Automated University Timetabling: The State of the Art

EDMUND BURKE, KIRK JACKSON[1], JEFFREY H. KINGSTON[2] AND RUPERT WEARE[3]

[1]*Department of Computer Science, University of Nottingham, Nottingham, UK*
[2]*University of Sydney, Sydney, Australia*
[3]*Cap Gemini UK plc, UK*
Email: ekb@cs.nott.ac.uk

Scheduling lectures or exams for universities is a large and complex task. There are many different departments and faculties, each with their own ideas about how and when their courses should be run. Furthermore, modularization means that students can take courses from a combination of several departments, possibly even in different faculties. Numerous university timetabling systems have been devised, and work is underway to create a standard which will enable objective performance comparisons. This paper briefly looks at various popular techniques and recent work on both exam and course scheduling around the world, and should serve as an introduction to the field.

Received December 23, 1996; revised July 7, 1997

1. WHAT IS THE TIMETABLING PROBLEM?

A timetable is a placement of a set of meetings in time. A meeting is a combination of resources (e.g. rooms, people and items of equipment), some of which may be specified by the problem, and some of which must be allocated as part of the solution. Timetabling has long been known to belong to the class of problems called NP-complete [1], i.e. no method of solving it in a reasonable (polynomial) amount of time is known.

There are several fundamental variations on the timetabling problem. University timetabling can be divided into two main categories: lectures and examinations. The major common differences between lecture scheduling and exam scheduling are:

- Exams must be scheduled so that no student has more than one exam at a time, but lectures must usually be scheduled before student enrolments are known.
- As space is often a constraint, exams often share rooms, or are split across rooms, but only one lecture may be held in a room at any time.

Carter *et al.*'s summary of the (exam) scheduling problem is that 'the basic challenge is to schedule examinations over a limited time period so as to avoid conflicts and to satisfy a number of side-constraints' [2]. The conflicts he refers to occur whenever a timetable requires any resource to be in two places at the same time. The side-constraints vary between institutions, and between exam and lecture scheduling, and the list is endless. Challenge is a very appropriate word to use. As Bloomfield and McSharry say: 'Depending on the size of the department and the diversity of the course offerings, the time required to produce such a schedule of classes can range from a perfunctory afternoon's work to an intensive month-long ordeal' [3].

The timetabling process is made more difficult by the fact that so many people are affected by its outcome. Romero identified three main stakeholders in this process each with their own set of aims and wants [4]:

1. The administration sets the minimum standards to which the timetable must conform. For example, some universities specify that no student should have to take two exams in consecutive periods.
2. The departments' concerns are more likely to be prominent in the course timetable. They will want the 'schedule to be consonant with the development of the subject taught' as well as making more specific demands for particular classrooms or labs. In an examinations context, they are likely to request that large exams be placed early in order to allow more time for marking.
3. The third group of stakeholders are the students, whose view of the timetable will be restricted to the part that affects them. Given the number of students involved it is difficult to obtain specific criteria as to what is the best timetable for students. Many students prefer not to have lectures late on a Friday, and to have a break between consecutive exams. Taking student preferences into account can considerably increase the difficulty of the problem.

Timetabling constraints are many and varied. Here are some of the most common types.

Resource assignment. A resource may be assigned to a resource of a different type or to a meeting. For example, a lecturer prefers to teach in a particular room (for whatever reason) or an exam should take place in a particular building.

Time assignment. A meeting or a resource may be assigned to a time. This constraint can be used to specify days on which a teacher is unavailable or to pre-assign a time to a particular meeting.

Time constraints between meetings. Common examples of this class of constraint are that one particular meeting must take place before another one or a set of exams must be sat simultaneously.

Meeting spread. Meetings should be spread out in time. For example, no student should have more than one exam in any day.

Meeting coherence. These constraints are designed to produce more organized and convenient timetables, and often run contrary to 'meeting spread' constraints (above). For example, a lecturer prefers to have all his lectures in three days, giving him two lecture-free days.

Room capacities. The number of students in a room may not exceed the room's capacity.

Continuity. Any constraints whose main purpose is to ensure that certain features of student timetables are constant or predictable. For example, lectures for the same course should be scheduled in the same room or at the same time of day.

