A Quadratic Propagator for the Inter-Distance Constraint

Claude-Guy Quimper
University of Waterloo
School of Computer Science
cquimper@uwaterloo.ca

Alejandro López-Ortiz
University of Waterloo
School of Computer Science
alopez-o@uwaterloo.ca

Gilles Pesant
École Polytechnique de Montréal
Department of Computer Engineering
pesant@crt.umontreal.ca

Abstract

We present a new propagator achieving bound consistency for the INTER-DISTANCE constraint. This constraint ensures that, among a set of variables $X_1, \ldots, X_n$, the difference between two variables is at least p. This restriction models, in particular, scheduling problems in which tasks require p contiguous units of a resource to be completed. Until now, the best known propagator for bound consistency had time complexity $O(n^3)$. In this work we propose a quadratic propagator for the same level of consistency. We then show that this theoretical gain gives savings of an order of magnitude in our benchmark of scheduling problems.

Introduction

The *cumulative scheduling problem* with one resource of capacity C consists of a set of tasks $T_1, \ldots, T_n$ to which we associate four integer variables: a release time r_i, a deadline d_i, a processing time p_i and a capacity requirement c_i. Each task T_i must start at time t_i such that $r_i \leq t_i \leq d_i - p_i$. Let $\Omega(t)$ be the set of tasks in process at time t, i.e. the tasks T_i such that $t_i \leq t \leq t_i + p_i$. We have the resource capacity constraint $\sum_{T_i \in \Omega(t)} c_i \leq C$. This problem is NP-Hard even in the case where $C = 1$ which we call, in this particular case, the *disjunctive scheduling problem*.

Edge finders (Carlier & Pinson 1994; Mercier & Van Hentenryck 2005) have largely been used to solve scheduling problems. This technique reduces the intervals $[r_i, d_i]$ by detecting time zones that must be allocated to a subset of the tasks making these zones unavailable for other tasks. The goal is to increase release times and reduce deadlines without eliminating any feasible solution. The problem is said to be *bound consistent* when intervals $[r_i, d_i]$ have been fully shrunk, i.e. when there exists at least one feasible schedule in which task T_i starts on time r_i and at least one feasible schedule in which task T_i finishes on time d_i. It is NP-Hard to make a scheduling problem bound consistent, even in the disjunctive case. For this reason, edge finders try to reduce, in polynomial time, the size of the intervals without necessarily achieving bound consistency. A backtracking search assigns starting times to tasks and uses the edge finder to reduce the size of the search tree.

We study the disjunctive scheduling problem when all tasks have the same processing time $p_i = p$. This problem can be solved in polynomial time (Garey *et al.* 1981) but traditional algorithms only return one solution that generally does not satisfy the side constraints. These side constraints can even make the problem NP-Hard. The flexibility that constraint programming offers to encode such problems is an asset. A single INTER-DISTANCE constraint can encode the disjunctive scheduling problem. This constraint ensures that starting times are pairwise distant by at least p units of time. The global constraint offers a stronger pruning than the $O(n^2)$ associated binary constraints $|X_i - X_j| \geq p$.

Artiouchine and Baptiste (Artiouchine & Baptiste 2005) recently proposed an $O(n^3)$ propagator that enforces bound consistency on the INTER-DISTANCE constraint. By achieving bound consistency, their propagator prunes the search space better than edge finding algorithms resulting in smaller choice points in the backtracking search and a better time performance. We propose in this work a quadratic propagator that is faster both in theory and in practice, even for small instances.

Throughout the paper, we will consider the set $[a, b]$ as the interval of integer values between a and b inclusively. If $a > b$, then the interval $[a, b]$ is empty. We nevertheless say that the lower bound of the interval is $\min([a, b]) = a$ and the upper bound is $\max([a, b]) = b$ as for non-empty intervals.

We first present some notions about how bound consistency can be enforced on the INTER-DISTANCE constraint. We then explain how the computation can be simplified. Based on this simplification, we present our algorithm and a data structure that ensures the quadratic behaviour of our propagator. Finally, we present some experiments proving the efficiency of our propagator.

The INTER-DISTANCE Constraint

Régin (Régin 1997) first introduced the INTER-DISTANCE constraint. The expression INTER-DISTANCE$([X_1, \ldots, X_n], p)$ holds if and only if $|X_i - X_j| \geq p$ for all $i \neq j$. When $p = 1$, the INTER-DISTANCE specializes into an ALL-DIFFERENT constraint (Régin 1994; Mehlhorn & Thiel 2000; López-Ortiz *et al.* 2003). Régin (Régin 1994) showed that a single global constraint, in many cases, causes

more domain reduction than the $\frac{n(n-1)}{2}$ equivalent binary constraints. This observation also applies to the INTER-DISTANCE constraint which is the general case. Artiouchine and Baptiste provided the first propagator for the bound consistency of the INTER-DISTANCE constraint. The running time complexity of this propagator is cubic.

