Arc-B-Consistency of the Inter-Distance Constraint

K. Artiouchine Ph. Baptiste *

February 8, 2009

Abstract

We study the "inter-distance constraint", also known as the global minimum distance constraint, that ensures that the distance between any pair of variables is at least equal to a given value. When this value is 1, the inter-distance constraint reduces to the all-different constraint. We introduce an algorithm to propagate this constraint and we show that, when variables domains are intervals, our algorithm achieves arc-B-consistency. It provides tighter bounds than generic scheduling constraint propagation algorithms (like edge-finding) that could be used to capture this constraint. The worst case complexity of the algorithm is cubic but it behaves well in practice and it drastically reduces the search space. Experiments on special Job-Shop problems and on an Air-Traffic problem known as the "Runway Sequencing" problem.

1 Introduction

We study a global constraint, the "inter-distance constraint" that ensures that the distance between any pair (S_i, S_j) of variables in some set $\{S_1, ..., S_n\}$ is not smaller than a given value p, *i.e.*, $\forall i \neq j, |S_i - S_j| \geq p$. To our knowledge, this constraint has been introduced for the first time by Régin [23]. When p is 1, the inter-distance constraint reduces to the well-known all-different constraint [22], [20], [15].

This study was motivated by an industrial application for Air Traffic Management in the Terminal RAdar Control Area of airports [2]. When aircraft reach the final descent in the "Terminal Radar Approach CONontrol" area (TRACON), a set of disjoint time windows in which the landing is possible, can be automatically assigned to each aircraft. The objective is then to determine landing times, within these time windows, which maximize the minimum time elapsed between consecutive landings. The decision variant of this problem, (*i.e.*, when the minimum time elapsed between consecutive landings is fixed and when the question is to determine if there are feasible

*CNRS LIX, Ecole Polytechnique, F-91128 Palaiseau, Philippe.Baptiste@polytechnique.fr

landing times or not), can be modeled with an inter-distance constraint. The inter-distance constraint is also useful to model scheduling situation in which all jobs that have to be processed on the same machine have the same processing time. This is often the case in manufacturing scheduling problems (see for instance the testbed proposed by Ilog `www2.ilog.com/masclib` described in [18]).

The objective of this paper is to present a global constraint propagation algorithm for the inter-distance constraint. As explained in Section 2, standard scheduling constraint propagation algorithm, like edge-finding or "Not-First/Not-Last", can be used to model this constraint. We also show that these more generic algorithms do not perform all possible deductions. An algorithm that determines whether the constraint is globally consistent or not is described in Section 3. We then introduce propagation rules (Section 4) and a polynomial time algorithm to propagate the inter-distance constraint (Section 5). We show that, when variables domains are intervals, our algorithm achieves the arc-B-consistency (*i.e.*, arc consistency restricted to the bounds of the domains of the variables [14]) of the global constraint and hence performs the best possible filtering. The worst case complexity of the algorithm is cubic but it behaves well in practice and it drastically reduces the search space. Experiments (Section 6) on special Job-Shop problems and on our industrial application are reported.

2 Inter-Distance Constraint *vs.* Scheduling Constraints

Régin [23] relies on the sequencing constraint [24] to propagate the "inter-distance constraint". However, we believe it is somewhat easier to consider this constraint as a pure scheduling constraint. To each variable S_i, we associate a job i starting at time S_i and whose processing time is p. The release date of i is the minimum value in the domain of S_i. The deadline of i is the maximum value in the domain of S_i plus p. The disjunctive constraint directly ensures that activities are not processed simultaneously and hence, the distance between any pair of starting times is at least p. So both models are identical.

Over the last decade, several resource constraint propagation algorithms have been designed to address a variety of scheduling situations (see [3] for a review). We first describe two constraint propagation schemes known as "Edge-Finding" and "Not-First/Not-Last" that are widely used in the literature for disjunctive scheduling. Second, we show that they can be improved when all processing times are equal.

2.1 Edge-Finding

The edge-finding algorithm [8], [9], [16] is one of the most well known Operations Research algorithm integrated in CP. This global constraint propagation algorithm for disjunctive scheduling

is a key ingredient for solving complex scheduling problems such as the Job-Shop Scheduling problem

The term "Edge-Finding" denotes both a "branching" and a "bounding" technique [1]. The branching technique consists of ordering jobs that require the same resource. At each node, a set of jobs Ω is selected and, for each job $i \in \Omega$, a new branch is created where i is constrained to execute first (or last) among the jobs in Ω. The bounding technique consists of deducing that some jobs from a given set Ω must, can, or cannot, execute first (or last) in Ω. Such deductions lead to new ordering relations ("edges" in the graph representing the possible orderings of jobs) and new time bounds, *i.e.*, strengthened earliest start times and latest end times of jobs.

In the following, r_Ω and d_Ω respectively denote the smallest of release dates and the largest deadline of the jobs in Ω. Moreover, p_Ω is the sum of the processing times of the jobs in Ω. Finally, let $i \gg \Omega$ mean that i executes after all the jobs in Ω and let S_i be the variable representing the starting time of the job i. The following rules capture the "essence" of the Edge-Finding bounding technique:

$$\forall \Omega, \forall i \notin \Omega, [d_\Omega - r_{\Omega \cup \{i\}} < p_\Omega + p_i] \Rightarrow [i \gg \Omega]$$
$$\forall \Omega, \forall i \notin \Omega, [i \gg \Omega] \Rightarrow [S_i \geq \max_{\emptyset \neq \Omega' \subseteq \Omega} (r_{\Omega'} + p_{\Omega'})]$$

An algorithm that performs all the time-bound adjustments in $O(n^2)$ is presented in [8]. Another variant of the Edge-Finding technique is presented in [9]. It runs in $O(n \log n)$ but requires more complex data structures.