Constraints are usually divided into 'hard' and 'soft' categories:

- **Hard constraints.** A timetable which breaks a hard constraint is not a feasible solution, and must be repaired or rejected by the timetabling algorithm. Hard constraints include first-order conflicts, i.e. no person may be required to attend more than one meeting at any time.
- **Soft constraints.** Soft constraints are less important than hard constraints, and it is usually impossible to avoid breaking at least some of them. Whichever timetabling method is applied, timetables are usually rated by a penalty function, which calculates the extent to which a timetable has violated its soft constraints. Some soft constraints are more important than others, and this is often specified with a priority value.

Given its complexity, there is great potential for computational techniques to help ease the task of university timetabling.

2. EXAMINATION TIMETABLING IN UK UNIVERSITIES

The timetabling problem can vary greatly between different universities. The Automated Scheduling and Planning (ASAP) Group at the University of Nottingham carried out a survey [5] of examination timetabling in UK universities. The survey was sent to 95 British universities, of which 56 (59%) replied.

The results show just how different timetabling problems at different institutions can be. For example, the number of exams to schedule can vary from 100 to 2500 in any one session, with anything from 500 to 20,000 students being examined. As one university timetabler replied. 'There are so many variations within the timetabling schedule we could write a book really!'

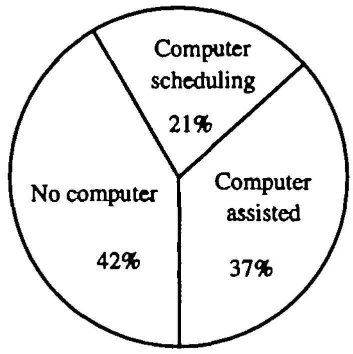

FIGURE 1. Computer usage in British university examination timetabling 1995.

Even one of the most common features of academic timetables, that of fixed timeslots, is not sacrosanct. For example, a university may allow exams to be scheduled at any time between 09:30 and 12:30, and 13:30 and 18:00. Although potentially making the timetabling process considerably more difficult, this does allow more efficient room utilization, especially where there is a large variation in exam lengths.

One of the biggest problems for universities is finding space to house the exams. Several universities overcome the problem of insufficient accommodation by hiring external halls. This solves the problem to a certain extent but, since the halls are also used for other functions, their availability can be unpredictable. It is also the case that these halls are not always close to the campus, thus causing further problems similar to that of split campuses where students must be allowed extra commuting time.

Of those universities responding to the survey, it is perhaps surprising that 42% do not use any form of computational help in the timetabling process (Figure 1). One possible reason for this is that in some institutions, there is no significant change in the timetable from year to year. The split between those that use last year's timetable and those that construct a completely new timetable each year is exactly even, but in every case, those that use neither the timetable from the previous year nor a computer take at least 4 weeks to produce a timetable.

Another reason for the lack of automation may be due to a certain inertia on behalf of the timetablers themselves. The examination department from one university said that they could 'see the disadvantages in a fully automated system of examination timetabling. There is (dare it be said) also a case for working through the initial stages of timetabling with a chart and a pencil! This allows instant overview of the progress of timetabling in the whole of the examination session rather than reliance on a system of queries and scrolling almost inevitable on a computer screen.'

Of those that do use some sort of computational aid, 21% have a facility to perform scheduling, although these may, in some cases, still require a certain amount of manual input and previous knowledge of the particular timetabling problem. A few more universities stated that they are either currently developing their own computer timetabling system or customizing a commercial package.

There are two main timetabling approaches used by those institutions that do not employ scheduling aid:

1. Let each school or department produce their own draft timetable and then use a central administration facility to merge them all into a master timetable. This assumes that the schools or departments are independent. However, many British universities are adopting a modular approach, which means that this assumption is no longer valid.
2. Construct the examination timetable by altering the teaching timetable. One institution, which is incidentally developing a new system, said, 'This, of course, doesn't work, since some lectures are repeated in different slots, or have more students in large lecture theatres than we can accommodate simultaneously in flat exam rooms.' This approach depends on the nature of the associated course timetable.

