In: Proceedings of the Fifteenth Workshop of the U.K. Planning Special Interest Group, Liverpool, United Kingdom, 1996.

Edge-Finding Constraint Propagation Algorithms for Disjunctive and Cumulative Scheduling

Philippe Baptiste and Claude Le Pape[1]

Abstract. Edge-finding bounding techniques are particular constraint propagation techniques which reason about the order in which several activities can execute on a given resource. The aim of the following paper is to provide a rather comprehensive (although necessarily incomplete) review of existing edge-finding algorithms. The simplest case of non-preemptive disjunctive scheduling is reviewed first, followed by generalizations to preemptive disjunctive and non-preemptive cumulative scheduling. A new quadratic algorithm for non-preemptive disjunctive scheduling is also introduced. This algorithm can be used in association with a traditional edge-finding algorithm, in order to update the earliest start time (or the latest end time) of an activity A, when it is shown that A cannot precede (respectively, cannot follow) all the activities in a set Ω. To our knowledge, this is the first reported algorithm to perform all the corresponding deductions in $O(n^2)$ where n denotes the number of activities that require the resource under consideration.

1 INTRODUCTION

Roughly speaking, scheduling is the process of assigning activities to resources in time. Several basic scheduling problems can be distinguished:

- In **disjunctive** scheduling, each resource can execute at most one activity at a time. In **cumulative** scheduling, a resource can run several activities in parallel, provided that the resource capacity is not exceeded.
- In **non-preemptive** scheduling, activities cannot be interrupted. Each activity A must execute without interruption from its start time to its end time. In **preemptive** scheduling, activities can be interrupted at any time, to let some other activities execute.

Most scheduling problems can easily be represented as instances of the **constraint satisfaction problem** (CSP) [Kumar 92]: given a set of **variables** and a set of **constraints** between the variables, assign a value to each variable, so that all the constraints are satisfied. In non-preemptive scheduling, two variables, start(A) and end(A), are associated with each activity A; they represent the start time and the end time of A. A preemptive scheduling problem is more difficult to represent: one must associate a set variable (i.e., a variable the value of which will be a set) set(A) with each activity A; set(A) represents the set of times at which A executes. Ignoring implementation details, let us note that:

- in the non-preemptive case, set(A) = [start(A) end(A)), with the interval [start(A) end(A)) closed on the left and open on the right so that |set(A)| = end(A) – start(A) = duration(A);
- assuming time is discretized, start(A) and end(A) can be defined, in both the preemptive and the non-preemptive case, by $start(A) = \min_{t \in set(A)}(t)$ and $end(A) = \max_{t \in set(A)}(t + 1)$; in the preemptive case, these variables are often needed to connect activities together by temporal constraints.

Constraint propagation is a deductive process which consists in deducing new constraints from existing constraints. Given a set of activities and a set of constraints (including temporal constraints, resource constraints, problem-specific constraints, constraints representing already made decisions, and bounds on one or several optimization criteria), constraint propagation determines conditions that a schedule must satisfy to meet all the considered constraints. These necessary conditions can then be used either to find an optimal solution and prove its optimality (for "small" problems) or to guide a heuristic search procedure (or the user of an interactive system) towards "good" solutions.

Edge-finding bounding techniques are particular constraint propagation techniques which reason about the order in which several activities can execute on a given resource. In the simplest case of non-preemptive disjunctive scheduling, edge-finding consists of determining whether an activity A can, cannot, or must, execute before (or after) a set of activities Ω which require the same resource. Two types of conclusions can then be drawn: new ordering relations ("edges" in the graph representing the possible orderings of activities) and new time-bounds, i.e., strengthened earliest and latest start and end times of activities. Cumulative and preemptive scheduling cases are more complex since several activities can overlap (on a cumulative resource) or preempt one another. Then edge-finding consists of determining whether an activity A can, cannot, or must, start or end before (or after) a set of activities Ω.

The aim of this paper is to provide a rather comprehensive (although necessarily incomplete) review of existing edge-finding algorithms. Section 2 presents the first edge-finding algorithm, developed in [Pinson 88] [Carlier & Pinson 90] for non-preemptive disjunctive scheduling, discusses some of its variants, and provides a precise characterization of the results the algorithm produces. The following sections present extensions to mixed preemptive and non-preemptive disjunctive scheduling (Section 3) and to cumulative scheduling (Section 4). Finally, Section 5 provides a new quadratic algorithm for non-preemptive disjunctive scheduling. This algorithm can be used in association with a traditional edge-finding algorithm, in order to update the earliest start time (or the latest end time) of an activity A, when it is shown that A cannot precede (respectively, cannot follow) all of the activities in a set Ω. To our knowledge, this is the first reported algorithm to perform all the corresponding deductions in $O(n^2)$ where n denotes the number of activities that require the resource under consideration.