We use the following problem as a running example.

Example 1 *Consider a problem with $n = 3$ tasks T_1, T_2, and T_3 with processing time $p = 6$ and the following release times and deadlines.*

$$r_1 = 2 \qquad r_2 = 10 \qquad r_3 = 4$$
$$d_1 = 12 \qquad d_2 = 20 \qquad d_3 = 21$$

After propagation of the constraint INTER-DISTANCE$([T_1, T_2, T_3], p)$*, we obtain the following release times and deadlines.*

$$r_1 = 2 \qquad r_2 = 14 \qquad r_3 = 8$$
$$d_1 = 8 \qquad d_2 = 20 \qquad d_3 = 14$$

Here, task T_1 must finish before or at time 8 in order to allow tasks T_2 and T_3 to meet their deadlines. Task T_2 cannot start before time 14 since the two other tasks are not completed before this time. Finally, task T_3 must be executed between tasks T_1 and T_2 forcing its release time to be increased and its deadline to be reduced.

Garey et al. (Garey *et al.* 1981) designed an algorithm that finds a solution satisfying the INTER-DISTANCE constraint in $O(n \log n)$ steps. Their algorithm proceeds in two phases. In the first phase, the algorithm computes a set of regions F in which no tasks are allowed to start. We call these regions the *forbidden regions*. Their number is bounded by n, the number of tasks. Once these forbidden regions are computed, the second phase uses a greedy strategy to schedule the tasks.

Artiouchine, Baptiste and Garey et al. use two basic functions as main pillars of their algorithm. Let $ect(F, r, q)$ be the *earliest completion time* of a schedule of q tasks starting at or after time r without ever starting in a forbidden region contained the set of forbidden regions F. Symmetrically, let $lst(F, d, q)$ be the *latest starting time* of a schedule of q tasks finishing at or before time d without ever starting in a forbidden region in F. Computing $ect(F, r, q)$ and $lst(F, d, q)$ can be done in $O(q)$ steps using the following recurrences where $ect(F, r, 0) = r$ and $lst(F, d, 0) = d$.

$$ect(F, r, q) = \min\{t \notin F \mid t \geq ect(F, r, q-1)\} + p$$
$$lst(F, d, q) = \max\{t \notin F \mid t \leq lst(F, d, q-1) - p\}$$

Using these two functions, Artiouchine and Baptiste describe two types of adjustment intervals necessary and sufficient to maintain bound consistency on the INTER-DISTANCE constraint. We first define $\Delta(r, d)$ to be the set of tasks whose release times and deadlines are contained in the interval $[r, d]$. An *internal adjustment interval* is an interval in which no task is allowed to start. The set of internal adjustment intervals is a superset of the forbidden regions F. Theorem 1 formally defines the internal adjustment intervals.

Theorem 1 ((Artiouchine & Baptiste 2005)) *Given two time points r, d, and an integer $0 \leq q < |\Delta(r, d)|$, no task can start in the interval $I_{r,d,q}$ with endpoints $lst(F, d, q+1) + 1$ and $ect(F, r, |\Delta(r, d)| - q) - 1$.*

The *external adjustment intervals* are intervals in which a subset of the tasks are not allowed to start. They are formally defined in Theorem 2.

Theorem 2 ((Artiouchine & Baptiste 2005)) *Given two time points r, d and an integer $0 \leq q < |\Delta(r, d)|$, a task $i \notin \Delta(r, d)$ cannot start in the interval $E_{r,d,q}$ with endpoints $lst(F, d, q+2) + 1$ and $ect(F, r, |\Delta(r, d)| - q) - 1$.*

Table 1 shows the internal and external adjustment intervals from Example 1.

Internal Adjustment Intervals

$r_i \backslash d_j$	12	20	21
2	$\{[7, 7]\}$	$\{[9, 7], [15, 13]\}$	$\{[3, 7], [9, 13], [16, 19]\}$
4	$\emptyset$	$\{[15, 9]\}$	$\{[9, 9], [16, 15]\}$
10	$\emptyset$	$\{[15, 15]\}$	$\{[16, 15]\}$

External Adjustment Intervals

$r_i \backslash d_j$	12	20	21
2	$\{[-3, 7]\}$	$\{[3, 7], [9, 13]\}$	$\{[-3, 7], [3, 13], [9, 19]\}$
4	$\emptyset$	$\{[9, 9]\}$	$\{[3, 9], [9, 15]\}$
10	$\emptyset$	$\{[9, 15]\}$	$\{[9, 15]\}$

Table 1: Internal and external adjustment intervals generated by a pair of time points (r_i, d_j) from Example 1. Intervals are written in decreasing order with respect to parameter q. The forbidden regions are $F = \{[-3, 1], [3, 3], [9, 9]\}$.

