University Course Timetabling Using ACO: A Case Study on Laboratory Exercises

Vatroslav Dino Matijaš, Goran Molnar, Marko Čupić, Domagoj Jakobović, and Bojana Dalbelo Bašić

Faculty of Electrical Engineering and Computing,
Unska 3, 10000 Zagreb, Croatia
University of Zagreb
dinomatijas@gmail.com, {goran.molnar2|marko.cupic|
domagoj.jakobovic|bojana.dalbelo-basic}@fer.hr

Abstract. The ant colony optimisation metaheuristic has shown promise on simplified artificial instances of university course timetabling problems. However, limited work has been done applying it to practical timetabling problems. In this paper, we describe the application of the ant colony optimisation to a highly constrained real–world instance of the university course timetabling problem. We present the design of the memory–efficient construction graph and a sophisticated solution construction procedure. The system devised here has been successfully used for timetabling at the authors' institution.

1 Introduction

Almost every type of human organisations is occasionally faced with some form of timetabling tasks. The University Course Timetabling Problem (UCTP) and its variations are parts of the larger class of timetabling and scheduling problems. A large number of university timetabling problems have been described in the literature, and they differ from each other based on the type of institution involved, the entities being scheduled and the constraints in the definition of the problem.

Due to inherent complexity and variability of the problem, most real–world timetabling problems are NP–complete [1]. This calls for use of heuristic algorithms which do not guarantee an optimal solution but can usually generate solutions that are good enough for practical use. Due to their manageability and good performance (if properly implemented), metaheuristic techniques have shown to be particularly suitable for solving these kinds of problems.

In this work[1], we focus on the *Laboratory Exercises Timetabling Problem* (LETP), framed as an example of the university course timetabling problem. The work described is a part of the research on two different metaheuristics for

[1] This work has been supported by the Ministry of Science, Education and Sports, Republic of Croatia and under the grants No 036-1300646-1986 and .....

timetable construction: *genetic algorithm* [2] and *ant colony optimisation metaheuristic*. Constructing a system for solving instances of LETP is challenging both technically and administratively. Along with the efficient timetable production system, a feature–rich model of the timetable constraints is devised. It is necessary for the system to be able to account for a complex set of constraints because various departments at our institution have very different ideas about what a good timetable should look like. The definition of LETP is devised in close coordination with the departments and is constantly evolving. Therefore, a good modular design that supports changes in the problem definition is necessary.

Our approach is based on the Ant Colony Optimisation metaheuristic (ACO), proposed by Dorigo et al. [3]. It is a distributed stochastic probabilistic constructive search procedure, inspired by the foraging behaviour of ants. It utilises a problem representation in the form of a *construction graph*, on which suggested solutions are constructed by artificial ants. ACO has already shown promise for various timetabling applications. It has been successfully applied to various artificially generated UCTP datasets in [4, 5] and in [6] its performance is compared with that of several other metaheuristics. ACO has also been applied to artificially generated instances of the examination timetabling problem, for example in [7]. However, as noted in [8],

> *"A major identified weakness in the current approach to Operational Research is described as follows: a gap still remains between the output of a successful research project and what is needed for direct use by industry. In general, the area of educational timetabling is one such area."*

Solving real such problems is a difficult task that involves modelling and handling various kinds of constraints which are usually simplified in academic problems. Moreover, only a few, relatively basic UCTP representations have been proposed in literature for use with ACO. It has been shown that these representations do not satisfy the requirements imposed by complex, large LETP instances.

This work is an effort to bridge this gap between the theory and practice of automated timetabling and to build an ACO–based system that satisfies the timetabling needs of our institution. The result of our work is an ACO–based timetabling system suitable for practical use.

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

2 University Course Timetabling Problem

2.1 Problem Statement

The timetable construction problem is a combinatorial optimisation problem that consists of four finite sets: (*i*) a set of meetings, (*ii*) a set of available resources (e.g. rooms, staff, students), (*iii*) a set of available time–slots, and (*iv*) 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. The University Course Timetabling Problem (UCTP) is a timetabling problem where the given data consists of a set of students and sets of courses that each of the students needs to attend. A *course* is a set of events that need to take place in the timetable. The main characteristic that distinguishes the university course timetabling problem from other types of timetabling problems is the fact that students are generally allowed to choose the courses in which they wish to enrol [9]. 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.