[1] Bouygues, Direction Scientifique, 1, Avenue Eugène Freyssinet,
F-78061 Saint-Quentin-en-Yvelines, France. Tel: +33 1 30 60 41 82.
Fax: +33 1 30 60 27 27. Email: baptiste@utc.fr lepape@dmi.ens.fr

2 EDGE-FINDING FOR NON-PREEMPTIVE DISJUNCTIVE SCHEDULING

In non-preemptive disjunctive scheduling, two activities A and B which require the same resource R must be ordered: either A precedes B or B precedes A. The edge-finding technique consists in determining whether an activity A must precede (or follow) a set of activities Ω which all require R. In the following, we assume the scheduling horizon is bounded, so that the domain of the start and end variables of activities are bounded by earliest and latest start and end times, possibly derived through the propagation of other constraints (such as precedence relations, release dates and due dates, etc). EST_A denotes the earliest start time of A, LET_A the latest end time of A, p_A the processing time (duration) of A, EST_Ω the smallest of the earliest start times of the activities in Ω, LET_Ω the greatest of the latest end times of the activities in Ω, and p_Ω the sum of the durations of the activities in Ω. A « B (respectively, A » B) means that A executes before (respectively, after) B and A « Ω (respectively, A » Ω) means that A executes before (respectively, after) all the activities in Ω. As we shall see, variants exist but the following rules capture most of the edge-finding technique:

$$\forall\Omega, \forall A \notin \Omega, \; LET_{\Omega\cup\{A\}} - EST_\Omega < p_\Omega + p_A \Rightarrow A \ll \Omega$$

$$\forall\Omega, \forall A \notin \Omega, \; LET_\Omega - EST_{\Omega\cup\{A\}} < p_\Omega + p_A \Rightarrow A \gg \Omega$$

$$A \ll \Omega \Rightarrow end(A) \le \min_{\Omega' \subseteq \Omega}(LET_{\Omega'} - p_{\Omega'})$$

$$A \gg \Omega \Rightarrow start(A) \ge \max_{\Omega' \subseteq \Omega}(EST_{\Omega'} + p_{\Omega'})$$

If n activities require the resource, there are a priori $O(n * 2^n)$ pairs (A, Ω) to consider. An algorithm that performs all the time-bound adjustments in $O(n^2)$ is presented in [Carlier & Pinson 90].[2] This algorithm consists of a "primal" algorithm to update earliest start times and a "dual" algorithm to update latest end times. The primal algorithm runs as follows:

- Compute "Jackson's Preemptive Schedule" (JPS) for the resource under consideration. JPS is the preemptive schedule obtained by applying the following priority rule: whenever the resource is free and one activity is available, schedule the activity A for which LET_A is the smallest. If an activity B becomes available while A is in process, stop A and start B if LET_B is strictly smaller than LET_A; otherwise continue A.
- For each activity A, compute the set Ψ of the activities which are not finished at $t = EST_A$ on JPS. Let p_B^* be the residual duration on the JPS of the activity B at time t. Take the activities of Ψ in decreasing order of latest end times and select the first activity C such that:

  $$EST_A + p_A + \sum_{B \in \Psi - \{A\} \mid LET_B \le LET_C} (p_B^*) > LET_C$$

  If such an activity C exists, then post the following constraints:

  $$A \gg \{B \in \Psi - \{A\} \mid LET_B \le LET_C\}$$

  $$start(A) \ge \max_{B \in \Psi - \{A\} \mid LET_B \le LET_C}(C_B^{JPS})$$

  where C_B^{JPS} is the completion time of activity B in JPS.

Several variants of this algorithm are available. [Nuijten 94] and [Martin & Shmoys 96] present variants which also run in $O(n^2)$, but do not require the computation of Jackson's Preemptive Schedule. An $O(n * \log(n))$ algorithm is presented in [Carlier & Pinson 94]. This algorithm is the best in terms of worst-case complexity, but relies on more complex data structures. [Caseau & Laburthe 94] presents another variant, based on the explicit definition of "task intervals." This variant runs in $O(n^3)$ in the worst case, but works in an incremental fashion and allows the performance of additional deductions (cf. Section 5). Finally, [Brucker & Thiele 96] proposes several extensions to take setup times into account.

As shown in [Lévy 96], the edge-finding algorithms above may perform different deductions than the more standard disjunctive constraint propagation algorithms [Erschler 76] [Le Pape 88], which consist of imposing a disjunct from a disjunction when all the other disjuncts have been proven impossible. In this case, the disjunctions are of the form (end(A) ≤ start(B) or end(B) ≤ start(A)) where A and B are any two activities requiring the same resource. Examples given in [Lévy 96] show that each of the two techniques (edge-finding and disjunctive constraint propagation) performs some deductions that the other technique does not perform. Examples given in [Baptiste 95] show that the same result applies to the edge-finding rules and the energetic reasoning rules of [Erschler et al 91]. In practice, an edge-finding algorithm is often coupled with a disjunctive constraint propagation algorithm to allow a maximal amount of constraint propagation to take place.

Coupled with branch-and-bound backtracking algorithms, these constraint propagation algorithms prove to be very successful on both academic and industrial problems [Carlier & Pinson 90] [Nuijten 94] [Caseau & Laburthe 95] [Baptiste & Le Pape 95]. For example, the following table provides the results obtained with the ILOG SCHEDULE scheduling tool (version 2.0) [Le Pape 95] on the ten 10x10 (10 machines, 10 jobs, 100 activities) instances of the Job-Shop Scheduling Problem (JSSP) used by Applegate and Cook in their computational study of the JSSP [Applegate & Cook 91]. (See [Blazewicz et al 96] for a review on the JSSP.) In this table, column "MAK" provides for each instance the optimal makespan, i.e., the minimal total duration of the schedule. 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. CPU times are expressed in seconds on an RS6000 workstation, rounded to the closest tenth of a second. The algorithm used to solve the problem is described in [Baptiste et al 95].

