Ferko Project – Intelligent Support for Course Management at Faculty Level

M. Čupić and T. Franović
Faculty of Electrical Engineering and Computing
University of Zagreb
Unska 3, 10000 Zagreb, Croatia
Phone: +385 1 6129 548 Fax: +385 1 6129 653 E-mail: Marko.Cupic@fer.hr, Tin.Franovic@fer.hr

Abstract – In this paper we discuss how computers can be used to support faculty staff with regard to creating several kinds of high-quality schedules, at faculty level. Implementations of adequate evolutionary computation-based algorithms are being developed as part of the Ferko project. Various parts of the Ferko project have already been is use for several years by thousands of students and faculty staff members at the Faculty of Electrical Engineering and Computing (FER).

I. INTRODUCTION

When talking about the role of computers in education, today's mainstream research topics are focused on helping students to learn by creating various formats of learning materials (e-learning, m-learning, Intelligent Tutoring Systems, etc.) and helping teachers to better assess students knowledge (e-assessment systems providing adaptive and intelligent assessments techniques). However, even today, the majority of courses are still being taught in classrooms, while assessments are still being conducted on-campus under staff supervision. For all but small-sized universities, creating high-quality lecture, assessment, and laboratory exercise schedules poses a serious challenge.

Being part of faculty administration is definitely not an easy job. This is even more so considering all the tasks needed to be done in light of the new Bologna reform. Out of all challenging tasks, we will single out several that are performed at faculty level: creating a semestral lecture schedule for all courses and enabling students to exchange groups in case of conflicting schedules, creating weekly laboratory schedules for all courses, creating assessment schedules for all courses, and supporting course staff to manage make-up or deferred exams more easily. With regards to computational complexity, all of those problems are combinatorial NP, and solving them by hand – even in an attempt to obtain a best-effort quality schedule – is unfeasible.

Fortunately, adequate algorithmic techniques have been developed for dealing with NP class combinatorial problems. Most prominent candidates are biologically inspired meta-heuristics researched under the Evolutionary Computation umbrella, such as Evolutionary Algorithms (e.g. Genetic Algorithm) [1,2,3], Swarm Algorithms (e.g. Particle Swarm Optimization and Ant Colony Optimization) [4,5,6], Artificial Immune Systems [7,8,9], and Differential Evolution Algorithms [10,11].

The goal of Ferko, a course management and support system we are developing, is to improve the quality of studying and to leverage faculty administration and course staff of many of the common and tedious tasks. In this paper, we present work carried out as part of the development of Ferko, addressing most of the above-mentioned problems by providing intelligent optimization methods based on Evolutionary Computation algorithms, capable of automatically producing good-quality schedules. We comment on the performance of developed algorithms, and discuss how they can be used to provide an intelligent support for course management at faculty level.

II. MOTIVATION

Since the alignment with the Bologna process five years ago, the semestral schedule has been constantly adjusted, in an attempt to enable the creation of better and stress-free lecture, laboratory and assessment schedules (see Fig. 1). The current semestral schedule is divided into three terms, each having four or five weeks for lectures interwoven with laboratory exercises, followed by two weeks of assessments. At the end of the semester there is another week reserved for make-up exams. This is depicted on Fig. 1.c.

The lecture schedule is created first. At the present time, the lecture schedule for undergraduate students alternates each week for most courses, but not all. For graduate students, the lecture schedule is fixed. On both study levels (undergraduate and graduate) course enrollment is based strictly on prerequisites. Combined with the fact that there are a number of elective courses in which a student may enroll, all described results with a lecture schedule having a certain number of students with conflicts. In order to amend this, as part of Ferko, we have implemented a lecture group market-place; a fully automated web-based application where students can auction for a group which

(A)	(B)	(C)
LECT	LECT	LECT + LAB
LECT	LECT	LECT + LAB
LECT	LECT	LECT + LAB
LECT	LECT	LECT + LAB
LAB	LECT + LAB	LECT + LAB
ME 1	LAB + ME1	ME 1
LECT	ME 1	ME 1
LECT	LECT	LECT + LAB
LECT	LECT	LECT + LAB
LECT	LECT	LECT + LAB
LAB	LECT + LAB	LECT + LAB
ME 1	LAB + ME2	ME 2
LECT	ME 2	ME 2
LECT	LECT	LECT + LAB
LECT	LECT	LECT + LAB
LECT	LECT	LECT + LAB
LECT	LECT + LAB	LECT + LAB
LAB	LAB + FE	FE
FE	FE	FE
MuE	MuE	MuE

Fig. 1. Several semestral structures for lectures (LECT), midterm exams (ME), laboratory exercises (LAB) and final exams (FE).

better suits them (they can offer their current group, and perform the exchange by accepting other students' offerings). This mechanism is implemented in a generic way, so it is also used later for fine-tuning students' laboratory exercise schedules.

