Some Novel Approaches to Lecture Timetabling

Tomáš Müller

Charles University, Faculty of Mathematics and Physics, Malostranské náměstí 2/25,
118 00 Praha 1, Czech Republic, e-mail: muller@kti.mff.cuni.cz

Abstract: Lecture timetabling consists of lectures, which have to be assigned to appropriate time periods, and resources – rooms, classes, and teachers respecting various constraints like precedence and room capacity. In this paper we outline a constraint model and a solution algorithm for interactive timetabling. Current and further improvements of the described timetabling algorithm are also presented in this paper.

Keywords: lecture timetabling, university timetabling, weekly-course timetabling, local search, forward search, graph colouring, genetic algorithm

1 INTRODUCTION

Timetabling can be seen as a form of scheduling where the task is to allocate activities to available slots in resources respecting some constraints. Typically, an activity is described by its duration and by resources required to process the activity (conjunction and disjunction of resources can be used). There are also precedence constraints between the activities stating which activity must proceed before another activity and resource constraints stating when and how many activities can be processed by a given resource. In addition to the above hard constraints, that cannot be violated, there are many preferences - soft constraints - describing the users' wishes about the timetable. Constraint programming (CP) is a natural tool for describing and solving such problems and there are many timetabling systems based on CP [1, 3, 7, 16].

In this paper we define a lecture timetabling problem and we describe an interactive algorithm to solve this problem. This forward-search algorithm combines both local search and backtracking approaches. Current work on the presented algorithm and several improvements are also discussed in the following sections of this paper.

The problem motivation, basic algorithm, and some preliminary results are also described in paper [12]. The interactive behaviour, as the major feature of the below presented algorithm, is also discussed there. The extensions of the problem and the solution algorithm to solve a real-life lecture timetabling problem at the Faculty of Mathematics and Physics at Charles University with outlined achieved results are presented in paper [13]. The generalization of this algorithm for solving various finite constraint satisfaction problems is presented in paper [14].

2 MODEL

We propose a generic model for weekly periodical, lecture timetabling problems consisting of a set of resources (teachers, classes, rooms etc.), a set of activities (lectures, courses, seminars etc.), and a set of dependencies between the activities. Time is divided into the time slots with the same duration. Every slot may have a constraint assigned, either a hard or a soft one: a hard constraint indicates that the slot is forbidden for any activity, a soft constraint indicates that the slot is not preferred. We call these constraints "time preferences". Every activity and every resource may have assigned a set of time preferences, which indicate forbidden and not preferred time slots.

An activity is identified by its name. Every activity is described by its **duration** (expressed as a number of time slots), by **time preferences**, and by a **set of resources**. This set of resources determines which resources are required by the activity. To model alternative as well as required resources, we divide the set of resources into several subsets – **resource groups**. Each group is either conjunctive or disjunctive: the conjunctive group of resources means that the activity needs all the resources from the group, the disjunctive group means that the activity needs exactly one of the resources (we can

select among several alternatives). For example, a lecture will take place in one of the possible classrooms and it will be taught for all of the selected classes. Note that we do not need to model the conjunctive groups explicitly because we can use a set of disjunctive groups containing exactly one resource instead (the set of required resources can be described in a conjunctive normal form). However, usage of both conjunctive and disjunctive groups simplifies modelling for the users.

A resource is also identified by its **name** and it is fully described by **time preferences**. There is a hard restriction that only one activity can use the resource at the same time. The resource represents a teacher, a class, a classroom, or another special resource in the timetabling problem.

Finally, we need a mechanism for defining and handling direct **dependencies** between activities. It seems sufficient to use binary dependencies only that define a relationship between two activities. Currently, we use three temporal constraints: an activity finishes before another activity, an activity finishes exactly at the time when the other activity starts, and two activities run concurrently (e.g. they have the same start time). The scheduling engine provides an interface for introduction of other binary dependencies. For example, in [13] a dependence is discussed which avoids teaching two non-alternative lectures of the same seminar in the same day.

