A Quadratic Propagator for the Inter-Distance Constraint

Claude-Guy Quimper
QUIMPER@ALUMNI.UWATERLOO.CA
University of Waterloo
School of Computer Science

Alejandro López-Ortiz
ALOPEZ-O@UWATERLOO.CA
University of Waterloo
School of Computer Science

Gilles Pesant
PESANT@CRT.UMONTREAL.CA
École Polytechnique de Montréal
Department of Computer Engineering

Editor: Pascal Van Hentenryck

Abstract

We present a new propagator achieving bounds consistency for the INTER-DISTANCE constraint. This constraint ensures that, among a set of variables $X_1, \dots, X_n$, 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 disjunctive resource to be completed. Until now, the best known propagator for bounds 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 the theoretical gain gives savings of an order of magnitude in our benchmark of scheduling problems. Our propagator is also the first one to filter the variable p.

1. Introduction

The *cumulative scheduling problem* with one resource of capacity C consists of a set of tasks $T_1, \dots, T_n$ 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 and Pinson (1994); Mercier and 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 *bounds 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 bounds 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 bounds 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. See, for instance, the runway scheduling problem (Artiouchine et al. (2004)). 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 (2005) recently proposed an $O(n^3)$ propagator that enforces bounds consistency on the INTER-DISTANCE constraint. By achieving bounds 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. We generalize the INTER-DISTANCE constraint by letting the distance P be a constrained variable whose domain can be pruned. This proves to be useful when one wants to maximize the distance between each pair of variables. The generalization of the INTER-DISTANCE constraint as well as the experiments with this new constraint extend the work previously presented in Quimper et al. (2006).

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 present in Section 2 some notions about how bounds consistency can be enforced on the INTER-DISTANCE constraint. We then explain in Section 3 how the computation can be simplified. Based on this simplification, we present in Section 4 our algorithm and a data structure that ensures the quadratic behaviour of our propagator. We show in Section 5 how bounds consistency can be enforced on the distance variable P. Finally, we present in Section 6 some experiments proving the efficiency of our propagator.

2. The INTER-DISTANCE Constraint

Régin (1997) first introduced the INTER-DISTANCE constraint under the name *global minimum 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 and Thiel (2000); López-Ortiz et al. (2003)). 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 bounds consistency of the INTER-DISTANCE constraint. The running time complexity of their propagator is cubic. Throughout the paper, we will assume that P is fixed to a value p and therefore can be assigned to this value. In Section 5, we generalize to the case where P is not fixed and show how to enforce bounds consistency on all constrained variables.

We use the following problem as a running example.

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

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

Figure 1 shows the release times and deadlines as well as a feasible schedule.

Let the domain of T_i be the interval $[r_i, d_i - p]$ i.e., $T_1 \in [2, 6]$, $T_2 \in [10, 14]$, and $T_3 \in [4, 15]$. After propagation of the constraint INTER-DISTANCE$([T_1, T_2, T_3], p)$*, the variables get assigned to the following values.*

$$T_1 = 2 \qquad T_2 = 14 \qquad T_3 = 8$$

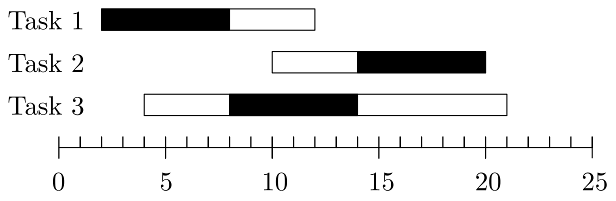

Figure 1: Release times and deadlines for the three tasks described in Example 1. The black rectangles represent a feasible schedule.

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. (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 intervals F in which no tasks are allowed to start. We call this set the *forbidden regions* and denote it by F. Their number is bounded above by n, the number of tasks. Once these forbidden regions are computed, the second phase uses a greedy strategy to schedule the tasks.