The above description of the UCTP defines a broad range of problems, whose complexity significantly depends on the specific constraints defined. Particular timetabling applications are usually focused on a more strictly defined subset of the problems, as the constraints and dimensions of the problem vary among institutions. We use the same approach, giving a detailed formal description of the problems for which our application is designed.

2.2 Definition of Laboratory Exercise Timetabling Problem

The *laboratory exercise timetabling problem* is defined 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 the university, R is a set of *rooms*, E is a set of *events* that need to be scheduled, S is a set of attending *students*, and C is a set of *constraints*. We assume that the durations of all the events can be quantified as multiples of a fixed value of time that we call a *time–quantum*. A *time–slot* is defined as one or more consecutive time–quanta in the timetable. The duration of the quantum reflects a trade–off between the precision of scheduling and the size of the search space.

The set of limited assets (resources) shared among the different exercises is denoted L. For each resource $l \in L$, a fixed number of workplaces can use the resource concurrently.

Each room is defined as a set of *workplaces*, atomic room resources 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. For each of the events, the desired number of students per workplace is defined. Since some rooms may not be available all the time, a set of time quanta $T_r \subseteq T$ in which the room is available is defined for each room.

Events have the following set of properties:

- Each event e has a duration, denoted $dur_e \in \mathbb{N}$, a multiple of a time quantum.
- Each event e has an acceptable room set $R_e \subseteq R$.
- Each event e has a suitable time quanta set, denoted $T_e \subseteq T$.
- The set of limited assets used by the event is denoted $L_e \subseteq L$.

– The number of staff available for each event and the number of staff needed for event e when held in the room r are given. Because each of the departments at our institution produces staff timetables independently of our system, staff is not defined as a separate entity of the LETP.
– An ordering relation, denoted $\succ_d$ can be defined for a pair of events. The relation $e_2 \succ_d e_1$ is true if and only if e_2 needs to be scheduled at least d days after e_1.
– The maximum number of rooms to be used concurrently for the event e can be defined.
– An event timespan can be defined to ensure that all of the time–quanta in which the event is scheduled are within a specified time interval.
– For each event e, the number of students per workplace is denoted $spw_e \in \mathbb{N}$.

The set S is the set of students that are to be scheduled. Each student $s \in S$ has the following set of properties: (i) a set of time quanta $T_s \subseteq T$ when each student s is free, and (ii) a non–empty set of events he or she needs to attend, denoted $E_s \subseteq E$.

The requirements of the courses are represented by 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:

– All of the properties of limited assets, rooms, events and students need to be satisfied in the timetable.
– Each room can only be occupied by one event at a time.
– Students can attend only one event at a time.
– Each event e occupies dur_e consecutive quanta of the room.
– At most $size_r \cdot spw_e$ students can be placed in room r used for the event e.
– Enough teaching staff must be available to attend each event.

The set of soft constraints C_s contains one element: the students must attend all the events they are enrolled in. Defining this constraint as soft may seem irrational, but the reasoning behind this is as follows: 'hard' constraints are simply those that are satisfied at all times in any solution suggestion 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 a solution that satisfies at least the hard constraints as defined above.

3 Solving LETP Using Ant Colony Optimisation

3.1 Construction Graph

The main issue in applying ACO to a problem is to find an appropriate representation that can be used by the artificial ants to build solutions [3]. This

representation is called the *construction graph*. To ensure that the problem representation is suitable for large instances of LETP, memory–efficiency is the main design goal. The construction graph we devised can be seen in figure 1. Semantically, each of the nodes represents one of the following: a student, an event or a *dock node*. A dock is an ordered pair of the room and beginning time in which an event can be scheduled, e.g., (ComputerLab-2, (2010-09-08, 10:00)).

