November 2005

Constructing university timetable using constraint satisfaction programming approach

L. Zhang
University of Wollongong

Sim Kim Lau
University of Wollongong, simlau@uow.edu.au

The timetabling problem consists of a set of subjects to be scheduled in different timeslots, a set of rooms in which the subjects can take place, a set of students who attend the subjects, and a set of subjects satisfied by rooms and required by timeslots. The heart of the problem is the constraints that exist as regulations within each resource and between resources. There are various solution approaches to solve the timetabling problem. This paper focuses on developing a constraint satisfaction problem model for a university timetabling problem. A solution of a constraint satisfaction problem is a consistent assignment of all variables to values in such a way that all constraints are satisfied. A sample case study problem is investigated and a constraint satisfaction programming approach is implemented using ILOG Scheduler and ILOG Solver. We use various goals in ILOG to investigate the performance of the CSP approach.

Constructing university timetable using constraint satisfaction programming approach

Lixi Zhang and SimKim Lau
Information Systems Discipline
School of Economics and Information System
University of Wollongong
Wollongong, NSW 2500, Australia
Email: {lxz01, simlau}@uow.edu.au

Abstract

The timetabling problem consists of a set of subjects to be scheduled in different timeslots, a set of rooms in which the subjects can take place, a set of students who attend the subjects, and a set of subjects satisfied by rooms and required by timeslots. The heart of the problem is the constraints that exist as regulations within each resource and between resources. There are various solution approaches to solve the timetabling problem. This paper focuses on developing a constraint satisfaction problem model for a university timetabling problem. A solution of a constraint satisfaction problem is a consistent assignment of all variables to values in such a way that all constraints are satisfied. A sample case study problem is investigated and a constraint satisfaction programming approach is implemented using ILOG Scheduler and ILOG Solver. We use various goals in ILOG to investigate the performance of the CSP approach.

1. Introduction

Every year or term in a university, every individual department has to design a new timetable for subjects. The timetabling problem consists of placing these subjects, which share resources, such as lecturers and classrooms, in a weekly calendar. The timetabling problem is a historic problem and much research has been investigated in this area. Solutions to timetabling problems have been proposed since the 1960s [1--20].

The timetabling problem exhibits the unwelcome nature of combinatorial problem. It is difficult to find an optimal solution when the number of resources and constraints increases. Actually, all problems related to building a timetable are known to be NP-Complete [21, 22, 8]. Various methods have been proposed to solve the timetabling problem such as graph-coloring problem [1, 3, 5] and integer linear programming technique [5]. There are also various meta-heuristic methods such as simulated annealing [6], tabu search [10, 19] and genetic algorithms [9, 16] that have been used to solve a variety of timetabling problems.

The remainder of this paper is structured as follows. Section 2 provides background knowledge of timetabling problem and the constraint satisfaction approach used to solve the problem. Detail discussion of search algorithms and consistency techniques in solving constraint satisfaction problem will be presented. Section 3 presents the model used to solve the timetabling problem. Section 4 presents the implementation of the Constraint satisfaction approach using the ILOG Scheduler and Solver. The results of the sample case study problem are given. Finally, promising paths of research are discussed in the conclusion.

2. The approaches to solve timetabling problem

Wren [23, p.46] defines the timetabling problem as a special case of scheduling**:** **“**Timetabling is the allocation, subject to constraints, of given resources to objects being placed in space time, in such a way as to satisfy as nearly as possible a set of desirable objectives.” Timetabling problem is generally considered as a resource allocation problem in Operations Research, where resources of lecturers, students, classrooms and subjects are to be allocated into timeslots of a weekly timetable to achieve an objective function subject to constraints among resources [18].

Timetabling problems is a type of assignment problems with large amount of complex constraints, thus usually can be easily modeled as constraint satisfaction problems (CSP) [17]. The application for solving the timetabling problem using constraint satisfaction programming approach allows the formulation of all the constraints of the problem in a more declarative way than other approaches [13, 14]. Thus the CSP 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.

Constraint satisfaction problem (CSP) deals with assignment of values from its domains to each variable such that no constraint is violated [24, 25]. CSP has three components: variables, values and constraints. In general, CSP consists of: a finite set of variable $X = \{x_1,\ldots,x_n\}$ with respective domains $D = \{D_1,\ldots, D_n\}$ which list the possible values for each variable $D_i = \{v_i,\ldots,v_k\}$ and a set of constraints $C = \{C_1, \ldots, C_t\}$ [25, p.31]. The constraints limit the possible values that a variable can have. A solution of a CSP is a consistent assignment of all variables to values in such a way that all the constraints are satisfied.