Artiouchine and Baptiste (2005) and Garey et al. (1981) use two basic functions as main pillars of their algorithm. We define the completion time of a schedule as the time of completion of the very last task and the starting time of a schedule as the starting time of the very first task. Let $ect(F, r, q)$ be the *earliest completion time* of a schedule of q tasks with release time r and unbounded deadline such that no task starts in a forbidden region contained in the set of forbidden regions F. Symmetrically, let $lst(F, d, q)$ be the *latest starting time* of a schedule of q tasks with deadline d and unbounded release time such that no task starts 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.

$$ect(F, r, q) = \begin{cases} r & \text{if } q = 0 \\ \min\{t \notin F \mid t \geq ect(F, r, q-1)\} + p & \text{if } q > 0 \end{cases} \tag{1}$$

$$lst(F, d, q) = \begin{cases} d & \text{if } q = 0 \\ \max\{t \notin F \mid t \leq lst(F, d, q-1) - p\} & \text{if } q > 0 \end{cases} \tag{2}$$

The function lst helps to explain the algorithm of Garey et al. (see Algorithm 1) that computes the forbidden regions F. Their algorithm starts with an empty set of regions $F = \emptyset$ and processes each release time r in decreasing order. Consider a deadline d and let $\Delta(r, d)$ be the tasks whose release times and deadlines are both contained in the interval $[r, d]$. The value $lst(F, d, |\Delta(r, d)|) - r$ represents the amount of slack in the interval $[r, d]$, i.e. an upper bound on the processing time that can be used in the interval $[r, d]$ for tasks that are not in $\Delta(r, d)$. If $lst(F, d, |\Delta(r, d)|) - r$ is smaller than p then there is not enough room to fit a whole task not in $\Delta(r, d)$. The algorithm therefore appends to F the forbidden region $[lst(F, d, |\Delta(r, d)| + 1) + 1, r - 1]$. Indeed, any task starting in this region consumes too much time in the interval $[r, d]$ to allow the completion of the tasks in $\Delta(r, d)$. If $lst(F, d, |\Delta(r, d)|) - r < 0$, there are too many tasks that require to be executed in the interval $[r, d]$, the problem is unsolvable. Upon termination, the set of forbidden regions contains up to n distinct regions. These regions are sufficient to find a solution to the problem but are insufficient to enforce bounds consistency on the INTER-DISTANCE constraint. Artiouchine and Baptiste explain how to compute a larger set of regions that are sufficient to filter the INTER-DISTANCE constraint.

Using the functions ect and lst, Artiouchine and Baptiste describe two types of adjustment intervals necessary and sufficient to maintain bounds consistency on the INTER-DISTANCE constraint. An *internal*

```
$F \leftarrow \emptyset$
$R \leftarrow$ the set of release times
$D \leftarrow$ the set of deadlines
for $r \in R$ in decreasing order do
    $d \leftarrow \operatorname{argmin}_{d \in D} lst(F, d, |\Delta(r, d)|) - r$
    $s \leftarrow lst(F, d, |\Delta(r, d)|) - r$
    if $s < 0$ then
        return The problem is unsolvable
    else if $s < p$ then
        $F \leftarrow F \cup [lst(F, d, |\Delta(r, d)| + 1) + 1, r - 1]$
return $F$
```

Algorithm 1: Algorithm that computes the forbidden regions. Garey et al. (1981) use special data structures to obtain a complexity of $O(n \log n)$.

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.

Definition 1 (Internal Adjustment Interval) *Given two time points r and d and an integer $0 \leq q < |\Delta(r, d)|$, the* internal adjustment interval $I_{r,d,q}$ *is defined as follows.*

$$I_{r,d,q} \quad = \quad [lst(F, d, q + 1) + 1, ect(F, r, |\Delta(r, d)| - q) - 1] \tag{3}$$

Theorem 2 presents in which context we will use internal adjustment intervals.

Theorem 2 (Artiouchine and Baptiste (2005)) *Given two time points r, d, and an integer $0 \leq q < |\Delta(r, d)|$, no task can start in the interval $I_{r,d,q}$.*

Internal adjustment intervals appear in problems where a group of variables must be assigned to values in an interval that is small enough to force a certain structure to be maintained. The internal adjustment intervals ensure that a single variable does not occupy the “space” of two variables.

The *external adjustment intervals* are intervals in which a subset of the tasks are not allowed to start.

Definition 3 (External Adjustment Intervals) *Given two time points r and d and an integer $0 \leq q < |\Delta(r, d)|$, the* internal adjustment interval $E_{r,d,q}$ *is defined as follows.*

$$E_{r,d,q} \quad = \quad [lst(F, d, q + 2) + 1, ect(F, r, |\Delta(r, d)| - q) - 1] \tag{4}$$

Theorem 4 shows the main property of external adjustment intervals.