An edge connecting a dock node and an event node means that the event can be scheduled in that time and place. This means that the dock represents the room and time that are suitable for that event. Dock nodes are connected to student nodes as well. Student nodes are only connected to docks under the following conditions: (i) the dock is connected to at least one of the events the student needs to attend and (ii) the student is free from pre–assignments in time–quanta represented by the dock and the dur_{e_s} consecutive time quanta. The event e_s is the shortest event enrolled by the student that can be held in the aforementioned dock, and its duration is denoted dur_{e_s}. To each of the graph's edges, a pheromone concentration value τ_{ij} and a heuristic information value η_{ij} are assigned. The LETP solution is a timetable with all of the events and students scheduled into the appropriate docks. A candidate solution (timetable) is represented as a subset of edges of the construction graph, connecting the timetable building blocks into a specific timetable.

For larger problem instances, the size of the construction graph can be considerable, and the efficiency of the search procedure strongly depends on the size of the graph. To reduce the size of the search space, an additional preprocessing step is performed. During that step, edges representing solution components of poor quality are removed. More precisely, an edge is removed if the number of students who can attend the corresponding event is less than 80% of the room capacity. Note that this value may vary in different problem instances. In the final step, all isolated docks are removed.

Hard constraints are included in the construction procedure through the *constraint fence* layer. This layer dynamically masks the edges of the construction graph that lead to infeasible solutions, based on each incomplete solution (partial tour) of the ants. Thus, ants moving through the graph are producing only feasible timetables. The construction graph is a very large structure, while the constraint fence is a small and expandable one. Ants access the graph exclusively through the constraint fence. They may move only on paths that are allowed by the constraint fence, and they behave as if the edges that are not allowed by the constraint fence did not exist at all. While the construction graph remains constant throughout the execution time, each ant that is moving through the graph is given its own instance of the constraint fence. The constraint fence evaluates each edge based on the desired set of timetable constraints. Each of the constraints from our library is implemented as an independent module. This makes it easy to update the constraints and to add new ones if needed. The computational cost of the constraint fence depends on the set of constraints used. If used with our current library, its computational complexity is $O(|R|)$, where R is the set of rooms in the LETP instance.

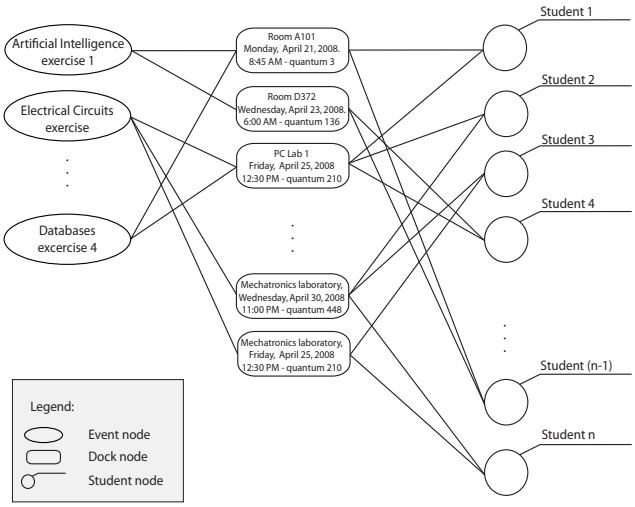

Fig. 1. LETP construction graph. Edges represent partial assignments of students and events to times and rooms. Note that the construction graph is not fully connected as some of the edges are implicit and implemented in the tour construction procedure.

3.2 Algorithm Description

Our approach uses a $\mathcal{MAX}-\mathcal{MIN}$ Ant System [3], since such systems have shown great promise on various different problems, including artificially generated timetabling problems [5]. A colony of m ants is used. At each algorithm iteration, each ant constructs a complete timetable (a candidate solution). In each of the generated solutions, all of the hard constraints are satisfied.

In choosing solution components, the probability p_{ij}^k that the ant k, currently at node i will choose the node j is calculated using the *random proportional rule* given by

$$p_{ij}^k = \frac{[\tau_{ij}]^\alpha [\eta_{ij}]^\beta}{\Sigma_{l \in \mathcal{N}_i^k} [\tau_{il}]^\alpha [\eta_{il}]^\beta} \ , \qquad j \in \mathcal{N}_i^k \ , \tag{1}$$

where τ_{ij} is the pheromone trail value on the edge connecting node i to node j, η_{ij} is the heuristic value of that edge, α and β are the parameters which determine the relative influence of pheromone trail and the heuristic information, and $\mathcal{N}_i^k$ is the feasible neighbourhood of ant k when it is at node i.

