

THREE MECHANISMS FOR MANAGING RESOURCE CONSTRAINTS IN A LIBRARY FOR CONSTRAINT-BASED SCHEDULING*

Claude Le Pape

ILOG S.A., 2 Avenue Galliéni, BP 85, F-94253 Gentilly Cedex

E-mail: lepape@ilog.fr

Url: <http://www.ilog.fr>

Abstract: ILOG SCHEDULE is a C++ library aimed at simplifying the development of industrial scheduling applications. SCHEDULE is based on SOLVER, a generic tool for object-oriented constraint programming. SCHEDULE includes three categories of predefined constraints: *temporal constraints*, *capacity constraints*, and *resource utilization constraints*. Three distinct mechanisms are available to deal with resource utilization constraints. The first mechanism relies on explicit *time-tables* to maintain information about the variations of resource utilization and resource availability over time. The second mechanism relies on a generic *disjunctive constraint* which ensures that incompatible activities (for example, activities which require a common resource of capacity 1) cannot overlap in time. The third mechanism, known as *edge-finding*, considers arbitrary tuples $\{A_1 \dots A_n\}$ of activities to prove that some activity A_i must execute first, or must execute last, among $\{A_1 \dots A_n\}$. The edge-finding mechanism is a priori more CPU-time consuming than the two other mechanisms, but results in the assignment of more precise earliest and latest start and end times to activities. Each of the three mechanisms is useful, as the time-table and disjunctive mechanisms enable the expression of wider classes of constraints, while edge-finding is in general the most effective in pruning the search space.

1. Introduction

Scheduling is the process of assigning activities to resources in time. Basically, the three main things to consider when building a scheduling system are:

- *The complexity of the scheduling problem.* Most scheduling problems are known to be NP-hard. NP-hard problems are problems for which it is conjectured that there exists no algorithm enabling to optimally solve the problem in an amount of time bounded by a polynomial function of the size of the data. In practice, this means that one must design robust approximate algorithms, to generate appropriate (possibly optimal but often sub-optimal) solutions in a bounded amount of time.
- *The specificity of the problems to address.* Different manufacturing environments induce different scheduling constraints, some of which may be very specific to the problem under consideration.
- *The integration with the overall manufacturing system.* A scheduling system must get its data from the information system globally in use in the factory, and must return its results (i.e., the constructed schedule) for factory-floor execution.

Scheduling problems are very different one from the other:

- First, three broad families of scheduling problems can be distinguished depending on the degrees of freedom in positioning resource supply and resource demand

* In: Proceedings of the INRIA/IEEE Conference on Emerging Technologies and Factory Automation, Paris, France, 1995.

intervals in time. In pure *scheduling* problems (e.g., job-shop machine scheduling), the capacity of each resource is defined over a number of time intervals and the problem consists of positioning resource-demanding activities over time, without ever exceeding the available capacity. In pure *resource allocation* problems (e.g., allocation of personnel to planes or trains), the demand for each resource is known beforehand and the problem consists of allocating resources in time to guarantee that the supply always equals or exceeds the demand. In *joint scheduling and resource allocation* problems, degrees of freedom exist for deciding both which activities to perform and when, and which resources to make available for these activities.

- Different environments are subjected to different constraints which more or less contribute to the complexity of the problem. For example, a factory scheduling problem may involve only machines as resources, while another may, in addition, require the consideration of the abilities of human operators.
- The size of a scheduling problem may vary from a few dozens activities to thousands of activities. For complexity reasons, the scheduling algorithms that work well for the small problems may not be applicable to the bigger problems.
- Depending on the environment, the suitable response time for the construction of a schedule may vary from a few microseconds to a few days. Also, it may be necessary to incrementally modify the schedule, either as a response to environmental changes, or because it is more appropriate for a human to “make the decisions.”

In response to the important variety and variability of scheduling problems, ILOG initiated the development of ILOG SCHEDULE, a C++ library enabling the representation of a wide collection of scheduling constraints in terms of *resources* and *activities* [Le Pape 94]. SCHEDULE is itself based on SOLVER, the generic software tool for object-oriented constraint programming developed and marketed by ILOG [Puget 94]. This enables users of SCHEDULE to benefit from the functionalities of SOLVER to develop specific types of constraints and implement specific problem-solving procedures, in response to the requirements of each particular application.

