Timetabling in Constraint Logic Programming

Maria Kambi, David Gilbert

e-mail: {cb173, drg}@cs.city.ac.uk

tel: +44 171 477 8444
fax: +44 171 477 8587

Department of Computer Science
The City University
Northampton Square
London, EC1V 0HB
UK

Abstract

In this paper we describe the timetabling problem and its solvability in a Constraint Logic Programming Language. A solution to the problem has been developed and implemented in ECL^{i}PS^e, since it deals with finite domains, it has well-defined interfaces between basic building blocks and supports good debugging facilities. The implemented timetable was based on the existing, currently used, timetables at the School of Informatics at out university. It integrates constraints concerning room and period availability.

1. Introduction

During the last twenty years many contributions related to timetabling have appeared and it will probably continue with the same rate for years. The reason for this could be the huge variety of problems which are included in the field of timetabling; or it could be the fact that educational methods are changing, so models have to be modified to meet those changes.

One specialised area of timetabling is the school or university timetables. In particular, course timetabling within a university has been a tedious task for university administration. Its aim is to arrange periods, modules, rooms and lecturers to all courses in an academic year. The limited number of human and material resources available, such as lecturers, rooms or time, and the restrictions of their use means that the timetables have to be constrained so that certain conditions are met.

The construction of timetables, involving three or more variables, taking values from domains, having hundreds of values with several constraints, is a very common problem. Various heuristic solutions have been proposed using results based on graph theory, mathematical programming and manual methods. In this paper, an approach based on Constraint Logic Programming is proposed.

Constraint Logic Programming is a relative newcomer to the area of timetabling although some work has been done in related areas. It is particularly well suited for timetabling problems, since it allows the formulation of all constraints of the problem in a more declarative way than other approaches. Constraints help to solve the problem in a finite amount of time, by stating various relationships between the variables.

2. The Timetabling Problem

Every year or term in a university, every individual department has to design a new timetable for courses. The timetabling problem consists of placing these courses (modules) which share resources, such as lecturers or classrooms, in a weekly calendar. In the School of Informatics at our university, a module usually consists of lectures of two hours per week and tutorials of one hour per week. Several constraints have to be considered for each timetable, for example no two lectures can take place at the same room at the same time, or the capacity of a room must not be less than the number of students attending a lecture, and so on.

Until now, the timetables are constructed by hand and consequently leading to problems concerning the correctness of the timetable, as well as time consuming problems. Usually, the timetabling process consists of two distinct phases:

- First, the curricula are defined for each module and the various resources (time, rooms, lecturers) must be assigned to the classes.
- Second, a feasible timetable must be found, which is compatible with the previously defined requirements.

Generally the role of computers is to handle the second phase.

3. CLP and ECL^{i}PS^{e}

Constraint Logic Programming (CLP) is a generalisation of Logic Programming (LP) where unification, the basic operation of LP languages, is replaced by constraint handling in a constraint system. The resulting languages combine the advantages of LP (declarative semantics, non-determinism, relational form) with the efficiency of constraint-solving algorithms. Therefore, the advantages of such languages rely on the fact that the user does not need to concern himself with search techniques, the stating of the constraints is straightforward and the programs can easily be modified and extended.

In the paradigm of constraint logic programming, a constraint satisfaction problem can be written in the form of Horn clause logic programs in which the clause bodies may contain constraints. Constraints are generated incrementally during run-time and passed to a constraint solving mechanism which applies a domain-dependent constraint satisfaction technique, such as linear programming, Boolean unification and so on, to find a feasible solution for the constraints. Such an approach is still search-based, but it can effectively reduce the search space and improve the inefficient 'generate and test' nature of problem solving. So CLP introduces a new method of computation known as the 'constrain and generate' approach.

ECL^{i}PS^{e} (ECRC[1] Common Logic Programming System) is a Constraint Logic Programming language. It is a Prolog based system whose aim is to serve as a platform for integrating various logic programming extensions. The kernel of ECL^{i}PS^{e} is an efficient implementation of standard Prolog and it is built around an incremental compiler which compiles the Prolog source into WAM[2]-like code [Ecl95a].

The ECL^{i}PS^{e} logic programming system is an integration of ECRC's SEPIA, MegaLog and CHIP system and newly developed libraries. This combination is now the default configuration of the system. ECL^{i}PS^{e} has become a tool which is powerful, flexible and general enough to be of use to all programmers in the LP field [Ecl95b]. In particular, the finite domains library of ECL^{i}PS^{e} implements constraints that involve integers as well as atomic data and supports the writing of user-defined constraints over variables with finite atomic or ground domains.

