Incorporating Efficient Operations Research Algorithms in Constraint-Based Scheduling[*]

Philippe Baptiste and Claude Le Pape and Wim Nuijten

ILOG S.A., 2 Avenue Gallieni, BP 85, F-94253 Gentilly Cedex, France
Email: {baptiste, lepape, nuijten}@ilog.fr
Url: http://www.ilog.fr

Abstract

We address the area of scheduling and the differences between the way operations research and artificial intelligence approach scheduling. We introduce the concept of constraint programming, and describe how operations research techniques can be integrated in constraint programming. Finally, we give a short overview of the results obtained with our approach.

1 Introduction

Baker [1974] defines scheduling as the problem of allocating scarce resources to activities over time. Scheduling problems arise in areas as diverse as production planning, personnel planning, computer design, and time tabling. Over the years, the theory and application of scheduling has grown into an important field of research, and an extensive body of literature exists on the subject. For more elaborate introductions to the theory of scheduling, we refer to [Baker, 1974], [Coffman, 1976] and [French, 1982].

Roughly speaking, we can distinguish two fields of research that pay attention to scheduling, viz., *operations research* (OR) and *artificial intelligence* (AI). Traditionally, a lot of the attention in OR has been paid to scheduling problems that are based on relatively simple mathematical models. For solving the problem at hand, the combinatorial structure of the problem is

[*]To appear in: Proceedings of the First International Joint Workshop on Artificial Intelligence and Operations Research, Timberline Lodge, Oregon, 1995.

heavily exploited, leading to improved performance characteristics. We could say that an OR approach often aims at achieving a high level of *efficiency* in its algorithms. However, when modeling a practical scheduling problem using these classical models, one is often forced to discard many degrees of freedom and side constraints that exist in the practical scheduling situation. Discarding degrees of freedom may result in the elimination of interesting solutions, regardless of the solution method used. Discarding side constraints gives a simplified problem and solving this simplified problem may result in impractical solutions for the original problem.

In contrast, AI research tends to investigate more general scheduling models and tries to solve the problems by using general problem solving paradigms. We could say an AI approach tends to focus more on the *generality of application* of its algorithms. This, however, implies that AI algorithms may perform poorly on specific cases, compared to OR algorithms.

So, on the one hand we have OR which offers us *efficient* algorithms to solve problems that however might not be well suited to be used in practice, and on the other hand we have AI that offers us algorithms that are more *generally applicable*, but that might suffer from somewhat poor performance. Naturally, we want the best of both worlds, i.e., we want efficient algorithms that we can apply to a wide range of problems. In this article, we introduce *constraint programming* which provides a general modeling and problem solving paradigm, and we show how to integrate it with OR algorithms, in order to improve performance of the approach.

2 Constraint Programming

Generally speaking, constraint programming is concerned with solving instances of the *Constraint Satisfaction Problem* (CSP). An instance of the CSP consists of a set of *variables*, a *domain* for each variable specifying the values to which the variable may be assigned, and a set of *constraints* on the variables. A constraint denotes a relation between the values of one or several variables. For instance, if x is an integer variable, $x < 10$ is a constraint on the variable x. Solving an instance of the CSP consists in assigning values to variables such that all constraints are satisfied simultaneously.

In constraint programming, constraints are exploited to reduce the amount of computation needed to solve the problem. More specifically, they are used to reduce the domains of the variables and to detect inconsistencies. This deductive process is called *constraint propagation.*

For example, from $x < y$ and $x > 8$, we deduce, if x and y denote integers, 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, no contradiction would be detected until the instantiation of both x and y.

For complexity reasons, constraint propagation is usually *incomplete.* This means that some but not all the consequences of constraints are deduced. In particular, constraint propagation cannot detect all inconsistencies. Consequently, heuristic search algorithms must be implemented to explore possible refinements of the instance of the CSP, e.g., by assigning a value to each variable, and exhibit solutions that are guaranteed to satisfy the constraints.