2. Constraint propagation

The interest of constraint programming lies in using constraints to reduce the computational effort needed to solve combinatorial problems. Constraints are used not only to test the validity of a solution, as in conventional programming languages, but also in a constructive mode to deduce new constraints and rapidly detect inconsistencies. For example, from “ $X < Y$ ” and “ $X > 8$ ”, we deduce, if X and Y are integer variables, that the value of Y is at least 10. If later we add the constraint “ $Y \leq 9$ ”, a contradiction is immediately detected. Without propagation, the contradiction would not be detected before the instantiation of X and Y .

For complexity reasons, constraint propagation is usually *incomplete*. This means that some but not all the consequences of posted constraints are deduced. In particular, constraint propagation cannot detect all inconsistencies. Consequently, heuristic search algorithms must be implemented to explore possible refinements of the constraints (e.g., order any two activities that require the same unary resource) and exhibit solutions that are guaranteed to satisfy the constraints.

Backtracking is a mechanism used to implement such algorithms: the algorithm is said to backtrack when it returns to a previous problem-solving state. SOLVER includes a set of non-deterministic control primitives that allow its users to implement backtracking search algorithms. These primitives rely on the concept of a non-deterministic goal which can be decomposed into a conjunction or a disjunction of subgoals.

For example, the following program states that to satisfy goal `f`, one must satisfy either the conjunction of goals `f1` and `f2` or the conjunction of goals `f3` and `f4`.

```
CTGOAL0(f) {  
    CtOr(CtAnd(f1(), f2()),  
         CtAnd(f3(), f4()));  
}
```

Goals may receive parameters as arguments. The syntax for calling a goal is identical to the syntax used to call regular C++ functions.

SCHEDULE is an add-on to SOLVER, used to develop finite capacity scheduling applications. SCHEDULE implements a natural object model for representing schedules in terms of *activities* that share and use *resources*. The model consists of a set of C++ classes, methods, and functions that implement the concepts of “resource” and “activity” in terms of SOLVER variables and constraints. The algorithm building and search programming primitives of SOLVER can then be used for solving the represented problem.

SCHEDULE includes three categories of predefined constraints: temporal constraints, capacity constraints, and resource utilization constraints.

- *Temporal constraints.* Users may link any two activities together by any type of precedence constraint (A starts after START(B), A starts after END(B), A ends after START(B), A ends after END(B)). In addition, minimum and maximum delays between activities can be imposed. When only temporal constraints are considered, constraint propagation is *complete*. This means that constraint propagation suffices (a) to determine whether a set of temporal constraints is consistent and (b) to compute the earliest and latest start and end times of activities [Le Pape 88].
- *Capacity constraints.* SCHEDULE offers many different ways to express that a resource is available in finite amounts over time: *unary resources* to represent resources of capacity one (like a specific person); *volumetric resources* to represent resource pools of many, non-differentiated resources (like a group of people with the same capabilities); *state resources* to represent situations where an activity uses a resource only under specific conditions (like waiting for an appropriate oven temperature). In addition, the capacity of a resource can be constrained either at each time unit (number of people available each day) or over given time periods (number of people-days available over one week).
- *Resource utilization constraints.* An activity may require, consume, provide and produce resources, in an amount represented either as a constant or as a constrained variable. This allows the representation of the case where the duration of the activity varies with the amount of resources assigned to the activity. The propagation of resource utilization constraints results in adjusting the earliest and latest start and end times (time bounds) of activities.

3. Propagation of resource utilization constraints

The problem of determining whether a set of resource utilization constraints is consistent is NP-hard. The propagation of resource utilization constraints is consequently incomplete. This means that, except in a few particular cases, it is necessary to explore the search space to generate a solution to the scheduling problem under consideration. Constraint propagation is useful in this process as it allows the pruning of many impossible decisions.

Three distinct mechanisms are available to propagate resource utilization constraints in the current version of SCHEDULE (version 1.1).

The first mechanism relies on explicit *time-tables* to maintain information about the variations of resource utilization and resource availability over time. Resource constraints are propagated in two directions: from the resources to the activities, in order to update activity time bounds according to the availability of the resources; and from the activities to the resources, in order to update resource utilization and availability according to the time bounds of activities. For example, when the latest start time LST of an activity is smaller than its earliest end time EET, it is sure that the activity will use the resource between LST and EET. Over this period, the corresponding resource amount is no longer available for other activities. The main advantage of the time-table mechanism for propagating resource constraints is that it applies to any type of resource [Le Pape 94]. Users of SCHEDULE can rely on time-tables to represent unary, volumetric, and state resources. Time-tables also enable to specify that at least some amount of resource capacity must be used over a given period or that the capacities of two or several resources must satisfy a specific constraint at any point in time. For example, [Le Pape 94] shows how to represent a constraint stating that a polyvalent operator, able to operate two types of machines, e.g., lathes and milling machines, must remain assigned to the same type of machine during each shift. The propagation of the constraint guarantees that, if at a particular time during the shift N polyvalent operators must be assigned to lathes, then during the shift they cannot be assigned to a milling machine, and vice versa.