Once the lecture schedule is completed, several tasks must be performed. A schedule for midterm and final exams must be created, as well as a make-up exams schedule. Ideally, since each is scheduled in a two-week period, schedules could be reused. Unfortunately, more often than not, the three two-week periods are not equal due to various holidays, so a schedule must be created for each separately.

At the same time, in accordance with course-specific requirements laboratory exercise schedules must be created for each week separately.

Additionally, at single course level, some scheduled events can be skipped (either due to faculty staff or student absence), so a make-up event must be scheduled without causing conflicts with other student assignments. This is not a problem when the number of students is small, but can be very difficult otherwise.

In order to provide support for all of the described activities conducted after the initial creation of a lecture schedule, we implemented a number of algorithms and systems which are described in the following sections.

III. Laboratory Exercise Schedule

There are several reasons why the creation of a laboratory exercise schedule is inherently hard. First and foremost, a course can be enrolled at the same time by students simultaneously enrolled in many other courses, each with its own lecture schedule (which possibly alternates). So a laboratory exercise schedule must be created in a way which does not introduce new conflicts with previously scheduled obligations. For a course staff member to handle such a problem it would require taking into consideration previously defined schedules for each enrolled student, and trying to produce a non-conflicting laboratory schedule. Additionally, this would require some kind of coordination among various courses, in order to avoid a situation in which two or more courses simultaneously produce laboratory schedules which are non-conflicting with previously scheduled students' obligations but are conflicting mutually.

The other difficulties arise from the fact that different courses require different kinds of laboratory rooms (computer-equipped rooms, electronic labs, etc) whose availability is limited, and that some courses share licensed software for which there is a fixed and limited number of licenses available. More often than not, the number of available course staff members is small, which also prohibits the creation of schedules in which there are many laboratory rooms simultaneously scheduled to the same course, etc. Taking all of the before-mentioned constraints into consideration, it is better to allow each course to provide its laboratory requirements, and then to centrally create a schedule for all courses at the same time.

In an effort to support this, a web application has been created in which for each course, a course staff member can enter all of the requirements, which are briefly summarized as follows:

- the number of laboratory exercises in a given week (can be more than one, but occurs rarely),
- the duration of each laboratory exercises (it is typically between half an hour and four hours),
- a selection of students for each exercise (typically, all enrolled students),
- a set of laboratory rooms acceptable for each exercise,
- the number of students that can be placed in each room (room capacity is dependent on the course and type of laboratory exercise),
- the number of course staff members needed to be assigned to each room,
- the number of course staff members available for each laboratory exercise,
- laboratory exercise ordering (in case that more than one exercise for same course should be scheduled in the same week),
- laboratory exercise spacing (in case that more than one exercise for same course should be scheduled in the same week, this is the minimal amount of time enforced between the end of one exercise and the beginning of another),
- acceptable time-span for course laboratory exercises (e.g. a time-span of 2 days would require that if the laboratory exercises for the first group of a specific course start on Monday, the last group of students for the same exercise should be scheduled no later than on Tuesday).

A number of other constraints are also available, but will not be mentioned here.

Once the requirements are entered, centralized scheduling is started. Schedules are created separately for each week, since not all of the courses have laboratory exercises every week.

To create a schedule, requirements are collected from the web application, and then used by two implementations of scheduling algorithms. The first one is based on the hybrid Genetic Algorithm described in [12] and the second one is based on the Ant Colony Optimization algorithm. The GA-based algorithm in all candidate-schedules always assigns all students to laboratory exercises, even if such an assignment creates conflicts. For that reason, we say that the GA-based implementation has found a good schedule when the number of conflicts becomes zero. The ACO-based implementation, on the other hand, creates schedules in which students are assigned to a laboratory exercise only if such an assignment will not create a conflict. So the quality of the schedule is measured by the number of unassigned students. A typical run of the ACO-based algorithm is shown on Fig. 2. Sudden worsening at approximate time 6000 is result of algorithm restart due to detected stagnation, and is normal for this implementation. On the same figure there is also depicted a strong influence of the implemented local search method on schedule quality.