In the remainder of this article, Sections 3 and 4 discuss the propagation of two types of constraints that occur in scheduling problems, viz., *temporal constraints* and *resource constraints.* Furthermore, Section 5 presents an overview of the results we obtained with our approach. Finally, Section 6 presents some conclusions.

3 Temporal Constraints

A temporal constraint is logically expressed by a formula $I_i + d \leq I_j$. I_i and I_j are time points and d a delay that must elapse between these time points. Generally, a time point corresponds to the beginning or the end of an activity. If A and B are two activities, and if a time unit corresponds to a minute, the temporal constraint $end(A) + 5 \leq start(B)$ means that at least 5 minutes must pass between the end of A and the beginning of B. A special time point O is often defined to represent the origin of time: $O + 60 \leq start(A)$ means that A cannot start before date 60; $end(A) - 120 \leq O$ means that A cannot end after date 120.

Ford's algorithm [Ford, 1956; Gondran & Minoux, 1984], which computes the length of the longest paths from a node N_0 to the other nodes N_1 ... N_n of a valued oriented graph, can be used to compute the earliest start and end times of activities. Informally, this algorithm can be described as follows.

1. Let $\pi(N_0) = 0$ and $\pi(N_i) = -\infty$ for every i between 1 and n.
2. For every j between 1 and n, replace $\pi(N_j)$ with the maximum of $\pi(N_j)$ and $\max(\pi(N_i) + l_{ij})$ where l_{ij} denotes the length of the arc going from N_i to N_j when this arc exists.

3. Iterate step 2 until $\pi(N_i)$ becomes stable for every i.

This algorithm can also be used to compute the latest start and end times of activities by exploring the graph in the reverse direction. When the temporal constraints are globally compatible, it updates the time-bounds of activities (earliest and latest start and end times) in $O(mn)$ where m is the number of constraints and n the number of activities. Note that one can easily implement an incremental version of the algorithm. Indeed, at each iteration, it is useless to re-compute $(\pi(N_i)+l_{ij})$ if the preceding iteration did not result in an update of $\pi(N_i)$. Consequently, the algorithm can be modified to make the update of $\pi(N_i)$ trigger, at the next iteration, the update of $\pi(N_j)$.

In a constraint programming framework, the constraint $I_i + d \leq I_j$ can be propagated in a very simple way. Each time the minimal value of I_i changes (becomes I_i^{min}), the minimal value of I_j is set to the maximum of $I_i^{min}+d$ and the current minimal value of I_j. In the reverse direction, each time the maximal value of I_j changes (becomes I_j^{max}), the maximal value of I_i is set to the minimum of $I_j^{max} - d$ and the current maximal value of I_i. The main advantage of the constraint programming implementation is that each constraint is considered separately. If other constraints exist, the algorithm that performs the propagation propagates also the other constraints. Other constraints can consequently be designed independently of the temporal constraints.

An issue that arises is the efficiency of a constraint programming implementation, compared to Ford's algorithm. It is shown in [Le Pape, 1988] that the constraint propagation steps can be ordered to obtain the same complexity as Ford's algorithm when the temporal constraints are globally compatible. In particular, the constraint propagation method which consists in using a first-in first-out queue of constraints updates the time-bounds of activities in $O(mn)$. This method is used in ILOG SCHEDULE [Le Pape, 1994], an add-on to ILOG SOLVER, a C++ library for constraint programming [Puget, 1994].

4 Resource Constraints

In scheduling problems, activities require resources which are available in finite amounts over time. Three main classes of resources can be distinguished:

- *Unary resources*: a unary resource is a resource of capacity one (e.g. a specific person). Two activities that require the same unary resource cannot overlap.

- *Volumetric resources*: a volumetric resource typically represents a pool of many non-differentiated resources (e.g. a group of people with the same capabilities). At any point in time, the number of units required by the executing activities cannot exceed the number of units that are available.
- *State resources*: a state resource is a resource that can be used for an activity only when it is in a given state. Two activities that require the same resource in different states cannot overlap.