The second mechanism can be applied to unary resources and state resources. It consists of a generic “disjunctive” constraint which ensures that the time intervals over which two activities require a unary resource (or require different states of a state resource) cannot overlap in time. For instance, if a resource is required (or provided) by two activities A and B throughout two time intervals $[START(A) \text{ END}(A)]$ and $[START(B) \text{ END}(B)]$, the disjunctive constraint states that either $END(A) \leq START(B)$, or $END(B) \leq START(A)$. Such a disjunctive constraint representation has been used in the scheduling domain for years [Erschler 76] [Carlier 84] [Esquirol 87] [Le Pape 88] [Varnier et al 93] [Smith and Cheng 93] [Laborie 94]. The propagation of the disjunctive constraint consists in reducing the set of possible values for the start and end times of the activities under consideration: whenever the smallest possible value of $END(A)$ exceeds the greatest possible value of $START(B)$, A cannot precede B; hence B must precede A; the time bounds of A and B can consequently be updated with respect to the new temporal constraint “ $END(B) \leq START(A)$ ”. Similarly, when the earliest possible end time of B exceeds the latest possible start time of A, B cannot precede A. When neither of the two activities can precede the other, a contradiction is detected. This way of propagating constraints enforces arc-B-consistency as defined by [Lhomme 93].

Extensions of the disjunctive constraint $[END(A) \leq START(B) \text{ OR } END(B) \leq START(A)]$ are discussed in [Baptiste and Le Pape 95]. This includes (1) activities that may or may not require the resource, (2) disjunctive constraints applied to state resources, and (3) setup times between activities that require the same resource. These extensions are all provided in ILOG SCHEDULE.

The third mechanism considers arbitrary tuples $\{A_1 \dots A_n\}$ of activities to prove that some activity A_i must execute first (or must execute last) among $\{A_1 \dots A_n\}$. In SCHEDULE 1.1, this mechanism is applied to unary resources only. Extensions to volumetric resources are considered for the next version (based on [Nuijten and Aarts 94] [Nuijten 94] [Nuijten and Aarts 95]). The algorithm implemented in SCHEDULE 1.1 is described in [Nuijten et al 93]. This algorithm implements the “edge-finding” (immediate selection of disjunctive constraints) technique of Carlier and Pinson [Pinson 88] [Carlier and Pinson 89] [Carlier and Pinson 90] [Carlier and Pinson 94] with $O(N^2)$ time and $O(N)$ space complexity, where N denotes the number of activities requiring the unary resource under consideration. This method is a priori more time consuming than the two others, but results in the assignment of more precise time bounds to the activities.

Each of the three mechanisms is useful. Indeed, as explained above, the time-table and disjunctive mechanisms enable the expression of many types of constraints, while the edge-finding mechanism is limited to the ordering of a set of activities requiring the same unary resource. On the other hand, edge-finding is in general more effective in pruning the search space.

For example, the following table provides the number of backtracks and the CPU time needed to optimally solve a bridge construction scheduling problem described in [Van Hentenryck 89]. The optimization algorithm consists of a branch-and-bound backtracking search, with constraint propagation being performed at each node of the search tree. Columns “BT” and “CPU” provide the total number of backtracks and CPU time needed to find an optimal solution and prove its optimality. Columns “BT(pr)” and “CPU(pr)” provide the number of backtracks and CPU time needed for the proof of optimality. Column “CPU(one)” provides the time needed to find a first problem solution, including the time needed for stating the problem and performing initial constraint propagation. Column “TM” provides the total amount of memory used to represent and solve the problem (in kilobytes). CPU times are expressed in seconds on a HP715/50 workstation, rounded to the closest tenth of a second. Details about the problem and the optimization algorithm are available in [Baptiste 94].