Artiouchine and Baptiste formally proved that the internal and external adjustment intervals are necessary and sufficient to enforce bound consistency on the INTER-DISTANCE constraint.

Towards a Quadratic Propagator

Internal and external adjustment intervals in the worst case may be computed with up to n possible release times r, n possible deadlines d and produce $O(n)$ adjustment intervals each. Therefore, $O(n^3)$ adjustment intervals could be checked in the worst case, hence the cubic time complexity of the Artiouchine-Baptiste propagator.

In reality, the union of all internal and external adjustment intervals consists of a maximum of $O(n^2)$ disjoint intervals. It is therefore possible to ignore intervals that are subsets of already discovered intervals in order to achieve a quadratic complexity. To avoid computing redundant adjustment intervals, we introduce the notion of dominance between two pairs of time points. When a pair of time points dominates another pair, the adjustment regions of the dominant pair contain some adjustment regions of the other pair.

Definition 1 (Dominance) *A pair of time points (r_i, d_j) dominates a pair of time points (r_k, d_l) if we have $\min(I_{r_i,d_j,q}) \leq \min(I_{r_k,d_l,q})$ and $\max(I_{r_i,d_j,q}) \geq \max(I_{r_k,d_l,q})$ for all $0 \leq q < \min(|\Delta(r_i, d_j)|, |\Delta(r_k, d_l)|)$. We write $(r_i, d_j) \succ (r_k, d_l)$.*

Notice that we usually have $|\Delta(r_i, d_j)| \neq |\Delta(r_k, d_l)|$. The definition of dominance only applies for q below $\min(|\Delta(r_i, d_j)|, |\Delta(r_k, dl)|)$. Also, for a fixed deadline d, the dominance operator ($\prec$) is transitive, i.e. if $(r_i, d) \prec (r_j, d)$ and $(r_j, d) \prec (r_k, d)$ hold, then $(r_i, d) \prec (r_k, d)$ holds. In Example 1 we have $(2, 21) \succ (4, 21)$.

The following lemmas describe a property of the ect and lst functions that will allow us to efficiently decide if a pair of time points dominates another one.

Lemma 1 *If* $ect(F, r_i, q_1) \leq ect(F, r_j, q_2)$ *then* $ect(F, r_i, q_1 + k) \leq ect(F, r_j, q_2 + k)$ *for any* $k \geq 0$.

Proof: The proof is by induction on k. The base case $k = 0$ is trivial. Suppose that the lemma holds for $k - 1$. We have $ect(F, r_i, q_1 + k) = ect(F, r_i, q_1 + k - 1) + p + s_i$ where s_i is a (potentially null) shift caused by the (potentially empty) forbidden region $F_i = [ect(F, r_i, q_1 + k - 1), ect(F, r_i, q_1 + k - 1) + s_i] \subseteq F$. Similarly we have $ect(F, r_j, q_2 + k) = ect(F, r_j, q_2 + k - 1) + p + s_j$ where s_j is the shift caused by the forbidden region $F_j = [ect(F, r_j, q_2+k-1), ect(F, r_j, q_2+k-1)+s_j] \subseteq F$. If s_i is large enough to obtain $ect(F, r_i, q_1+k) \geq ect(F, r_j, q_2+k)$, then we have $F_j \subseteq F_i$. Since both forbidden regions intersect, both functions are shifted to the same value and we obtain $ect(F, r_i, q_1 + k) = ect(F, r_j, q_2 + k)$ which completes the induction step. □

Lemma 2 *If* $lst(F, d_i, q_1) \leq lst(F, d_j, q_2)$ *then* $lst(F, d_i, q_1 + k) \leq lst(F, d_j, q_2 + k)$ *for any* $k \geq 0$.

Proof: Symmetric to the proof of Lemma 1. □

We now describe three different situations in which a pair of time points dominates another one. The first case is described in Lemma 3.

Lemma 3 *Consider the pairs of time points* (r, d_i) *and* (r, d_j) *such that* $d_i < d_j$. *If* $k = |\Delta(r, d_i)| = |\Delta(r, d_j)|$ *then* $(r, d_i) \succ (r, d_j)$.