Theorem 4 (Artiouchine and 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 $E_{r,d,q}$.*

External adjustment intervals appear in problems where a group of variables compete for an interval of values. The variables whose domain is not restricted to this small interval and hence do not belong to this group of competing variables must be assigned to values outside of the interval.

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

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

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 $F = \{[-3,1],[3,3],[9,9]\}$.

3. Towards a Quadratic Time 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 fact, 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 5 (Dominance) *A pair of time points (r, d) dominates a pair of time points (r', d') if we have $\min(I_{r,d,q}) \leq \min(I_{r',d',q})$ and $\max(I_{r,d,q}) \geq \max(I_{r',d',q})$ for all $0 \leq q < \min(|\Delta(r,d)|, |\Delta(r',d')|)$. We write $(r,d) \succ (r',d')$.*

Notice that we usually have $|\Delta(r,d)| \neq |\Delta(r',d')|$. The definition of dominance only applies for q below $\min(|\Delta(r,d)|, |\Delta(r',d')|)$. 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 6 *If $ect(F, r, q) \leq ect(F, r', q')$ then $ect(F, r, q+k) \leq ect(F, r', q'+k)$ 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 $ect(F, r, q+k) = ect(F, r, q+k-1) + p + s$ where s is a (potentially null) shift caused by the (potentially empty) forbidden region $F_i = [ect(F, r, q+k-1), ect(F, r, q+k-1) + s] \subseteq F$. Similarly we have $ect(F, r', q'+k) = ect(F, r', q'+k-1) + p + s'$ where s' is the shift caused by the forbidden region $F_j = [ect(F, r', q'+k-1), ect(F, r', q'+k-1) + s'] \subseteq F$. If s is large enough to obtain $ect(F, r, q+k) \geq ect(F, r', q'+k)$, then we have $F_j \subseteq F_i$. Since F_j is a subset of F_i, both functions $ect(F, r, q+k)$ and $ect(F, r', q'+k)$ are shifted to the same value. We therefore obtain $ect(F, r, q+k) = ect(F, r', q'+k)$ which completes the induction step. ■

Lemma 7 *If $lst(F, d, q) \leq lst(F, d', q')$ then $lst(F, d, q+k) \leq lst(F, d', q'+k)$ for any $k \geq 0$.*

Proof Symmetric to the proof of Lemma 6. ■

We now describe three different situations in which a pair of time points dominates another one. The first case is described in Lemma 8.

Lemma 8 *Consider the pairs of time points* (r, d) *and* (r, d') *such that* $d < d'$. *If* $|\Delta(r, d)| = |\Delta(r, d')|$ *then* $(r, d) \succ (r, d')$.

Proof Let $k = |\Delta(r, d)| = |\Delta(r, d')|$. We have $lst(F, d, 0) < lst(F, d', 0)$ and by Lemma 7, for any $0 \leq q < k$, we have $lst(F, d, q + 1) \leq lst(F, d', q + 1)$. This implies $\min(I_{r,d,q}) \leq \min(I_{r,d',q})$ and since we have $\max(I_{r,d,q}) = \max(I_{r,d',q})$ we have $(r, d) \succ (r, d')$. ■

From Lemma 8 we conclude that $(10, 20) \succ (10, 21)$ in Example 1. Similarly, we have the following Lemma.

Lemma 9 *Consider the pairs of time points* (r, d) *and* (r', d) *such that* $r < r'$ *and* $|\Delta(r, d)| = |\Delta(r', d)|$. *Then* $(r, d) \prec (r', d)$.

Proof Let $k = |\Delta(r, d)| = |\Delta(r', d)|$. We have $ect(F, r, 0) < ect(F, r', 0)$ and by Lemma 6, for any $0 \leq q < k$, $ect(F, r, k - q) \leq ect(F, r', k - q)$. This implies $\max(I_{r,d,q}) \leq \max(I_{r',d,q})$ and since we have $\min(I_{r,d,q}) = \min(I_{r',d,q})$ we have $(r, d) \prec (r', d)$. ■

In Example 1, Lemma 9 detects $(4, 20) \prec (10, 20)$. We show a last case where a pair of time points dominates another one.

Lemma 10 *Let* (r, d) *and* (r', d) *be two pairs of time points such that* $|\Delta(r, d)| = |\Delta(r', d)| + k$ *and* $ect(F, r, k) \leq ect(F, r', 0)$. *Then* $(r', d) \succ (r, d)$.