The parallelization of schedule creation, at the present time, is trivial: the algorithms are started on several computers, and usually left running during night. Then, the best found solution is used and inspected. If anomalies are found (e.g. scattered room assignments), the specification is adjusted, and the process is repeated until a satisfactory schedule is obtained. For weeks in which laboratory schedules must be created for many large courses (e.g. for a total of 25 courses), it is often impossible to find ones without conflicts with the lecture schedule. In that case, a schedule with the minimal number of conflicts is used. However, it should be noted that in that case the number of remaining conflicts is reasonably small. In case that the number of conflicts can not be lowered to an acceptable amount, problematic conflict-causing courses are identified, and in cooperation with course staff members the requirements are adjusted so that a low-conflict (or no-conflict) schedule can be created. One often identified source of conflicts is a densely constrained specification for long duration exercises (e.g. exercises constrained to only two days in a week, with duration of four hours). Once this is identified as a conflict source, simply allowing the algorithm to use more than just two days in a week usually resolves the issue. For reasonably simple schedules, the algorithm run is typically over in several minutes (up to half an hour). However, when dealing with many courses, it is typically run for 12 to 24 hours.

Finally, once the schedule is created, it is published in course calendars and students' personal calendars in Ferko. Then, course staff members can enable the usage of the market place for the scheduled laboratory group, allowing students to exchange scheduled terms and thus creating for themselves even higher quality schedules.

IV. Team Scheduling

One of the notorious problems, when dealing with scheduling, is the fact that there is no universal scheduler. Each course will have its own peculiarities, and for the majority, a framework can be created which is expressive enough (but at the same time efficient) to encompass them – but not all of them.

One such peculiarity is team scheduling. Our currently implemented laboratory exercise scheduler does not have built-in support for teams, so a new specialized scheduler was implemented, in order to provide us with team-centered scheduling capabilities. We will define a team as a predetermined group of students which must be scheduled atomically. Depending from course to course, in a laboratory room a team can occupy only one place, or as many places as there are students in it. An additional request came from one of the courses which needed a laboratory schedule: teams also have a predetermined course staff member. However, since the number of course staff members is smaller than the number of teams, multiple teams are assigned to each staff member.

Bearing this in mind, a laboratory schedule must be created for that course, so that there are no conflicts with the lecture schedule, and no conflicts with previously created laboratory exercise schedules for the majority of "regular" courses. Additionally, two teams being assigned to the same course staff member cannot be scheduled at the same time.

To cope with those requirements, a scheduler implementation was created, based on the Clonal Selection Algorithm [7,8,9] (a member of Artificial Immune Algorithms). This implementation was successfully used to create a schedule for three laboratory exercises. The work of the algorithm used to schedule 47 teams in a single day is illustrated on Fig. 3. In this case, the average number of students per team was 8, and there were only 5 course staff

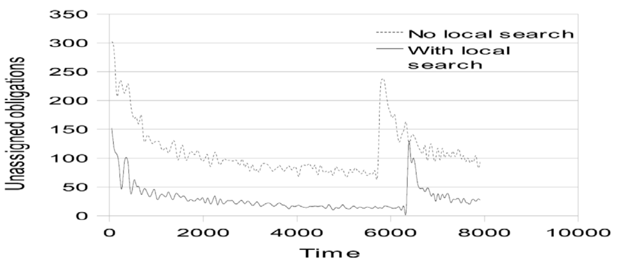

Fig. 2. Typical run of ACO-based laboratory exercise scheduler, and influence of local search on performance.

members, each having an average of 9 teams. The quality of the schedule is measured as a total number of conflicting minutes among the created schedule and the previously created lecture schedule and laboratory schedules, for the scheduled students. The algorithm quickly satisfies all of the hard constraints (e.g. that two teams being assigned to the same course staff member cannot be scheduled simultaneously), so we plotted only conflicting minutes. The best found solution had 330 minutes of conflicts with previously created schedules (only 11 conflicting students).

Three runs depicted on Fig. 3. illustrate a run with no local search, a run with local search applied only to the current best schedule, and a run with local search applied to the current best schedule as well as to 1% of the other created schedules. It is clearly visible that the last scenario gives the best results.