Proof: We have $lst(F, d_i, 0) < lst(F, d_j, 0)$ and by Lemma 2 $lst(F, d_i, q + 1) \leq lst(F, d_j, q + 1)$. This implies $\min(I_{r,d_i,q}) \leq \min(I_{r,d_j,q})$ for any $0 \leq q < k$ and since we have $\max(I_{r,d_i,q}) = \max(I_{r,d_j,q})$ we have $(r, d_i) \succ (r, d_j)$. □

From Lemma 3 we conclude that $(10, 20) \succ (10, 21)$ in Example 1. Similarly, we have the following Lemma.

Lemma 4 *Consider the pairs of time points* (r_i, d) *and* (r_j, d) *such that* $r_i < r_j$ *and* $k = |\Delta(r_i, d)| = |\Delta(r_j, d)|$. *Then* $(r_i, d) \prec (r_j, d)$.

Proof: We have $ect(F, r_i, 0) < ect(F, r_j, 0)$ and by Lemma 1 $ect(F, r_i, k - q) \leq lst(F, r_j, k - q)$. This implies $\max(I_{r_i,d,q}) \leq \max(I_{r_j,d,q})$ for any $0 \leq q < k$ and since we have $\min(I_{r_i,d,q}) = \min(I_{r_j,d,q})$ we have $(r_i, d) \prec (r_j, d)$. □

In Example 1, Lemma 4 detects $(4, 20) \prec (10, 20)$. We show a last case where a pair of time points dominates another one.

Lemma 5 *Let* (r_i, d) *and* (r_j, d) *be two pairs of time points such that* $|\Delta(r_i, d)| = |\Delta(r_j, d)| + k$ *and* $ect(F, r_i, k) \leq ect(F, r_j, 0)$. *Then* $(r_j, d) \succ (r_i, d)$.

Proof: Clearly, for $0 \leq q < |\Delta(r_j, d)|$, the internal adjustment intervals $I_{r_i,d,q}$ and $I_{r_j,d,q}$ share the same lower bound. For the upper bounds, we have the following:

$$\begin{aligned}\max(I_{r_i,d,q}) &= ect(F, r_i, |\Delta(r_i, d)| - q) - 1 \\ &= ect(F, r_i, |\Delta(r_j, d)| + k - q)) - 1\end{aligned}$$

and by Lemma 1

$$\begin{aligned}&\leq ect(F, r_j, |\Delta(r_j, d)| - q) - 1 \\ &\leq \max(I_{r_j,d,q})\end{aligned}$$

Therefore we have $(r_j, d) \succ (r_i, d)$. □

In Example 1, we have $(10, 20) \succ (2, 20)$ from Lemma 5.

Lemma 5 is crucial to obtaining a quadratic algorithm. Consider a deadline d and a sequence of release times $r_1 < r_2 < \ldots < r_k$ such that $(r_1, d) \prec (r_2, d) \prec \ldots \prec (r_k, d)$. There can be up to $O(n^2)$ internal adjustment intervals associated to these pairs of time points. Nevertheless, the union of all $O(n^2)$ intervals can be given by the union of only $O(n)$ intervals. Given two integers j and q such that $1 \leq j \leq k$ and $0 \leq q < |\Delta(r_j, d)|$, we first notice that the following intervals all share a same lower bound. The union of the intervals is therefore equal to the interval whose upper bound is the greatest.

$$\begin{aligned}\bigcup_{i=1}^{j} I_{r_i,d,q} &= [\min(I_{r_j,d,q}), \max_{1 \leq i \leq j} \max(I_{r_i,d,q})] \\ &= [\min(I_{r_j,d,q}), \max(I_{r_j,d,q})] \\ &= I_{r_j,d,q}\end{aligned}$$

We see that up to $O(n)$ adjustment intervals can be contained in a single interval. Using this observation, we compute the union of all adjustment intervals formed by the pairs $(r_1, d), \ldots, (r_k, d)$ using the following equation. To simplify the notation, we let $|\Delta(r_{k+1}, d)| = 0$ since r_{k+1} is undefined.

$$\bigcup_{i=1}^{k} \bigcup_{q=0}^{|\Delta(r_i,d)|-1} I_{r_i,d,q} = \bigcup_{i=1}^{k} \bigcup_{q=|\Delta(r_{i+1},d)|}^{|\Delta(r_i,d)|-1} I_{r_i,d,q} \quad (1)$$