4. Timetabling in a CLP language

Constraint Logic Programming is particularly well suited for timetabling problems. It allows the formulation of all the constraints in a declarative way. Although its performance can be greatly affected by minor changes in problem formulation, the alternatives in formulation are still within the bounds of logical equivalence.

The timetable problem is a large scale combinatorial problem, with an extremely large search space. Thus, by solving the problem in a CLP language, the search space can effectively be reduced to smaller search trees that lead to results within a finite amount of time.

In a CLP language that makes use of finite domains, e.g. ECL^{i}PS^{e} or CLP(FD), the time periods and rooms can be expressed in terms of integers within given domains. Then, most of the constraints can be integrated as arithmetic constraints. As in all CLP programs used to solve Constraint Satisfaction Problems, the overall structure of a

[1] ECRC is the name for the European Computer Industry Research Centre.
[2] WAM are the initials for Warren Abstract Machine.

timetable program would first state the domains of the problem variable, then state the constraints and finally generate the values.

5. Design and Implementation

At this time, the construction of the timetables in our university is paper based. The main procedures for constructing a timetable by hand, as explained by the School of Informatics' timetabler [Fel95] are discussed below.

First of all most of the information comes from the course director, concerning the modules offered to students and the lecturers teaching each module. Then all the lecturers state their preferences e.g. if they like double period lectures, on which days they prefer to give lectures, how they like their tutorials to be taught (e.g. in small groups) and where (room or a laboratory) etc. Information about the periods that must be reserved for student union actions and lunch breaks are also taken into account. In addition, the time length of lectures and generally of the periods must be known.

In the case of the MSc course, where both full-time and part-time students are taking various modules, the vast majority of the lectures are taught on two certain days.

With this information in mind, a timetable is written on paper. No program is used for completing the timetable, although a word-processing package is used at the end to write the timetable in a presentable form.

The timetabler is not responsible for the room allocation. This is the job of certain people that allocate rooms for the timetables of the whole university. The School of Informatics timetabler just presents his requirements for rooms along with the adjusted capacity and the periods during which the rooms are needed, to these people and they, in turn, accept or reject his requirements and allocate the rooms. When a correct version of the timetable is ready, it is presented to the course director. If it is approved, it is passed to the lecturers and students.

In a similar way a CLP program will follow the main steps in order to produce a feasible timetable program.

The facts that will be given to the program are:

- The teaching periods,
- The lecturers and which modules they can teach,
- The rooms and their capacities,
- Various constraints.

The program will then:

- Allocate lecturers to modules,
- Allocate teaching periods to modules,
- Allocate rooms to teaching periods,
- Generate a timetable.

The timetable to be constructed consists of 45 periods. The first lecture on one day starts at 9:00am, while the last one starts at 5:00pm. Thus, each day comprises of 9 periods and each week of 45 periods. Period 1 is the first period on a Monday, period 2 is the second period on a Monday, period 10 is the first period on a Tuesday, and so on. All lectures start at 9:00, 10:00, 11:00 etc. with a 10 minute break at 50 minutes past each hour. Thus each lecture lasts for 50 minutes.
The timetable program will schedule its main elements (lecturers, periods, modules and rooms) in a way that:

- No two modules must be taught in the same room at the same time.
- All modules must be scheduled.
- All rooms must be large enough to hold the classes assigned to them.
- No lecturer can be teaching in two rooms at the same time.

The above consist the main constraints of the program that must be satisfied in all instances of any timetable. Some additional constraints integrated in this specific timetable are stated below:

- Periods 24-27 (Wednesday afternoon) are reserved for student union activities.

- Periods 5, 14, 23, 32, 41 (periods between 1:00-2:00 each day) are available for lunch.

- Each module is taught in two periods of lectures and one period of tutorial/lab.
- All modules taught to the first year students are obligatory.
- Double lectures cannot begin on one day and finish on the next day.
- There must be no nine lectures taught on one day.
- All double periods must be stated and there must be a free period before and after each double period.

The timetable will be represented as a set of quadruples, each quadruple representing a period along with the module that is taught at that period, the lecturer teaching that module and the room in which the lecture will take place. The constraints on the periods, modules, lecturers and rooms will be represented as arithmetic constraints. The program will then assign a period to a module, at which period that module is to be taught. The program must also find a room that will fit all the students attending that lecture. The capacities of the room and the minimum and maximum numbers of students attending each lecture are given as data to the program.