Propagation method	CPU(one)	BT	CPU	BT(pr)	CPU(pr)	TM
Time-table	.1	563	1.3	209	.4	96
Disjunctive constraint	.1	176	.3	149	.1	76
Edge-finding	.1	14	.5	12	.1	72

On this experiment, we see that the number of backtracks significantly decreases when more propagation is being performed. Also, we see that on such a problem, the algorithm based on arc-consistency performs better than the others in terms of CPU time: the cost of additional constraint propagation is not balanced by the reduction of the search effort.

Such is not the case for other problems. For example, the use of the edge-finding mechanism enables the resolution of “hard” job-shop scheduling problem instances such as those used by Applegate and Cook in their computational study of the job-shop scheduling problem (cf. [Applegate and Cook 91]). The following table shows the results obtained on ten “hard” instances of the job-shop scheduling problem. CPU times are expressed in seconds on a RS6000 workstation. The algorithm used to solve the problem is described in [Bapstiste et al 95]. The total number of backtracks over the ten instances is 215256 for the SCHEDULE algorithm, while the total number of nodes explored by Applegate and Cook's algorithm is 674128.

We were unable to solve these problem instances to optimality using the time-table or the disjunctive constraint mechanism without edge-finding [Baptiste 94].

Problem instance	CPU(one)	BT	CPU	BT(pr)	CPU(pr)	TM
MT10	.2	13684	184.8	4735	52.8	227
ABZ5	.2	19303	226.6	4519	49.7	216
ABZ6	.2	6227	80.3	312	3.8	198
LA19	.3	18102	219.2	6561	74.7	235
LA20	.2	40597	407.3	20626	186.9	220
ORB1	.2	22725	323.7	6261	85.3	243
ORB2	.2	31490	416.4	14123	189.3	210
ORB3	.3	36729	488.4	22138	277.6	262
ORB4	.2	13751	169.9	1916	19.0	210
ORB5	.2	12648	168.5	2658	29.7	232

4. Summary and conclusion

In this paper, we have described three generic mechanisms, which are used in ILOG SCHEDULE to propagate resource utilization constraints. These mechanisms are complementary. Each of the *time-table* and the *disjunctive constraint* mechanism enables the propagation of constraints which cannot be expressed in the context of the two other mechanisms. The *edge-finding* mechanism is, in the current version, limited to the ordering of a set of activities which require the same unary resource. It is, on the other hand, more effective in pruning the search space. Its integration in SCHEDULE allows users to enjoy the flexibility inherent to constraint programming, with performance in the same range of efficiency as specific operations research algorithms, such as the one reported in [Applegate and Cook 91].

SCHEDULE is based on SOLVER, an extensible tool for constraint programming. This is quite important from an industrial point of view: users can extend the scheduling model and define solution search strategies with respect to the requirements of a specific application. The fact that SOLVER and SCHEDULE are C++ libraries (and not new programming languages) is also very important. The use of standard programming languages like C and C++ simplifies the integration of constraint-based scheduling algorithms with other parts of a scheduling application, such as a Gantt chart graphical editor, a database management system, or a rule-based execution monitoring system. The use of other C++ tools provided by ILOG, such as ILOG VIEWS for the implementation of graphical interfaces, ILOG DBLINK for communication with relational databases, ILOG RULES [Albert and Fages 92] for the development of an execution monitoring system, and ILOG SERVER and BROKER [Ceugniet et al 93] for the development of distributed scheduling applications, may even facilitate the development of these “other parts.” However, it is not compulsory; any other software tool can be used provided that it communicates easily with C++ or C. In this respect also, users of SCHEDULE can *de facto* perform the extensions needed for their application.

References

- [Albert and Fages 92]
Patrick Albert and François Fages.
XRETE : un outil pour les systèmes experts temps-réel.
Génie logiciel et systèmes experts, (28):22-34, 1992 (in French).
- [Applegate and Cook 91]
David Applegate and William Cook.
A Computational Study of the Job-Shop Scheduling Problem.
ORSA Journal on Computing, 3(2):149-156, 1991.
- [Baptiste 94]
Philippe Baptiste.
Constraint-Based Scheduling: Two Extensions.
MSc Thesis, University of Strathclyde, 1994.
- [Baptiste and Le Pape 95]
Philippe Baptiste and Claude Le Pape.
Disjunctive Constraints for Manufacturing Scheduling: Principles and Extensions.
3rd International Conference on Computer Integrated Manufacturing, Singapore, 1995.

[Baptiste et al 95]
Philippe Baptiste, Claude Le Pape and Wim Nuijten.
Constraint-Based Optimization and Approximation for Job-Shop Scheduling.
AAAI-SIGMAN Workshop on Intelligent Manufacturing Systems, IJCAI, Montréal,
Québec, 1995.

