DISJUNCTIVE CONSTRAINTS FOR MANUFACTURING SCHEDULING: PRINCIPLES AND EXTENSIONS

PHILIPPE BAPTISTE and CLAUDE LE PAPE
ILOG S.A., 2 Avenue Gallieni, BP 85
F-94253 Gentilly Cedex FRANCE
E-mail: baptiste@ilog.fr lepape@ilog.fr
Url: http://www.ilog.fr or http://www.ilog.com

ABSTRACT

Disjunctive constraints are widely used to ensure that the time intervals over which two activities require the same resource do not overlap in time. Two types of extensions of disjunctive constraints are proposed: (1) extensions allowing the representation of more complex constraints including activities that may or may not require the resource, "state resources" to represent activities which may use resources only under specific conditions, and a mechanism to represent setup times between activities; (2) extensions of the disjunctive constraint propagation algorithm to deduce more precise time-bounds. These extensions are integrated in ILOG SCHEDULE, a C++ library for constraint-based scheduling.

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 (Garey and Johnson 1979). 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 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 also 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, 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 this important variety and variability of scheduling problems, ILOG initiated the development of ILOG SCHEDULE, a C++ library enabling the representation of scheduling constraints in terms of *resources* and *activities*. SCHEDULE (Le Pape 1994a) is itself based on SOLVER, the generic software tool for object-oriented constraint programming developed and marketed by ILOG (Puget 1994). This enables the user 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.

The interest of constraint-based 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 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, the "$y \leq 9$" test could not be performed before the instantiation of y: no contradiction would be detected at this stage of the problem-solving process.

SCHEDULE includes three categories of predefined 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 suffices to determine whether a set of temporal constraints is consistent and to compute the earliest and latest start and end times of activities (Le Pape 1988).
- **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.

The problem of determining whether a set of resource utilization constraints is consistent with given time-bounds is NP-hard. The propagation of resource utilization constraints is consequently incomplete. This means that it is usually 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 constraint propagation methods are available in the current version of SCHEDULE (version 1.1).

- The first method relies on a generic mechanism allowing the definition of time-tables as discrete arrays or sequential lists of constrained variables. This mechanism is presented in details in (Le Pape 1994a). It applies to all types of resources: unary, volumetric and state resources.
- The second method applies to unary resources and state resources. It consists of posting a generic "disjunctive" constraint to ensure 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 throughout two time intervals $[s_i\ e_i)$ and $[s_j\ e_j)$, the disjunctive constraint states that either e_i is less than or equal to s_j, or e_j is less than or equal to s_i. Such a disjunctive representation has been used in the scheduling domain for years (Erschler 1976) (Carlier 1984) (Le Pape 1988). It is a priori more time-consuming but often results in more precise time-bounds than the propagation of the corresponding time-table constraints. Section 2 presents the basic principles used to propagate such constraints. Section 3 presents three extensions made in SCHEDULE to (1) allow activities that may or may not require the resource, (2) apply disjunctive constraint propagation to state resources, and (3) provide a mechanism to deal with sequence-dependent setup times between activities.
- The third method extends the disjunctive constraint propagation algorithm to deduce more precise time-bounds for each activity. Section 4 presents this extension.

2. Propagation of Disjunctive Constraints for Scheduling

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 stated as follows:

$$[end(A) \leq start(B)] \; or \; [end(B) \leq start(A)]$$

In this formula, $start(A)$, $end(A)$, $start(B)$ and $end(B)$ denote constrained variables. This means that the values of $start(A)$, $end(A)$, $start(B)$ and $end(B)$ are initially unknown. Constraint propagation consists in reducing the set of possible values for these variables: whenever the minimal possible value of $end(A)$ (earliest possible end time of A) exceeds the maximal possible value of $start(B)$ (latest possible start time of 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.

Notice that the disjunctive formulation above does not necessarily imply the explicit creation of a disjunctive constraint for each pair of activities. In ILOG SCHEDULE, a unique global constraint is created for the overall set $\{A_1 \; ... \; A_n\}$ of activities that require a given unary resource R. The propagation of this constraint is equivalent to applying the process described above to the $n(n-1)/2$ pairs of activities $\{A_i \; A_j\}$. Three extensions are described in the next section.

3. Extended Functionalities

3.1. Optional Activities and Resource Alternatives

In manufacturing scheduling, it is often the case that several resources can be used to perform the same activity. These resources can be strictly equivalent (e.g., n identical machines), in which case a discrete resource is used. They can also be different (e.g., have different operating speeds), in which case it is absolutely necessary to determine which resource is going to be used for which activity.

Some activities can also be optional if, for example, the production of parts can be either done "in house," or sub-contracted to another company.

In both of these cases, it is necessary to represent the situation in which an activity may or may not require a resource. This can be done in two ways:

- The fact that an activity may or may not require a resource can be stated by attaching a Boolean variable *demand* to the resource requirement constraint. In a given solution, *demand* is either 0 or 1. When *demand* is 1, it means that the resource **is** used for executing the activity. When *demand* is 0, it means that the resource **is not** used for executing the activity. A choice between m resources can consequently be implemented by using m Boolean variables, and stating that exactly one of these Boolean variables must be 1.