2.2 Not-First/Not-Last

The algorithms presented earlier focus on determining whether a job i must execute first (or last) in a set of jobs $\Omega \cup \{i\}$ requiring the same resource. A natural complement consists of determining whether i can execute first (or last) in $\Omega \cup \{i\}$. If not, i is "Not-First" and cannot start before at least one job in Ω is completed. This leads to the following rules [19]:

$$\forall \Omega, \forall i \notin \Omega, [d_\Omega - r_i < p_\Omega + p_i] \Rightarrow \neg(i \ll \Omega)$$
$$\forall \Omega, \forall i \notin \Omega, \neg(i \ll \Omega) \Rightarrow S_i \geq \min_{j \in \Omega} S_j + p_j$$

A quadratic algorithm is described in [4]. Alternative approaches can be found in [27], [10], [28].

2.3 Missed Deductions, Special Cases

It is well known that Edge-Finding and Not-First/Not-Last propagation rules do not ensure the consistency of the global disjunctive constraint (determining whether this constraint is consistent is NP-Complete in the strong sense as it reduces to scheduling a set of jobs with arbitery time windows on a single machine [11]). What happens when processing times are equal ?

To answer this question, we first recall a characterization of Edge-Finding from [3]: *The Edge-Finding algorithm computes the smallest start time at which each job i could start if all jobs, except i, could be interrupted.* As interruptions (or preemptions) are always integral (see [5] for an overview of structural properties of preemptive schedules), this property directly ensures that *when processing times are 1, edge-finding achieves Are-B-Consistency on the Inter-Distance constraint.*

Second, consider the general case where the processing time p is greater than or equal to 2. The following examples show that Edge-Finding and Not-First/Not-Last do not perform all possible deductions.

In the example described in Figure 1, 6 jobs with processing time 2 are to be scheduled on a single machine. The time window (release dates / deadlines) associated to a job is drawn as white rectangle inside which the job itself (black rectangle) has to be scheduled. There is no feasible schedule (the machine is full from 3 to 9 because of the last three jobs, hence 3 jobs have to be scheduled in two disjoint time windows with size 3; contradiction) but neither edge-finding nor Not-First/Not-Last rule detect this. In the example of Figure 2, 3 jobs with processing time

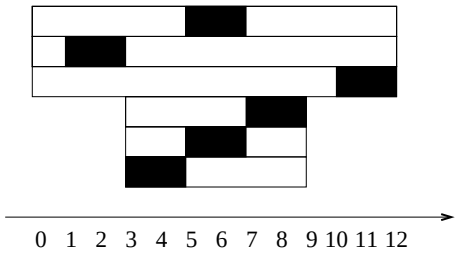

Figure 1: Edge-Finding and Not-First/Not-Last do not detect all inconsistencies

5 have to be scheduled on a single machine. There is a feasible schedule but neither edge-finding nor Not-First/Not-Last make any deductions. While it is clear that the third job cannot start earlier than 10.

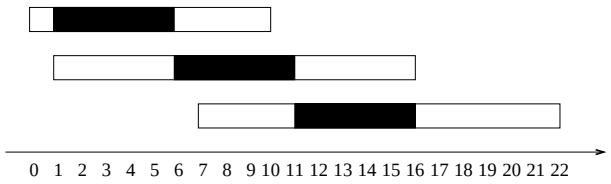

Figure 2: Edge-Finding and Not-First/Not-Last do not achieve Arc-B-Consistency

3 Feasibility test

From now on, we focus on the scheduling problem with identical processing times. Garey, Johnson, Simons, and Tarjan [12] have introduced an $O(n \log n)$ algorithm to solve this problem. We describe a cubic version of this algorithm based on similar techniques as in [12]. We have however modified the presentation of the algorithm (and also the proofs) to be able to introduce the adjustments of release dates and deadlines described in Sections 4 and 5.

3.1 Why EDD Fails

EDD (Earliest DeaDline) is a dispatching rule that builds a schedule chronologically as follows: Whenever the machine is idle, select among jobs that are released before or at the current time point, the job with minimal deadline. Schedule the job and iterate. When preemption is allowed, the preemptive EDD rule computes a feasible schedule if one such exists [8] (this also holds in the general case with arbitrary processing times). It is well known that this is not the case in the non-preemptive case. We show in Figure 3 a 3 job instance with identical processing times for which EDD fails while there is a feasible schedule. When processing times are all equal, Garey,

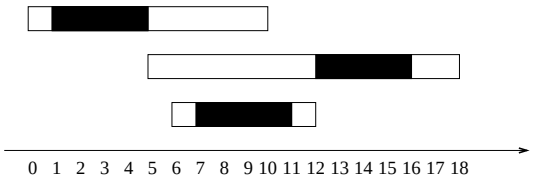

Figure 3: EDD fails while there is a feasible schedule.

Johnson, Simons, and Tarjan propose to modify the EDD rule to ensure that it builds a feasible schedule, if one exits. The modification consists in introducing a set of forbidden regions F in which no job can start on any feasible schedule.

Given a set of forbidden regions F, the modified EDD rule keeps the schedule idle when the current time point t belongs to F (see Algorithm 1). Throughout this algorithm, U denotes the set of yet unscheduled jobs. At each iteration one job of U is scheduled.

Algorithm 1 Modified EDD schedule

```
1: U := {1, ..., n}
2: t := min_{i∈U} r_i
3: while U ≠ ∅ do
4:    t := max(t, min_{i∈U} r_i)
5:    if t ∈ F then
6:       t := min{t' ≥ t : t' ∉ F}
7:    end if
8:    Find k in U with smallest deadline s.t. r_k ≤ t
9:    Start job k at time t, U := U − {k}, t := t + p
10: end while
```

3.2 Computing Forbidden Regions

The crucial point is how to compute the set F of "forbidden regions" . We first provide an *intuitive description* of this mechanism. F is built step by step, starting from $F = \emptyset$. Given a set of jobs X, compute an upper bound *lst* (latest start time) of the largest time point such that there is a feasible schedule of the jobs in X that is idle before *lst* (*i.e.*, no job is scheduled before *lst*) and in which no job starts in F. If *lst* is smaller than $\min_{i\in X} r_i$ then there is no feasible schedule. If $lst - p < \min_{i\in X} r_i$ then no job can start after $lst - p$ and before $\min_{i\in X} r_i$ (if this were the case, the job would finish after *lst* and, thus we would not have a feasible schedule). So no job starts in the interval $[lst - p + 1, \min_{i\in X} r_i - 1]$ and this interval is added to the set of forbidden regions F.