The pheromone trail between dock node i and event/student node j marks the desirability of scheduling that event/student in that dock. In each algorithm iteration only one of the ants updates the pheromone trails based on the quality of the constructed solution candidate. The heuristic value η_{ij} is controlled by

the constraint fence for each ant. It is important to note that the heuristic value is dynamically modified throughout the algorithm execution. It is set to zero when it is determined that the constraints of the problem would be violated if the edge (i, j) were included in the the tour, and to one otherwise.

Solutions are constructed one event at a time and the scheduling of each event is performed in two phases. In the first phase, the event is placed into suitable rooms and time–slots. This is done by assigning a sufficient number of docks to the event, one dock at a time, until a sufficient amount of room is reserved for the event. The decision of which dock to assign to the event is a biased random choice that is influenced by the pheromone trail in the feasible neighbourhood of the event node. To determine the probability that an edge will be selected, the random proportional rule is used. In the second phase of the scheduling of the event, students are assigned to the docks chosen in the first phase. The decision of which student is to be placed in which dock is also biased by the pheromone trails deposited by previous ant generations, using the random proportional rule. After a partial timetable for a single event is constructed, the ant constructs partial timetables for all of the events that have not yet been scheduled. The order in which the ants schedule the events is heuristically determined at the start of each iteration of the algorithm, so that the events with more unscheduled students in the best tour of the previous iteration have greater priority. Each of the solutions proposed by the ants is improved using the local search routine. The local search rearranges the students among the docks assigned to the same event. It tries to find suitable docks for students that are not scheduled to the event they need to attend. The search is performed by checking whether room can be made by for an unscheduled student by switching one of the scheduled students into another dock.

After the solution construction and local search is finished for all ants, each of the solutions is evaluated. We define the *penalty function* as the total number of unscheduled obligations in a candidate solution (the exact definition of an obligation is given in Section 3.3). The number of unscheduled obligations is also used as the optimised variable. After the evaluation, the pheromone updating step is performed as described in the following subsection.

3.3 Pheromone Update

The pheromone trail value is updated at the end of each algorithm iteration. The pheromone is updated by the best–so–far ant or the iteration–best ant. The probability that the best–so–far ant is allowed to update the pheromone trail is 5%. Unlike usual ACO approaches, the pheromone gain value in our approach is not the same for all of the edges of a single tour. Instead, the pheromone gain is defined for each event individually. The quality value is assigned to each event of the timetable by counting the number of obligations left unscheduled for that event. The intention is to improve the search process by using the quality of the solution components to determine the pheromone gain value for individual edges in the solution suggestion.

Usually, more than one event can be scheduled in a single dock. Nevertheless, after event e is scheduled in dock d, no other event can be scheduled into d. Therefore, the quality of a partial schedule of a single event cannot be measured without considering its influence on other events. For example, suppose that event e_1 can be scheduled in the set of docks $\{d_0, d_1, d_2\}$ and event e_2 can be scheduled only in dock d_1. Dock d_1 may seem to be appropriate for event e_1 since it would leave zero students that need to attend e_1 unscheduled. However, this is a very poor choice considering that d_1 is the only suitable dock for e_2, so assigning it to e_1 would leave all of the students who need to attend e_2 unscheduled. The level of influence between two events is modelled by the influence function $f : E \times E \rightarrow [0, 1]$. When $f_{a,b} = 0$, event a has no influence on event b, while $f_{a,b} = 1$ means that event a is scheduled in the entire set of docks suitable for the event b. More formally, the influence of event a on b, denoted $f_{a,b}$, is defined as:

$$f_{a,b} = \frac{|chosenD_a \cap D_b|}{|D_b|} \ ,$$

where $chosenD_a \subseteq D_a$ is a subset of docks in which event a is scheduled in a given suggested solution, and $D_b \subseteq D$ is the set of docks suitable for the event b.