The solution of the problem defined by the above model is a timetable, where every scheduled activity has assigned its start time and a set of reserved resources, which are needed for its execution (the activity is allocated to respective slots of the reserved resources). This timetable must satisfy all the hard constraints, namely:

- every scheduled activity has all the required resources reserved, i.e., all the resources from the conjunctive groups and one resource from each disjunctive group of resources,
- two scheduled activities cannot use the same resource at the same time,
- no activity is scheduled into a time slot where the activity or some of its reserved resources has a hard constraint in the time preferences,
- all dependencies between the scheduled activities must be satisfied.

Furthermore, we want to minimise the number of violated soft constraints in the time preferences of resources and activities. We do not formally express this objective function; these preferences are used as a guide during search for the solution.

Last but not least, due to the requested interactive feature (discussed in [12, 13]), we need to present some solution to the user at each time. Therefore, we will work with partial solutions where some of the activities are not scheduled yet. Still, these partial solutions must satisfy the above constraints. Note that using partial solutions also allows us to provide a reasonable result even in case of over-constrained problems.

3 ALGORITHM

To satisfy the needs of interactive timetabling we designed an algorithm for solving the above described timetabling problem, which combines two basic constraint programming approaches: the local search and the backtrack-based search. Namely, we required working with feasible (perhaps incomplete) timetables, where some activities are not scheduled yet and we demanded the timetable not to differ much from step to step (from one feasible solution to another) during the search.

We propose an interactive scheduling algorithm that works in iterations. This algorithm uses two basic data structures: a set of activities that are not scheduled and a partial feasible timetable. At each iteration step, the algorithm tries to improve the current partial timetable towards a complete one. The scheduling starts with an empty partial schedule (which is a feasible schedule), i.e. all the activities are in the list of non-scheduled activities. Then it goes repeatedly from one partial feasible solution to another feasible solution until all the activities are scheduled or the maximum number of iterations is reached.

Let's have a look now at what is going on in the iteration step. First, the algorithm selects an un-scheduled activity and computes the locations where the activity can be allocated (locations with the hard constraints are not assumed). Every location is described by a start time (the number of the first slot for the activity) and by a set of required resources. Then every location is evaluated using a heuristic function over the current partial schedule. Finally, the activity is placed to the best location that may cause some conflicts with already scheduled activities. Such conflicting activities are removed from the schedule and they are put back to the list of non-scheduled activities. (see Figure 1)