V. Make-up Assignments Scheduling

The FerSched system is a Ferko module developed in order to assist course staff members in creating students' schedules for make-up assignments. The make-up assignments include laboratory exercises, lectures, quizzes, etc. The system consists of two main components: a Java applet [13] for the definition of schedules, and a locally runnable application which creates the schedules from the input data.

The applet is a component of the Ferko system, and enables the user to define in detail the constraints of each schedule that is to be created. It is possible to define the constraints on three levels: the plan, event and term levels. Each level allows the user to define three main constraints: the available students, time spans and rooms. If a constraint is defined on one of the levels, it is locked and

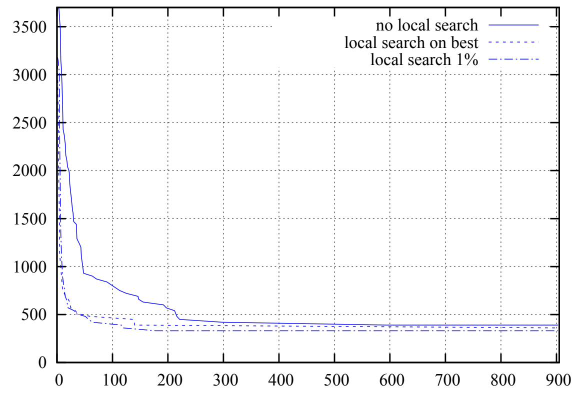

Fig. 3. Schedule quality for team scheduler, and influence of local search method.

cannot be defined on any of the other two levels.

The laboratory exercises for demonstrators for the Digital logics course could serve as an illustrative example. Firstly, the user defines all the students on the plan level (in the example, there are 51 of them). The time span can as well be defined on the plan level, since both events should occur in the same week. Secondly, the user creates two events: the Theory event, and the Practice event. Then, it is necessary to define the durations of both events (2 hours for Theory, and 1 hour for Practice), and the preconditions (Theory must occur at least 1 day before Practice). The next step is to define the available rooms for each of the events (we can limit the room capacity for the rooms assigned to Practice to 18 students, in order to avoid overcrowding). Finally, the user selects the given distribution (to be explained later) for both events, and selects 1 term for Theory, and 3 for Practice. After following these steps, the user has created a valid plan, which is ready for processing.

The plan level represents the schedule as a whole, and every solution is eventually presented in form of a plan. Each plan consists of a number of events. An event is one make-up assignment and has its own duration, distribution and preconditions. A term is a realization of an event, meaning that it is uniquely defined by its start time and duration, room, and a list of assigned students.

The plan level offers the user the possibility to define only the main constraints, which are then transferred to each of the events. The event level offers the possibility of defining more constraints, apart from the main ones. These constraints include the duration, distribution and precondition of the event. The duration defines the maximum duration of each term representing the event. The distribution can be random or given. A random distribution enables the user to define the minimal and maximal number of terms, while the given distribution offers the possibility to manually create a number of terms for the event and define term constraints. The preconditions determine the events which must occur before this event, and the time margin between events. The term level, similarly to the plan level, allows only the definition of the main constraints.

After the successful definition of a schedule, the applet extracts from Ferko the data containing information about student and room occupancies and creates an executable Java application (.jar) which creates the schedule.

The schedule creation application processes the occupancy data provided from Ferko and attempts to create a valid schedule (the term will be defined later in the text) with the aid of evolutionary algorithms. The algorithms implemented in the application are as follows: Bee Colony Optimization [14], Clonal Selection Algorithm [15], Genetic Algorithm [12], Harmony Search [16], Particle Swarm Optimization [17] and Stochastic Diffusion Search [18].

All of the algorithms implement the same interface which enables them to easily exchange temporary solutions. Solution exchange is a mechanism which provides the algorithms with quality solutions which enhance and diversify their population. The algorithm which is to execute the next iteration of the schedule generation process is randomly chosen based on roulette wheel selection, where the intervals corresponding to every algorithm are set by the user when starting the schedule generation process.

During the process, the user is provided progress feedback by means of a JFreeChart [19] bar chart (as shown in Fig. 4) whose bars represent each of the following constraints: unsatisfied preconditions, room conflicts, student conflicts, overcrowded rooms, number of terms and vacant places. The constraints are divided into hard and soft constraints. The hard constraints (unsatisfied preconditions, room and students conflicts and overcrowded rooms) represent the constraints which must be satisfied in order for the schedule to be valid. The remaining two constraints are a measure of optimality of the solution. The student and room conflicts are presented in units denoting the number of 15 minute intervals where conflicts exist.