	MAK	BT	CPU	BT(pr)	CPU(pr)
MT10	930	13684	184.8	4735	52.8
ABZ5	1234	19303	226.6	4519	49.7
ABZ6	943	6227	80.3	312	3.8
LA19	842	18102	219.2	6561	74.7
LA20	902	40597	407.3	20626	186.9
ORB1	1059	22725	323.7	6261	85.3
ORB2	888	31490	416.4	14123	189.3
ORB3	1005	36729	488.4	22138	277.6
ORB4	1005	13751	169.9	1916	19.0
ORB5	887	12648	168.5	2658	29.7

[2] $O(n^2)$ corresponds to applying the rules once. In constraint programming, one would apply these rules (and other rules, used to propagate other constraints) in an incremental fashion until the domains of all the problem variables become stable. In the worst case, this could lead to executing the edge-finding algorithm O(Dn) times where D is a bound on the size of the domains of the variables involved in the propagation. In most cases, however, this does not occur, i.e., each propagation algorithm is applied only a couple of times.

[Baptiste 95] and [Martin & Shmoys 96] provide an interesting characterization of edge-finding algorithms: considering only the resource constraint and the current time-bounds of activities, the primal algorithm computes the earliest start time at which each activity A could start if all the other activities were preemptable. This suggests a logical extension of the technique to preemptive and mixed cases where some activities are preemptable and some are not: for each activity A requiring the resource, if A is not preemptable, the non-preemptive edge-finding bound applies; if A is preemptable then, considering only the resource constraint and the current time-bounds, it would be nice to determine the earliest start and end times between which A could execute if all the activities were preemptable. This is the topic of the next section.

3 EDGE-FINDING FOR PREEMPTIVE AND MIXED DISJUNCTIVE SCHEDULING

Let us define $\rangle\rangle$ so that $A \rangle\rangle \Omega$ means "A ends after all the activities in Ω" and substitute $\rangle\rangle$ for » in the rules of the primal algorithm.

$$\forall\Omega, \forall A \notin \Omega, \text{LET}_\Omega - \text{EST}_{\Omega\cup\{A\}} < p_\Omega + p_A \Rightarrow A \rangle\rangle \Omega$$

$$A \rangle\rangle \Omega \Rightarrow \text{start}(A) \geq \max_{\Omega' \subseteq \Omega} (\text{EST}_{\Omega'} + p_{\Omega'})$$

When A cannot be interrupted, these two rules remain valid (even if other activities can be interrupted) and the adjustment of EST_A is the same than in the non-preemptive case. When A can be interrupted, the first rule is still valid but the second is not. However, the second rule can be replaced by a weaker one:

$$A \rangle\rangle \Omega \Rightarrow \text{end}(A) \geq \max_{\Omega' \subseteq \Omega} (\text{EST}_{\Omega'\cup\{A\}} + p_{\Omega'\cup\{A\}})$$

This leads to a more general primal edge-finding algorithm:

- Compute Jackson's Preemptive Schedule JPS.
- For each activity A, compute the set Ψ of the activities which are not finished at $t = \text{EST}_A$ on JPS. Let p_B^* be the residual duration on the JPS of the activity B at time t. Take the activities of Ψ in decreasing order of latest end times and select the first activity C such that:
  $$\text{EST}_A + p_A + \sum_{B \in \Psi - \{A\} \mid \text{LET}_B \leq \text{LET}_C} (p_B^*) > \text{LET}_C$$
  If such an activity C exists, then post the following constraints:
  $A \rangle\rangle \{B \in \Psi - \{A\} \mid \text{LET}_B \leq \text{LET}_C\}$
  If A cannot be interrupted:
  $\text{start}(A) \geq \max_{B \in \Psi - \{A\} \mid \text{LET}_B \leq \text{LET}_C} (C_B^{JPS})$
  If A can be interrupted:
  $\text{end}(A) \geq \text{EST}_A + p_A + \sum_{B \in \Psi - \{A\} \mid \text{LET}_B \leq \text{LET}_C} (p_B^*)$

It is proven in [Baptiste 95] that considering only the resource constraint and the current time-bounds of activities, this algorithm computes:

- when A is not preemptable: the earliest time at which A could start if all the other activities were preemptable.
- when A is preemptable: the earliest time at which A could end if all the other activities were preemptable.

[Le Pape & Baptiste 96] presents a variant of this algorithm (which extends the basic edge-finding algorithm of [Nuijten 94]) and a series of experiments which show that the edge-finding technique performs better than other techniques on the Preemptive Job-Shop Scheduling Problem (PJSSP). The following table provides the results obtained on the preemptive variants of the ten JSSP instances of [Applegate & Cook 91], except ORB3 which remains open, with a lower bound of 971 and an upper bound of 973 for the optimal makespan. The algorithm used to solve the problem is a variant of the algorithm described in [Baptiste & Le Pape 96]. It is implemented in CLAIRE [Caseau & Laburthe 96a]. CPU times are expressed in seconds on a PC Dell GXL 5133, rounded to the closest tenth of a second.