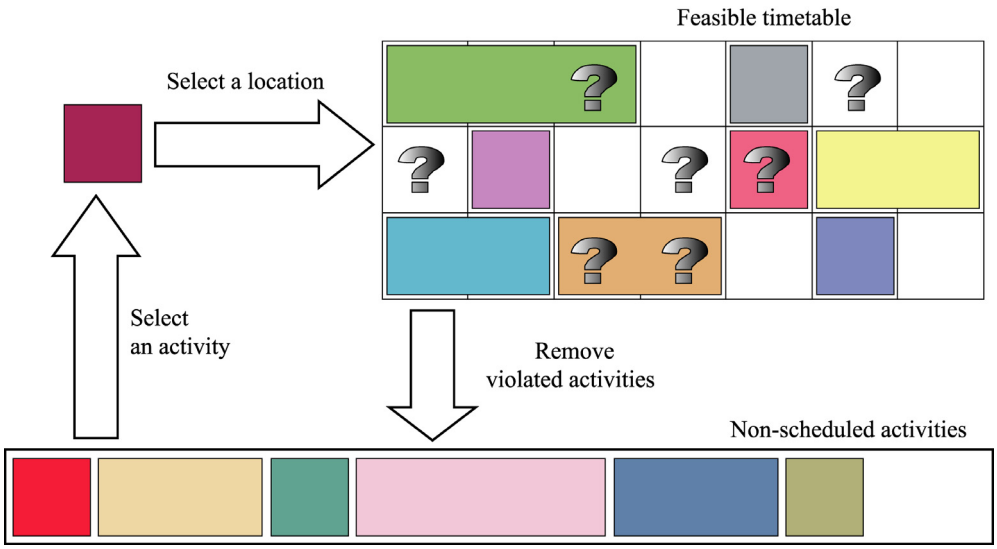

Figure 1. An iteration step of the timetabling algorithm

The above algorithm schema is parameterised by two functions: the activity selection and the location selection. The activity selection problem is equivalent to what is known as a variable selection criterion in constraint programming. There are several general guidelines how to select a variable. In the local search, the variable participating in the largest number of violations is usually selected first. In backtracking based algorithms, the first-fail principle is often used, i.e., the variable whose instantiation is the most complicated is selected first. This could be the variable involved in the largest set of constraints (a static criterion) or the variable with the smallest domain (a dynamic criterion) etc. [4, 8, 11]. In the current implementation a weighted sum of several criteria (such as in how many dependencies does the activity participate or in how many places can the activity be placed) is used and the activity with the minimal heuristic value is selected. Using such formula gives us more flexibility when tuning the system for a particular problem.

After selecting an activity we need to find a location where to place it. This problem is usually called the value selection in constraint programming. Typically, the most useful advice is to select the best-fit place [4, 8, 11]. So, we are looking for a place, which is the most preferred for the activity and also where the activity causes less trouble. It means that we need to find a place with minimal potential future conflicts with other activities. Note that we are not using constraint propagation explicitly in our algorithm, the power of constraint propagation is hidden in the location selection (it roughly corresponds to a forward checking method). Similarly to the activity selection criterion, a weighted sum of several criteria is used and a location with the minimal value is selected. For example, the number of violated soft constraints or the number of conflict activities is counted for each location.

Moreover, because of removing activities from the partial schedule, we need a mechanism to prevent cycles. In the current implementation of the scheduling engine we use a technique based on a tabu list [4, 10, 17]. Tabu list is a FIFO (first in first out) structure of pairs (variable, value) with a fixed length. When a value V is assigned to the variable X, a new pair (X,V) is added to the end of the tabu list and the first pair is removed from the tabu list. Now, we can avoid a repeated selection of the same value by applying a tabu rule which says: if the pair (X,V) is already in the tabu list then do not select the value V for X again (until the pair disappears from the tabu list). The tabu rule prevents short cycles (a cycle length corresponds to the length of the tabu list) and it can be broken only via so called aspiration criterion. If the aspiration criterion is satisfied for a pair (X,V) then V can be assigned to X even if the pair (X,V) is in the tabu list.

4 IMPROVEMENTS

Although the above described algorithm is pretty successful both in randomly generated and in the real-life timetable problems (as shown in [12, 13]), it also gives us a lot of space for further improvements. These improvements can be oriented both towards acceleration of the search and towards problem enlargements and tuning the sense of adding new constraints or requirements. Several algorithm improvements are described in the following sections.

4.1 Current Improvements

Some improvements are already discussed in papers [12, 13]. The first improvement is included in the activity selection criterion. It is possible to select the worst activity among all non-scheduled

activities but due to the complexity of computing the heuristic value, this could be rather expensive. Therefore we proposed to select a subset of non-scheduled activities randomly (we use a probability of selection 0.2) and then to choose the worst activity from this subset. The results will not be much worse and we can select the activity approximately five times faster (see Figure 2 and 3).

The second improvement is the randomisation of the location selection method (e.g. to remove cycling). For example, it is possible to select five best locations for an activity and then choose one other randomly. Or, it is possible to select a set of locations such that the heuristic value for the worst location in this group is maximally twice as large as the heuristic value of the best location. Again, the location is selected randomly from this group. This second rule inhibits randomness if there is a single very good location.

On the following figures some results achieved with the described improvements are presented. These results were obtained using randomly generated problems with the size of 20 classes, 20 rooms, and 20 teachers and 10 slots per day (5 days). See [12] or visit our benchmark page [15] for more details about the generated timetables.