It is clear from the survey that there is a great need for automated assistance for examinations timetabling help. Having said that, any system, if it is to be useful, must satisfy a number of difficult criteria. The main problem, of course, is that it must be able to generate high-quality timetables despite the huge variations in the type of problems found in the different universities. It must also be compatible with present systems, be easy to use and perform all the functions required by the exams office.

As far as the authors are aware, no similar recent survey of course scheduling has been carried out.

3. BENCHMARKS

Many different methods have been developed and used successfully to solve real timetabling problems [6, 7, 8, 9]. In virtually every case though, these methods are only ever used in one particular department or institution, and are rarely properly compared with each other. Accurate comparison is vital in determining which are the best computational methods given various types of timetabling data. When the techniques used were relatively simple, they were fairly easy to reproduce. As timetabling techniques become more sophisticated, partially in response to the need to cope with more varied constraints, this has become much harder.

Data from real university timetabling problems is available on-line (from the University of Toronto [`ftp://ie.utoronto.ca/mwc/testprob/`] and the University of Nottingham [`ftp://ftp.cs.nott.ac.uk/ttp/Data/`]), but there is as yet no benchmark to facilitate objective and unambiguous comparison of different timetable solutions to these problems.

It is not easy to express clearly and precisely the requirements for a real-life timetabling problem. The basic requirements such as every lecture should have a lecturer and no-one should have to be in two places at once are easy enough, but the side-constraints on detailed points such as how many examinations are permitted in any given period, or whether students may interrupt a laboratory session to attend a lecture, and so on, are known only by expert timetablers or buried deeply inside computer programs and are rarely written down in a form that others can read and understand.

This is the major reason why so little data is exchanged among automated timetabling researchers at the moment.

A project underway at the University of Nottingham, and in conjunction with Napier University, the University of Reading, the University of Sydney and the University of Toronto, is attempting to find a way to write down such requirements, using formulae based on logic so that there can be no room for uncertainty or disagreement about their meaning. This will permit independent computerized checking of proposed solutions against the requirements and (in the longer term) automatic conversion between the data formats used by different researchers.

4. APPROACHES TO AUTOMATED TIMETABLING

Simple, problem-specific heuristic methods can produce good timetables, but the size and complexity of modern university timetabling problems has provoked a trend, which was confirmed by the presentations at the first international conference on the Practice And Theory of Automated Timetabling [10], towards more general problem-solving algorithms, or metaheuristics, such as simulated annealing, evolutionary algorithms and tabu search. Problem-specific heuristics may be employed in the context of such an algorithm to reduce the number of possible solutions processed, or to locally optimize a solution. Constraint logic programming (CLP) is also a popular approach.

4.1. Genetic algorithms

Genetic algorithms (GAs) are analogous to neo-Darwinian evolution. A population of feasible timetables is maintained. The fittest timetables (those with the lowest penalty values) are selected to form the basis of the next iteration, or generation, thus improving the overall fitness whilst maintaining diversity.

The most common genetic representation for a timetable is a long string encoding when and where each meeting is to take place. Thus, pairs of selected timetables may be crossed over, the strings cut and spliced to create new timetables. However, Corne *et al.* found an intelligent mutation operator to be more successful than two-parent crossover [11]. Their system, GATT, is now being used successfully to timetable courses at the University of Edinburgh, the Harvard Business School, Kingston University, the University of Reading and several other institutions.

Paechter *et al.* have developed Neeps and Tatties, a system which is being used to schedule courses at Napier

University's Computer Science department. Its GA encodes timetables as an ordering of events, which must be input to a special program which uses the order to produce a timetable [12]. This necessitates a different sort of recombination operator, which takes elements of the order from each parent to produce a new ordering.

The ASAP group at the University of Nottingham has developed GAs for examinations scheduling which employ a large degree of heuristic knowledge, both to seed the initial population, and to improve the standard genetic operators [13, 14, 15]. Nottingham crossover operators work on a timetable period level, taking meetings scheduled in both parents first, and then others according to a heuristic ordering, until no more may be placed in the child timetable without conflict or seating problems. The use of this type of crossover operator can lead to very efficient room usage as it always tries to fill up space where ever possible.