	MAK	BT	CPU	BT(pr)	CPU(pr)
MT10	900	140903	5659.5	41255	1679.0
ABZ5	1203	1192553	42172.8	338597	11943.0
ABZ6	924	17699	849.5	8157	366.9
LA19	812	34637	1453.2	10928	454.0
LA20	871	2779	151.8	998	56.5
ORB1	1035	347653	13864.0	85091	3422.5
ORB2	864	53127	1844.7	16189	574.6
ORB3	OPEN				
ORB4	980	97654	3318.7	37122	1271.6
ORB5	849	10380	420.2	4151	163.0

4 EDGE-FINDING FOR NON-PREEMPTIVE CUMULATIVE SCHEDULING

Given an activity A requiring a resource of capacity $C > 1$, let W(A) be the "energy" required to execute A. Assuming A requires the same amount c_A of the resource at any time during its execution, W(A) is equal to the product $p_A * c_A$. In the following, we consider the case where p_A and c_A are given constants. However, some of the following edge-finding rules could be modified to apply when p_A and c_A are constrained variables, or when c_A is allowed to vary over time (between a minimum c_A^{min} and a maximum c_A^{max}).

Given a set Ω of activities, let $W(\Omega) = \sum_{A \in \Omega} W(A)$ be the "energy" required to execute Ω. Note that if for a given set Ω, $W(\Omega)$ exceeds the energy $C * (\text{LET}_\Omega - \text{EST}_\Omega)$ "provided" by the resource, an inconsistency is detected. This type of reasoning is heavily exploited in [Nuijten 94] [Nuijten & Aarts 96] where the following deduction rules (and their symmetric counterparts) are introduced.

Rule 1. $\forall\Omega, \forall A \notin \Omega$, if $C * (\text{LET}_\Omega - \text{EST}_{\Omega\cup\{A\}}) < W(\Omega\cup\{A\})$, then $A \rangle\rangle \Omega$.

Proof. If $\exists B \in \Omega$ such that A ends before B, then A ends before LET_Ω and thus all activities in $\Omega\cup\{A\}$ have to be scheduled during $[\text{EST}_{\Omega\cup\{A\}}\ \text{LET}_\Omega)$. Thus, $C * (\text{LET}_\Omega - \text{EST}_{\Omega\cup\{A\}}) \geq W(\Omega\cup\{A\})$. Absurd.

This deduction rule allows to update EST_A. For any non-empty subset Ω' of Ω such that $W(\Omega') > (C - c_A) * (\text{LET}_{\Omega'} - \text{EST}_{\Omega'})$, we can deduce $\text{EST}_A \geq \text{LET}_{\Omega'} - \lfloor (C * (\text{LET}_{\Omega'} - \text{EST}_{\Omega'}) - W(\Omega')) / c_A \rfloor$. Indeed, if $t \leq \text{LET}_{\Omega'}$ denotes a time at which all the activities in Ω' are completed, $W(\Omega')$ exceeds $(C - c_A) * (t - \text{EST}_{\Omega'})$, so A cannot execute all over $[\text{EST}_{\Omega'}, t)$. Since A must finish after all the activities in Ω', this implies that A must start after $\text{EST}_{\Omega'}$ and that a capacity of c_A cannot be used for activities in Ω' from the start time of A up to $\text{LET}_{\Omega'}$. Thus, the product $c_A * (\text{LET}_{\Omega'} - \text{EST}_A)$ cannot exceed the existing slack $C * (\text{LET}_{\Omega'} - \text{EST}_{\Omega'}) - W(\Omega')$. This implies $\text{EST}_A \geq \text{LET}_{\Omega'} - \lfloor (C * (\text{LET}_{\Omega'} - \text{EST}_{\Omega'}) - W(\Omega')) / c_A \rfloor$.

Rule 2. $\forall\Omega$, $\forall A \notin \Omega$, if $EST_\Omega < EST_A < \min_{B\in\Omega}(EST_B + p_B)$ and $C * (LET_\Omega - EST_\Omega) < W(\Omega) + c_A * (\min(LET_\Omega, EST_A + p_A) - EST_\Omega)$, then at least one activity B in Ω must precede A.

Proof. If A starts before the end of every activity B in Ω, then before the end of A, no activity in Ω can use the c_A resource units required by A. Thus, the energy $c_A * (\min(LET_\Omega, EST_A + p_A) - EST_\Omega)$ cannot be assigned to the activities of Ω during the interval $[EST_\Omega\ LET_\Omega)$. Therefore, if the provided energy $C * (LET_\Omega - EST_\Omega)$ is less than $W(\Omega) + c_A * (\min(LET_\Omega, EST_A + p_A) - EST_\Omega)$, at least one activity B in Ω must precede A.

This deduction rule allows to update EST_A to the new value $\min_{B\in\Omega}(EST_B + p_B)$.

