Inter-Distance Constraint: An Extension of the All-Different Constraint for Scheduling Equal Length Jobs

Konstantin Artiouchine[1,2] and Philippe Baptiste[1]
Konstantin.Artiouchine@polytechnique.org,
Philippe.Baptiste@polytechnique.fr

[1] CNRS LIX, Ecole Polytechnique, 91128 Palaiseau, France
[2] Thales TRT, Domaine de Corbeville, 91404 Orsay CEDEX, France

Abstract. We study a global constraint, the "inter-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 domains of the variables 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 industrial problem are reported.
Keywords: Global Constraint, Scheduling, Constraint Propagation.

1 Introduction

We introduce 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.*, $\forall i, j, |S_i - S_j| \geq p$. To our knowledge, this constraint has been introduced for the first time by Régin [20]. When p is 1, the inter-distance constraint reduces to the well-known all-different constraint [19], [18], [13].

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 CONtrol" 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 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 [16]).

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 algorithms, 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 [12]) 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 [20] relies on the sequencing constraint [21] 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 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. Then we show that they can be improved when all processing times are equal.

2.1 Edge-Finding

The edge-finding algorithm [7], [8], [14] is one of the most well known OR 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 [7]. Another variant of the Edge-Finding technique is presented in [8]. Its time somplexity is $O(n \log n)$ but it requires much 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 [17]:

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

A quadratic algorithm is described in [4]. Alternative approaches can be found in [23], [9], [24].

2.3 Missed Deductions

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). What happens when processing times are equal ? 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) of a job is drawn as white rectangle. The job itself (black rectangle) has to be scheduled inside this window. There is no feasible schedule (the machine is occupied from 3 to 9 because of the last three jobs, hence 3 jobs have to

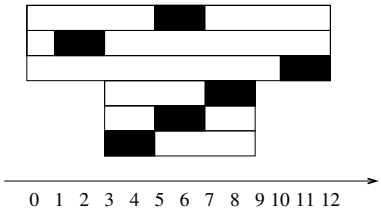

Fig. 1. Edge-Finding and Not-First/Not-Last do not detect all inconsistencies

be scheduled in two disjoint time windows of 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 5 have to be scheduled on a single machine. There is a feasible schedule but neither edge-finding nor Not-First/Not-Last deduce that the third job cannot start earlier than 10.

Fig. 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 [10] 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 [10]. 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 the 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 [7] (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, Johnson, Simons and Tarjan pro-

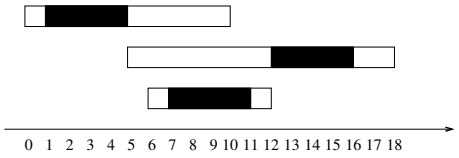

Fig. 3. EDD fails while there is a feasible schedule.

pose to modify the EDD rule to ensure that it builds a feasible schedule, if one exists. 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: $t := \min_{i \in U} r_i$
3: **while** $U \neq \emptyset$ **do**
4: $\quad t := \min(t, \min_{i \in U} r_i)$
5: $\quad$ **if** $t \in F$ **then**
6: $\quad\quad t := \min\{t' \geq t : t' \notin F\}$
7: $\quad$ Let k be the job with smallest deadline s.t. $r_k \leq t$
8: $\quad$ Start job k at time t, $U := U - \{k\}$, $t := t + p$

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* of the largest time point such that there is a feasible schedule of the jobs in X that is idle before *lst* and in which no job starts in F (*lst* stands for latest start time). 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*. 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.

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

Definition 1. *Given a time point t, an integer q and a set of forbidden regions 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 $lst(F,t,q)$. The proof of correctness is easy to make by induction on q and is skipped.

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

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

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

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

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.