The program integrates the timetables for the first and second year of a university degree. In the same way it can be extended to include as many years as required.

5.1 Set Theoretic Specification

The variables in the timetable problem are the periods, the modules, the rooms and the lecturers. Their domains, using a mathematical description, based on set theory, are the following:

Period	1..45
Mod	M
Modpart	$M \rightarrow \{1,2,3\}$
Room	R
Lecturer	L

Periods can take values form 1 to 45 and modules can take values from M, where M is the set of the module names. Since each module is taught in three periods, *Modpart* is used which maps each module to the domain {1,2,3}, thus forming the three taught periods (module parts) for each module. Finally, rooms take values from the domain R and lecturers take values from the domain L.

The main functions on the above domains are the following:

taughtby:	Modpart $\rightarrow$ L
when:	Modpart $\rightarrow$ Period
where:	Modpart $\rightarrow$ R

The function *taughtby* denotes which lecturer teaches which module part. The function *when* gives the period in which a module part is taught and the function *where* gives the room in which it is taught.

The invariant is a rule which, when recognised, provides understanding. The invariant properties specify the relationship that must hold between the values of the objects in the timetable system. The invariants for the above functions are the following:

- No room can be used for more than one module part at a time. This is represented as

  $$(\text{where } \& \text{ when})^{-1} \in (\text{Period x R}) \mapsto \text{Modpart}$$

  The & operator is defined as

  $$\&: \quad (X \leftrightarrow Y) \text{ x } (X \leftrightarrow Z) \rightarrow (X \rightarrow (Y \text{x} Z))$$

  So (where & when) is the function (Modpart→(Period x R)) and the inverse of that is the function ((Period x R)→ Modpart). But the rule states that the latter is a partial function ($\mapsto$), i.e. an instance of the set (Period x R) that denotes a room at a certain period, can only map to one module part.

- Every module part is taught by the same lecturer. This is represented as

  $(\mathrm{dom} \circ (\mathrm{taughtby})^{-1})^{-1} \in M \to L$

  The *dom* operator gives the domain of a function. The inverse of the function *taughtby* is L→Modpart and consequently L→{1,2,3}, having L as its domain. The inverse of that will now be {1,2,3}→L and will map the module parts to lecturers. But, as stated by the rule, this belongs to the set M→L, which maps each module to a lecturer. So all module parts representing the same module must be taught by the same lecturer.

- No lecturer can be teaching in two rooms at the same time. This is represented as

  $\mathrm{codom}((\mathrm{taughtby} \,\&\, \mathrm{when}) \,\&\, \mathrm{where}) \in (L \times \mathrm{Period}) \mapsto \mathrm{Modpart}$

  The *codom* operator gives the range of a function. According to the definition of the & operator, the function ((taughtby & when) & where) becomes the function (Modpart → (L x Period x R)) which has the range (L x Period x R). But this belongs to the partial function ((L x Period) ↦ Modpart) which states that an instance of the set (L x Period) can only map to one value of Modparts and thus denoting that at a given period a lecturer can only teach one module part.

5.2 High Level Design

The high level design of the timetable program can be derived from the mathematical description of the problem, defined in the previous section using set theory.

In the high level design the periods are stated as before and can take values from 1 to 45. The *Mod* and *Modpart* variables in the mathematical description are now collapsed to form one set of variables representing the various parts of all the modules, i.e. if there are two modules, A and B, the module set will contain the variables A(1), A(2), A(3), B(1), B(2), B(3), where A(1), A(2), A(3) are the module parts for module A and B(1), B(2), B(3) are the module parts for module B. The rooms are enumerated and can take values from 1 to 20. Lecturers are also enumerated.

The timetable I designed includes two years of a degree, each one having five modules. Each module is taught in three periods, so we have a total of thirty taught periods, thirty rooms that must be reserved and ten lecturers teaching the modules. The set of periods, rooms and lecturers, are restricted so that they do not have repeated values. Consequently, they can be represented as lists.