Rule 3. $\forall\Omega$, $\forall A \notin \Omega$, if $EST_A < EST_\Omega$ and $EST_\Omega < EST_A + p_A$ and $C * (LET_\Omega - EST_\Omega) < W(\Omega) + c_A * (EST_A + p_A - EST_\Omega)$, then $A \rangle\rangle \Omega$.

Proof. If $\exists B\in\Omega$ such that A ends before B, then A ends before LET_Ω and thus at least $EST_A + p_A - EST_\Omega$ time units of A overlap $[EST_\Omega\ LET_\Omega)$. Therefore, if the provided energy $C * (LET_\Omega - EST_\Omega)$ is less than $W(\Omega) + c_A * (EST_A + p_A - EST_\Omega)$, A must end after all activities in Ω.

This deduction rule allows to update EST_A. As for rule 1, we can deduce $EST_A \geq LET_\Omega, - \lfloor (C * (LET_\Omega, - EST_\Omega,) - W(\Omega')) / c_A \rfloor$ for any subset Ω' of Ω such that $W(\Omega') > (C - c_A) * (LET_\Omega, - EST_\Omega,)$.

[Nuijten 94] presents constraint propagation algorithms which achieve the time-bound adjustments corresponding to the three rules above. If n is the number of activities which require the resource, and n_c is the number of distinct values assumed by c_A over all the activities A, the time complexity of these algorithms are respectively $O(n^2 * n_c)$ for rule 1 and $O(n^3 * n_c)$ for rules 2 and 3. A nice property of the first algorithm is that when the resource capacity is one, this algorithm runs in $O(n^2)$ and computes exactly the same time-bounds as the edge-finder described in Section 2.

In [Caseau & Laburthe 96b], the concept of "task interval" is applied to non-preemptive cumulative scheduling. A task interval is a set of activities $\Omega_{I,J} = \{K \mid EST_I \leq EST_K \text{ and } LET_K \leq LET_J\}$. The following rules are described:

- $\forall I, \forall J$, if $W(\Omega_{I,J}) > C * (LET_J - EST_I)$, then an inconsistency is detected.
- $\forall I, \forall J, \forall A$ such that $EST_A < EST_I$ or $LET_J < LET_A$, let slack(I, J) be the slack of $\Omega_{I,J}$ (slack(I, J) = $C * (LET_J - EST_I) - W(\Omega_{I,J})$) and overlap(A, I, J) be the product of c_A with the duration of the overlap of the intervals $[EST_I\ LET_J)$ and $[EST_A\ (EST_A + p_A))$. If overlap(A, I, J) exceeds slack(I, J), then $A \rangle\rangle \Omega_{I,J}$ (see proof of rule 3). This implies $EST_A \geq LET_J - \lfloor slack(I, J) / c_A \rfloor$.

The overall constraint propagation algorithm runs in $O(n^3)$ in the worst case. However, the quantities $W(\Omega_{I,J})$ can be incrementally maintained. A priori, the last deduction rule leads to less precise time-bound adjustment than those obtained by rules 1, 2 and 3, but its straightforward implementation results in a much more incremental algorithm than those described in [Nuijten 94].

[Carlier & Pinson 96] studies a pseudo-preemptive relaxation of the "parallel-machine" resource constraint, a specialization of the cumulative constraint to the case where each activity requires one unit of the resource ($\forall A$, $c_A = 1$). Just as for disjunctive scheduling, a particular schedule, "Jackson's Pseudo-Preemptive Schedule" (JPPS), is introduced. The makespan of JPPS can be computed in $O(n * \log(n) + n * C * \log(C))$ time and is a tight lower bound for the parallel-machine problem with earliest and latest start and end times. A key improvement would be, like for disjunctive scheduling, to use JPPS to obtain tight time-bound adjustments, but to the best of our knowledge no such technique is available yet.

In most cases, cumulative edge-finding algorithms are coupled with (explicit or implicit) time-table mechanisms [Le Pape & Smith 88] [Fox 90] [Le Pape 94] which keep track of information about resource utilization and resource availability over time. For example, when the latest start time LST of an activity A is smaller than its earliest end time EET, it is guaranteed that A will execute between LST and EET. Over this period, c_A units of capacity are marked as no longer available for other activities B. When c_B exceeds the capacity that remains available at time t, this leads to updating the time-bounds of B. In most cases, the edge-finding and the time-table techniques effectively complement each other, i.e., the combination of both techniques provides more precise time-bounds than each technique considered separately.[3] As shown in [Lock 96], "energetic reasoning" extensions [Erschler et al 91] [Lopez 91] [Beck 92] [Baptiste & Le Pape 95] of the time-table mechanism can also be considered, but it is still unclear how much energetic reasoning can be used without incurring an excessive cost in CPU time.