There does not appear to exist algorithms as general as Ford's algorithm to optimally update the time-bounds of activities submitted to resource constraints. As a matter of fact, the problem of eliminating all the impossible start and end times of activities submitted to resource constraints is *NP-hard* [Garey & Johnson, 1979]. Even when only one unary resource is considered, the problem of determining whether there exists a schedule satisfying given time-bounds for each activity is NP-hard [Garey & Johnson, 1979]. As a result, many OR algorithms have been developed to compute time-bounds for specific problems. Each of these algorithms corresponds to a different tradeoff between the generality of the algorithm, i.e., the class of problems to which the algorithm applies, the precision of the computed time-bounds, and the computational complexity of the algorithm.

One of the most successful OR algorithms for updating time-bounds of activities submitted to unary resource constraints was proposed by Carlier & Pinson [1990]. The main principle of this algorithm is to compare the temporal characteristics of an activity A to those of a set of activities Ω which require the same resource. Let est_A denote the earliest possible start time of A, let_A the latest possible end time of A, and p_A the processing time of A. Let est_Ω denote the smallest of the earliest start times of the activities in Ω, let_Ω denote the greatest of the latest end times of the activities in Ω, and p_Ω denote the sum of the processing times of the activities in Ω. The following rules apply:

$$\begin{bmatrix} let_\Omega - est_\Omega < p_A + p_\Omega \\ let_A - est_\Omega < p_A + p_\Omega \end{bmatrix} \Rightarrow [A \textit{ is before all activities in } \Omega]$$

$$\begin{bmatrix} let_\Omega - est_\Omega < p_A + p_\Omega \\ let_\Omega - est_A < p_A + p_\Omega \end{bmatrix} \Rightarrow [A \textit{ is after all activities in } \Omega]$$

New time-bounds can consequently be deduced. When A is before all activities in Ω, the end time of A is necessarily at most $let_\Omega - p_\Omega$. When A is after all activities in Ω, the start time of A is necessarily at least $est_\Omega + p_\Omega$.

The technique which consists in applying these rules is known as *edge-finding*. Notice that if n activities require the resource, there are potentially $O(n * 2^n)$ pairs (A, Ω) to consider. An algorithm that performs all of the possible time-bound adjustments in $O(n^2)$ is presented in [Carlier & Pinson, 1990]. Impressive results have been obtained by applying variants of this algorithm as part of a tree search procedure to the Job Shop Scheduling Problem (JSSP) as defined in [Garey & Johnson, 1979]. Examples of such approaches are [Carlier & Pinson, 1990], [Applegate & Cook, 1991], and [Carlier & Pinson, 1994]. However, as in the case of temporal constraints, it is unclear what ought to be done when a problem requires the satisfaction of other types of constraints, in addition to temporal and unary resource constraints. Clearly, Carlier & Pinson's algorithm could still be useful, but a specific implementation may not be easy to integrate with the other components needed to solve the problem.

This contrasts with the work done in the field of AI. In this field, various classes of resource constraints are naturally integrated in a global framework that incorporates other types of constraints as well. Two main mechanisms are used to propagate resource constraints.

- The first mechanism uses *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; 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. In ILOG SCHEDULE [Le Pape, 1994], time-tables can be used to represent unary, volumetric, and state resources. They also enable to state that at least some amount of resource capacity must be used over a given period.
- The second mechanism consists of posting disjunctive constraints. The most basic disjunctive constraint states that two activities A and B that require the same unary resource R cannot overlap in time: either

A precedes B or B precedes A. This can be written as

$$end(A) \leq start(B) \ \vee \ end(B) \leq start(A).$$

Constraint propagation consists in reducing the set of possible values for the start and end variables: 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 [1993]. Several extensions of the basic disjunctive constraint are discussed in [Baptiste & Le Pape, 1995]. 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.