A high level representation of the timetable program will be the following:

```
Periods1st = [P1,P2,P3,P4,P5,P6,P7,P8,P9,P10,P11,P12,
               P13,P14,P15],
Periods2nd = [P16,P17,P18,P19,P20,P21,P22,P23,P24,P25,
               P26,P27,P28,P29,P30],
Room = [R1,R2,R3,R4,R5,R6,R7,R8,R9,R10,R11,R12,R13,R14,
         R15,R16,R17,R18,R19,R20,R21,R22,R23,R24,R25,R26,
         R27,R28,R29,R30],
Lecturer = [L1,L2,L3,L4,L5,L6,L7,L8,L9,L10],
Periods1st :: 1..45,
Periods2nd :: 1..45,
Room :: 1..20,
Lecturer :: [white,peterson,novac,smith,black,johnson,
              daniels,conlon,fisher,roberts],
alldifferent(Periods1st),
alldifferent(Periods2nd),
alldifferent(Lecturer).
```

Periods1st and Periods2nd express the taught periods for the first and the second year respectively. The variable *Room* can take values from 1 to 20, since there are twenty rooms in the building.

The invariant denoting that no room can be used for more than one module part at a time will be implemented in association with the periods. In one year all modules are obligatory and are taught at different times, so there will be no room booked for two different module parts at the same time. But when we have two years and two modules (the first module from one year and the second module from another year) that can appear at the same time, a clause must be defined that ensures that these modules are not taught in the same room.

The invariant denoting that every module part must be taught by the same lecturer will be implemented by having the same lecturer variable appearing as the lecturer for all parts of a module, e.g. lecturer L1 will be teaching modules A(1), A(2) and A(3), where A(1),A(2),A(3) are the module parts for module A.

5.3 Implementation

The timetable program is written in ECL^{i}PS^e.

At the beginning of the program, data about which modules are taught by which lecturers, the number of students attending the modules and the capacities of all the rooms are stated.

For example the fact

```
teaches(white,databases).
```

states that Mr. White may teach the module 'databases'. In the same way all the modules that the lecturers teach are stated.

A clause of the form

```
find_room(db(D), Num):-D#<3,Num#<=60,Num#>=20.
```

denotes that the two lectures of the module 'databases' (db) have 20-60 students attending it.

Then a clause of the form

```
capacity(1,Capacity):-Capacity#<=30,Capacity#>=15.
```

denotes that room 1 can fit 15-30 students.

The program, by combining the above two clauses, can then allocate a room to a module, according to the capacity of the room, for example the rule

```
find_room(db(1),Capacity), capacity(R1,Capacity)
```

will find a room for the first lecture of the module 'databases', by matching the number of students attending that lecture (variable *Capacity* in the *find_room* predicate) with the capacity of a room (variable *Capacity* in the *capacity* predicate).

According to the design, the timetable is represented as a list of quadruples of the form

```
m(P1,db(1),R1,L1)
```

where P1 is the period at which the first lecture of the module databases (db(1)) is taught. The lecture will be taught in room R1 by lecturer L1.

The variables for periods, lecturers and rooms, and their domains are written in ECL^{i}PS^e as they appear in the high level design.

All the periods in the first year must be unique, since all the modules are obligatory and must not be taught at the same time. The same holds for the periods in the second year. So the following predicates are used, that make sure that the periods in the first and second year are unique

```
alldifferent(Periods1st),
alldifferent(Periods2nd)
```

At the beginning of the program, the list of the free periods for the first and the second year are given in the lists *FreePeriods1st* and *FreePeriods2nd*. The free periods are reserved either for student activities and are fixed, or they vary according to the allocation of double periods (explained later).

The constraints about the periods and the rooms take the form of arithmetic constraints, for example the constraint

```
P2#<10
```

states that period 2, representing the second taught period of the module databases is on Monday.

Constraints using the predicate *between* state that the taught period for a lecture will take values from a more restricted domain. For example the constraint

```
between(P9,27,37)
```

states that period P9, representing the third period for the module 'computer hardware' must appear between periods 27 and 37 , i.e. on Tuesday.

In addition to the above, a period constraint can force a period to take values from a given domain, by using the built-in predicate *member*. So, the constraint

```
member(P10,[6,15,33,42])
```

denotes that P10, representing a lecture for software engineering, can take the values 6, 15, 33 or 42.

At the beginning of the program two lists of free periods are stated. The clause *notin* will constraint the taught periods of the first or the second year so that they do not appear in the corresponding list of the free periods for that year.

A clause *double_period* will constrain two periods of the same module to be consecutive, thus forming a double period. The clause also makes sure that the double period is not split in two days, i.e. the first half of the double period cannot appear as the last period of one day and the second half as the first period of the next day.