5 A QUADRATIC ALGORITHM FOR THE NOT-FIRST / NOT-LAST PROBLEM

The algorithms presented in the preceding sections mostly focus on determining whether an activity A **must** execute before (or after) a set of activities Ω requiring the same resource. A natural complement consists of determining whether A **can** execute before (or after) Ω. In the non-preemptive disjunctive case, this leads to the following rules [Pinson 88] [Carlier & Pinson 90] [Caseau & Laburthe 94]:

$$\forall\Omega, \forall A \notin \Omega, LET_\Omega - EST_A < p_\Omega + p_A \Rightarrow \neg(A \ll \Omega)$$

$$\forall\Omega, \forall A \notin \Omega, LET_A - EST_\Omega < p_\Omega + p_A \Rightarrow \neg(A \gg \Omega)$$

$$\neg(A \ll \Omega) \Rightarrow start(A) \geq \min_{B\in\Omega}(EST_B + p_B)$$

$$\neg(A \gg \Omega) \Rightarrow end(A) \leq \max_{B\in\Omega}(LET_B - p_B)$$

The problem which consists of performing all the time-bound adjustments corresponding to the first and third rules can be called the "not-first" problem, since it consists of updating the earliest start time of every activity A which cannot be first to execute in a set $\Omega\cup\{A\}$. Similarly, the problem which consists of performing all the time-bound adjustments corresponding to the second and fourth rules can be called the "not-last" problem: it consists of updating the latest end time of every activity A which cannot be last to execute in a set $\Omega\cup\{A\}$. Most researchers who have been working on edge-finding techniques have considered the "not-first" and "not-last" rules above [Pinson 88] [Carlier & Pinson 90] [Caseau & Laburthe 94] [Nuijten 94] [Baptiste & Le Pape 95] [Lévy 96], but in the absence of low-polynomial algorithms for solving the complete "not-first" problem, the rules had to be applied in an incomplete way, allowing only **some** but not all of the possible time-bound adjustements. In this section, we present an $O(n^2)$ time and O(n) space algorithm to solve the "not-first"

[3] In the disjunctive case, the time-table mechanism is generally not considered, as it is subsumed by the disjunctive constraint propagation mechanism mentioned in Section 2 [Le Pape & Baptiste 96].

problem. The "not-last" problem is solved in a symmetric fashion. To our knowledge, this is the first reported algorithm to perform all the deductions allowed by the rules above in quadratic time.

Let us first introduce some assumptions and notations. We assume that the relation $EST_A + p_A \leq LET_A$ holds for every activity A. Otherwise, the scheduling problem clearly allows no solution and the constraint propagation process can stop. We also assume that the activities A_1 ... A_n which require the resource under consideration are sorted in non-decreasing order of due-dates (this can be done in $O(n * \log(n))$ time). EST_i, LET_i and p_i denote the earliest start time, latest end time, and processing time of A_i. Hence, $i \leq j$ implies $LET_i \leq LET_j$. For a given j and a given k, $\Omega(jk)$ denotes the set of indices m in $\{1 ... k\}$ such that $EST_j + p_j \leq EST_m + p_m$ and $\Omega(ijk)$ denotes $\Omega(jk) - \{i\}$. Hence, if i does not belong to $\Omega(jk)$, $\Omega(ijk)$ is equal to $\Omega(jk)$. Let $S_{j,k} = p_{\Omega(jk)}$ if $j \leq k$ and $S_{j,k} = -\infty$ otherwise. Let $\delta_{j,k} = \min_{l \leq k}(LET_l - S_{j,l})$.

Lemma 1. For a given j, the values $\delta_{j,1}$ ... $\delta_{j,n}$ can be computed in O(n) time.

Proof. Indeed, the values of $S_{j,k}$ and $\delta_{j,k}$ can be computed in constant time from the values of $S_{j,k-1}$ and $\delta_{j,k-1}$. One just has to test whether k verifies $EST_j + p_j \leq EST_k + p_k$ or not.

Lemma 2. If the "not-first" rules applied to activity A_i and set Ω allow to update the earliest start time of A_i to $EST_j + p_j$ then there exists an index $k \geq j$ such that the "not-first" rules applied to activity A_i and set $\Omega(ijk)$ allow to update the earliest start time of A_i to $EST_j + p_j$.

Proof. Let k be the maximal index of the activities in Ω. Ω is included in $\Omega(ijk)$ and LET_Ω is equal to $LET_{\Omega(ijk)}$. Hence the rules can be applied to A_i and $\Omega(ijk)$ and provide the conclusion that A_i cannot start before $EST_j + p_j$ since every m in $\Omega(ijk)$ satisfies $EST_j + p_j \leq EST_m + p_m$.

Lemma 3. Let i and j be such that $EST_i + p_i < EST_j + p_j$. In this case, the "not-first" rules allow to update the earliest start time of A_i to $EST_j + p_j$ if and only if $EST_i + p_i > \delta_{j,n}$.

Proof. $\Rightarrow$ Let us assume that the rules allow to update the earliest start time of A_i to $EST_j + p_j$. According to lemma 2, there exists $k \geq j$ such that $\Omega(ijk)$ is not empty and $LET_k - EST_i < p_{\Omega(ijk)} + p_i$. Since $EST_i + p_i < EST_j + p_j$ implies that i does not belong to $\Omega(jk)$, this implies $EST_i + p_i > LET_k - S_{j,k} \geq \delta_{j,n}$.

$\Leftarrow$ Let us assume that $EST_i + p_i > \delta_{j,n}$. $\delta_{j,n}$ is finite, so there exists an index $k \geq j$ such that $\delta_{j,n} = LET_k - S_{j,k}$. Since i does not belong to $\Omega(jk)$, this implies that $LET_k - EST_i < p_{\Omega(ijk)} + p_i$. So, the rules allow to update the earliest start time of A_i to $EST_j + p_j$.