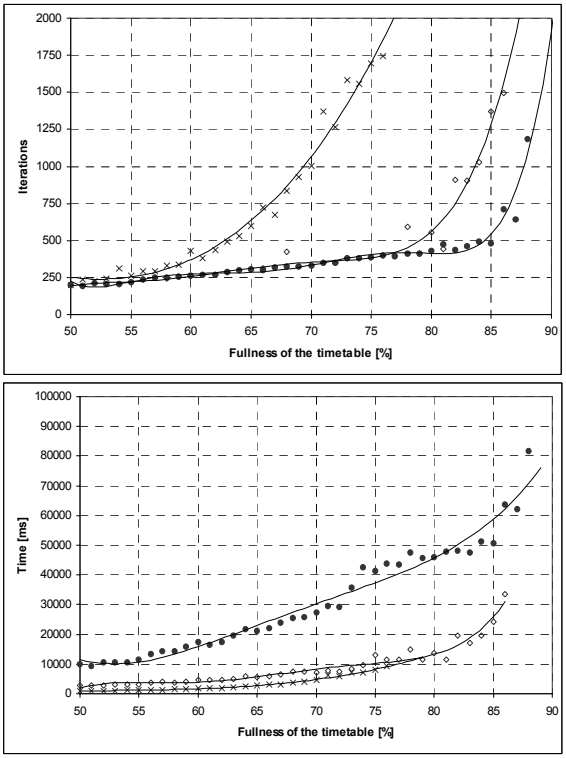

Figure 2. Comparison of the number of iterations and time (measured in milliseconds) needed to solve the generated timetable for three basic variable selection criteria (× randomly selected activity, ◊ the worst activity among 20% randomly selected activities, • the worst activity among all activities).

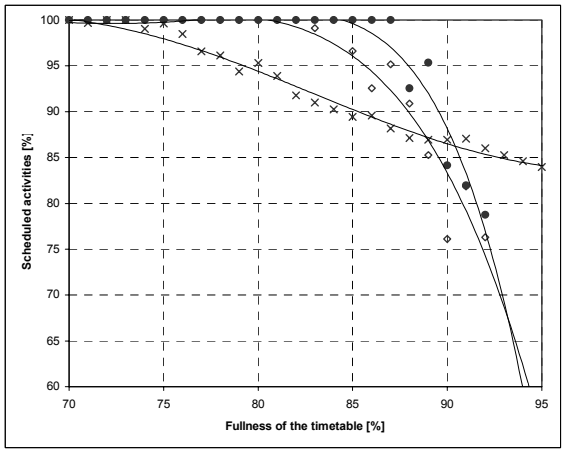

Figure 3. Comparison of the number of scheduled activities (as percent of all activities) for three basic variable selection criteria (× randomly selected activity, ◊ the worst activity among 20% randomly selected activities, • the worst activity among all activities).

We compared the three basic activity selection criteria, namely random selection of the activity, selection of the worst activity among 20% randomly selected non-scheduled activities, and selection of the worst activity among all non-scheduled activities. Feasible problems with different number of activities were used (the number of activities is measured in percent of the timetable space).

It is not surprising that the selection of the activity from all non-scheduled activities leads to the smallest number of iterations and to schedules with the largest number of scheduled activities. However, this method also brings some overhead so it needs more time to find a schedule. Random selection of the activity is very fast but it requires higher number of iterations to find a schedule and in cases with more activities the number of scheduled activities decreases. Our experiments confirm that the idea of combination of random selection with activity selection heuristic could bring interesting results. The number of iterations and successfully scheduled activities is comparable to the method where all non-scheduled activities are explored. However, the overall time is lower because we are choosing from about 20% of randomly selected activities only.

4.2 Graph Colouring Approach

As it is presented in [5], we can use the graph colouring approach for splitting up the activities into different groups. Each activity is represented as a different vertex with an edge between vertices where two respective activities conflict in some way. Colouring the graph is the process of allocating different colours to each vertex so that no two adjacent (conflicting) vertices have the same

colour. Each colour corresponds to one period in the timetable. Therefore, ignoring the constraints imposed by the availability of rooms and other resources and duration of the activities, the smallest timetable length possible is the same as the minimum number of colours needed to colour the graph (the chromatic number). The graph colouring problem have been proved to be NP-complete.