There are two approaches to solving CSP. One is using the search algorithms and the other is using the consistency technique. Consistency techniques have been widely studied to simplify constraint network before or during the search of solutions. Dechter [25] defines arc-consistency as a process that ensures any valid value in the domain of a single variable has a valid match in the domain of any other variables in the problem. Arc $(V_i,V_j)$is arc consistent if for every value x in the current domain of V_i there is some value y in the domain of V_j such that V_i=x and V_j=y is permitted by the binary constraint between V_i and V_j. The concept of arc-consistency is directional. If the process involves three variables then it is known as path consistency. In general a graph is k-consistent if there exists (k-1) variables that satisfy all the constraints among these variables and there also exists a value for this k^{th} variable that satisfies all the constraints among these k variables [25].

Most algorithms for solving the CSP search systematically through the possible assignments of values to variables. Such algorithms are guaranteed to find a solution if one exists or to prove that the problem has no solution, but this process may take a very long time. Backtracking is the most common method for performing systematic search. In the backtracking algorithm, the current variable is assigned a value from its domain. This assignment is then checked against the current partial solution. If any of the constraints between this variable and the last variables is violated, the assignment is abandoned and another value for the current variable is selected [25]. There are three disadvantages of backtracking approach: thrashing, redundant work and late detection of conflict [26]. Thus look-ahead scheme is proposed to overcome some or all of these problems. The look-ahead scheme is invoked whenever the algorithm is preparing to assign a value to the next variable [25]. There are two approaches in the look ahead scheme. The first approach is called forward checking. This approach checks only the constraints between the current variable and the future variables. When a value is assigned to the current variable, any value in the domain of a future variable, which results in conflicts with this assignment, is removed from the domain. This means if the domain of the future variable is empty, it infers that the current partial solution is inconsistent and another value should be tried or it should backtrack to the previous variable [27]. The second approach is called (full) look ahead or maintaining arc-consistency. This is an approach that uses full arc-consistency during the look ahead scheme. It allows branches of the search tree that will lead to failure to be pruned earlier [28].

Look back schemes are invoked when the algorithm encounters a dead-end and prepares for the backtracking step [25]. All look back schemas share the disadvantage of late detection of the conflict. It solves the inconsistency when it occurs but does not prevent the inconsistency from occurring. There are two approaches to look back scheme: backjumping and backmarking. Backjumping works the same way as backtracking. The difference is during the backtracking step. In backjumping, it analyses the situation in order to identify the source of inconsistency. Backjumping backtracks to the most recent conflicting variable, whereas backtracking backtracks to the immediate past variable [27]. In backmarking, it avoids redundant constraint checking by recording the highest level that is last backtracked to. This helps to reduce repetitive consistency checking by remembering the success and failure of compatibility checks, which have already been performed [27].

3. CSP model for timetabling problem

The problem consists of scheduling a set of classes (lectures and tutorials) in different timeslots subject to satisfying the following constraint: no student attends more than one class at the same time, the room must be big enough for all the attending students, no core subject is scheduled at the same time, and only one class is scheduled in one room at any one timeslot.

Let

$S = \{s_1, s_2, \ldots s_n\}$ be the set of students;

$L= \{l_1, l_2, ...l_o\}$ be the set of subjects taught;
$T= \{t_1, t_2, ..., t_m\}$ be the available teaching periods ;
$R = \{r_1, r_2, ..., r_p\}$ be the set of rooms available.
where
S_l represents the set of students who take the subject l;
T_l is the set of timeslots allocated to the subject l;
R_l is the set of rooms assigned to the subject l.

Then $T_{l\ i}$ is the number of teaching periods for subjects l_i . S_{li} is the set of student's wishes to attend the subject l_i and R_{li} is the set of rooms that can be assigned to the subject $l_{i.}$ A feasible timetable is one in which all events have been assigned a timeslot and a room so that the following constraints are satisfied:

- no student attends more than one subject at the same time;

$$\forall(S_{li} = S_{lj})\ \exists\ T_{li} \neq T_{lj}$$

where $S_{li} = S_{lj}$ represents the student who take two subjects l_i and l_j , these two subjects can not be held at the same time.