Lemma 4. Let i and j be such that $EST_i + p_i \geq EST_j + p_j$. In this case, the "not-first" rules allow to update the earliest start time of A_i to $EST_j + p_j$ if and only if either $EST_i + p_i > \delta_{j,i-1}$ or $EST_i > \delta_{j,n}$.

Proof. $\Rightarrow$ Let us assume that the rules allow to update the earliest start time of A_i to $EST_j + p_j$. According to lemma 2, there exists $k \geq j$ such that $\Omega(ijk)$ is not empty and $LET_k - EST_i < p_{\Omega(ijk)} + p_i$. Two cases, $k < i$ and $i < k$, can be distinguished. If $k < i$, i does not belong to $\Omega(jk)$. This implies $EST_i + p_i > LET_k - S_{j,k} \geq \delta_{j,i-1}$. On the contrary, if $i < k$, i belongs to $\Omega(jk)$. Then $p_{\Omega(jk)} = p_{\Omega(ijk)} + p_i$ and $EST_i > LET_k - S_{j,k} \geq \delta_{j,n}$.

$\Leftarrow$ If $EST_i + p_i > \delta_{j,i-1}$, $\delta_{j,i-1}$ is finite, so there exists an index $k < i$ such that $\delta_{j,i-1} = LET_k - S_{j,k}$. Since i does not belong to $\Omega(jk)$, this implies that $LET_k - EST_i < p_{\Omega(ijk)} + p_i$. So, the rules allow to update the earliest start time of A_i to $EST_j + p_j$. Let us now assume that $EST_i + p_i \leq \delta_{j,i-1}$ and $EST_i > \delta_{j,n}$. Then there exists an index $k \geq j$ such that $\delta_{j,n} = LET_k - S_{j,k}$. Note that $k \geq i$ (otherwise, $\delta_{j,n} = \delta_{j,i-1} < EST_i$ contradicts $EST_i + p_i \leq \delta_{j,i-1}$). Consequently, i belongs to $\Omega(jk)$. In addition, $\Omega(jk)$ is not reduced to $\{i\}$, otherwise we would have $EST_i > \delta_{j,n} = LET_k - S_{j,k} = LET_k - p_i \geq LET_i - p_i$ which contradicts the initial assumption that $EST_i + p_i \leq LET_i$ for all i. Hence, $\Omega(ijk)$ is not empty and satisfies $LET_k - EST_i < p_{\Omega(ijk)} + p_i$. So, the rules allow to update the earliest start time of A_i to $EST_j + p_j$.

Theorem. The following algorithm performs the same time-bound adjustments than the "not-first" rules. It runs in $O(n^2)$ time and O(n) space.

```
ITERATE ON Aj
  • COMPUTE δj,1 ... δj,n
  • ITERATE ON Ai
      IF ESTi + pi < ESTj + pj
      THEN
          IF ESTi + pi > δj,n
          THEN ESTi = max(ESTi, ESTj + pj)
          END IF
      ELSE
          IF ESTi + pi > δj,i-1 or ESTi > δj,n
          THEN ESTi = max(ESTi, ESTj + pj)
          END IF
      END IF
    END ITERATE
END ITERATE
```

Proof. Lemmas 3 and 4 imply that the algorithm performs exactly the deductions implied by the rules. The algorithm runs in $O(n^2)$ steps since for each j in the outer loop, O(n) steps are required to compute $\delta_{j,1}$ ... $\delta_{j,n}$ and for each i in the inner loop, O(1) steps are required to perform the relevant tests. In addition, the algorithm requires a linear amount of memory space since only the values $\delta_{j,1}$ ... $\delta_{j,n}$ for a given j are required.

Let us note that when the durations of activities are fixed, the "not-first" and "not-last" rules subsume the disjunctive constraint propagation technique mentioned in Section 2. Hence, no disjunctive constraint propagation algorithm is needed when the "not-first" algorithm above and its dual "not-last" algorithm are applied.

6 CONCLUSION AND PERSPECTIVES

This paper has presented two extensions of the basic disjunctive edge-finding technique: preemptive edge-finding and cumulative edge-finding. We believe that much more work remains, especially in the cumulative case for which the constraint propagation algorithms are still rather costly in terms of CPU time. We also think that the preemptive cumulative case which, to our knowledge, remains unexplored is of great theoretical and practical interest.

ACKNOWLEDGEMENTS

Part of the work presented in this paper was done while the second author was finishing a Master's thesis at ILOG S.A. The authors want to thank Bruno de Backer, Henri Béringer, Jacques Carlier, Yves Caseau, Ulrich Junker, François Laburthe, Michel Leconte, Wim Nuijten, Jean-François Puget, and Jean-Charles Régin, for many enlightening discussions on constraint programming, resource constraints, and edge-finding.

REFERENCES

[Applegate & Cook 91]
D. Applegate and W. Cook. *A Computational Study of the Job-Shop Scheduling Problem.* ORSA Journal on Computing, 3(2):149-156, 1991.

[Baptiste & Le Pape 95]
Ph. Baptiste and C. Le Pape. *A Theoretical and Experimental Comparison of Constraint Propagation Techniques for Disjunctive Scheduling.* Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, 1995.