A few issues are raised by the integration of algorithms such as the edge-finder of Carlier & Pinson in a generic constraint propagation framework.

- First, it is necessary to identify the events that should trigger the execution of the algorithm. When activities are certain to use the resource and their durations are fixed, only the modification of the earliest start and latest end times of the activity can lead the edge-finder to deduce more precise time-bounds. When the processing time of an activity is itself a constrained variable, the situation is different: to deduce only correct information, the edge-finder must rely on the smallest possible duration of the activity; when the smallest possible duration increases, the edge-finder must be called again to ensure that all the possible adjustments are done.
- Second, efficiency issues have to be considered. Indeed, when both temporal constraints and resource constraints apply, it is intuitively more efficient to deduce all the consequences of temporal constraints prior to apply the edge-finder. The main reason for that is that the propagation of each temporal constraint has a very low cost in comparison to an application of the edge-finder. The edge-finder is consequently delayed until the other constraints have been propagated.

- Implementation issues also have to be considered. In the context of an incremental constraint satisfaction framework, it must be possible to add a new activity at any stage of the problem-solving process. This modifies the data structures that can be used as a basis for the implementation of the edge-finder.

A variant of the edge-finder of Carlier & Pinson [1990] is presented in [Nuijten, 1994]. This algorithm has the same complexity but has a much simpler structure. It, furthermore, can easily be generalized to be used for instance for volumetric resources, as is shown in [Nuijten, 1994]. There, the algorithm is already integrated in a constraint-based approach.

Both the [Carlier & Pinson, 1990] and the [Nuijten, 1994] versions were tested on unary resources in the context of ILOG SCHEDULE. We remark that Carlier & Pinson [1994] presents a variant running in $O(n * log(n))$ that we have not tested yet. The experimental results presented in [Baptiste, 1994] show that the two $O(n^2)$ versions are roughly equivalent in terms of CPU time. They also show edge-finding to be extremely powerful in comparison to arc-B-consistency. As the [Nuijten, 1994] version was simpler, it was retained and implemented in version 1.1 of ILOG SCHEDULE. This allows the users of ILOG SCHEDULE to enjoy the efficiency of the edge-finder in the flexible context of constraint programming. In particular, the use of the edge-finder does not prevent the user from defining activities of initially unknown duration, activities that may or may not require the resource, setup times between activities, or any other user-defined constraint.

5 Computational Results

This section presents some of the results we obtained by incorporating OR algorithms in constraint-based approaches to scheduling.

5.1 Job Shop Scheduling Approximation

Nuijten [1994] presents an approximation approach based on constraint satisfaction techniques which is applied to a number of scheduling problems amongst which the JSSP and a generalization of the JSSP in which machines can have an arbitrary capacity, called the Multiple Capacitated Job Shop Scheduling Problem (MCJSSP) [Nuijten & Aarts, 1994].

The results on these two problems are good. Out of 43 well known instances of the JSSP, 31 instances are solved to optimality, including the notorious MT10 instance, and the deviation of the minimum found makespan from the best known lower bound is on average 0.72%. Vaessens, Aarts & Lenstra [1994] compare 20 of the best approaches to the JSSP for 13 instances of the aforementioned set of 43 instances. Ranking the approaches that were used according to effectiveness, this approach comes in the sixth place with an average deviation of 2.26%, where the *tabu search* approach of Nowicki & Smutnicki [1993] performs best with an average deviation of 0.54%. The results on the MCJSSP are also good; out of 30 instances, 22 are solved to optimality and the deviation of the upper bounds from the lower bounds on the minimal makespan is on average 0.52%. For more extensive results we refer to [Nuijten, 1994].