- A fully optional activity can be represented by an activity the duration of which can be either 0 or the time actually necessary to execute it. When the duration is 0, the activity does not really require the resource.

The disjunctive constraint can easily be modified to represent these two cases. For any two activities A and B that require the resource R, the disjunctive formula to satisfy is:

$$[end(A) \leq start(B)] \; or \; [end(B) \leq start(A)]$$
$$or \; [duration(A) = 0] \; or \; [duration(B) = 0]$$
$$or \; [demand(A) = 0] \; or \; [demand(B) = 0]$$

The propagation of this constraint consists in imposing one of its six disjuncts when the five other disjuncts are proven false.

3.2. State Resources

A state resource is a resource (a priori of infinite capacity) which can operate in different states. Each activity may require the resource to remain in a particular state throughout its execution. Consequently, two activities which require distinct states cannot overlap. The formula to satisfy is:

$$[end(A) \leq start(B)] \; or \; [end(B) \leq start(A)]$$
$$or \; [duration(A) = 0] \; or \; [duration(B) = 0]$$
$$or \; [state(A) = state(B)]$$

Notice that the states required by A and B can be variables. This allows, for example, the representation of the situation where a batch of products can be "cooked" at different temperatures, with the duration of the activity or the quality of products varying with the chosen temperature.

SCHEDULE also allows the representation of constraints such that: (1) activity A requires resource R in any state different from a given state s (the state may change during the execution of A, but can never be s); (2) activity A requires resource R in any state that belongs to a given set of states $\{s_1 \; ... \; s_n\}$ (the state may change during the execution of A, but must remain in the given set of states); (3) activity A requires resource R in any state that does not belong to a given set of states $\{s_1 \; ... \; s_n\}$ (the state may change during the execution of A, but can never belong to the given set of states). The disjunctive constraint can be generalized to handle constraints of types (1) and (3) when the number of possible states for the resource is infinite. When the number of possible states is finite, or when constraints of type (2) are used, the time-table mechanism (Le Pape 1994a) must be used. Nevertheless, it may still be interesting to propagate the redundant disjunctive constraint in these cases, as the disjunctive constraint may deduce more precise time-bounds.

3.3. Transition Times Between Activities

Another extension allows the definition of "transition times" between any two activities that require the same unary or state resource. Given two activities A and

B, the "transition time" between A and B is an amount of time that must elapse between the end of A and the beginning of B when A precedes B. Transition times typically exist in factories where tools must be set up prior to the execution of a manufacturing operation: the setup time varies not only with the operation for which the setup is made, but also with the preceding operation.

The disjunctive constraint can easily be modified to take transition times into account. If $tt(A, B)$ denotes the transition time between A and B, the formula to satisfy is:

$$[end(A) + tt(A, B) \leq start(B)] \; or \; [end(B) + tt(B, A) \leq start(A)]$$
$$or \; [duration(A) = 0] \; or \; [duration(B) = 0]$$
$$or \; [demand(A) = 0] \; or \; [demand(B) = 0]$$

The tt function is defined by the user of SCHEDULE with respect to the particular needs of his or her specific application. The default tt function always returns 0. Hence, if the user does not redefine tt, the disjunctive constraint behaves as if there were no setup times between activities. The main advantage of this implementation is that it does not necessitate the creation of a matrix containing $tt(A, B)$ for every A and every B. For example, if A and B are painting operations and if the setup time depends only on the colors used for A and B, $tt(A, B)$ could be defined by a matrix the dimensions of which are the number of colors — a number potentially much smaller than the number of activities.

Transition times can also apply to state resources, assuming that the transition time between two activities that require the same state is 0.

4. Extended Propagation

The previous section has shown how disjunctive constraints can be extended to represent a variety of manufacturing scheduling situations. This section presents an extension of the propagation process: the meaning of the disjunctive constraint remains the one developed in Section 2; but the constraint propagation algorithm is improved to allow the deduction of more precise time-bounds. In the context of a particular scheduling application, more CPU time may be spent propagating the constraints, but this extra propagation may result in a better exploration of the search space and, consequently, in a drastic improvement of the overall CPU time.

The extended propagation algorithm still allows the consideration of activities that may or may not require the resource, and of sequence-dependent transition times between activities. In fact, the extension of the propagation process does not interfere with the extensions of functionalities proposed in Section 3, thereby allowing the combination of both types of extensions in the same propagation algorithm.