4.2. Memetic algorithms

Memetic algorithms are an extension of GAs, based on a model of how ideas evolve. The basic units of ideas are memes, which, unlike genes, can be improved during their lifetime. Thus a hill-climbing function is used at set intervals, ensuring that the timetable population members are all local optima.

The University of Nottingham is currently developing memetic algorithms for examinations scheduling [16]. A light or heavy mutation operator is applied to selected members of the population. After each timetable is disrupted by a mutation operator, it is locally optimized by a hill-climbing algorithm. This combination generates good quality new solutions.

Napier University is working on memetic lecture scheduling [17]. Their timetable encoding specifies a list of suggested timeslots for each event. These timeslots are tried in order, and the successful timeslot is moved to the top of the list, thus improving the memetic material. Their recombination operator builds new suggestion lists by taking timeslot suggestions alternately from each of the parents' corresponding suggestion lists. Since each parent's suggestion lists were in turn zipped together from their parents' suggestion lists, this memetic improvement, or writeback, means that each suggestion list is built using the results of its ancestors' timeslot searches.

4.3. Simulated annealing

Simulated annealing is a search strategy which keeps track of one feasible timetable. On each iteration, a neighbour is generated—another feasible timetable, slightly altered at random from the current one. This neighbour is accepted as the current timetable if it has a lower penalty. If the neighbour has a higher penalty, it may be accepted according to a probability which is related to a control parameter called temperature. The temperature, and thus the probability of inferior neighbours being accepted, is decreased each iteration or (more usually) after a particular number of iterations (this number may be constant or it can increase as the temperature decreases). The process is analogous to the cooling process in actual annealing. One drawback with simulated annealing is that the cooling process can take a long time in order to achieve good results.

Simulated annealing has been successfully applied to the timetabling problem in Swansea's TISSUE examinations scheduling system [18, 19]. The authors, Thompson and Dowsland, use a graph-colouring model. To simplify the highly constrained timetabling problem, they divide it into two phases, first finding a feasible solution, then optimizing secondary constraints. However, this approach results in a considerably reduced, and possibly disconnected, search space. Hence, they use Kempe chains to provide a more connected search space. Rather than simple geometric cooling, Dowsland uses a non-monotonic cooling schedule which actually increases the temperature when a move is rejected. Furthermore, the ratio of rejected to accepted moves is geometrically increased as the search progresses.

4.4. Tabu search

Like simulated annealing, tabu search remembers just one currrent feasible timetable. The difference is in the method by which moves to new timetables are accepted. A tabu search maintains a list of tabu moves, representing timetables which, having been visited recently, are forbidden in order to prevent the search from staying in the same area, and thus escape local optima. The tabu list is usually of a fixed size, with the oldest moves being removed as new moves are added. Because tabu moves may prevent the search from reaching new improved solutions, an aspiration level is often maintained, which represents the best solution visited so far. If a tabu timetable reaches the aspiration level, it may be removed from the tabu list.

Tabu search has been successfully applied by Boufflet and Nègre to generate examinations timetables at the University of Technology of Compiègne [20]. Their tabu list contains the seven most recent moves. If the current neighbourhood does not contain an improved solution, the aspiration function may select one from the tabu list.

Formulating course scheduling as an assignment problem, Hertz developed and applied the TATI tabu algorithm [21] which he later adapted for a more complex and constrained real-life course scheduling problem [22]. The length of a lecture is not fixed in advance and there are ten different types of moves (e.g. moving a lecture to another day, changing the duration of the lecture etc.). When the schedule of a particular lecture in a particular day is changed it may be moved to another period (possibly in another day). However, for a given number of iterations it is tabu to move the lecture to a period in the original day.

4.5. Constraint logic programming

Timetabling can be modelled as a constraint satisfaction problem (CSP). In a CSP, values which satisfy a set of constraints must be found for a set of discrete variables

TABLE 1.

Attribute name	Attribute values
OHP screen	True iff the room has a screen for an overhead projector
Purposes	The uses the room may be put to, e.g. lecture theatre, computer lab, etc.
Building	The building in which the room is situated
Capacity	The number of students which the room can contain

TABLE 2.