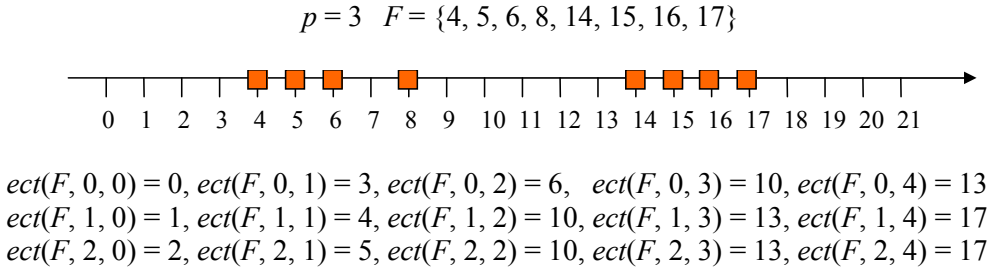

Figure 4: Computing *ect* values

To give a formal description of this principle, we introduce the two following notations:

Definition 1. *Given a time point t, an integer q and a set of forbidden region F, $ect(F, t, q)$ and $lst(F, t, q)$ respectively denote the* ***earliest completion time (resp. latest start time)*** *of a schedule of q jobs, with no release date and no deadline, starting after or at (resp. completed*

before or at) t.

Algorithms 2 and 3 compute respectively $ect(F, t, q)$ and $lst(F, t, q)$. In the following, Algorithm 2 is called the "Forward Scheduling Algorithm" and Algorithm 3 is called the "backward Scheduling Algorithm". The proof of correctness is easy to make by induction on q and is skipped. Figure 4 illustrates these definitions.

Algorithm 2 Earliest Completion Time of q jobs starting after or at t

$t' = t$
for $i = 1$ to q **do**
  if $t' \in F$ **then**
    $t' := \min\{\theta \geq t' : \theta \notin F\}$
  end if
  $t' := t' + p$
end for
$ect(F, t, q) = t'$

Algorithm 3 Latest Start Time of q jobs to be completed before or at t

$t' = t$
for $i = 1$ to q **do**
  $t' := t' - p$
  if $t' \in F$ **then**
    $t' := \max\{\theta \leq t' : \theta \notin F\}$
  end if
end for
$lst(F, t, q) = t'$

We are now ready to explain Algorithm 4 that computes all forbidden regions. In the following, $\mathbf{\Delta}(\mathbf{r}, \mathbf{d})$ **stands for the set of jobs** $\{\mathbf{i} : \mathbf{r} \leq \mathbf{r_i}, \mathbf{d_i} \leq \mathbf{d}\}$ and $|\Delta(r, d)|$ is the cardinality of this set.

Proposition 1. *If Algorithm 4 fails then there is no feasible schedule. Moreover, in any feasible schedule, jobs do not start in* F.

Proof. Assume that the proposition holds for all the regions that have been added by Algorithm 4 up to the current iteration (r, d). If $lst < r$ then in any feasible schedule, one job of $\Delta(r, d)$ starts before the minimal release date in this set; contradiction. Now assume that $r \leq lst < r + p$ (otherwise no forbidden region is added to F and our claim holds up to the next iteration). If

Algorithm 4 Forbidden Regions

```
1: F := ∅
2: for all release date r taken in non-increasing order do
3:    lst := ∞
4:    for all deadline d taken in non-increasing order do
5:       lst := min(lst, lst(F, d, |Δ(r, d)|))
6:    end for
7:    if lst < r then
8:       There is no feasible schedule
9:    else if r ≤ lst < r + p then
10:      F := F ∪ [lst − p + 1, r − 1]
11:   end if
12: end for
```

there is a feasible schedule in which a job u is completed at $t \in [lst+1, r+p-1]$ then $u \notin \Delta(r,d)$ and thus no job of $\Delta(r,d)$ starts before $t \geq lst+1$ which contradicts the definition of lst. □

Proposition 2. *Given two time points $t_1 \leq t_2$, an integer q and a set of forbidden regions F, $ect(F, t_1, q) > t_2$ if and only if $lst(F, t_2, q) < t_1$.*

Proof. If $ect(F, t_1, q) \leq t_2$ then there is a schedule of q jobs that can be executed between t_1 and t_2. Hence a schedule of q jobs completed at t_2 with no starting times in F can start after or at t_2. Hence $lst(F, t_2, q) \geq t_1$. □

Proposition 3. *If Algorithm 4 does not fail, there is a feasible schedule.*

The proof of this Proposition can be found in [12]. The following proof is slightly different and is important to understand the paper.

Proof. We claim that, given the set of forbidden regions F computed by the Algorithm 4, a feasible schedule is built by Algorithm 1 when applied to the set of forbidden regions F. Let us assume this is not true and let k' denote the first job that is completed after its deadline by Algorithm 1. We iteratively build a set of jobs S'. Initially, $S' = \{k'\}$; we then add in S' all the jobs preceding k' until we either reach a time point $t \notin F$ at which all jobs released before t are completed or until we reach a job u with $d_u > d_{k'}$.

- If we have reached a time point $t \notin F$ at which all jobs released before t are scheduled then, let j' be the job in S' with minimal release date. So, at some step of the algorithm the interval $r_{j'}, d_{k'}$ was considered. All jobs of S' belong to this interval. When building the

EDD schedule of this set, note that the forbidden regions that are encountered are those in the set F as built at iteration r. Indeed, forbidden regions added at a later iteration end before r so they do not interact with jobs considered at this step.

Also note that time points at which the machine is idle are forbidden (these time points do not correspond to release dates because of the construction of S'). Hence the shape of the EDD schedule after $r_{j'}$ up to $d_{k'}$ is exactly the same as the one obtained by the forward scheduling algorithm (Algorithm 2) when computing $ect(F, r_{j'}, |\Delta(r_{j'}, d_{k'})|)$. Hence $ect(F, r_{j'}, |\Delta(r_{j'}, d_{k'})|) > d_{k'}$. So, according to Proposition 2, $lst(F, d_{k'}, |\Delta(r_{j'}, d_{k'})|) < r_{j'}$. Hence Algorithm 4 would declare the failure.