Notice that the left hand side of Equation 1 has $O(n^2)$ intervals while the right hand side has only $O(n)$ intervals. Indeed, the number of intervals to union is given by $\sum_{i=1}^{k}(|\Delta(r_i, d)| - |\Delta(r_{i+1}, d)|)$. The telescopic series simplifies to $|\Delta(r_1, d)| - |\Delta(r_{k+1}, d)| = |\Delta(r_1, d)|$. In Example 1 since we have $(2, 20) \prec (10, 20)$ we obtain the following:

$$\begin{aligned}(I_{2,20,0} \cup I_{2,20,1}) \cup (I_{10,20,0}) &= I_{2,20,1} \cup I_{10,20,0} \\ &= [9, 7] \cup [15, 15] \\ &= [15, 15]\end{aligned}$$

A Quadratic Propagator

General Scheme

The idea behind the algorithm is the following. We process each deadline in increasing order. If two deadlines d_i and d_j are equal and their associated release times satisfy $r_j \leq r_i$, we process both deadlines at the same time but use d_i as a reference. For every deadline d_i, we compute the longest sequence of release times $r_{x_1} < r_{x_2} < \ldots < r_{x_k}$ such that $(r_{x_1}, d_i) \prec (r_{x_2}, d_i) \prec \ldots \prec (r_{x_k}, d_i)$. Using this sequence and Equation 1, we compute the union of all internal adjustment intervals generated by the pairs of time points whose deadline is d_i. To build the sequence, we iterate through all release times in non-decreasing order. Two cases can occur where we can safely skip a release time r_j.

Case 1 ($d_j > d_i$): Suppose that the deadline d_j associated to r_j has not been processed yet, i.e. $d_j > d_i$. For such a release time r_j, two cases can occur. We choose the smallest release time r_k whose deadline has already been processed and such that $r_k > r_j$. If such a r_k exists, then $|\Delta(r_j, d_i)| = |\Delta(r_k, d_i)|$ and Lemma 4 insures that $(r_k, d_i) \succ (r_j, d_i)$. All adjustment intervals from (r_j, d_i) will be taken into account when iterating through r_k. If no such r_k exists, then we have $\Delta(r_j, d_i) = \emptyset$ and no adjustment intervals are associated to the pair (r_j, d_i). In either case, the pair (r_j, d_i) can be ignored.

Case 2 ($r_j > r_i$): A release time r_j greater than r_i can also be safely ignored. Let d_l be the deadline processed before d_i. Since $|\Delta(r_j, d_i)| = |\Delta(r_j, d_l)|$ Lemma 3 insures that we have $(r_j, d_l) \succ (r_j, d_i)$ and adjustment intervals from (r_j, d_i) have already been taken into account when processing d_l.

Suppose while processing d_i we find the subsequence $(r_j, d_i) \prec (r_k, d_i)$, we add to the set U_i the tuples (j, q) for $|\Delta(r_j, d_i)| \leq q < |\Delta(r_k, d_i)|$. Each tuple $(j, q) \in U_i$ will be later used to create the adjustment intervals $I_{r_j,d_i,q}$ and $E_{r_j,d_i,q}$. According to Equation 1, these intervals are the only useful ones. Following (Artiouchine & Baptiste 2005), external adjustment intervals are computed in an order that ensures that the processed variable is not contained in any $\Delta(r_i, d_j)$ considered so far.

Algorithm 1 prunes the release times. Notice that variables are indexed in non-decreasing order of release times. Should two tasks share the same release time, the task with the smallest deadline has the smallest index. Following (Puget 1998), one can prune the upper bounds by creating the symmetric problem where task T'_i has release time $r'_i = -d_i$ and deadline $d'_i = -r_i$. Algorithm 1 can then prune the lower bounds in the symmetric problem, which prunes the upper bounds in the original problem.

We assume that the forbidden regions F have already been computed in $O(n \log n)$ (see (Garey *et al.* 1981)) and that release times are sorted such that $r_i \leq r_{i+1}$ and $r_i = r_{i+1} \Rightarrow d_i \leq d_{i+1}$. The function $lst(F, d_i, q)$ can be implemented with a table L where $lst(F, r_i, q) = L[i][q]$. Such a table requires $O(n^2)$ steps to initialize and supports function evaluation in constant time. We use a similar table

Let D be the set of deadlines sorted in increasing order. If two deadlines are equal, exclude from D the one whose associated release time is the smallest.
$P \leftarrow \emptyset$, $A \leftarrow \emptyset$, $U_i \leftarrow \emptyset$, $\forall\ 1 \leq i \leq n$
for $d_i \in D$ **do**
  $P \leftarrow P \cup \{j \mid d_j = d_i\}$
  $l \leftarrow \min(P)$
  for $j \in P \cap [l, i]$ **do**
    $a \leftarrow |\Delta(r_j, d_i)|$
    $b \leftarrow |\Delta(r_l, d_i)|$
    if $ect(F, r_l, b - a) < r_j$ **then**
      $U_i \leftarrow U_i \cup \{(l, q) \mid a \leq q < b\}$
      $l \leftarrow j$;
  $U_i \leftarrow U_i \cup \{(l, q) \mid 0 \leq q < |\Delta(r_l, d_i)|\}$
1 **for** $(j, q) \in U_i$ **do** $A \leftarrow A \cup I_{r_j,d_i,q}$
for *all deadlines* d_i *in non-decreasing order* **do**
2 $r'_i \leftarrow \min\{t \notin A \mid t \geq r_i\}$
  if $d_i \in D$ **then**
3 **for** $(j, q) \in U_i$ **do** $A \leftarrow A \cup E_{r_j,d_i,q}$
$\forall i$, $r_i \leftarrow r'_i$