In the above presented timetable algorithm, the graph colouring engine can be used for splitting up the activities into several independent groups or for finding the maximal set of independent activities (or almost maximal set when a heuristic colouring algorithm is used). A similar approach, where the search for almost maximal independent set of the activities is used with combination of the knapsack filling problem, is presented in [5], used in the exam scheduling at the University of Nottingham.

We propose to use the search of an almost maximal independent set of activities in the activity selection criterion. Two different activities can be located in the same independent set if there is no direct dependence between them and they do not share a common resource for their execution (even if the resource is in the disjunctive group for one of the activities). Typically, many lectures in the lecture timetabling problem have no restriction for the room where it should be taught, and therefore these lectures can be seen as conflict ones; for that reason, we propose not to use the room resource constraints for specifying the conflicts between the activities. Two different lectures will be in the conflict (cannot be in the same independent set) if there is a direct dependence (e.g. one should precede the other) between them and they are not taught by the same teacher or for the same class (e.g. the intersection of the sets of classes, for which are the lectures taught, has to be empty). Two activities cannot use the same special resource (e.g. overhead projector) either.

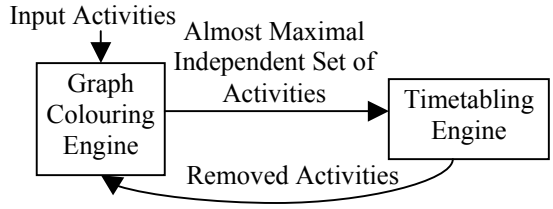

Figure 4. Timetabling with the graph colouring approach

On the above figure, one possible use of graph colouring approach in the lecture timetabling algorithm is shown. At first, the almost maximal independent set of activities from all unscheduled activities is found. These activities are scheduled via the above presented timetabling algorithm, the removed activities are placed back to the set of the unscheduled activities. The activity selection criterion can be the same as before, but it will operate only over the independent set of activities (not over all unscheduled activities). The location criterion should be slightly modified to take into account also the independency of the given activities (e.g. it should try to schedule them in the same time).

Unfortunately, we cannot give any overview now, whether this improvement helps the timetabling engine or not, or how exactly should be the timetabling algorithm or its heuristics altered in order to profit from the independency of the given activities. This feature is not yet fully implemented into our timetabling program. The graph colouring algorithm can use the similar idea for colouring vertices as the timetabling algorithm – when some vertex is coloured, colours from all adjacent conflicting vertices are removed. Similar heuristics for selecting a vertex to be coloured or the vertex's new colour can be also introduced there.

4.3 Evolutionary Approach

Genetic algorithms are powerful general purpose optimisation tools which model the principles of evolution [2, 6, 7, 9]. They are often capable of finding globally optimal solution even in the most complex search spaces. They operate on a population of solutions which are selected according to their quality and then used as the basis for a new generation of solutions found by combining (crossover) and/or altering (mutating) current individuals. Traditionally, the search mechanism has been domain independent, that means the crossover and mutation operators have no knowledge of what a good solution would be. However, it seems that better results can be achieved by using domain dependent operators [6]. In the following paragraphs, we will discuss how the presented algorithm can be used as a part of the genetic timetabling algorithm.

In our case, a solution is a feasible timetable, which consists of a partial sound timetable and a set of unscheduled activities. A genetic algorithm starts by generating a set (population) of timetables randomly. Next, every randomly generated timetable is made consistent by removing the conflict activities. Starting timetables can be also generated from free timetables (where no activity is scheduled) by performing at most N iterations with random activity selection criterion (where N is a suitably chosen constant). These starting timetables are evaluated according to some kind of criteria. For example, a number of unscheduled activities or a number of violated constraints can be used in the evaluation. On the basis of this evaluation the population members (timetables) are chosen as parents for the next generation of timetables.