We use the number of unscheduled *obligations* as the optimised variable. An obligation is defined as an assignment of a given student to one of the laboratory exercises he or she needs to attend. In our problem representation, this is done by assigning students to the dock nodes during the second phase of the construction procedure for a single event, as described in Section 3.2. The solution quality function Q_e for the event e and the solution suggestion Sug is defined as:

$$Q_e = \frac{assignedObligations(e, Sug)}{numberOfObligations(e)} \cdot spaceEfficiency(e, Sug)^3 \ ,$$

$$spaceEfficiency(e, Sug) = \left[\frac{assignedObligations(e, Sug)}{reservedSeats(e, Sug)} \right] \ .$$

The space efficiency factor ensures that better solutions use the dock space more efficiently. The pheromone gain for each event $\Delta\tau_e$ is given by

$$\Delta\tau_e = \left(\frac{\sum_{e_i \in E} (f_{e,e_i} \cdot Q_{e_i})}{|E|} \right)^4 \ , \qquad e \in E \ .$$

The pheromone gain $\Delta\tau_e$ is deposited on the edges of the tour connecting the event e to the dock d and on the edges connecting a student s to any of the docks in which the event is scheduled.

3.4 $\mathcal{MAX} - \mathcal{MIN}$ Ant System Parameters

Several configurations of the metaheuristic were evaluated, and the best results were achieved using the settings in Table 1. The pheromone values are initially

set to τ_{max}. The pheromone trails are updated after each algorithm iteration, as described in Section 3.3, and evaporation is used for each edge, according to the rule $\tau_{ij} \leftarrow (1 - \rho) \cdot \tau_{ij}$.

Our system uses either 10000 iterations or $penalty = 0$ as the stop criteria. To prevent stagnation, if the best solution found is not improved after 125 consecutive iterations, the pheromone trails are reset by setting pheromone value on each edge back to τ_{max}.

Table 1. Ant colony optimisation algorithm parameters

Parameter	Value	Parameter	Value
number of ants (m)	5	ρ	0.02
α	1.0	$\tau_{max} = 1/\rho$	50
β	1.0	τ_{min}	0.5

4 Results

The system described here was successfully applied to the laboratory exercises timetabling problem at the authors' institution. The performance of the algorithm on several datasets is presented in Table 2. These problem instances have different durations and widely varying numbers of events and attending students. Two values, $S_{e,s}$ and Ts are given as measures of the problem instance complexity. The *student event sum*, denoted $S_{e,s}$, is defined as the aggregate number of events that each of the students needs to attend. More formally, $S_{s,e} = \sum_{s \in S} |E_s|$, where E_s is the set of events that student s needs to attend. The *timespan* of the problem instance, denoted Ts, is defined as the number of days on which the events need to be scheduled. For each dataset, 30 independent runs were performed, and each run was limited to one hour. For the purpose of comparison, the maximum run–time limit is added to the usual stop criteria described in Section 3.4. Note that these datasets are instances of real–world timetabling problems that had appeared at our faculty. Anonymised version of these datasets is publicly available for download at `http://morgoth.zemris.fer.hr/jagenda`. To illustrate the effectiveness of our approach, the results are compared with a GRASP technique (Table 2). The tested GRASP technique uses a construction search (different from the search procedure defined for our ants) to build solutions satisfying hard constraints, after which a local search that optimises the schedule of students is performed. We used the Mann–Whitney test to check the H0 hypothesis that the distribution functions of the algorithm performances were the same for the results of both the ACO and GRASP techniques. For each of the datasets, the p values were well below 0.05. On each problem instance, with very high statistical significance, we conclude that the ACO technique performs better than the GRASP technique.

Note that although the Genetic algorithm has also been applied to the LETP problem as a part of a sister project at our institution [2], the performance of these approaches cannot be directly compared since different quality measures and optimised variables are used in these approaches.

Table 2. Algorithm performance on different *laboratory exercise timetabling problem* instances.