Algorithm 1: Enforcing bound consistency for the INTER-DISTANCE constraint. We assume that the forbidden regions F have already been computed and that release times are sorted such that $r_i \leq r_{i+1}$ and $r_i = r_{i+1} \Rightarrow d_i \leq d_{i+1}$.

to evaluate $ect(F, r, q)$. The function $|\Delta(r_j, d_i)|$ can trivially be computed in $O(n)$ steps. The function $|\Delta(r_{j+1}, d_i)|$ can later be computed in $O(1)$. The running time complexity of Algorithm 1 is $O(n^2)$ provided that Lines 1, 2, and 3 have time complexity $O(n)$. The next section describes how the adjustment data structure A can meet these requirements.

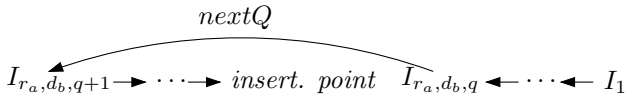
Keeping Track of Adjustment Intervals

To guarantee a quadratic running time, we must carefully design the data structure A that contains the adjustment intervals. We use a doubly-linked list containing all adjustment intervals sorted by lower bounds, including empty intervals. Each interval $I_{r_i,d_j,q}$ has a pointer $next(I_{r_i,d_j,q})$ and $previous(I_{r_i,d_j,q})$ pointing to the next and previous intervals in the list. The first interval has its $previous$ pointer undefined as the last interval has its $next$ pointer undefined. Each interval has also a pointer $nextQ(I_{r_i,d_j,q})$ pointing to $I_{r_k,d_j,q+1}$ where r_k and r_i might be equal. If the interval $I_{r_k,d_j,q+1}$ does not exist, the pointer is undefined. The data structure initially contains an empty interval with lower bound $-\infty$ used as a sentinel.

We implement Line 1 of Algorithm 1 as follows. We insert the intervals in decreasing order of lower bounds. Since we process variables by increasing deadlines, the lower bound of $I_{r_j,d_i,0}$ is larger or equal to any lower bound inserted in A and is therefore inserted at the end of the linked list.

Suppose we have inserted the interval $I_1 = I_{r_j,d_i,q}$ and we now want to insert the interval $I_2 = I_{r_k,d_i,q+1}$. Algorithm 2 computes the insertion point in the linked list.

The algorithm follows the $previous$ pointers starting from I_1 until it either finds the insertion point or finds an interval $I_{r_a,d_b,q}$ whose $nextQ$ pointer is assigned. In the latter case, the algorithm follows the $nextQ$ pointer to finally follow the $next$ pointers until the insertion point is reached. When following the $nextQ(I_{r_a,d_b,q})$ pointer, the algorithm necessarily goes to or beyond the insertion point since we have $\min(I_{r_a,d_b,q}) < \min(I_1)$ and by Lemma 2 we have $\min(nextQ(I_{r_a,d_b,q})) \leq \min(nextQ(I_1))$ and therefore $\min(I_{r_a,d_b,q+1}) \leq \min(I_2)$.



$I \leftarrow previous(I_{r_j,d_i,q})$
$I_2 \leftarrow I_{r_j,d_i,q+1}$
while $nextQ(I)$ *is undefined* $\wedge \min(I) > \min(I_2)$ **do**
  $I = previous(I)$
if $\min(I) > \min(I_2)$ **then**
  $I \leftarrow nextQ(I)$
  while $\min(next(I)) < \min(I_2)$ **do**
    $I \leftarrow next(I)$
Insert I_2 after I

Algorithm 2: Compute the insertion point of $I_{r_j,d_i,q+1}$ provided that $I_{r_j,d_i,q}$ has already been inserted.