- only one subject is in each room at any timeslot;

$$\forall(l_i = l_j)\ \exists\ T_{li} \neq T_{lj}$$

where $T_{li} \neq T_{lj}$ represents the timeslot is allocated to the subject l_i and $l_{j..}$ When the two subjects l_i and l_j are held in the same room, they need to be held at different timeslots.

- the room size RSize(R_j) of room R_j is satisfied for all the features required by the subjects R_{li} and is big enough for all the attending students size SSize(S_i);

$$\forall(\ R_{li} = R_j)\ \exists\ RSize(R_j) > SSize(S_i)$$

where $SSize(S_{li})$ represents the size of students who attend the subject $l_{i.}$

- some of the timeslots have been reserved for special events E. Therefore, these timeslots should not be assigned any subjects;

$$\forall(T_j = E)\ \exists\ T_l \neq T_j$$

where $T_l \neq T_j$ means the set of timeslots is allocated to the subject l can not be equal to the special timeslot T_j when T_j is equal to special event E.

In the sample case study, we have 24 subjects that made up of 203 timetable items to be scheduled into 12 rooms in 54 timeslots. A timetable item refers to either a lecture or a tutorial class. Background and problem description of the case study is described as follows. A subject on offer is always made up of weekly lecture and tutorial. One or more lecturers can teach a subject. Each lecture usually runs for two hours. However a lecturer may request to have a one-hour lecture only. There are two types of tutorial: classroom-based tutorial or laboratory-based tutorial. Both types of tutorial can either be one or two hour duration. In each case, the lecturer will specify the maximum number of students allowed to enrol in each tutorial group (this way the number of classes or tutorial groups that are required for the subject can be computed by the system). Other constraints under consideration are in the form of regulations such as lecture time can only be scheduled from 8 o'clock in the morning to 6 o'clock in the evening and tutorials not to be scheduled after 8pm. In addition, no lecture or tutorial must be scheduled between 1pm and 2pm on Wednesday to allow the teaching staff to attend meetings or seminars. In addition, a lecturer may make special requests so that individual requirements can be taken into consideration during the timetabling planning process. Example of such request includes a certain lecturer can only teach in a particular day or time of the week due to the nature of the employment such as part-time lecturer. In addition due to the way the subject is designed, a lecturer may make request such that a student can only take tutorial class after the lecture is conducted. Other examples of pre-specified requirements include a repeat lecture which caters mainly for part-time students must be held in the evening, and if a subject can only be taught by one lecturer then different tutorial groups cannot be scheduled concurrently.

The model we propose for a timetabling problem as a CSP is as follows: a timetable is a constrained variable the value of which is a function associating a value to each slot in time t. The timetable item is given by the set of subjects. Note that the subject can be offered as a lecture or a tutorial, which is considered as a timetable item. Basically our task consists in instantiation of the set of three tuples CSP (timetable item, classroom, time), i.e., each lecture or tutorial of a subject has assigned its set of classroom and time.

We use ILOG to solve the timetabling problem in our research. ILOG was created in 1987 to industrialise the expertise of INRIA (the National Institute for Research in Computer Science and Control), Europe's largest computer research centre in the field of symbolic computer languages and object-oriented environments [29]. ILOG decomposes the problem by separating the models from the search algorithms. This way, it is easy to change different algorithms applied to the same model [29]. We use two modules of ILOG in our implementation: Scheduler and Solver.

In solving the problem, we define each timetable item as an activity and the room as resource. In this timetabling problem, we use `IloActivity` to represent the subjects which is represented as the timetable item of lecture and tutorial. We use

IloUnaryResource to represent the room resource. Then we use the ILOG Scheduler to build the model of the CSP. After the model is complete, it is extracted to the ILOG Solver. ILOG Solver is needed to generate the goal and to search for a solution. When the solution is obtained, the timetable in term of subject number, time and room number are displayed. ILOG Solver also provides a set of control primitives that allow user to implement his/her own heuristic search algorithm.

4. Implementation and Experimental Results

We will present and analyze the results obtained using different goals in ILOG Solver. The programs are run on a DELL personal computer with Intel Pentinum 4, 1.6 G CPU, 512 M memory and Linux 2.4 operating system.