5.2 Job Shop Scheduling Optimization

We also devised an optimization algorithm for the JSSP [Baptiste, 1994]. This algorithm is based on Branch-and-Bound backtracking search with constraint propagation being performed at each node of the search tree. It was tested on more than eighty instances. Table 1 gives the results obtained on the ten 10x10 JSSP instances used by Applegate & Cook in their computational study of the JSSP [Applegate & Cook, 1991]. In Table 1, columns "BT" and "CPU" give the total number of backtracks and CPU time needed to find an optimal solution and prove its optimality. Columns "BT(pr)" and "CPU(pr)" give the number of backtracks and CPU time needed for the proof of optimality. Column "CPU(1)" gives the CPU time needed to find the first solution, including the time needed for stating the problem and performing initial constraint propagation. Column "TM" gives the total amount of memory used to represent and solve the problem (in kilobytes). All CPU times are given in seconds on a HP715/50 workstation.

Ilog Schedule performs better or about as well as the specific procedure of Applegate & Cook [1991] on five problems in terms of CPU time, and on seven problems in terms of the number of backtracks needed to solve the problem. Over the ten problems, the total number of backtracks for Ilog Schedule is 579711, while the total number of nodes explored by Applegate & Cook's algorithm is 674128. The integration of an edge-finder within Ilog Schedule allows its users to enjoy the flexibility inherent to constraint programming, with performance in the same range of efficiency as specific OR algorithms.

Instance	CPU(1)	BT	CPU	BT(pr)	CPU(pr)	TM
MT10	.6	69758	1916.3	7792	230.6	140
ABZ5	.6	17636	401.7	5145	115.0	136
ABZ6	.5	898	27.3	291	8.4	136
LA19	.6	21910	529.6	5618	137.9	140
LA20	.6	74452	1521.0	22567	443.8	136
ORB1	.7	13944	412.6	5382	165.4	144
ORB2	.6	114715	3552.3	30519	927.4	140
ORB3	.6	190117	5597.0	25809	770.7	140
ORB4	.5	64652	2004.1	22443	701.8	140
ORB5	.6	11629	303.6	3755	96.8	140

Table 1: Results on ten 10x10 instances of the JSSP

5.3 A Practical Problem

The edge-finder was also tested on an industrial project scheduling problem submitted by a customer of Ilog, and found to be extremely difficult to solve. The problem consists of scheduling two projects that require common resources. There are 45 activities and five resources to consider. Each activity requires up to four resources. Within each project, activities are subjected to precedence constraints. Three of the resources are unary resources. The number of activities that require each unary resource is close to 30. The two other resources are volumetric resources with capacity greater than one.

There are two optimization criteria, viz., minimization of the end times of two specific activities, one for each project. As the projects rely on common resources, these two optimization criteria are conflicting. As a result, the final user wants to impose upper bounds on the two criteria, and wants the system to tell whether there exists a solution satisfying these upper bounds.

Table 2 reports the CPU times, in seconds, obtained for different values of the upper bounds of the two criteria. We compared two algorithms, one using arc-B-consistency and one using the edge-finder of Nuijten [1994]. If an algorithm is incapable of solving an instance within one hour of CPU time, that is reported by "–". Solving an instance means either finding a solution or proving that there is no solution. Numbers in bold correspond to the upper bounds for which there is no solution. Table 2 clearly shows that the edge-finder strongly outperforms arc-B-consistency.

	Algorithm	120	125	130	134	135	136	140
120	arc-B-consistency	**6**	**39**	**131**	**805**	**1188**	621	1
	edge-finder	**1**	**1**	**1**	**1**	**31**	1	1
125	arc-B-consistency	**40**	**387**	**1771**	–	–	–	1
	edge-finder	**1**	**1**	**1**	**1**	**50**	1	2
130	arc-B-consistency	**100**	**1172**	–	–	–	–	1
	edge-finder	**1**	**1**	**1**	**1**	**117**	2	2
134	arc-B-consistency	**276**	–	–	–	–	–	1
	edge-finder	**1**	**1**	**1**	**1**	**215**	2	2
135	arc-B-consistency	**356**	–	–	–	–	–	1
	edge-finder	**10**	**58**	**171**	**265**	**299**	113	2
136	arc-B-consistency	247	–	–	–	–	–	1
	edge-finder	1	1	2	2	2	2	2
140	arc-B-consistency	1	1	1	1	1	1	1
	edge-finder	1	1	1	2	2	2	2