- If we have reached a job u with $d_u > d_{k'}$. Then All jobs i in S' are such that $d_i \leq d_{k'}$ and $r_i > r_u$ (otherwise EDD would have scheduled another job instead of u). As before, let j' denote the job in S' with minimal release date. So S' is a subset of $\Delta(r_{j'}, d_{k'})$.

  Let t_u denote the starting time of u on the schedule. Algorithm 1 fails hence, $ect(F, t_u + p, |S'|) > d_{k'}$. So, according to Proposition 2, $lst(F, d_{k'}, |S'|) < t_u + p$. Since no failure has been triggered by Algorithm 4, we have also $t_u < r_{j'} \leq lst(F, d_{k'}, |S'|)$. Hence, according to Algorithm 4, t_u belongs to a forbidden region; contradiction.

□

3.3 Runtime Analysis

Note that any forbidden region is associated to a release date. So, there are no more than n forbidden regions and thus Algorithms 2 and 3 can be implemented in quadratic time. This would lead to an overall $O(n^4)$ algorithm. We can improve this to cubic time.

When a new forbidden region is created (Algorithm 4), its endpoint is smaller than or equal to the endpoints of the previously build forbidden regions. So, we can easily maintain – in constant time – the set of forbidden regions as a *list of ordered disjoint intervals* whenever a new interval is added.

Now that F is a list of at most n ordered disjoint intervals, we can incrementally compute $\min\{\theta \geq t' : \theta \notin F\}$ (Algorithms 2) since t' increases at each iteration. The same remark applies for $\min\{t' \geq t : t' \notin F\}$ (Algorithm 1) and for $\max\{\theta \leq t' : \theta \notin F\}$ (Algorithm 3). Hence the overall complexity of Algorithm 1 is $O(n^2 + |F|)$. Following the same remarks, both Algorithms 2 and 3 run in $O(q + |F|)$.

This directly ensures that Algorithm 4 runs in $O(n^3)$. Garey, Johnson, Simons, and Tarjan [12] have introduced an $O(n \log n)$ algorithm relying on a specific data structure.

4 Propagation Rules

From now on, we assume that there is a feasible schedule. We denote by F the set of forbidden regions computed by Algorithm 4. The following propositions characterize a set of time points at which jobs cannot start. Proposition 7 shows that all other time points are possible start times.

Proposition 4 (Internal Adjustment)**.** *Given two time points r, d, and an integer $0 \leq q \leq |\Delta(r,d)| - 1$, job i cannot start in*

$$I_{r,d,q} = [lst(F, d, |\Delta(r,d)| - q) + 1, ect(F, r, q+1) - 1].$$

Proof. Assume there is a feasible schedule in which a job i starts at time $t < ect(F, r, q+1)$, for some $q \in \{0, ..., |\Delta(r,d)| - 1\}$ (if $t \geq ect(F, r, q+1)$ then the job does not start in $I_{r,d,q}$). Given the definition of *ect*, at most q jobs are completed strictly before $ect(F, r, q+1)$. Hence $|\Delta(r,d)| - q$ jobs are completed after t. So $t \leq lst(F, d, |\Delta(r,d)| - q)$, *i.e.*, job i does not start in the interval $[lst(F, d, |\Delta(r,d)| - q) + 1, ect(F, r, q+1) - 1]$. □

Note that in the above proposition, we could restrict to jobs $i \in \Delta(r,d)$ but it also works for jobs $i \notin \Delta(r,d)$.

Proposition 5 (External Adjustment)**.** *Given two time points r, d and an integer $0 \leq q \leq |\Delta(r,d)| - 1$, a job $i \notin \Delta(r,d)$ cannot start in*

$$E_{r,d,q} = [lst(F, d, |\Delta(r,d)| - q + 1) + 1, ect(F, r, q+1) - 1].$$

Proof. Assume there is a feasible schedule in which a job $i \notin \Delta(r,d)$ starts at time $t < ect(F, r, q+1)$, for some $q \in \{0, ..., |\Delta(r,d)| - 1\}$. Given the definition of *ect*, at most q jobs are completed strictly before $ect(F, r, q+1)$. Hence $|\Delta(r,d)| - q$ jobs of $\Delta(r,d)$ as well as job i are completed after t. So $t \leq lst(F, d, |\Delta(r,d)| - q + 1)$, *i.e.*, job i does not start in the interval $[lst(F, d, |\Delta(r,d)| - q + 1) + 1, ect(F, r, q+1) - 1]$. □

In the following, we say that a time point $t \geq r_i$ is a *candidate starting time* for job i if it has not been discarded by internal and/or external adjustment (Propositions 4, 5). Given a candidate starting time t for job i, we note I' the instance obtained from I in which we have replaced r_i by t and d_i by $t + p$ and F' the set of forbidden intervals associated to I. So in I' the job i is fixed, and our objective is to prove that there is a feasible schedule for this instance. This claim is proven in Proposition 8 but we first need some technical propositions.

Definition 2. *The **associated deadline** of a release date r is the largest deadline d such that $lst(F, d, |\Delta(r,d)|)$ is minimal.*

Proposition 6. *If Algorithm 4 declares a forbidden region* $[lst' - p + 1, r - 1]$ *for the instance* I' *that strictly extends the forbidden region* $[lst - p + 1, r - 1]$ *computed for the instance* I *then the associated deadline of* r *for the instance* I' *is greater than or equal to* $t + p$.

Proof. We introduce a contradiction by letting r be the largest release date of instance I' such that

- its associated deadline d is strictly lower than $t + p$
- and $lst(F', d, |\Delta(r, d)|) < lst(F, d, |\Delta(r, d)|)$.