In ILOG, a goal is used to define the search for a solution to a model. The model for which an instance of a goal will search for a solution is specified via the IloSolver. In our experiment, we will use the predefined functions such as IloRankForward, IloRankBackward, IloSetTimesForward and IloSetTimesBackward that return a goal to assign start times to activities in a schedule.

To discuss the result of various goals for the sample case study problem, we analyse the result from the perspectives of number of fails and number of choice points. Failure refers to the node which backtracks when the search cannot find the goal. The number of fails refers to the number of backtrack in the search process until a goal is found. Choice point refers to the node that has been explored or visited in the search process. Therefore the number of choice points refers to the number of nodes that has been visited in the search process [31]. Figure 1 shows the number of failures and number of choice points in the search tree.

In Figure 1, the grey circle represents choice point, and black circle represents dead end. The black square represents the goal is found, and the arrow represents the failure. For this four level search tree, there are seven choice points and three failures. A choice point is created by the execution of the goal IlcOr. Backtracking occurs as long as no subgoal succeeds. Thus if no subgoal succeeds, the choice point is considered as fail. We will also compare the result using the total running time that is explored in seconds. However, the time displays here returns the elapsed time, sometimes known as wall clock time.

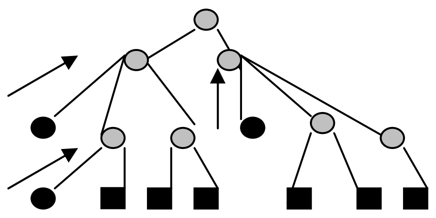
Figure 1. Number of failures and number of choice points

We have used the following various scenarios as a discussion basis in terms of the goals in the Solver. These include: Ranking goals and SetTime goals. For each goal we have used various enforcement levels for each of the scenario.

The first scenario uses the ranking goals approach. As explained, ILOG uses the AC-5 algorithm that is the default search strategy in ILOG to remove inconsistent values from variables domain [30]. When a constraint is ranked first, the activity corresponding to it is positioned at the head of the activities not already ranked. A set of instances of IloResourceConstraint may be ranked (ordered along the time line) for a resource. Ranking is defined for the classes IloUnaryResource and IloStateResource.The resource constraint selector selects the next resource constraint to be ranked first. A resource is chosen and the activities at each iteration, which require the chosen resource, are put in order. For this ordering at each iteration, a resource constraint is chosen [32].

The ranking goals include IloRankForward and IloRankBackward. IloRankForward creates and returns a goal that ranks all resource constraints of unary resource. By default (when no resource constraint selector is given as an argument), the resource constraint selector selects the next resource constraint to be ranked first. The difference with IloRankForward is that the next resource to be ranked backward by using IloRankBackward. The results show that these two ranking goals have similar performance. From the detail timetable schedule produced, the difference is that subjects were not held on the same time in different goals.

In terms of SetTime goals, we use IloSetTimesForward and IloSetTimesBackward. When we use the goal IloSetTimesBackward, the Solver will choose the latest timeslots for the activities (subjects) to build the timetable. On the contrary, the Solver will choose the start timeslots for the activities (subjects) when the goal IloSetTimesForward is selected. When the IloSetTimesForward goal is applied, the Solver puts the lecture at the earliest timeslots. In the sample case study, most of the tutorials are run after the lecture. When the temporal constraints such as tutorial must be scheduled after the lecture it is thus easy to satisfy. However for the `IloSetTimesBackward` goal, it will put the lecture at the latest timeslots. So the lecture needs to move backward to another timeslots when the program tries to satisfy the above constraints of scheduling tutorial after the lecture that will result in more failures than using the `IloSetTimesForward` goal. From the results obtained, we can see that using `IloSetTimesBackward` results in more number of failures and choice points than using `IloSetTimesForward`. The CPU time of these two scenarios only has a subtle difference due to small sample data. Obviously, the running time of using `IloSetTimesBackward` is more than using `IloSetTimesForward`. Due to a higher number of failures in `IloSetTimesBackward`, results obtained using this goal requires more CPU time.