[Carlier 84]
Jacques Carlier.
Problèmes d'ordonnancement à contraintes de ressources : algorithmes et complexité.
Thèse de Doctorat d'Etat, Université Paris VI, 1984 (in French).

[Carlier and Pinson 89]
Jacques Carlier and Eric Pinson.
An Algorithm for Solving the Job-Shop Problem.
Management Science, **35**(2):164-176, 1989.

[Carlier and Pinson 90]
Jacques Carlier and Eric Pinson.
A Practical Use of Jackson's Preemptive Schedule for Solving the Job-Shop Problem.
Annals of Operations Research, **26**:269-287, 1990.

[Carlier and Pinson 94]
Jacques Carlier and Eric Pinson.
Adjustment of Heads and Tails for the Job-Shop Problem.
European Journal of Operational Research, **78**:146-161, 1994.

[Ceugniet et al 93]
Xavier Ceugniet, Claude Fornarino and Vincent Lextrait.
Gestion de la cohérence dans les systèmes orientés objets distribués.
6^{èmes} journées internationales sur le génie logiciel et ses applications, Paris La Défense,
1993 (in French).

[Erschler 76]
Jacques Erschler.
Analyse sous contraintes et aide à la décision pour certains problèmes d'ordonnancement.
Thèse de Doctorat d'Etat, Université Paul Sabatier, 1976 (in French).

[Esquirol 87]
Patrick Esquirol.
Règles et processus d'inférence pour l'aide à l'ordonnancement de tâches en présence de
contraintes.
PhD Thesis, University Paul Sabatier, 1987 (in French).

[Laborie 94]
Philippe Laborie.
Planifier avec des contraintes de ressources.
2^{èmes} rencontres des jeunes chercheurs en intelligence artificielle, Marseille, France, 1994
(in French).

[Le Pape 88]
Claude Le Pape.
Des systèmes d'ordonnancement flexibles et opportunistes.
PhD Thesis, University Paris XI, 1988 (in French).

- [Le Pape 94]
Claude Le Pape.
Implementation of Resource Constraints in ILOG SCHEDULE: A Library for the Development of Constraint-Based Scheduling Systems.
Intelligent Systems Engineering, 3(2):55-66, 1994.
- [Lhomme 93]
Olivier Lhomme.
Consistency Techniques for Numeric CSPs.
13th International Joint Conference on Artificial Intelligence, Chambéry, France, 1993.
- [Nuijten et al 93]
W. P. M. Nuijten, E. H. L. Aarts, D. A. A. van Erp Taalman Kip and K. M. van Hee.
Job-Shop Scheduling by Constraint Satisfaction.
Computing Science Note, Eindhoven University of Technology, 1993.
- [Nuijten and Aarts 94]
W. P. M. Nuijten and E. H. L. Aarts.
Constraint Satisfaction for Multiple Capacitated Job-Shop Scheduling.
11th European Conference on Artificial Intelligence, Amsterdam, The Netherlands, 1994.
- [Nuijten 94]
W. P. M. Nuijten.
Time and Resource Constrained Scheduling: A Constraint Satisfaction Approach.
PhD Thesis, Eindhoven University of Technology, 1994.
- [Nuijten and Aarts 95]
W. P. M. Nuijten and E. H. L. Aarts.
A Computational Study of Constraint Satisfaction for Multiple Capacitated Job-Shop Scheduling.
European Journal of Operational Research (to appear).
- [Pinson 88]
Eric Pinson.
Le problème de job-shop.
PhD Thesis, University Paris VI, 1988 (in French).
- [Puget 94]
Jean-François Puget.
A C++ Implementation of CLP.
Technical Report, ILOG S.A., 1994.
- [Smith and Cheng 93]
Stephen F. Smith and Cheng-Chung Cheng.
Slack-Based Heuristics for Constraint Satisfaction Scheduling.
11th National Conference on Artificial Intelligence, Washington, District of Columbia, 1993.
- [Van Hentenryck 89]
Pascal Van Hentenryck.
Constraint Satisfaction in Logic Programming.
MIT Press, 1989.

[Varnier et al 93]

Christophe Varnier, Pierre Baptiste and Bruno Legeard.

Le traitement des contraintes disjonctives dans un problème d'ordonnancement : exemple du Hoist Scheduling Problem.

2^{èmes} journées francophones de programmation logique, Nîmes et Avignon, France, 1993 (in French).