When the user is satisfied with the result shown as feedback on the screen, he can stop the process, and is then provided with an overview of the schedule, including the remaining student conflicts in case a valid schedule could not be generated. The user is also offered the option to export the generated schedule into XML, which is then transferred to Ferko in order to create a new student assignment.

Fig. 5 shows a graph representing the change in constraint values through iterations. The most notable decrease is present in the number of student conflicts, which decreases exponentially.

The algorithms which have proved to work successfully together in terms of solution exchange are the Genetic Algorithm, Clonal Selection Algorithm and Harmony Search. The remaining algorithms have longer initialization times, and thus produce better results when used alone.

As far as the conducted tests show, the algorithms are relatively successful in schedule creation. The distribution of 51 students in 2 events, having 1 and 3 terms, respectively takes approximately 8 seconds to complete, while a schedule containing more than 600 students and 4 events with 4 terms each presents a notable decrease in conflicts in approximately 45 seconds.

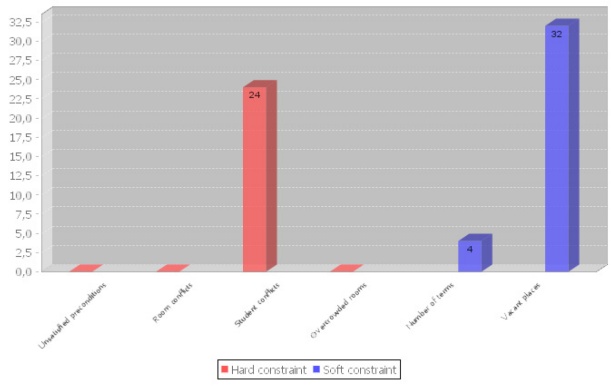

Fig. 4. An example of the feedback provided to the user during FerSched schedule creation.

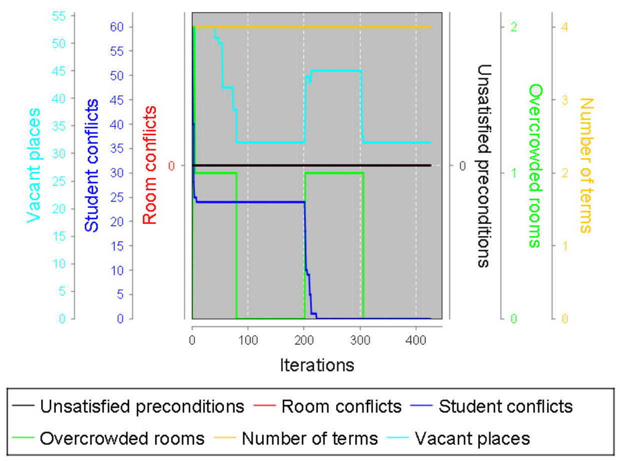

Fig. 5. Constraint change through iterations during a single run of FerSched schedule creation.

VI. Examination Scheduling

One of the most challenging scheduling problems is to produce a high-quality exam schedule (often referred to as examination timetable). This is particularly important since at the present time, all schedules are condensed in two-week periods, and for students those are highly stressful times. Since the final grades are generated by ranking, each score counts, and it is not enough to solve the exams well – for an excellent grade it is important to solve it better than most of the other students. Also, frequently there are thresholds on midterm exams and final exams; students who fail to achieve the threshold fail the course completely. They get another chance on make-up exams, but a student that fails two thresholds on the same course automatically fails the course and prolongs the study for another year.

So in order to produce a high-quality schedule, it is not enough to only ensure that all students can attend to all scheduled exams for the enrolled courses (this we will proclaim to be a hard constraint). It is of paramount importance for a student to have equally distributed exams throughout the examination weeks. Namely, everyone can have a bad day – but on that day, ideally, there should be no more than one exam scheduled. And if possible, the day after that, there should be no exams either. So having those ideas in mind, we defined the following hard constraints:

- no exams should be scheduled at the same time if they share enrolled students (student conflicts),
- for each available term, multiple course exams can be scheduled, but term capacity cannot be exceeded,

and the following soft constraints:

- it is bad for a student to have more than one exam on the same day,
- it is bad for a student to have more exams on two (or more) successive days.