Algorithm 4 Forbidden Regions

1: $F := \emptyset$
2: **for all** release date r taken in non-increasing order **do**
3: $lst := \infty$
4: **for all** deadline d taken in non-increasing order **do**
5: $lst := \min(lst, lst(F,d,|\Delta(r,d)|))$
6: **if** $lst < r$ **then**
7: There is no feasible schedule
8: **else if** $r \leq lst < r + p$ **then**
9: $F := F \cup [lst - p + 1, r - 1]$

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

Lemma 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 $lst(F, t_2, q) \geq t_1$. □

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

Proof. We claim that, given the set of forbidden regions F computed by 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 scheduled 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. 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 computed by the forward scheduling algorithm when applied to the same set of jobs with the forbidden list built while processing the release date r. Hence $ect(F, r_{j'}, |\Delta(r_{j'}, d_{k'})|) > d_{k'}$. So, according to Lemma 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 than u at time r_u). As before, let j' denote the job in S' with minimal release date. So S' is a subset of $\Delta(r_{j'}, d_{k'})$.
  Algorithm 1 fails, hence, $ect(F, r_u + p, |S'|) > d_{k'}$. So, according to Lemma 2, $lst(F, d_{k'}, |S'|) < r_u + p$. Since no failure has been triggered, we have also $r_u < r_{j'} \leq lst(F, d_{k'}, |S'|)$. Hence, r_u must belong to a forbidden region declared by Algorithm 4, but it is not the case. □

3.3 Runtime Analysis

It is easy to see that there are no more than n forbidden regions hence 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 as follows:

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. Finally, note that forbidden regions end right before a release date. Hence, it is easy to maintain the set F in such a way that we have no more than n forbidden regions throughout the algorithm.

Moreover, we can incrementally compute $\min\{t' \geq t : t' \notin F\}$ since t increases at each iteration. Moreover, inside the loop, all operations can be done in linear time except $\min\{t' \geq t : t' \notin F\}$. Hence the overall complexity of Algorithm 1 is $O(n^2 + |F|)$ since overall time needed for the computations of $\min_{i \in U} r_i$ is linear if all jobs are sorted in increasing order of release dates. Following the same remarks, both Algorithms 2 and 3 run in $O(q + |F|)$.

This directly ensures that Algorithm 4 runs in $O(n^3)$.

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 lemmas characterize a set of time points at which jobs cannot start. Lemma 7 shows that all other time points are possible starting times.

Lemma 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 lemma, we could restrict to jobs $i \in \Delta(r,d)$ but it also works for jobs $i \notin \Delta(r,d)$.

Lemma 5 (External Adjustment). *Given two time points r, d and an integer $0 \leq q \leq |\Delta(r,d)|$, 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 (Lemmas 4, 5). Given a candidate starting time t for job i, we note I' the the instance obtained from I in which we have replaced r_i by t and d_i by $t + p$. 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 Lemma 8 but we first need some technical lemmas.

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.*

Lemma 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. Let r be the largest release date of instance I' such that

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

Note that if r does not exist then our lemma 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 t be the largest starting time in this backward schedule that belongs to F' but not to F and let $r' > r$ be the release date that makes t forbidden in the new instance. As $r' > r$, we know that the associated deadline d' of r' is greater than or equal to $t + p$. When back-scheduling $|\Delta(r', d)|$ jobs from d using F', no starting time belongs to F' (recall that t is maximal). Hence, the corresponding latest start time is larger than $t + p$ (on the back-schedule) otherwise $t \in F$. So we have

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

Now note that $lst(F', d', |\Delta(r, d')|) = lst(F', lst(F', d', |\Delta(r', d')|), q)$ where q is exactly $|\{i : r \leq r_i < r', d_i \leq d\}|$, or similarly,

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

As the function $h \rightarrow lst(F', h, q)$ is non decreasing, we have

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

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

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

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

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

Proof. Let 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 Lemma 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 lemma shows that we achieve Arc-B-Consistency on the global constraint.

Lemma 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 Lemma 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 $E_{r,d,q}$ are completed at the same 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 $E_{r,d,q}$ as we cannot merge all these intervals since external adjustments are only valid for jobs that are not in $\Delta(r, d)$. To solve this issue, we consider jobs in non-decreasing order of deadlines and we add, at each iteration, all intervals corresponding to external adjustments associated to this deadline. This is valid since these intervals are used (Algorithm 5) to adjust release dates of jobs with a greater deadline.

The algorithm runs in cubic time. Indeed, there are $O(n^2)$ values to precompute and each time, this can be done in linear time. Moreover, the union of two intervals can be done in constant time and finally, 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 since there are $O(n^2)$ intervals to consider. 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 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.