We use `IloEnforcementLevel` to allow how much effort is specified in a given resource constraint. The enforcement level allows specifying with how much effort a given resource constraint may be expressed on a given resource. `IloBasic` is the default enforcement level. There are other enforcement levels that represent degrees of enforcement lower or higher than the `IloBasic`. Each level represents a certain degree of effort spent by the Scheduler to enforce constraints. `IloMediumHigh`, `IloHigh` and `IloExtended` correspond to a scale of enforcement levels higher than the default level `IloBasic`. When the enforcement level of a type of constraint is higher than `IloBasic`, then the Scheduler will spend more effort at enforcing those constraints than it would by default. When higher enforcement levels is applied it causes more propagation of constraints, this results in fewer failures and fewer choice points, but more CPU time consumption in each search state. On the other hand, `IloLow` and `IloMediumLow` represent enforcement levels lower than the default level `IloBasic`. Thus Scheduler will spend less effort at enforcing those constraints than it would by default. The higher enforcement levels typically cause more propagation of constraints; this results in fewer fails and fewer choice points, but more CPU time consumption in each search state. Also, the use of enforcement level should be chosen in accordance with the resource and how it is being used [32].Table 1 shows the summary of the various results for different scenarios.

Table 1. Comparisons of results for different scenarios

Scenarios	Number of fails	Number of choice points	CPU time (seconds)
IloRankForward	373	761	4.91
IloRankBackward	380	766	4.96
IloSetTimesForward	2377	2767	5.86
IloSetTimesBackward	6460	6856	8.72
IloLow+IloRankForward	373	761	5.23
IloLow+IloRankBackward	380	766	5.3
IloLow+IloSetTimesForward	2377	2767	5.87
IloLow+IloSetTimesBackward	6460	6856	8.72
IloHigh+IloRankForward	3	391	3.91
IloHigh+IloRankBackward	10	396	3.93
loHigh+IloSetTimesForward	2007	2397	5.25
IloHigh+IloSetTimesBackforwad	6090	6486	9.33
IloExtended+IloRankForward	3	391	20.3
IloExtended +IloRankBackward	10	396	20.75
IloExtended +IloSetTimesForward	2007	2397	31.34
IloExtended+IloSetTimesBackward	6090	6486	68.25

5. Conclusions and Recommendations

In this paper, we have demonstrated that it is possible to apply the CSP approach to solve a university timetabling problem. The data for sample case study problem were derived from a department in the local university. We have used the CSP model to solve the problem. ILOG Solver and ILOG Scheduler tools are used to solve the CSP problem. Various scenarios in term of using different goals have been conducted and the results obtained are satisfactory. From the test results, we can see that using `IloHigh+IloRankForward` can lead to the better result compare to other goals. This means by enforcing tight constraint level and ranking the constraints by positioning the constraints at the beginning of the activities can lead to better result. In addition we find that using Ranking goals can obtain better result compare to using SetTime goals.

There are two enhancements that can be made to the program. We propose future work to be conducted in the following areas: ILOG Solver has its default search algorithm that is similar to AC-5. As a matter of conclusion, we will highlight that ILOG can make its own algorithm to search the solution. For future comparison, these algorithms can be implemented and compare to the results obtained here. Currently the program just read the data from a data file. Future enhancement can be conducted to design a graphical user interface and a subject database connection. In addition the users can specify whether the preferences are of hard or soft constraints. This way when no optimal schedule is found, we can remove those preferences that are of soft constraints.

6. References

[1] Almond, M., 1966, An algorithm for constructing university timetables, *Computer Journal*, vol.8, pp.331-340.

[2] Brittan, J. N. G. and Farley, F. J. M., 1971, College timetable construction by computer, *The Computer Journal*, vol.14, pp.361–365.

[3] Vit'anyi, P. M. B., 1981, How well can a graph be n coloured, *Discrete mathematics*, vol.34, pp.69-80.

[4] Tripathy, A., 1984, School Timetabling--A Case in Large Binary Integer Linear Programming, *Management Science*, vol.30, pp.1473–1489.

[5] de Werra, D., 1985, An introduction to timetabling, *European Journal of Operations Research*, vol.19, pp.151–162.

[6] Abramson, D., 1991, Constructing schools timetables using simulated annealing: sequential and parallel algorithms, *Management Science*, vol.1, No.37, pp.98-113.

[7] Abramson, D. and Abela, J., 1991, A Parallel Genetic Algorithm for Solving the School Timetabling Problem, *Technical Report*, Division of Information Technology, C.S.I.R.O.

[8] Hertz, A., 1992, Finding a feasible course schedule using tabu search, *Discrete Applied Mathematics*, vol.35, pp.255-270.