Soft-constraints are handled by multiplying the number of violations with the appropriate factor, and then summing it up. The goal of scheduling is to find such a timetable (which satisfies all of the hard constraints) from a set of all valid timetables which has the best quality (minimal violation sum).

To tackle this problem, we have developed two implementations. One is based on Genetic Algorithm [20], and the other one on Ant Colony Optimization. At our institution, exam schedules created this way have now been in use for two years. Two exam weeks are separated in a total of 40 non-overlapping terms in which around 130 courses are scheduled.

A typical run of both schedulers is shown on Fig. 6. As we can see, there is almost no difference among the average run of the GA-based scheduler and the ACO-based scheduler. The quality of the obtained schedules is also reasonably high. Just as an illustration, the majority of students on the first year of the undergraduate study program are enrolled in five courses, and the created schedule had exams distributed on Wednesday and Friday of the first week, and on Monday, Wednesday and Friday of the second week. While talking to students on schedule quality, many of them said that not all course exams are equally difficult, and that in order to produce even better schedules, this kind of information (subjective or perceived exam difficulty) should be taken into account, and this is something we are currently trying to implement.

Make-up exams (scheduled at the end of semester) require separate handling. Since the available time is only half of the time available for the final exams, courses are scheduled more densely. However, considering the fact that only a moderately small number of students are actually present on make-up exams, this is not a problem. During the make-up exam scheduling, a care must be taken not to schedule a make-up exam for a course too close to the final exam of the same course.

An additional challenge we are introducing this semester is a fully automated exam timetable with automatically allocated exam rooms. This is a shift from "seeing a term as something capable of accepting 1000 students" to "seeing a term as a collection of a number of rooms each with its own capacity". It might seem that there is nothing new here. However, looking from this perspective, we can now additionally try to create such a schedule that requires a globally minimal number of course staff members, and even better, to allocate rooms from several buildings in such a way to try to ensure minimal distance for a course staff member visiting all allocated rooms cyclically and answering to students' questions. If five courses are scheduled in the same term, solutions to five Travelling Salesperson Problems (TSPs)

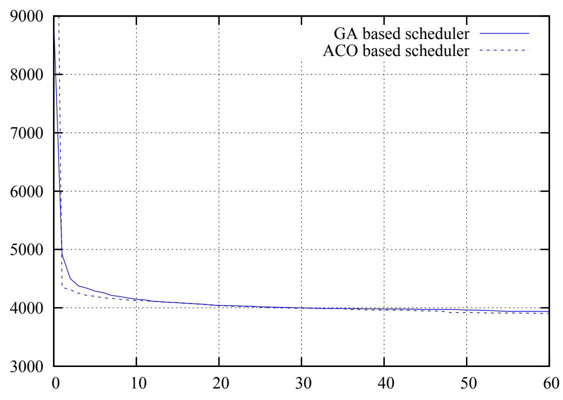

Fig. 6. The quality at typical run of GA-based and ACO-based exam scheduler implementations (sum of weighted soft constraint violations)

should be found, just to compare two schedules. Combined with the fact that there are courses requiring allocation of twenty or more rooms, and that TSPs are NP-hard combinatorial problems, a simplification is being developed, that will not produce optimally short routes, but still acceptable ones.

VII. CONCLUSION AND FUTURE WORK

In this paper we have described a number of situations constantly being presented to faculty staff members which are, in fact, various cases of scheduling problems. Formally, most of those problems are, from the computation complexity point of view combinatorial NP-hard problems.

As part of the Ferko project a number of algorithms are being developed in order to tackle those problems and to help create better conditions for students and faculty and course staff. Some of those algorithms are presented in this paper.

It is important to stress that schedules created this way significantly helped us to conduct the alignment with the Bologna process, to organize laboratory exercises in prerequisite based course-enrollment and to try to create better and less-stressful exam timetables.

At a single course level, significant efforts were also made in order to help course staff to better and more easily organize all course-related events. Specifically, algorithms were developed and integrated into Ferko which now allow course staff members to relatively easily schedule various kinds of make-up assignments without causing conflicts with previously scheduled students' obligations.

As part of future work, there is much room left for enhancements: either from the standpoint of expressiveness, or from the standpoint of efficiency, quality, and better and more usable integration with Ferko. All of those are areas of future research.