Note that if r does not exist then our proposition holds. When scheduling backward from d with the forbidden set F, at least one starting time must belong to F' but not to F otherwise $lst(F, d, |\Delta(r, d)|) = lst(F', d, |\Delta(r, d)|)$. Let then θ be the largest starting time in this backward schedule that belongs to F' but not to F. Finally, let r' be the release date that makes θ forbidden in the new instance and let d' denote the associated deadline of r'. We have $r' > r$ otherwise θ would be smaller than r and thus $lst(F, d, |\Delta(r, d)|) = lst(F', d, |\Delta(r, d)|)$. As $r' > r$, d' is greater than or equal to $t + p$ (because r is maximal). Alltoghether, we have $r < r'$ and $d < t + p \leq d'$. According to the definition of associated deadlines, we have

$$lst(F', d', |\Delta(r', d')|) \leq lst(F', d, |\Delta(r', d)|).$$

Now note that

$$lst(F', d, |\Delta(r, d)|) = lst(F', lst(F', d, |\Delta(r', d)|), q) lst(F', d', |\Delta(r, d')|) = lst(F', lst(F', d', |\Delta(r', d')|)$$

where $q = |\Delta(r, d)| - |\Delta(r', d)|$ and $q' = |\Delta(r, d')| - |\Delta(r', d')|$. As $q' \geq q$ and $lst(F', d', |\Delta(r', d')|) \leq lst(F', d, |\Delta(r', d)|)$ we have

$$lst(F', lst(F', d', |\Delta(r', d')|), q') \leq lst(F', lst(F', d, |\Delta(r', d)|), q)$$

which leads in turn to

$$lst(F', d', |\Delta(r, d')|) \leq lst(F', d, |\Delta(r, d)|).$$

This contradicts the fact that d is the associated deadline of r. □ □

In the proofs of the subsequent propositions, we use the following notation: given a release date r and deadline d,

$$\Theta_i(r, d) = \begin{cases} 0, & \text{if } r \leq r_i \leq d_i \leq d \\ 1, & \text{otherwise} \end{cases}$$

Proposition 7. *If t has not been discarded by the propagation, for any v, $lst(F,t,v) \notin F'$.*

Proof. We introduce a contradiction by letting v be the first integer value such that $lst(F,t,v) \in F'$. Let then $r > lst(F,t,v)$ be the release date that made the time point $lst(F,t,v)$ forbidden. According to Proposition 6, d, the associated deadline of r, is greater than or equal to $t+p$. So we have,

$$lst(F,t,v) < r \leq lst(F', d, |\Delta(r,d)| + \Theta_i(r,d)) < lst(F,t,v) + p$$

Now let q denote the largest integer such that $lst(F', d, q) \geq t$. Given this definition, we have

$$lst(F', d, |\Delta(r,d)| + \Theta_i(r,d)) \geq lst(F', t, |\Delta(r,d)| + \Theta_i(r,d) - q)$$

Because t has not been discarded by propagation, we immediately have

$$lst(F', t, |\Delta(r,d)| + \Theta_i(r,d) - q) \geq r$$

and thus we must have

$$|\Delta(r,d)| + \Theta_i(r,d) - q < v$$

and because of our hypothesis on v,

$$lst(F', t, |\Delta(r,d)| + \Theta_i(r,d) - q) = lst(F, t, |\Delta(r,d)| + \Theta_i(r,d) - q) < lst(F,t,v) + p$$

This contradicts the fact that the distance between two starting times in any back-schedule is at least p. □

The following proposition shows that we achieve Arc-B-Consistency on the global constraint.

Proposition 8 (Feasible Starting Times)**.** *If t has not been discarded by the propagation, there is a feasible schedule in which i starts at t.*

Proof. If the instance I' is not feasible the Algorithm 4 fails at some iteration. Let then r and d be the corresponding release date and deadline. First assume that $d < t+p$ then, when applying the back-scheduling algorithm from d, we must have at least one starting time in F' and not in F (otherwise, we would have the same lst value). By the same reasoning as in Proposition 6 we could prove that there is also a deadline $d' \geq t+p$ such that the backward schedule fails.

So now assume that $d \geq t+p$. We have

$$lst(F', d, |\Delta(r,d)| + \Theta_i(r,d)) < r.$$

As we know that i starts exactly at t, we can decompose the backward scheduling (before t and after $t+p$). And we get a better bound on the latest possible start time, *i.e.*,

$$\max_q \{lst(F', t, q) : lst(F', d, |\Delta(r,d)| + \Theta_i(r,d) - q) \geq t\} < r.$$

The back-scheduling algorithm computes exactly the same schedules before t when applied either to F or to F'. Moreover, for any v, $lst(F,t,v) \notin F'$. So, the above equation leads to

$$\max_q\{lst(F,t,q) : lst(F,d,|\Delta\,(r,d)| + \Theta_i(r,d) - q) \geq t\} < r.$$

So propagation would have detected that t is not a possible starting time. □

5 A Constraint Propagation Algorithm

For any deadline d, all intervals $I_{r,d,q}$ and $E_{r,d,q}$ end at time $ect(F,r,q+1)-1$. So we define $I_{r,q}$ as the maximum, over all d, of $I_{r,d,q}$. It is then easy to compute all intervals $I_{r,q}$ in cubic time. The situation is a bit more complex for intervals $E_{r,d,q}$: We cannot merge all these intervals since external adjustments are valid only for jobs that are not in $\Delta\,(r,d)$. To solve this issue, we iterate over jobs in non-decreasing order of deadlines. At each iteration we consider a new deadline and we add all intervals corresponding to the external adjustments associated to this deadline. This is valid since these intervals are used (Algorithm 5, line 11) to adjust release dates of jobs with a greater deadline.