Table 2: Results on an industrial project scheduling problem

6 Conclusions

We have shown how to combine operations research and artificial intelligence, in particular constraint programming, in a way that preserves the best of both, i.e., we preserved the efficiency provided by operations research and the generality of approach offered by constraint programming. We have shown that a good performance can be obtained on such a classical scheduling problem as the JSSP as well as on generalizations thereof, and on real-life scheduling problems as described in Section 5.3. In short, we think that in ILOG SCHEDULE we have found a powerful combination of techniques that allows us to tackle a broad range of practical scheduling problems in an efficient way.

References

Applegate, D., and W. Cook [1991], A computational study of the job-shop scheduling problem, *ORSA Journal on Computing* **3**, 149–156.

Baker, K.R. [1974], *Introduction to Sequencing and Scheduling*, Wiley & Sons.

Baptiste, P. [1994], Constraint-based scheduling: Two extensions, Master's thesis, University of Strathclyde.

Baptiste, P., and C. Le Pape [1995], Disjunctive constraints for manufacturing scheduling: Principles and extensions, *Proc. 3rd International Conference on Computer Integrated Manufacturing.*

Carlier, J., and E. Pinson [1990], A practical use of Jackson's preemptive schedule for solving the job shop problem, *Annals of Operations Research* **26**, 269–287.

Carlier, J., and E. Pinson [1994], Adjustment of heads and tails for the job-shop problem, *European Journal of Operational Research* **78**, 146–161.

Coffman, Jr., E.G. (ed.) [1976], *Computer & Job Shop Scheduling Theory*, Wiley, New York.

Ford, Jr., L.R. [1956], *Network Flow Theory*, Technical report, Rand Corporation.

French, S. [1982], *Sequencing and Scheduling: An Introduction to the Mathematics of the Job-Shop*, Wiley & Sons.

Garey, M.R., and D.S. Johnson [1979], *Computers and Intractability; A Guide to the Theory of NP-completeness*, W.H. Freeman and Company, New York.

Gondran, M., and M. Minoux [1984], *Graphs and Algorithms*, John Wiley and Sons.

Le Pape, C. [1988], *Des systèmes d'ordonnancement flexibles et opportunistes*, Ph.D. thesis, University Paris XI, in French.

Le Pape, C. [1994], Implementation of resource constraints in ILOG SCHEDULE: A library for the development of constraint-based scheduling systems, *Intelligent Systems Engineering* **3**, 55–66.

Lhomme, O. [1993], Consistency techniques for numeric CSPs, *Proc. 13th International Joint Conference on Artificial Intelligence.*

Nowicki, E., and C. Smutnicki [1993], *A Fast Taboo Search Algorithm for the Job Shop Problem*, Preprinty nr. 8/93, Instytut Cybernetyki Technicznej, Politechnicki Wroclawskiej, Poland.

Nuijten, W.P.M., and E.H.L. Aarts [1994], Constraint satisfaction for multiple capacitated job shop scheduling, in: A. Cohn (ed.), *Proc. 11th European Conference on Artificial Intelligence*, John Wiley & Sons, 635–639.

Nuijten, W.P.M. [1994], *Time and Resource Constrained Scheduling: A Con-*

straint Satisfaction Approach, Ph.D. thesis, Eindhoven University of Technology.

PUGET, J.-F. [1994], *A C++ Implementation of CLP*, Technical Report 94-01, ILOG S.A., Gentilly, France.

VAESSENS, R.J.M., E.H.L. AARTS, AND J.K. LENSTRA [1994], *Job Shop Scheduling by Local Search*, COSOR Memorandum 94-05, Eindhoven University of Technology.