Attribute type	Example	Values
Simple	OHP screen	true or false
List	Purposes	Any number of items from a list of possible values
Exclusive	Building	Exactly one item from a list of possible values
Number	Capacity	A positive integer

with finite domains. Constraint logic programming (CLP) is based on the application of declarative logic programming languages such as Prolog to CSPs. A Prolog program consists of a set of clauses, which in CLP may contain constraints, the satisfiability of which is checked during computation.

In CLP, a labelling strategy dictates the order in which the search space is traversed, which is vital to an effective search. There are two orderings: the order in which the variables are instantiated (e.g. meetings placed) and the order in which the values (e.g. times and rooms) are assigned.

A Prolog lecture scheduling system has been developed by White *et al.* at the University of Ottawa [23, 24]. Constraints are divided into primary and secondary. Primary constraints must always be satisfied, but secondary constraints may be relaxed.

The secondary constraints are partitioned into six groups, each of which may be relaxed, as required, during the scheduling. Each course is tentatively scheduled in turn. If it is not possible to assign a time and room without violating constraints, the program backtracks, relaxing secondary constraints one by one until the course can be scheduled. If the course still cannot be scheduled, a similar course will be temporarily unscheduled, and the search tried again. Unschedulable courses are left to be manually placed. Boizumault, Delon and Péridy have designed an examinations scheduling program [25] in CHIP, a CLP language based on Prolog, which provides several types of constraint formulation. CHIP's new cumulative constraint limits the amount of a resource which can be used at any time, and Boizumault *et al.* use this to implement the room capacity constraint. Their labelling strategy schedules largest exams first, in the period with the smallest total number of students. Boizumault, Guéret and Jussien have also implemented a lecture scheduling system in CHIP called FELIAC [26, 27]. Longest lectures are scheduled first, in the day which has the shortest total length of clashing lectures. Relaxation of constraints is essential for highly constrained CSPs such as timetabling. (A problem in which constraints may be relaxed is called a *dynamic* CSP.) For each failed assignment, FELIAC stores a 'justification', which identifies the constraints which the assignment violated. These justifications are used to undo the effects of a constraint when it is relaxed.

Lajos also used an implementation of CHIP to construct a lecture scheduling system to cope with modularization at the University of Leeds [28] His labelling heuristic schedules the most conflicting and longest lectures first. A day which has not been chosen recently is chosen at random, which spreads lectures out without the need for an explicit constraint. This system was the precursor to KnowTIS. Information about this can be found on the Web at `http://agora.leeds.ac.uk/gyuri/knowtis.html`.

5. MODELLING PROBLEM DATA

Timetabling problems vary considerably between universities, so in order to be applicable to more than one institution, a system must be flexible in the way in which resources and constraints are specified.

For example, a university timetabling package being developed by the ASAP group at the University of Nottingham incorporates a highly flexible system by which objects, specifically meetings and resources, may be defined and grouped. For simplicity, the following description talks about resources, but meetings may be defined and grouped in the same way.

Each type of resource, e.g. rooms, has a set of attributes which are used to define individual resources. For example, rooms may have the four attributes shown in Table 1. Every room would possess values for each of these attributes. This example uses the four types of attribute shown in Table 2. The timetabler may add, modify or remove attributes of any type in order to alter the information held about resources of any particular type.

All of the resources of a particular type which have the value true for a particular Simple attribute make up a set; for example, the set of rooms which have an OHP screen. Furthermore, each of the possible values for a List or

Exclusive attribute can also be viewed as the set of resources which take that value for that attribute; for example, the set of rooms which belong to the Tower Building.

The timetabler thus has two ways of viewing and manipulating resource data:

1. Via resources, looking at and changing their attribute values. For example, the timetabler may wish to view which building a room is in, or place a room in a different building.
2. Via sets, seeing which resources are in a set, and adding and removing resources from a set. For example, the timetabler may wish to view which rooms are in a building, or add or remove rooms to or from that building.

Nottingham's ASAP object and attribute editing system provides a powerful platform for the specification of constraints. A resource assignment constraint links a list of resources to a list of meetings or a list of resources to a different type of resource. Each list may be made up of individual objects, or sets of objects.

For example, a single resource assignment constraint could specify that all Computer Science and Electronic Engineering lectures should take place in the Tower Building, or that all people who are wheelchair-bound should not be required to attend meetings in rooms which do not have wheelchair access.