We show that Algorithm 2 inserts a sequence of $O(n)$ intervals in the linked list A in $O(n)$ steps. There is a maximum of n intervals in A whose $nextQ$ pointer is undefined, therefore the first while loop runs in $O(n)$. Let I_4 be an interval explored by the second while loop. The interval I_4 lies between $nextQ(I)$ and the insertion point. By Lemma 2, if an interval I_3 was pointing to I_4 with its $nextQ$ pointer, the interval I_3 would lie between I and I_1. Since $I_3 \neq I$, we conclude that no intervals point to I_4 with its $nextQ$ pointer. There is a maximum of n such intervals. The second while loop runs in $O(n)$. We therefore showed that Line 1 can be implemented in $O(n)$ steps. Since $\min(E_{r_i,d_j,q}) = \min(I_{r_k,d_j,q+1})$, Line 3 can be implemented by simply changing the upper bounds of internal adjustment intervals that were already inserted in A.

Line 2 of Algorithm 1 can be implemented in $O(\alpha(n))$ steps where α is the inverse of Ackermann's function. We create a union-find data structure S with elements from 1 to n. For each element i, we associate a time t_i initially set to r_i and a pointer p_i initially unassigned. When inserting adjustment intervals in A in decreasing order of lower bounds, we simultaneously iterate in decreasing order the sets in S. If an interval I is inserted such that $t_i \in I$, we set t_i to $\max(I) + 1$. We then follow the $next$ pointers from I to check if other intervals intersect t_i. If t_i becomes greater or equal to t_{i+1}, we merge the set in S containing i with the set containing $i + 1$. The pointer p_i is used to keep track of the last interval I tested with t_i in order to not check twice a same interval. When executing Line 2 of Algorithm 1, we simply retrieve from S the set s containing i and return t_j where $j = \max(s)$.

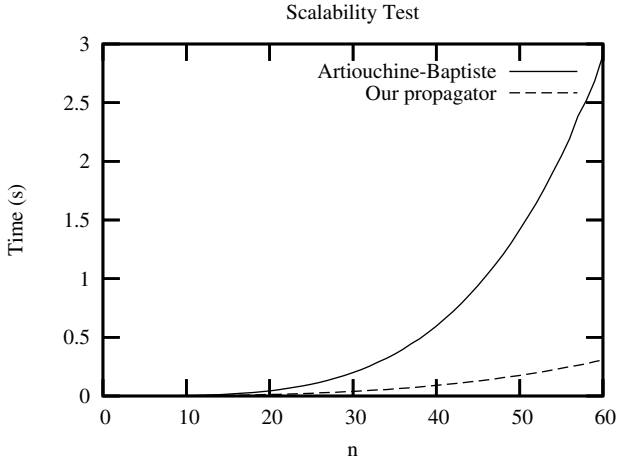


Figure 1: Running time of the Artiouchine-Baptiste propagator ($O(n^3)$) and our propagator ($O(n^2)$) as a function of the number of tasks. For this scalability test, we set all release times to $r_i = 0$ and deadlines to $d_i = 6n$.

Experiments

We implemented our algorithm using the ILog Solver C++ library, Version 4.2 (ILOG 1998)[1]. The library already provides a propagator for the INTER-DISTANCE constraint called *IlcAllMinDistance* and offers two levels of consistency, namely *IlcBasic* and *IlcExtended*. We also implemented the Artiouchine-Baptiste propagator (Artiouchine & Baptiste 2005). All experiments were run on a Pentium III 900 MHz with 256 Mb of memory. All reported times are averaged over 10 runs.

Scalability Test

In order to test the scalability of our propagator, we first consider a scheduling problem with a single INTER-DISTANCE constraint over n tasks whose release times are $r_i = 0$ and deadlines are $d_i = np$ for all tasks. This problem has a trivial solution and is solved without backtracking. We clearly see on Figure 1 that our propagator has a quadratic behaviour while the Artiouchine-Baptiste propagator has a cubic behaviour. This observation is supported by the study of the third and second derivative.

Runway Scheduling Problem

We then study the runway scheduling problem (Artiouchine, Baptiste, & Dürr 2004). In this problem, n airplanes have certain time intervals in which they can land. Airplane number i has s_i time intervals $[r_i^1, d_i^1], \ldots, [r_i^{s_i}, d_i^{s_i}]$. Following (Artiouchine & Baptiste 2005), we create for each airplane a variable t_i with domain $[r_i^1, d_i^{s_i}]$ representing the landing time and a variable c_i with domain $[1, s_i]$ representing the landing time interval. We have the constraints $c_i \geq k \iff t_i \geq r_i^k$ and $c_i \leq k \iff t_i \leq d_i^k$. Finally, we

[1]The code discussed in this section is available upon request from the first author.