			ACO penalty			median comparison	
instance	$S_{s,e}$	Ts	min	max	st.dev. (σ)	ACO	GRASP
C1	2104	5	66	150	23.31	102	330
C2	7081	9	1	14	3.46	8	71
C4	4868	5	5	73	12.42	13	125
C8	5430	4	34	146	31.00	58	190
C12	5934	5	32	78	9.31	41	333

In some problem instances, a solution where all students are scheduled could not be found. This was usually caused by a constellation of conflicting events, or events with infeasible requirements posed by the course organisers. In such instances, the system is used as a tool for identifying the problematic events. In practice, the process of scheduling is usually an iterative process of querying the system for the best results, interpreting those results, and allowing the staff to make informed decisions.

5 Conclusion

This paper presents a case study of applying ACO metaheuristic for solving a complex large–scale timetabling problem at our institution. Our solution uses a relatively general problem representation, suitable for different types of institutions. We present an innovative, memory–efficient problem representation that is appropriate for large problem instances. Moreover, the modular design of the constraint library facilitates the addition of new constraints. It makes the system manageable, which is extremely important for practical timetabling applications. The exact problem we are solving is formulated as the laboratory exercise timetabling problem, a subset of the university course timetabling problem.

This work arose out of the specific timetabling needs of one institution. However, the approach described here is not limited to LETP, since it shares many commonalities with other UCTP instances. Thus, it is likely that the challenges we faced, such as reducing the memory footprint of the construction graph or ensuring the ease of adaptation to problem modifications, will also be faced by other researchers. Other authors may use our approach without modifying the

problem representation. The only necessary adaptation may be the implementation of additional constraints that are not supported by our current library.

Furthermore, since prior work on ant colony optimisation mainly considered artificially generated UCTP instances, our work proves that ACO can be highly successful in solving real–world timetabling problems. It is an effort to help bridge the gap between theoretical research and practical adaptation of metaheuristic techniques that is currently so prevalent in the area of automated timetabling. Our work can also be viewed as an additional confirmation that ACO is not only an interesting academic research topic, but also a manageable and efficient approach able to solve highly complex real–world problems.

References

1. Cooper, T.B., Kingston, J.H.: The complexity of timetable construction problems. In: Proceedings of the First International Conference on the Practice and Theory of Automated Timetabling (ICPTAT '95). (1995) 511–522
2. Bratković, Z., Herman, T., Omrčen, V., Čupić, M., Jakobović, D.: University course timetabling with genetic algorithm: A laboratory excercises case study. In Cotta, C., Cowling, P., eds.: Proceedings of EvoCOP 2009 - 9th European Conference Evolutionary Computation in Combinatorial Optimization. Volume 5482 of Lecture Notes in Computer Science., Springer, Heidelberg (2009) 240–251
3. Dorigo, M., Stutzle, T.: Ant Colony Optimization. Bradford Books. The MIT Press (july 2004)
4. Socha, K., Sampels, M., Manfrin, M.: 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. Volume 2611 of Lecture Notes in Computer Science., Springer Verlag, Berlin, Germany (2003) 334–345
5. Socha, K., Knowles, J., Sampels, M.: A $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System for the University Timetabling Problem. In Dorigo, M., G. Di Caro, 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 (September 2002) 1–13
6. 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 DifferentMetaheuristics on the Timetabling Problem. In: The Practice and Theory of Automated Timetabling IV: Revised Selected Papers from the 4th International conference, Gent 2002. Volume 2740 of Springer Lecture Notes in Computer Science., Springer, Berlin, Germany (2003) 329–351
7. Azimi, Z.: Comparison of Methheuristic Algorithms for Examination Timetabling Problem. Applied Mathematics and Computation **16**(1) (2004) 337–354
8. McCollum, B.: University timetabling: Bridging the gap between research and practice. In E Burke, H.R., ed.: PATAT 2006 — Proceedings of the 6th international conference on the Practice And Theory of Automated Timetabling, Masaryk University (2006) 15–35
9. Gross, J.L., Yellen, J.: A Handbook of Graph Theory. Discrete Mathematics and Its Applications. CRC books (December 2003)