Any period before and after a double period must be free, in order to avoid any displeasure of students. Such periods are expressed using arithmetic constraints and then adding these constraints in the free periods list. For example the predicate

```
BefrDBdouble#=P1-1
```

finds the period before P1 (P1 representing the first half of the databases double period). Then the period *BefrDBdouble* is added to the *FreePeriods1st* list.

In order to prevent any clash of the rooms, i.e. allocate the same room at the same period for two different lectures, the clause *noclash* is used. At the beginning of the program two lists representing the periods at which the rooms are occupied for the first and the second year. These lists are the following:

```
Rooms1st=[(P1,R1),(P2,R2),...(P15,R15)],
Rooms2nd=[(P16,R16),(P17,R17),...,(R30,R30)]
```

Then a call of the clause

```
noclash(Rooms1st,Rooms2nd)
```

will prevent such a clash.

Since the periods can take values from a large domain, the constraints cannot restrict them enough in order to give just one value for each period. Instead, the solution of the timetable will give a range of the values the periods can take. A real-life timetable would be useless if it was presented in such a away. In order to avoid this, the periods are instantiated at the end of the program. This is accomplished by using the built-in predicate *labeling*. So, the program finishes with the predicates:

```
labeling(Periods1st),
labeling(Periods2nd).
```

It must also be noted that the order of the clauses in the program is very important. Any constraints must be stated at the beginning of the program, then the code for the various clauses must follow and at the end the labelling procedure is added. If the constraints and the code are not given in this order, then the program might fail to give a solution.

Finally, once the timetable is solved, the periods are sorted in an ascending order, the free periods are inserted and the whole timetable is presented in a readable form.

6. Testing

Once the timetable program was compiled and executed, it gave as output the timetables for the first and the second year of a university degree. The program produced all the possible solutions for the two timetables.

The tests of the program were executed during or after the implementation of the program. They included testing the correctness of the output routine, the labelling procedure, the ordering of the code and tests to ensure there were no clashes between any booked rooms. Finally, the program was tested and was able to check the feasibility of a given timetable.

7. Conclusions

This work has solved the timetable problem using the Constraint Logic Programming language ECL^{i}PS^e. The time taken to implement, debug and test the program was about ten weeks. The timetable was implemented successfully, giving correct results.

The use of ECL^{i}PS^e proved to be very useful, since it could produce complicated timetables in very short time (in terms of seconds). However, ECL^{i}PS^e could not be used to implement the optional constraints, that would be very important in real-life timetables. This could however be implemented in the future by using Hierarchical Constraint Logic Programming.

References and Bibliography

[AB94] F. Azevedo and P. Barahoma. Timetabling in Constraint Logic Programming. In *Proceedings of World Congress on Expert Systems ’94*, Estoril, January 1994.

[ABR93] Logic Programming Languages: Constraints, Functions and Objects, edited by K. R. Apt, J. W. de Bakker and J. J. Rutten. The MIT Press, 1993.

[BC93] Constraint Logic Programming: Selected Research, edited by Frederick Benhamou and Alain Colmerauer. The MIT Press, 1993.

[CKLW95] C. Cheng, L. Kang, N. Leung and G. White. Investigations of a Constraint Logic Programming Approach to University Timetabling. In *Proceedings of the 1st International Conference on the Practice and Theory of Automated Timetabling (ICPTAT ’95)*, pp. 82-93, Napier University, Edinburgh, August-September 1995.

[CL95] A. Colijn and C. Layfield. Interactive Improvement of Examination Schedules. In *Proceedings of the 1st International Conference on the Practice and Theory of Automated Timetabling (ICPTAT ’95)*, pp. 112-121, Napier University, Edinburgh, August-September 1995.

[CLPR] The CLP($\mathcal{R}$) Programmer’s Manual, version 1.2.

[DH92] Yves Deville and Pascal van Hentenryck. Construction of CLP Programs. In *Logic Programming: New Frontiers*, edited by D. R. Brough. Klwwer Academic Publishers, 1992.

[Dre95] Dr Alain Dresse. A Constraint Programming Library Dedicated to Timetabling. In *Proceedings of the 1st International Conference on the Practice and Theory of Automated Timetabling (ICPTAT ’95)*, pp. 122-131, Napier University, Edinburgh, August-September 1995.

[Ecl95a] ECLiPSe 3.5 User Manual, Munich, February 1995.

[Ecl95b] ECLiPSe 3.5 Extensions User Manual, Munich, December 1995.