n	#	Our solution	A.-B.	IlcBasic	IlcExt.
15	1	0.09	0.16		
15	40	0.07	0.12	19.27	64.32
30	54	0.42	1.65		
45	20	0.59	3.40		
60	40	2.38	18.72		
75	30	4.04	37.67		
90	10	5.64	60.84		

Table 2: Time in seconds to solve some representative problems from the benchmark. n is the number of variables. # is the problem number in the benchmark. Blank entries represent problems that remained unsolved after 2 minutes of computation.

n	a	b	c	d	Our solution	A.-B.	Fails
20	10	10	5	6	0.11	0.25	28
20	10	10	5	6	0.09	0.17	4
30	8	15	3	6	3.40	14.26	2111
40	7	10	5	6	0.95	4.74	183
50	10	10	5	6	0.72	4.23	5
50	10	10	5	6	0.63	3.81	27
55	7	10	5	6	1.03	6.68	16
60	8	15	3	6	1.27	9.46	11
20	10	10	5	6	0.09	0.29	34
20	8	6	3	6	0.09	0.19	13
40	10	20	3	6	0.70	3.54	67
40	7	10	5	6	0.47	2.14	19
50	10	10	5	6	3.99	26.66	435
50	8	6	3	6	0.83	5.26	35
55	8	15	3	6	0.89	6.19	16
60	8	15	3	6	1.18	8.95	25
60	8	15	3	6	1.64	12.54	44

Table 3: Time in seconds to solve the runway problems where landing time intervals have size a, the gap between landing time intervals is of size b, and where $c \leq s_i \leq d$ holds.

have the constraint INTER-DISTANCE$([t_1, \ldots, t_n], p)$ that ensures a gap of p between each landing. For security reasons, we want to maximize the time p between each landing. We first solve the problem with $p = 1$ and double the value of p until the problem becomes infeasible. Suppose the problem becomes infeasible for the first time at $p = 2x$. We perform a binary search in $[x, 2x)$ to find the greatest p satisfying the problem.

Using the same benchmark as (Artiouchine & Baptiste 2005), we obtain the running times listed in Table 2 on random runway problems where the sizes of intervals and the gap between intervals may vary. For problems with 90 airplanes, we obtain savings of an order of magnitude. Some propagators could not solve all problems within 2 minutes. All problems that could be solved with the propagators provided in ILog could also be solved by the Artiouchine-Baptiste propagator. All problems solved by the Artiouchine-Baptiste propagator could be solved at greater speed by our propagator.

We then consider the runway problem where all intervals have the same length (see Table 3). In these problems, Ilog propagators were unable to solve problems in less than two minutes. We obtain an improvement over the Artiouchine-Baptiste propagator proportional to n. This observation is compatible with the running time complexities of the algorithms.

Conclusion

We presented a new propagator achieving bound consistency for the INTER-DISTANCE constraint. The running time complexity of $O(n^2)$ improves by a factor of n the previous best known propagator. This theoretical improvement gives practical savings in scheduling problems.

It is still an open problem whether there exists an $O(n \log n)$ propagator for the INTER-DISTANCE constraint achieving bound consistency. It would also be interesting to study how the constraint could be generalized for the cumulative scheduling problem. For some optimization problems, it would be convenient to consider p as a constrained variable on which we could enforce bound consistency.

References

Artiouchine, K., and Baptiste, P. 2005. Inter-distance constraint: An extension of the all-different constraint for scheduling equal length jobs. In *Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming*, pp. 62–76.

Artiouchine, K.; Baptiste, P.; and Dürr, C. 2004. Runway scheduling with holding loop. In *Proceedings of Second International Workshop on Discrete Optimization Methods in Production and Logistics*, pp. 96–101.

Carlier, J., and Pinson, E. 1994. Adjustment of heads and tails for the job-shop problem. *European Journal of Operation Rsearch* 78:146–161.

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

ILOG. 1998. *ILOG Solver 4.2 user's manual*.

López-Ortiz, A.; Quimper, C.-G.; Tromp, J.; ; and van Beek, P. 2003. A fast and simple algorithm for bounds consistency of the alldifferent constraint. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, pp. 245–250.

Mehlhorn, K., and Thiel, S. 2000. Faster algorithms for bound-consistency of the sortedness and alldifferent constraint. In *Proceedings of the Sixth International Conference on Principles and Practice of Constraint Programming*, pp. 306–319.

Mercier, L., and Van Hentenryck, P. 2005. Edge finding for cumulative scheduling. Submitted for publication.

Puget, J.-F. 1998. A fast algorithm for the bound consistency of alldiff constraints. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pp. 359–366.

Régin, J.-C. 1994. A filtering algorithm for constraints of difference in CSPs. *Proceedings of AAAI-94* pp. 362–367.

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