[Baptiste et al 95]
Ph. Baptiste, C. Le Pape and W. Nuijten. *Constraint-Based Optimization and Approximation for Job-Shop Scheduling.* Proceedings of the AAAI SIGMAN Workshop on Intelligent Manufacturing Systems, IJCAI, 1995.

[Baptiste 95]
Ph. Baptiste. *Resource Constraints for Preemptive and Non-Preemptive Scheduling.* MSc Thesis, University Paris VI, 1995.

[Baptiste & Le Pape 96]
Ph. Baptiste and C. Le Pape. *A Constraint-Based Branch-and-Bound Algorithm for Preemptive Job-Shop Scheduling.* Proceedings of the International Workshop on Production Planning and Control, 1996.

[Beck 92]
H. Beck. *Constraint Monitoring in TOSCA.* Proceedings of the AAAI Spring Symposium on Practical Approaches to Planning and Scheduling, 1992.

[Blazewicz et al 96]
J. Blazewicz, W. Domschke and E. Pesch. *The Job-Shop Scheduling Problem: Conventional and New Solution Techniques.* European Journal of Operational Research, 93(1):1-33, 1996.

[Brucker & Thiele 96]
P. Brucker and O. Thiele. *A Branch & Bound Method for the General-Shop Problem with Sequence-Dependent Setup-Times* . OR Spektrum (to appear).

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

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

[Carlier & Pinson 96]
J. Carlier and E. Pinson. *Jackson's Pseudo-Preemptive Schedule for the Pm/ri,qi/Cmax Scheduling Problem.* Technical Report, Université de Technologie de Compiègne, 1996.

[Caseau & Laburthe 94]
Y. Caseau and F. Laburthe. *Improved CLP Scheduling with Task Intervals.* Proceedings of the Eleventh International Conference on Logic Programming, 1994.

[Caseau & Laburthe 95]
Y. Caseau and F. Laburthe. *Disjunctive Scheduling with Task Intervals.* Technical Report, Ecole Normale Supérieure, 1995.

[Caseau & Laburthe 96a]
Y. Caseau and F. Laburthe. *CLAIRE: A Parametric Tool to Generate C++ Code for Problem Solving.* Working Paper, Bouygues, Direction Scientifique, 1996.

[Caseau & Laburthe 96b]
Y. Caseau and F. Laburthe. *Cumulative Scheduling with Task Intervals.* Proceedings of the Joint International Conference and Symposium on Logic Programming, 1996.

[Erschler 76]
J. 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).

[Erschler et al 91]
J. Erschler, P. Lopez et C. Thuriot. *Raisonnement temporel sous contraintes de ressource et problèmes d'ordonnancement.* Revue d'Intelligence Artificielle, 5(3):7-32, 1991 (in French).

[Fox 90]
B. R. Fox. *Chronological and Non-Chronological Scheduling*. Proceedings of the First Annual Conference on Artificial Intelligence, Simulation and Planning in High Autonomy Systems, 1990.

[Kumar 92]
V. Kumar. *Algorithms for Constraint Satisfaction Problems: A Survey*. AI Magazine, 13(1):32-44, 1992.

[Le Pape & Smith 88]
C. Le Pape and S. F. Smith. *Management of Temporal Constraints for Factory Scheduling*. In: C. Rolland, F. Bodart and M. Léonard (editors), "Temporal Aspects in Information Systems," North-Holland, 1988.

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

[Le Pape 94]
C. 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.

[Le Pape 95]
C. Le Pape. *Three Mechanisms for Managing Resource Constraints in a Library for Constraint-Based Scheduling.* Proceedings of the INRIA/IEEE Conference on Emerging Technologies and Factory Automation, 1995.

[Le Pape & Baptiste 96]
C. Le Pape and Ph. Baptiste. *Constraint Propagation Techniques for Disjunctive Scheduling: The Preemptive Case.* Proceedings of the Twelfth European Conference on Artificial Intelligence, 1996.

[Lévy 96]
M.-L. Lévy. *Méthodes par décomposition temporelle et problèmes d'ordonnancement.* PhD Thesis, Institut National Polytechnique de Toulouse, 1996 (in French).

[Lock 96]
H. C. R. Lock. *An Implementation of the Cumulative Constraint*. Working Paper, University of Karlsruhe, 1996.

[Lopez 91]
P. Lopez. *Approche énergétique pour l'ordonnancement de tâches sous contraintes de temps et de ressources*. PhD Thesis, Université Paul Sabatier, 1991 (in French).

[Martin & Shmoys 96]
P. Martin and D. B. Shmoys. *A New Approach to Computing Optimal Schedules for the Job-Shop Scheduling Problem.* Proceedings of the Fifth Conference on Integer Programming and Combinatorial Optimization, 1996.

[Nuijten 94]
W. P. M. Nuijten. *Time and Resource Constrained Scheduling: A Constraint Satisfaction Approach.* PhD Thesis, Eindhoven University of Technology, 1994.

[Nuijten & Aarts 96]
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, 90:269-284, 1996.

[Pinson 88]
E. Pinson. *Le problème de job-shop*. PhD Thesis, University Paris VI, 1988 (in French).