The extended propagation algorithm works as follows: rather than just looking at pairs of activities $\{A \; B\}$, the algorithm compares the temporal characteristics of A to those of a set of activities $\Omega = \{B_1 \; ... \; B_n\}$. Let est_A denote the earliest possible start time of A, let_A denote the latest possible end time of A, and p_A denote the smallest possible duration of A. Let est_Ω denote the smallest of the

earliest possible start times of the activities in Ω, let_Ω denote the greatest of the latest possible end times of the activities in Ω, and p_Ω denote the sum of the smallest possible durations of the activities in Ω. The following rules apply:

$$let_{\Omega\cup\{A\}} - est_\Omega < p_A + p_\Omega \Rightarrow A \; is \; before \; all \; activities \; in \; \Omega$$
$$let_\Omega - est_{\Omega\cup\{A\}} < p_A + p_\Omega \Rightarrow A \; is \; after \; all \; activities \; in \; \Omega$$

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

The technique which consists in applying these rules is known as the *edge-finding* technique. Notice that if n activities require the resource, there are potentially $O(n*2^n)$ pairs $\{A\,\Omega\}$ to consider. (Carlier and Pinson 1990) presents an algorithm that performs all of the possible time-bound adjustments in $O(n^2)$. A variant of this algorithm, described in (Nuijten et al 1993), is used in SCHEDULE. (Carlier and Pinson 1994) presents a variant running in $O(n * log(n))$ that we have not tested yet.

The following table provides results obtained on ten 10x10 instances of the job-shop scheduling problem used by (Applegate and Cook 1991). In this table, BT and CPU denote the total number of backtracks and CPU time needed to find an optimal solution and prove its optimality. BT(pr) and CPU(pr) denote the number of backtracks and CPU time needed for the proof of optimality. CPU(one) denotes the CPU time needed to find one solution. This includes the time needed for stating the problem and performing initial constraint propagation. Finally, TM denotes the total amount of memory used to represent and solve the problem (in kilobytes). CPU times are expressed in seconds on a 100MHz 486DX4 PC under SOLARIS. The SCHEDULE program performs better or about as well as the specific procedure of Applegate and Cook on five problems in terms of CPU time, and on seven problems in terms of the number of backtracks needed to solve the problem. Other results are available in (Baptiste 1994) and (Le Pape 1994b).

Instance	CPU(one)	BT	CPU	BT(pr)	CPU(pr)	TM
MT10	.4	69758	1498.9	7792	179.5	140
ABZ5	.5	17636	322.6	5145	92.5	136
ABZ6	.4	898	22.4	291	6.8	136
LA19	.4	21910	424.1	5618	109.6	140
LA20	.4	74452	1207.2	22567	353.8	136
ORB1	.5	13944	318.2	5382	121.4	144
ORB2	.4	114715	2838.4	30519	742.8	140
ORB3	.5	190117	4485.5	25809	620.7	140
ORB4	.4	64652	1601.5	22443	561.5	140
ORB5	.5	11629	241.3	3755	76.9	140

5. Conclusion

Disjunctive constraints have been used by many researchers and practitioners (e.g., (Erschler 1976) (Carlier 1984) (Le Pape 1988)) to represent activities competing for the same unary resource. Two types of extensions have been proposed in this paper, to allow the representation of more complex constraints and to deduce more precise time-bounds for each activity.

These extensions are available in ILOG SCHEDULE, a C++ library aimed at simplifying the representation and the resolution of industrial scheduling problems. The integration of these extensions within the same constraint programming tool provides a strong basis for the development of scheduling applications that fit the constraints of complex manufacturing environments.

6. References

1. D. Applegate and W. Cook, "A Computational Study of the Job-Shop Scheduling Problem," *ORSA Journal on Computing* **3** (1991) 149-156.
2. P. Baptiste, *Constraint-Based Scheduling: Two Extensions* (MSc Thesis, University of Strathclyde, 1994).
3. J. Carlier, *Problèmes d'ordonnancement à contraintes de ressources : algorithmes et complexité* (Thèse de Doctorat d'Etat, University Paris VI, 1984).
4. J. Carlier and E. Pinson, "A Practical Use of Jackson's Preemptive Schedule for Solving the Job-Shop Problem," *Annals of Operations Research* **26** (1990) 269-287.
5. J. Carlier and E. Pinson, "Adjustment of Heads and Tails for the Job-Shop Problem," *European Journal of Operational Research* **78** (1994) 146-161.
6. J. Erschler, *Analyse sous contraintes et aide à la décision pour certains problèmes d'ordonnancement* (Thèse de Doctorat d'Etat, University Paul Sabatier, 1976).
7. M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness* (W. H. Freeman and Company, 1979).
8. C. Le Pape, *Des systèmes d'ordonnancement flexibles et opportunistes* (PhD Thesis, University Paris XI, 1988).
9. C. Le Pape, "Implementation of Resource Constraints in ILOG SCHEDULE: A Library for the Development of Constraint-Based Scheduling Systems," *Intelligent Systems Engineering* **3** (1994a) 55-66.
10. C. Le Pape, "Using a Constraint-Based Scheduling Library to Solve a Specific Scheduling Problem," in *Proceedings of the AAAI-SIGMAN Workshop on Artificial Intelligence Approaches to Modelling and Scheduling Manufacturing Processes* (TAI, New Orleans, Louisiana, 1994b).
11. 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).
12. J.-F. Puget, *A C++ Implementation of CLP* (Technical Report, ILOG S.A., 1994).