The algorithm runs in cubic time. Indeed, there are $O(n^2)$ values to pre-compute and each time, this can be done in linear time. Moreover, the union of two intervals can be done in constant time and finally, we claim that the adjustment $r_k = \min\{t \geq r_k, t \notin \cup_{r,q} I_{r,q} \cup E_{r,q}\}$ can be computed in quadratic time as follows. Note that for a given (r,q), intervals $I_{r,q}$ and $E_{r,q}$ end at the same time point $ect(F,r,q+1)-1$. Hence, we can "sort" the intervals in $\cup_{r,q} I_{r,q} \cup E_{r,q}$ according to their end point. This can be done in advance (*i.e.*before entering the loops of Algorithm 5) and as there are $O(n^2)$ intervals, this precomputation can be done in $O(n^2 \log n)$. Thanks to this, we can compute in $O(n^2)$ an ordered list of disjoint intervals whose union is exactly $\cup_{r,q} I_{r,q} \cup E_{r,q}$. Then, to compute $r_k = \min\{t \geq r_k, t \notin \cup_{r,q} I_{r,q} \cup E_{r,q}\}$, we can scan the ordered list and this costs $O(n^2)$. To simplify the presentation of the algorithm, we do not explicitly define the data structure in which we store the intervals $I_{r,q}$ and $E_{r,q}$. In practice, we rely on a quadratic array indexed by jobs.

A similar algorithm can be used to adjust deadlines.

6 Experiments

Our constraint propagation algorithm has been tested on two disjunctive scheduling problems. The first one is a special case of the Job-Shop scheduling problem in which all operations on the same machine have the same processing time. The second one is a combinatorial problem from Air-Traffic Management (ATM). In both case, we briefly describe the problem and the

Algorithm 5 An $O(n^3)$ Constraint Propagation Algorithm

Pre-compute all $lst(F, d, i)$ and $ect(F, r, i)$ values
Initialize $I_{r,q}$ and $E_{r,q}$ to $\emptyset$
for all deadline d **do**
  for all release date r **do**
    for all $0 \leq q \leq |\Delta(r, d)| - 1$ **do**
      $I_{r,q} = I_{r,q} \cup [lst(F, d, |\Delta(r, d)| - q) + 1, ect(F, r, q + 1) - 1]$
    end for
  end for
end for
for all job k taken in non-decreasing order of deadlines **do**
  $r_k = \min\{t \geq r_k, t \notin \cup_{r,q}(I_{r,q} \cup E_{r,q})\}$
  for all release date r **do**
    for all $0 \leq q \leq |\Delta(r, d_k)| - 1$ **do**
      $E_{r,q} = E_{r,q} \cup [lst(F, d_k, |\Delta(r, d_k)| - q + 1) + 1, ect(F, r, q + 1) - 1]$
    end for
  end for
end for

CP model but we do not describe the branching scheme nor the heuristics used. The objective of this section is only to evaluate the efficiency of the Inter-Distance Constraint Propagation Scheme.

6.1 Job-Shop Scheduling

The Job-Shop Scheduling Problem consists of n jobs that are to be executed using m machines. Each job consists of m operations to be processed in a specified order. Each operation requires a specified machine and each machine is required by a unique activity of each job. The Job-Shop is an optimization problem. The goal is to determine a solution with minimal makespan and prove the optimality of the solution. In this paper we study a variant of the problem in which processing times of operations that require the same machine are identical. Even with this restriction, the problem is strongly NP-Hard ([13]).

We use a standard model for the Job-Shop (see [3]) where starting times of operations are integer constrained variables and the makespan is represented as an integer variable constrained to be greater than or equal to the end of any job. Arc-B-Consistency is applied on precedence constraints between operations. Machine constraints are enforced either with Edge-Finding (EF)

or with the Inter-Distance Constraint (IDC). The branching scheme and the heuristics are those provided by default in ILOG SCHEDULER, the constraint based scheduling tool of ILOG. Note that we use the edge-finding constraint propagation scheme of ILOG SCHEDULER which seems to incorporate, some of the Not-First/Not-Last propagation rules (the exat set of propagation rules used is not documented).

As for the standard Job-Shop problem, randomly generated instances are very easy to solve with EF. Among 150 random instances with up to 15 jobs and 15 machines, 34 instances requiring a significant amount of time to be solved (more than 10 seconds on Dell Latitude D600 running XP) were selected. For each instance, the two variants (EF and IDC) have been run for up to 3600 seconds. EF is able to solve 23 instances while IDC can solve 29 instances. On the average, less than 27 seconds were required by IDC to solve the 6 instances that could not be solved within one hour by EF. Among the 23 instances solved by both variants, EF requires 588999 backtracks and 249 seconds while IDC requires 249655 backtracks and 232 seconds. In term of CPU, the relatively low improvement comes, we believe, from the fact that we compare the highly optimized Edge-Finding implementation of Ilog Scheduler with a straightforward implementation of IDC.

6.2 Runway Sequencing with Holding Patterns

Spacing Aircraft to Reduce the Wake Vortex Effects

Airport arrival and departure management is a very complex problem that plays a crucial role for airports. As both the number of runways and the number of air traffic controllers are limited, the traffic has to be carefully planned to limit peaks of activity (take-off and landing) while matching as much as possible airlines landing times requirements. Good plannings allow airports to reduce delays while keeping safety standards high.

However unpredictable delays make it hardly impossible to precisely schedule aircraft in advance. So the initial planning has to be refined on line when aircraft are close enough to the airport, *i.e.*, when they reach the final descent in the TRACON (Terminal Radar Approach CONontrol). Typically, the TRACON controls aircraft approaching and departing between 5 and 50 miles of the airport. In this final scheduling process, air traffic controllers make some aircraft wait before landing. Unfortunately this "waiting" process is complex as aircraft follow predetermined routes and their speed cannot change much (speed is heavily constrained by the type of aircraft, the altitude, the weather, *etc.*). To reach some degree of flexibility in the process, two kinds of delaying procedures are used (see [6] for details):

- First, the speed of the aircraft can be slightly decreased to increase the arrival time. Second, the flight plan can be lengthened by a "Vector For Spacing": Instead of following

the shortest route between two points, the aircraft makes a turn with a small rotation.

In both cases, this allows air traffic controllers to slightly increase the time taken by an aircraft to fly from one point to another.

- "Holding Patterns" generate a constant prescribed delay for an aircraft. Several holding patterns may exist in the same TRACON (see Figure 5 for an example) and an aircraft can enter such an holding pattern several times.

Air traffic controllers usually dislike holding patterns and often try to reduce the total number of holding patterns.