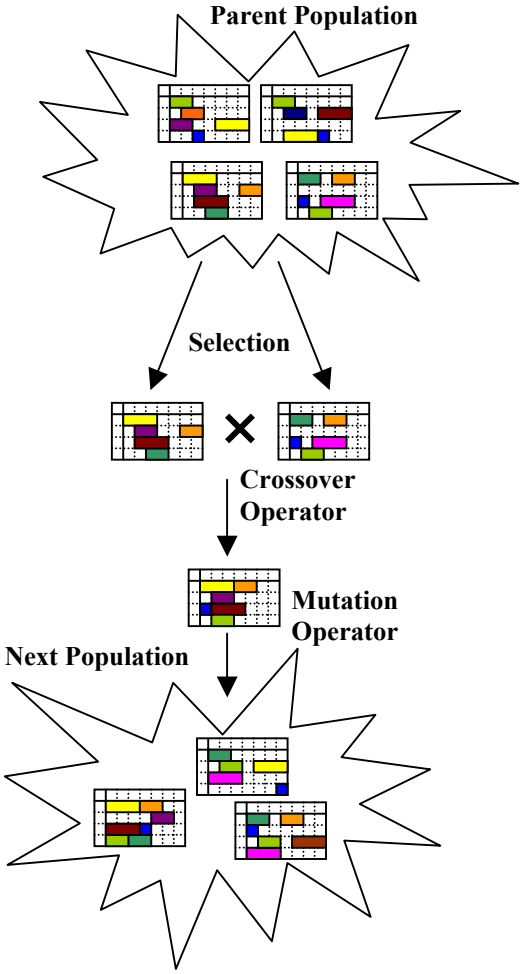

Figure 5. The genetic algorithm

On Figure 5, an iteration of the presented genetic algorithm is shown. Every iteration starts from the parent population and produces next population of feasible timetables, which is the parent population for the next iteration. This process stops when a solution with all activities scheduled is produced (and when this solution meets all additional requirements, for example when the solution satisfies at least 95% of all the soft constraints).

Until the next population has the required number of members, two members from the parent population are selected. These parent timetables are combined via the crossover operator into a new timetable, which is mutated by the mutation operator and then added into the next population (which is empty at the beginning of the iteration).

All timetables in the parent population are evaluated and the selection of two parent timetables is provided via some individual selection function. Good timetables (e.g. with less unscheduled activities) should be more likely to be chosen than the bad ones.

A major reason, whether the genetic algorithm is successful or not, lies in the crossover operator. This operator should combine the good properties from both parents and produce a timetable, which should intimately correspond to both parents and which should be better than the parent timetables. We propose an operator that works in two phases: First, it takes all scheduled activities from both parent timetables and puts them into the new timetable to the same locations where they were in one of the parent timetables. When an activity was scheduled only in a single parent timetable, it goes exactly to the same place. Otherwise, when an activity was scheduled in both parent timetables, these two locations are evaluated (we can use the same evaluation function as in the above presented location selection criterion) and the activity is placed to the more preferred location. Activities, which are unscheduled in both parent timetables, will be unscheduled again in the produced timetable. In the second step, when all scheduled activities are placed in the new timetable, the timetable is made feasible by removing conflicting activities. A heuristic criterion can also be used there, which can for example order the conflict activities (not all the activities, which are in conflict, have to be removed).

Next, when a new timetable is produced, the mutation operator is performed on it. The sense of this operator here is to improve the timetable by scheduling some of the unscheduled activities. As the mutation operator, we propose to use a limited number of iterations (e.g. at most N iterations) of the previously described algorithm.

We believe that the presented algorithm will be successful, but unfortunately there are no results, which can be presented right now – the algorithm is not fully implemented and balanced right now (via setting the population size, number of iterations in the mutation operations and parameters of all used heuristics etc.). The use of the feasible timetables seems to be a great advantage of this algorithm.

In the sense of interactivity, we need to present only one solution during the search. We also need to be able to continue scheduling after the user stops the timetable process and alters the timetable somehow. In this case, we can always present the best timetable from a population during the search. So, when the algorithm should start from some partial solution it can first make the solution feasible (by removing conflicting activities). Next, it can produce some starting population from one timetable by applying randomised mutation operator (e.g. random activity selection) several times on the altered timetable and then it can start searching (a genetic algorithm process) from this population. Unfortunately, the differences of two following timetables can be huge, which can be a bit confusing for the users.