[9] Burke, E., Elliman, D. and Weare, R., 1994, A genetic algorithm based university timetabling system, *East-West Int. Conf. Computer Technologies in Education* vol.1, pp.35-40.

[10] Costa, D., 1994, A tabu search algorithm for computing an operational timetable, *European Journal of Operational Research*, vol.76, No.1, pp.98–110.

[11] Jaffar, J. and Maher, M. J., 1994, Constraint logic programming: A survey, *The Journal of Logic Programming*, vol.19~20, pp.503-581.

[12] Gunadhi, H., Anand, V. J. and Yeong, W. Y., 1996, Automated timetabling using an object-oriented scheduler, *Expert systems with Applications*, vol.10, No. 2, pp.243-256.

[13] Guéret, C., Jussien, N., Boizumault, P. and Prins, C., 1996, Building university timetables using constraint logic programming, In: Burke, E. and Ross, P. (Eds.), *Practice and Theory of Automated Timetabling*, Springer-Verlag LNCS 1153, pp. 130–145.

[14] Lajos, G., 1996, Complete university modular timetabling using constraint logic programming, In: Burke, E. and Ross, P. (Eds.), *Practice and Theory of Automated Timetabling*, Springer-Verlag LNCS 1153, pp. 146–161.

[15] Deris, S. B., Omatu, S., Ohta, H. and Samat, P. A. B. D., 1997, University timetabling by constraint-base reasoning: A case study, *Journal of the Operational research Society*, vol.48, pp.1178-1190.

[16] Terashima-Marín, H., 1998, Combinations of GAs and CSP Strategies for Solving the Examination Timetabling Problem, *PHD thesis.*

[17] Brailsford, S. C., Potts, C. N. and Smith, B. M., 1999, Constraint satisfaction problems: algorithms and applications, *European Journal of Operational Research*, vol.119, pp.557-581.

[18] Schaerf, A., 1999a, A survey of automated timetabling, *Artificial Intelligence Review*. vol.13, pp.87-127.

[19] Schaerf, A., 1999b, Local search techniques for large high school timetabling problems, *IEEE Transactions on Systems Man and Cybernetics Part A – Systems and Humans*, vol.29, pp.368-377.

[20] Abdennadher, S. and Marte, M. 2000, University course timetabling using Constraint Handling Rules, *Journal of Applied Artificial Intelligence*, vol.14, No.4, pp.311–326.

[21] Duncan, A. K., 1965, Letters to the editor: Further results on a computer construction of school timetables, *Communications of the ACM*, vol.8, No.1, pp.72.

[22] Even, S., Itai, A., and Shamir, A., 1976, On the complexity of timetable and multicommodity flow problems, *SIAM J. Computing*, vol.5, No.4, pp.691–703.

[23] Wren, A., 1996, Scheduling, Timetabling and Rostering–A special Relationship, In: Burke, E. and Ross, P. (Eds.), *Practice and Theory of Automated Timetabling*, Springer-Verlag LNCS 1153, pp.46-75.

[24] Nadel, B. A., 1989, Constraint Satisfaction Algorithms, *Computational Intelligence*, vol.5, pp.188-224.

[25] Dechter, R., 2003, *Constraint Processing*, Morgan Kaufmann.

[26] Kumar, V., 1992, Algorithms for constraint Satisfaction Problems: A Survey, *AI magazine*, vol.13, No.1, pp.32-44.

[27] Barták, R., 1998, *On-line Guide to Constraint Programming*, <http://kti.ms.mff.cuni.cz/~bartak/constraints/> (Access: 25 March, 2004).

[28] Sabin, D. and Freuder, E., 1994. Contradicting conventional wisdom in constraint satisfaction, In: Cohn, A.G. (Eds.), *Proceedings of European Conference on Artificial Intelligence (ECAI-94)*. Wiley, Chichester, UK, pp. 125-129.

[29] IlOG, 2004, < http://www.ilog.com>, (Access: 25 July, 2004).

[30] Van Hentenryck, P., Deville, Y., Teng, C. M., 1992, A generic arc-consistency algorithm and its specializations, *Artificial Intelligence*, vol.57, pp.291-321.

[31] ILOG, Inc. 2001a, *ILOG Solver 5.1 User's Manual*, ILOG Inc.

[32] ILOG, Inc. 2001b, ILOG Scheduler 5.1 User's Manual, ILOG Inc.