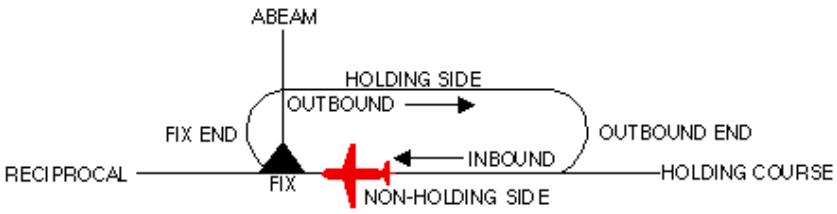

Figure 5: A simple Holding Pattern as described in pilots manuals.

Relying on the above described delaying mechanisms, a set of *time windows* in which the landing is possible can be associated to each aircraft entering the TRACON area. Each time window corresponds to a combination of holding patterns together with some vectors for spacing. As holding patterns usually induce larger delays than "Vectors For Spacing", we can assume that each interval corresponds to a single holding pattern (except the first one). Computing all these windows is out of the scope of this paper but this problem is addressed for instance in [6].

At landing time, air-traffic controllers space aircraft to reduce the wake vortex effects. Indeed, the danger of the wake turbulence is one of the most limiting factors for the take-off and landing frequency in the airports. Wake vortex effects are generally proportional to aircraft weight and the lighter the following aircraft, the more it suffers from wake vortex effects, demanding greater separation from the leading aircraft.

Problem Definition

We assume there are n aircraft in the TRACON. When entering this area, a set of s_i time intervals is assigned to each aircraft i. In the following, we assume that all these intervals are disjoint — which simplifies notation and is motivated by long holding pattern delays in real life — and that each interval corresponds to one holding pattern (*i.e.*, first interval corresponds to the case where the aircraft does not enter a holding pattern, and the $k+1$th interval corresponds to

the case where the aircraft enters k times a holding pattern). Aircraft schedulers are mainly interested in two objective functions:

1. Given a maximal number of times a plane can enter a holding pattern, maximize the minimum time elapsed between two consecutive landings.

2. Given a minimum time which can elapse between two consecutive landings, minimize the maximal number of times a plane enters a holding pattern.

We claim that the decision variants of these two problems are identical that can be stated as a single machine scheduling problem in which n "landing" jobs with identical processing time p have to be scheduled within some time windows on a single "runway" machine. This claim is obvious for the first objective. Now consider the second one and assume that each individual aircraft enters a Holding Pattern at most k times. We then remove for each aircraft i the intervals $\{k+2, ..., s_i\}$ (they correspond to situations in which aircraft i has entered the Holding Pattern more than k times) and we have a decision variant of the scheduling problem.

More formally, the decision problem can be described as follows.

THE RUNWAY SCHEDULING PROBLEM (DECISION VARIANT)

input integers $n, p, (s_1, \ldots, s_n), (r_{1,1}, d_{1,1}, \ldots, r_{1,s_1}, d_{1,s_1}), \ldots, (r_{n,1}, d_{n,1}, \ldots, r_{n,s_n}, d_{n,s_n})$.

meaning each job i has processing time p and has to be fully scheduled (*i.e.*, started and completed) in one of the intervals $[r_{iu}, d_{iu}]$. We wish to find a schedule such that every job is scheduled non-preemptively, and no two jobs overlap.

output a set of starting times $S_1, \ldots, S_n \in \mathbb{N}$ such that (1) $\forall i \in \{1, \ldots n\}$, $\exists j \in \{1, \ldots, s_i\}$, $S_i \in [r_{ij}, d_{ij} - p]$ and (2) $\forall i, k \in \{1, \ldots n\}$ with $k \neq i$, $|S_i - S_k| \geq p$.

We refer to [6], [7] and [2] for a complete description of the problem together with MIPs and Branch and Cut procedures to solve it.

We build a constraint based model as follows. For each aircraft i, and each time window $[r_{iu}, d_{iu}]$, we have a binary decision variable X_{iu} that determines whether i is scheduled in this window ($X_{iu} = 1$) or not ($X_{iu} = 0$). Of course, we have $\forall i, \forall u, \sum_u X_{iu} = 1$. We also associate a start time variable S_i for each job i. We have $\forall i, \forall u, S_i \geq r_{iu} X_{iu}$ and $\forall i, \forall u, S_i + p \leq d_{iu} X_{iu}$. The fact that jobs do not overlap in time is modeled as an Inter-Distance constraint.

To solve the problem, we look for an assignment of the X_{iu} variables and at each node of the search tree we test whether the IDC constraint is consistent or not (Section 3). This directly

ensures that, when all X_{iu} variables are bound, we have a solution to the scheduling problem. Two variants have been tested. In the first one, the machine constraint is propagated with Edge Finding (EF) while in the second one we use the Inter-Distance Constraint (IDC) propagation algorithm.

Two sets of instances have been generated. The first set of instances corresponds to "mono-pattern" problems in which all aircraft have the same number of time windows, each of them having the same size and being equally spaced. The second set of instances corresponds to the general problem. Instances with up to 90 jobs have been randomly generated (see [2] for details). For this problem, all tests were made on top of Eclair [17]. Within 1 minute of CPU time, 189 and 32 instances of the first and second set of instances could be solved with EF while with IDC, we can solve respectively 192 and 46 instances. Among instances solved by EF and IDC the number of backtracks is reduced of 60 % (first set) and 91% (second set) when using IDC. The CPU time is also decreased of 4 % and 73 %. Instances can be downloaded at http://www.lix.polytechnique.fr/~baptiste/flight_scheduling_data.zip.

7 Conclusion and Future Work

We have introduced a new global constraint, the "inter-distance constraint" that ensures that the distance between any pair of variables in some set is at least equal to a given value. We have introduced a constraint propagation algorithm that achieves arc-B-consistency on this constraint and we have shown that it allows to drastically reduce the search space on some combinatorial problems.