6. CONCLUSIONS

The long and arduous task of creating a timetable for a university can be considerably eased by computer automation. However, the immense complexity and variety of the problem means that there is much room for improvement on current systems. The most successful algorithms combine modern metaheuristics with a large amount of problem-specific heuristic knowledge.

There has been a consistent flow of automated timetabling papers over the last 40 years or so, but in recent years there has been a large increase in the interest shown by researchers all over the world. One indicator of this is the success of the recent conference on the Practice and Theory of Automated Timetabling (PATAT'97) in Toronto [29] and its predecessor in Edinburgh [10]. Details about these conferences can be found at `http://www.asap.cs.nott.ac.uk/ASAP/ttg/patat.html`. Another indicator is the setting up of a new EURO Working group on Automated TimeTabling (WATT). Details about this group and how to join can be found at `http://www.cs.rdg.ac.uk/cs/research/pedal/watt/`.

Many researchers around the world are developing new timetabling software, and a huge volume of work has been published [9]. As yet, there is no benchmark standard to enable comparison of different methods, but researchers at a group of universities are working on this important problem. With the PATAT conferences now established as a regular event, it seems there will be much more work in this area.

ACKNOWLEDGEMENTS

The authors would like to thank Dave Corne, Alain Hertz, Jim Newall, Johnathan Thompson and George White for helpful comments on the paper.

REFERENCES

[1] Cooper, T. B. and Kingston, J. H. (1996) The Complexity of Timetable Construction Problems. In Burke, E. K. and Ross, P. (eds), *The Practice and Theory of Automated Timetabling*, pp. 283–295. Springer-Verlag, Berlin.

[2] Carter, M. W., Laportel, G. and Chinneck, J. W. (1994) A general examination scheduling system. *Interfaces*, **24**, 109–120.

[3] Bloomfield, S. D. and McSharry, M. M. (1979) Preferential course scheduling. *Interfaces*, **9**, 24–31.

[4] Romero, B. P. (1982) Examination scheduling in a large engineering school: a computer-assisted participative procedure. *Interfaces*, **12**, 17–23.

[5] Burke, E. K., Elliman, D. G., Ford, P. H. and Weare, R. F. (1996) Exam Timetabling in British Universities A Survey. In Burke, E. K. and Ross, P. (eds), *The Practice and Theory of Automated Timetabling*, pp. 76–90. Springer-Verlag, Berlin.

[6] Bardadym, V. A. (1996) Computer-Aided School and University Timetabling: The New Wave. In Burke, E. K. and Ross, P. (eds), *The Practice and Theory of Automated Timetabling*, pp. 22–45, Springer-Verlag, Berlin.

[7] Carter, M. W. (1986) A survey of practical applications of examination timetabling algorithms. *Operations Res.*, **34**, 193–201.

[8] Carter, M. W. and Laportel, G. (1996) Recent Developments in Practical Examination Timetabling. In Burke, E. K. and Ross, P. (eds), *The Practice and Theory of Automated Timetabling*, pp. 3–21. Springer-Verlag, Berlin.

[9] Kingston, J. H., Bardadym, V. A. and Carter, M. W. *Bibliography on the Practice and Theory of Automated Timetabling*. `ftp://ftp.cs.usyd.edu.au/jeff/timetabling/timetabling.bib.gz`.

[10] Burke, E. K. and Ross, P. (eds) (1996) *The Practice and Theory of Automated Timetabling*, Springer-Verlag, Berlin. *Lecture Notes in Computer Science*, **1153**.

[11] Corne, D., Ross, P. and Fang, H. L. (1994) Fast Practical Evolutionary Timetabling. Artificial Intelligence and Simulation of Behaviour (AISB) Workshop on Evolutionary Computing, University of Leeds, UK, 11–13 April. *Lecture Notes in Computer Science*, **865**, 251–263. Springer-Verlag, Berlin.

[12] Paechter, B., Cumming, A., Luchian, H. and Petriuc, M. (1994) Two Solutions to the General Timetable Problem using Evolutionary Methods. *Proc. IEEE Conference on Evolutionary Computation*.