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

Precompute 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 $q \leq |\Delta(r, d)| - 1$ **do**
      $I_{r,q} = I_{r,q} \cup [lst(F, d, |\Delta(r, d)| - q) + 1, ect(F, r, q + 1) - 1]$
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 $q \leq |\Delta(r, d_k)|$ **do**
      $E_{r,q} = E_{r,q} \cup [lst(F, d_k, |\Delta(r, d_k)| - q + 1) + 1, ect(F, r, q + 1) - 1]$

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 ([11]).

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.

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

We study a scheduling problem that occurs when aircraft reach the final descent in the "Terminal Radar Approach CONtrol" area (TRACON) of an airport with a single runway. When entering the TRACON, a set of disjoint time windows in which the landing is possible, can be automatically assigned to each aircraft. Roughly speaking, the distance between two consecutive windows corresponds to a waiting procedure known as a "Holding Pattern". The objective is then to determine landing times, within these time windows, which maximize the minimum time elapsed between consecutive landings. More formally, the decision variant of this problem can be described as follows.

THE RUNWAY SCHEDULING PROBLEM

Input integers $n, p, (s_1, \ldots, s_n)$, $(r_1^1, d_1^1, \ldots, r_1^{s_1}, d_1^{s_1})$, …, $(r_n^1, d_n^1, \ldots, r_n^{s_n}, d_n^{s_n})$.
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) $\forall i \in \{1, \ldots n\}$, $\exists j \in \{1, \ldots, s_i\}$ such that $S_i \in [r_{ij}, d_{ij} - p]$ and (2) $\forall i, k \in \{1, \ldots n\}$ with $k \neq i$, $|S_i - S_k| \geq p$.

This problem is NP-Complete in the strong sense [2]. We refer to [5], [6] 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, we have a decision variable P_i that determines whether i is scheduled in its j-th time window $[r_{ij}, d_{ij}]$, $(P_i = j)$ or not $(P_i \neq j)$. We also associate a start time variable S_i for each job $i : P_i \geq j \iff r_j \leq S_i$ and $P_i \leq j \iff d_j \geq S_i + p$.

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 P_i 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 P_i 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 (instances can be downloaded at http://www.lix.polytechnique.fr/~baptiste/flight_scheduling_data.zip). 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©[15]. 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 %.

7 Conclusion

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). 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 $O(n^2)$.

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 immediately useful for car-sequencing problems where "a/b" constraints (no more than a cars with some special feature among b consecutive ones) can be expressed in scheduling terms: Schedule identical jobs with processing time b on a 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 algorithm of Simons [22]. However, no specific constraint propagation algorithm is known.

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. 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.
6. 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.*

7. 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.
8. J. Carlier and E. Pinson. Adjustment of Heads and Tails for the Job-Shop Problem. *European Journal of Operational Research*, 78:146–161, 1994.
9. U. Dorndorf, E. Pesch and T. Phan-Huy. Solving the Open Shop Scheduling Problem. *Journal of Scheduling*, 4, 157–174, 2001.
10. 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.
11. J.K. Lenstra and A.H.G. Rinnooy Kan. Computational complexity of discrete optimization problems. *Ann. Discrete Math.*, 4:121-140, (1979).
12. O. Lhomme. Consistency Techniques for numeric CSPs. *Proceedings of the thirteenth International Joint Conference on Artificial Intelligence*, Chambéry, France, (1993).
13. A. Lopez-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.
14. 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.
15. N. Museux, L. Jeannin, 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.
16. 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.
17. E. Pinson. *Le problème de Job-Shop*. Thèse de l'Université Paris VI, 1988.
18. J.-F. Puget. A fast algorithm for the bound consistency of all-diff constraints. *Proc. 15th National Conference on Artificial Intelligence*, 1998.
19. J.-C. Régin. A filtering algorithm for constraints of difference in CSPs. *Proc. 12th National Conference on Artificial Intelligence*, 1994.
20. J.-C. Régin. The global minimum distance constraint. Technical report, ILOG, 1997.
21. 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.
22. B. Simons. A fast algorithm for single processor scheduling. *19th Annual Symposium on the Foundations of Computer Science*, pages 246–252, October 1978.
23. Ph. Torres and P. Lopez. On Not-First/Not-Last conditions in disjunctive scheduling. *European Journal of Operational Research*, 127:332–343, 2000.
24. 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.