[EK95] Wilhelm Erben and Jurgen Keppler. A Genetic Algorithm solving a Weekly Course-Timetabling Problem. In *Proceedings of the 1st International Conference on the Practice and Theory of Automated Timetabling (ICPTAT '95),* pp. 21-32, Napier University, Edinburgh, August-September 1995.

[Fel95] Terry Felstead. Personal communication. 1995.

[FHS95] H. Frangouli, V. Harmandas and P. Stamatopoulos. UTSE: Construction of Optimum Timetables for University Course - A CLP based approach. In *Proceedings of the Third International Conference and Exhibition on Practical Applications of Prolog (PAP '95),* pp. 225-243, Paris, April 1995.

[GJBP95] ChristelleGueret, Narendra Jussien, Patrice Boizumault and Christian Prins. Building University Timetables using Constraint Logic Programming. In *Proceedings of the 1st International Conference on the Practice and Theory of Automated Timetabling (ICPTAT '95),* pp. 393-408, Napier University, Edinburgh, August-September 1995.

[Hen89] Pascal van Hentenryck. Constraint Satisfaction in Logic Programming. The MIT Press,1989.

[Hen91] Pascal van Hentenryck. Constraint Logic Programming. *The Knowledge Engineering Review,* Vol 6:3, 1995, pp. 151-154.

[Hog84] Christopher J. Hogger. Introduction to Logic Programming. Academic Press, Inc., 1984.

[JM94] Joxan Jaffar and Michael J. Maher. Constraint Logic Programming: A Survey. *Journal of Logic Programming 19/20,* 1994, pp. 503-581.

[Joh80] K. Johnson. Timetabling. Hutchinson Group Ltd, 1980.

[KM95] A. C. Kakas and A. Michael.. Timetabling in an Integrated Abductive and Constraint Logic Programming Framework. In *Proceedings of the 1st International Conference on the Practice and Theory of Automated Timetabling (ICPTAT '95),* pp. 167-176, Napier University, Edinburgh, August-September 1995.

[KW92] Le Kang and George M. White. A Logic Approach to the Resolution Of Constraints in Timetabling. *European Journal of Operational Research 61,* 1992, pp. 306-317.

[Laj95] Gyuri Lajos. Complete University Modular Timetabling using Constraint Logic Programming. In *Proceedings of the 1st International Conference on the Practice and Theory of Automated Timetabling (ICPTAT '95),* pp. 364-375, Napier University, Edinburgh, August-September 1995.

[Lel88] Wm Leler. Constraint Programming Languages: their Specification and Generation. Addison - Wesley, 1988.

[MA95] Micha Meier and Alexander Herold. CLP in ECRC. In *Proceedings of First International Conference on Principles and Practice of Constraint Programming (CP '95),* pp. 205-221, Cassis, France, September 19-22, 1995.

[MTU94] B. Mayoh, E. Tyugu and T. Uustalu. Constraint Satisfaction and Constraint Programming: A Brief Lead-In. In *Constraint Programming*, edited by Brian Mayoh, Enn Tyugu and Jaan Penjam. NATO ASI Series, Series F: Computer and Systems Sciences, Vol 131,1994.

[RBP95] Gilles Roux, Jean-Louis Bouquard and Pierre-Yves Partant. Solving school timetabling problems with CHIP. In *Proceedings of the Third International Conference and Exhibition on Practical Applications of Prolog (PAP '95),* pp. 511-515, Paris, April 1995.

[RH95] Vincent Robert and Alain Hertz. How to decompose constrained course scheduling problems into easier assignment type subproblems. In *Proceedings of the 1st International Conference on the Practice and Theory of Automated Timetabling (ICPTAT '95),* pp. 177-184, Napier University, Edinburgh, August-September 1995.

[SS94] Leon Sterling and Ehud Shapiro. The Art of Prolog. The MIT Press, 1994.

[Wal94] Mark Wallace. Applying Constraints for Scheduling. In *Constraint Programming,* edited by Brian Mayoh, Enn Tyugu and Jaan Penjam. NATO ASI Series, Series F: Computer and Systems Sciences, Vol 131,1994.

[Wer85] D. de Werra. An introduction to timetabling. *European Journal of Operational Research,* Volume 19, Number 1, January 1985, pp. 151-162.

[Wer95] D. de Werra. Some Combinatorial Models for Course scheduling. In *Proceedings of the 1st International Conference on the Practice and Theory of Automated Timetabling (ICPTAT '95),* pp. 1-13, Napier University, Edinburgh, August-September 1995.