[13] Burke, E. K., Elliman, D. G. and Weare, R. F. (1995) The automation of the timetabling process in higher education. *J. Educational Technol. Systems*, **23**, 257–266.

[14] Burke, E. K., Elliman, D. G. and Weare, R. F. (1995) A Hybrid Genetic Algorithm for Highly Constrained Timetabling Problems. *6th International Conference on Genetic Algorithms (ICGA'95)*, Pittsburgh, PA, 15–19 July. Morgan Kaufmann, San Francisco, CA.

[15] Burke, E. K., Elliman, D. G. and Weare, R. F. (1995) Specialised Recombinative Operators for Timetabling Problems. *Proc. of the AISB (Artificial Intelligence and Simulation of*

Behaviour) Workshop on Evolutionary Computing, University of Sheffield, UK, 3–7 April, pp. 75–85. Springer-Verlag, Berlin.

[16] Burke, E. K., Newall, J. P. and Weare, R. F. (1996) A Memetic Algorithm for University Exam Timetabling. In Burke, E. K. and Ross, P. (eds), *The Practice and Theory of Automated Timetabling*, pp. 241–250. Springer-Verlag, Berlin.

[17] Paechter, B., Cumming, A., Norman, M. G. and Luchian, H. (1996) Extensions to a Memetic Timetabling System. In Burke, E. K. and Ross, P. (eds), *The Practice and Theory of Automated Timetabling*, pp. 251–265. Springer-Verlag, Berlin.

[18] Thompson, J. and Dowsland, K. A. (1996) Variants of simulated annealing for the examination timetabling problem. *Ann. Operations Res.*, 63, 105–128.

[19] Thompson, J. and Dowsland, K. A. (1996) General Cooling Schedules for a Simulated Annealing based Timetabling System. In Burke, E. K. and Ross, P. (eds), *The Practice and Theory of Automated Timetabling*, pp. 345–363. Springer-Verlag, Berlin.

[20] Boufflet, J. P. and Nègre, S. (1996) Three Methods used to solve an Examination Timetable Problem. In Burke, E. K. and Ross, P. (eds), *The Practice and Theory of Automated Timetabling*, pp. 327–344. Springer-Verlag, Berlin.

[21] Hertz, A. (1991) Tabu search for large-scale timetabling problems. *Eur. J. Operations Res.*, No 54, 39–47.

[22] Hertz, A. (1992) Finding a feasible course schedule using Tabu search. *Discrete Appl. Math.*, 35, 255–270.

[23] Kang, L. and White, G. M. (1992) A logic approach to the resolution of constraints in timetabling. *Eur. J. Operational Res.*, 61, 306–317.

[24] Cheng, C., Kang, L., Leung, N. and White, G. M. (1996) Investigations of a Constraint Logic Programming Approach to University Timetabling. In Burke, E. K. and Ross, P. (eds), *The Practice and Theory of Automated Timetabling*, pp. 112–129. Springer-Verlag, Berlin.

[25] Boizumault, P., Delon, Y. and Péridy, L. (1995) A CLP Approach for Examination Planning, Constraint Processing. Selected Papers, *Lecture Notes in Artificial Intelligence*, Springer-Verlag, Berlin.

[26] Boizumault, P., Guéret, C. and Jussien, N. (1994) Efficient Labelling and Constraint Relaxation for Solving Timetabling Problems. *Proc. Workshop on Constraint Languages and their use in Problem Modelling*, International Logic Programming Symposium, Ithaca, NY, 18 November, pp. 116–130.

[27] Guèret, C., Jussien, N., Boizumault, P. and Prins, C. (1996) Building University Timetables Using Constraint Logic Programming. In Burke, E. K. and Ross, P. (eds), *The Practice and Theory of Automated Timetabling*, pp. 130–145. Springer-Verlag, Berlin.

[28] Lajos, G. (1996) Complete University Modular Timetabling using Constraint Logic Programming. In Burke, E. K. and Ross, P. (eds), *The Practice and Theory of Automated Timetabling*, pp. 146–161, Springer-Verlag, Berlin.

[29] Burke, E. K. and Carter, M. W. (eds) (1998) *The Practice and Theory of Automated Timetabling II*, in press. Springer, Berlin.