5 CONCLUSION

We presented a promising algorithm for solving timetabling problems, which combines principles of

the local search with other techniques for solving constraint satisfaction problems. Although the basic motivation was to design a generic algorithm with interactive features for solving school timetabling problems, the proposed principles can be applied to other constraint satisfaction problems especially when interactive behaviour is required. The algorithm and its current implementation can be easily extended to cover additional hard and soft constraints. Several potential improvements and extensions were presented as well.

Currently, we are working on further empirical studies of this algorithm with a particular emphasis on studies how the weights influence efficiency. Further research is oriented both theoretically, to formalise the techniques and to put them in a wider context of constraint programming, and practically, to implement the above described two major improvements of the algorithm – the almost maximal independent set of activities and the genetic algorithm.

6 REFERENCES

1. S. Abdennadher and M. Marte. *University timetabling using constraint handling rules*. Journal of Applied Artificial Intelligence, Special Issue on Constraint Handling Rules, 1999.
2. D. Abramson and J. Abela. A parallel genetic algorithm for solving the school timetabling problem, Technical report, Division of Information Technology, C.S.I.R.O., 1991.
3. R. Barták. Dynamic Constraint Models for Planning and Scheduling Problems. In New Trends in Constraints, LNAI 1865, pp. 237-255, Springer, 2000.
4. R. Barták. *On-line Guide to Constraint Programming*, http://kti.mff.cuni.cz/~bartak/constraints, 1998.
5. E. K. Burke, D. G. Elliman, R. F. Weare. *A University Timetabling System Based on Graph Colouring and Constraint Manipulation*, Journal of Research on Computing in Education, 1993.
6. E.K. Burke, D. G. Elliman, R. F. Weare. *A Genetic Algorithm Based University Timetabling System*. East-West Conference on Computer Technologies in Education, Crimea, Ukraine pp35-40, 1994.
7. J. P. Caldeira and C. R. Agostinho. *School Timetabling Using Genetic Search*, Practice and Theory of Automated Timetabling, Toronto, 1997
8. D. Clements, J. Crawford, D. Joslin, G. Nemhauser, M. Puttlitz, M. Savelsbergh. *Heuristic optimization: A hybrid AI/OR approach.* In Workshop on Industrial Constraint-Directed Scheduling, 1997.
9. W. Erben and J. Keppler. *A Genetic Algorithm solving a Weekly Course-Timetabling Problem.* In Proceedings of ICPTAT'95, pp. 21-32, Napier University, Edinburgh, 1995.
10. P. Galinier and J.K. Hao. *Tabu search for maximal constraint satisfaction problems*. In Proceedings of CP'97, G. Smolka ed., LNCS 1330, pp. 196-208, Schloss Hagenberg, Austria, Springer, 1997.
11. Z. Michalewicz, D. B. Fogel. *How to Solve It: Modern Heuristics*. Springer, 2000.
12. T. Müller, R. Barták. *Interactive Timetabling*. In Proceedings of the ERCIM workshop on constraints, Prague, 2001.
13. T. Müller and R. Barták. *Interactive Timetabling: Concepts, Techniques, and Practical Results*. In Proceedings of the PATAT conference, Gent, 2002.
14. T. Müller. *Interactive Heuristic Search Algorithm.* In Proceedings of the Doctoral Programme of the CP-2002 conference, Ithaca, 2002
15. T. Müller. *Interactive Timetabling – Benchmarks*, http://kti.mff.cuni.cz/~muller/ttbench, 2002
16. A. Schaerf. *A survey of automated timetabling.* Technical Report CS-R9567, Centrum voor Wiskunde en Informatica (CWI), Amsterdam, The Netherlands, 1996.
17. A. Schaerf. *Tabu search techniques for large high-school timetabling problems*. In Proceedings of the Fourteenth National Conference on Artificial Intelligence, pp. 363-368, Portland, Oregon, USA,1996.