Our constraint propagation algorithm is more costly than the edge-finding algorithm. Although it is much more powerful and achieves the best possible bounds (arc-B-consistency). Its complexity can be reduced to $O(n^2 \log n)$ but the algorithm requires specific data structures that are not in the scope of this paper. An open question is whether the worst case complexity of the constraint propagation algorithm can be reduced to $O(n^2)$. Quimper, López-Ortiz, and Pesant [21] have solved very recently this question. Their solution relies on the same mechanism as those described in this paper (this result was not know at the time of the submission of the manuscript).

We also believe that a *generalization of this constraint to the situation where m identical parallel machines are available could be interesting.* Such a constraint would be useful for car-sequencing problems where "m/p" constraints (no more than m cars with some special feature among p consecutive ones) can be expressed in scheduling terms: Schedule identical jobs with processing time p on m parallel identical machines (each time point in the scheduling model corresponds to a slot in the sequence of cars). The global consistency of the corresponding

constraint can be achieved in polynomial time thanks to a beautiful $O(n^3 \log\log n)$ algorithm of Barbara Simons [26]. Relying on this algorithm, we have designed a polynomial time algorithm that achieves the Arc-B-Consistency of the global constraint in $O(n^5 \times \log n \times \log\log n)$. The complexity is much too high for any practical purpose and we are currently looking for slightly less efficient constraint propagation rules that would lead to fast constraint propagation algorithms.

References

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

[2] K. Artiouchine, Ph.Baptiste and C.Dürr. Runway Sequencing with Holding Patterns. *http://www.lix.polytechnique.fr/Labo/Konstantin.Artiouchine/ejor04.pdf*

[3] Ph. Baptiste, C. Le Pape, and W. Nuijten. *Constraint-based Scheduling*. Kluwer Academic Publishers, 2001.

[4] Ph. Baptiste and C. Le Pape. Edge-Finding Constraint Propagation Algorithms for Disjunctive and Cumulative Scheduling. *Proc. 15th Workshop of the UK Planning Special Interest Group*, 1996.

[5] Ph. Baptiste, J. Carlier, A. Kononov, M. Queyranne, S. Sevastianov and Maxim Sviridenko. Structural Properties of Preemptive Schedules. Submitted.

[6] A. M. Bayen and C. J. Tomlin, Real-time discrete control law synthesis for hybrid systems using MILP: Application to congested airspace, Proceedings of the *American Control Conference*, 2003.

[7] A. M. Bayen, C. J. Tomlin, Y. Ye, J. Zhang, MILP formulation and polynomial time algorithm for an aircraft scheduling problem, *cherokee.stanford.edu/˜bayen/publications.html*.

[8] 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.

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

[10] U. Dorndorf, E. Pesch and T. Phan-Huy. Solving the Open Shop Scheduling Problem. *Journal of Scheduling*, 4, 157–174, 2001.

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

[12] M.R. Garey, D.S. Johnson, B.B. Simons, and R.E. Tarjan. Scheduling unit-time tasks with arbitrary release times and deadlines. *SIAM Journal on Computing*, 10(2), 256–269, 1981.

[13] J.K. Lenstra and A.H.G. Rinnooy Kan. Computational complexity of discrete optimization problems. *Ann. Discrete Math.*, 4:121-140, (1979).

[14] O. Lhomme. Consistency Techniques for numeric CSPs. *Proceedings of the thirteenth International Joint Conference on Artificial Intelligence*, Chambéry, France, (1993).

[15] A. López-Ortiz, C.-G. Quimper, J. Tromp, and P. van Beek. A fast and simple algorithm for bounds consistency of the alldifferent constraint. Proceedings of the 18th *International Joint Conference on Artificial Intelligence*, Acapulco, Mexico, 2003.

[16] P. D. Martin and D. B. Shmoys. A New Approach to Computing Optimal Schedules for the Job-Shop Scheduling Problem. *Proc. 5th Conference on Integer Programming and Combinatorial Optimization*, 1996.

[17] N. Museux, L. Jeannun, P. Savéant, F. Le Huédé, F.-X. Josset and J. Mattioli. Claire/Eclair : Un environnement de modélisation et de résolution pour des applications d'optimisations combinatoires émbarquées, *Journées Francophones de Programmation en Logique et de programmation par Contraintes*, 2003.

[18] W. Nuijten, T. Bousonville, F. Focacci, D. Godard and C. Le Pape. Towards an industrial Manufacturing Scheduling Problem and Test Bed, Proc. of the 9th *International Workshop on Project Management and Scheduling*, 2004.

[19] E. Pinson. *Le problème de Job-Shop*. Thèse de l'Université Paris VI, 1988.

[20] J.-F. Puget. A fast algorithm for the bound consistency of all-diff constraints. *Proc. 15th National Conference on Artificial Intelligence*, 1998.

[21] Claude-Guy Quimper and Alejandro López-Ortiz and Gilles Pesant. A Quadratic Propagator for the Inter-Distance Constraint. *Proc. of the Twenty-First National Conference on Artificial Intelligence. Menlo Park, Calif.: AAAI Press.*, 2006.

[22] J.-C. Régin. A filtering algorithm for constraints of difference in CSPs. *Proc. 12th National Conference on Artificial Intelligence*, 1994.

[23] J.-C. Régin. The global minimum distance constraint. Technical report, ILOG, 1997.

[24] J.-C. Régin and J.-F. Puget. A filtering algorithm for global sequencing constraints. Proceedings of the Third International Conference on Principles and Practice of Constraint Programming, 1997.

[25] B. Simons. A fast algorithm for single processor scheduling. *19th Annual Symposium on the Foundations of Computer Science*, pages 246–252, October 1978.

[26] B. Simons. Multiprocessor Scheduling of Unit-Time Jobs with Arbitrary Release Times and Deadlines. *SIAM J. Comput.*, 12(2), 294–299, 1983.

[27] Ph. Torres and P. Lopez. On Not-First/Not-Last conditions in disjunctive scheduling. *European Journal of Operational Research*, 127:332–343, 2000.

[28] P. Vilím. $o(n \log n)$ filtering algorithms for unary resource constraint. In Jean-Charles Régin and Michel Rueher, editors, *Proceedings of CP-AI-OR*, volume 3011 of *LNCS*, pages 335–347. Springer, 2004.