ACKNOWLEDGMENT

The authors would like to acknowledge the students Mihej Komar and Toni Pivčević for their work on the application for gathering laboratory schedule specifications, the students Zlatko Bratković, Tomislav Herman, Vjera Omrčen, Vatroslav Dino Matijaš and Goran Molnar for their work on the laboratory schedulers, the team of students who created the FerSched module, and all the others participating in the development of Ferko.

REFERENCES

[1] K. A. De Jong, *Evolutionary Computation,* MIT Press, Cambridge, 2006.

[2] K. Deb, *Multi-Objective Optimization using Evolutionary Algorithms*, Wiley, New York, 2009.

[3] M. Affenzeller and S. Wagner, *Genetic Algorithms and Genetic Programming*, Modern Concepts and Practical Applications, CRC Press, Boca Raton, 2009.

[4] R.C. Eberhart and J. Kennedy. "A new optimizer using particle swarm theory", *Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, Nagoya, Japan, pp. 39-43, 1995.

[5] M. A. Montes de Oca, T. Stützle, M. Birattari, and M. Dorigo. "Frankenstein's PSO: A Composite Particle Swarm Optimization Algorithm", *IEEE Trans. on Evolutionary Computation*. 13(5):1120-1132, 2009.

[6] M. Dorigo, and T. Stützle. *Ant Colony Optimization*. MIT Press, Cambridge, MA, 2004.

[7] L.N. de Castro and F.J. Von Zuben, "The Clonal Selection Algorithm with Engineering Applications", *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '00)*, Workshop on Artificial Immune Systems and Their Applications, Las Vegas, Nevada, USA, pp. 36-37, 2000.

[8] L.N. de Castro and J. Timmis. *Artificial Immune Systems: A new computational intelligence approach*, Great Britain: Springer-Verlag, 2002.

[9] V. Cutello and G. Nicosia. "Chapter VI. The Clonal Selection Principle for In Silico and In Vitro Computing", *Recent Developments in Biologically Inspired Computing*, eds. Leandro Nunes de Castro and Fernando J. Von Zuben. Hershey, London, Melbourne, Singapore: Idea Group Publishing, pp. 104-146., 2005.

[10] V. Feoktistov, Differential Evolution. In Search of Solutions., Springer, New York, 2006.

[11] K.V. Price, R.M. Storn, J.A.Lampinen, *Differential Evolution*. A practical Approach to Global Optimization. Springer-Verlag, Berlin, 2005.

[12] Z. Bratković, T. Herman, V. Omrčen, M. Čupić, D. Jakobović. "University Course Timetabling with Genetic Algorithm: a Laboratory Excercises Case Study". In: *Proceedings of the 9th European Conference*, EVO COP 2009, Tübingen, Germany, 2009.

[13] Java Applet Technology, http://java.sun.com/applets/

[14] C. Chong, M.H. Low, A. Sivakumar, K. Gay: "A bee colony optimization algorithm to job shop scheduling". In: *Proceedings of the 2006 Winter Simulation Conference*, Monterey, CA USA (2006) 1954–1961

[15] J. Brownlee: "The Clonal Selection Classification Algorithm (CSCA)". In: *Tech. Report 2-01*, Centre for Intelligent Systems and Complex Processes, Faculty of Information and Communication Technologies, Swinburne University of Technology, Victoria, Australia, (2005)

[16] Z.W. Geem, *Music-inspired Harmony Search Algorithm Theory and Applications*. Springer-Verlag, Germany, 2009.

[17] M. Clerc: "The swarm and the queen: towards a deterministic and adaptive particle swarm optimization". In: *Proceedings of the I999 ICEC*, Washington, DC USA, 1951-1957, 1999.

[18] K.D. Meyer, S.J. Nasut, M. Bishop: "Stochastic diffusion search: Partial function evaluation in swarm intelligence dynamic optimization". In: Abraham, A., Grosan, C., Ramos, V., eds.: *Studies in Computational Intelligence*. Volume 31. Springer-Verlag, Germany, 2006.

[19] JFreeChart, http://www.jfree.org/jfreechart/

[20] M. Čupić, M. Golub, D. Jakobović. "Exam Timetabling Using Genetic Algorithm". *Proceedings of the 31st International Conference on Information Technology Interfaces*, University Computing Centre, SRCE, 357-362, 2009.