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

$$\begin{aligned}
\max(I_{r,d,q}) &= ect(F, r, |\Delta(r, d)| - q) - 1 \\
&= ect(F, r, |\Delta(r', d)| + k - q)) - 1 \\
&\leq ect(F, r', |\Delta(r', d)| - q) - 1 \qquad \text{Using Lemma 6 and } ect(F, r, k) \leq ect(F, r', 0) \\
&\leq \max(I_{r',d,q})
\end{aligned} \tag{5}$$

Therefore we have $(r', d) \succ (r, d)$. ■

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

Lemma 10 is crucial to obtain 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.

$$\begin{aligned}
\bigcup_{i=1}^{j} I_{r_i,d,q} &= [\min(I_{r_j,d,q}), \max_{1 \leq i \leq j} \max(I_{r_i,d,q})] && (6) \\
&= [\min(I_{r_j,d,q}), \max(I_{r_j,d,q})] && (7) \\
&= I_{r_j,d,q} && (8)
\end{aligned}$$

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.

$$\bigcup_{i=1}^{k} \bigcup_{q=0}^{|\Delta(r_i,d)|-1} I_{r_i,d,q} = \bigcup_{i=1}^{k} \bigcup_{q=|\Delta(r_{i+1},d)|}^{|\Delta(r_i,d)|-1} I_{r_i,d,q} \tag{9}$$

Notice that the left hand side of Equation 9 has $O(n^2)$ intervals while the right hand side has only $O(n)$ intervals. Indeed, the number of intervals to unite 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:

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

Theorem 11 *There are $O(n^2)$ internal adjustment intervals that subsume any other internal adjustment interval.*

Proof Consider a deadline d and two release times $r_i \leq r_j$. For every value $0 \leq q < |\Delta(r_j, d)|$, we have $\min(I_{r_i,d,q}) = \min(I_{r_j,d,q})$. We have $\max(I_{r_i,d,q}) \geq \max(I_{r_j,d,q})$ if and only if $(r_i, d) \succ (r_j, d)$. Consequently, we have $I_{r_j,d,q} \subseteq I_{r_i,d,q}$ if and only if $(r_i, d) \succ (r_j, d)$.

Consider the list of release times $r_1 \leq r_2 \leq \ldots \leq r_n$ sorted in non-decreasing order. If for some k we have $(r_k, d) \succ (r_{k+1}, d)$, we can safely ignore the release time r_{k+1} since for every interval we have $I_{r_{k+1},d,q} \subseteq I_{r_k,d,q}$ for $0 \leq q < |\Delta(r_{k+1}, d, q)|$. After removing all dominated release times, we obtain a list of release times $r_{k_1} \prec r_{k_2} \prec \ldots \prec r_{k_m}$. Equation 9 shows how the internal adjustment intervals $I_{r,d,q}$ for any r and q are subsumed by $O(n)$ intervals. Since there are $O(n)$ deadlines d, there are $O(n^2)$ internal adjustment intervals that subsume any other internal adjustment interval. ■

4. A Quadratic Propagator

4.1 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 $(r_{x_1}, d_i) \prec (r_{x_2}, d_i) \prec \ldots \prec (r_{x_k}, d_i)$ as we did in Theorem 11. Using this sequence and Equation 9, 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 the 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 9 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 8 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.

We prove that Algorithm 2 constructs for every deadline d_i a sequence $r_{j_1} < r_{j_2} < \ldots < r_{j_k}$ such that $(r_{j_1}, d_i) \prec (r_{j_2}, d_i) \prec \ldots \prec (r_{j_k}, d_i)$. This sequence with Equation 9 compute the adjustment intervals.

```
    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.
    $P \leftarrow \emptyset$, $A \leftarrow \emptyset$, $U_i \leftarrow \emptyset$, $\forall\ 1 \le i \le n$
1   for $d_i \in D$ do
        $P \leftarrow P \cup \{j \mid d_j = d_i\}$
        $l \leftarrow \min(P)$
2       for $j \in P \cap [l, i]$ do
            $a \leftarrow |\Delta(r_j, d_i)|$
            $b \leftarrow |\Delta(r_l, d_i)|$
3           if $ect(F, r_l, b - a) \le r_j$ then
4               $U_i \leftarrow U_i \cup \{(l, q) \mid a \le q < b\}$
5               $l \leftarrow j$;
6       $U_i \leftarrow U_i \cup \{(l, q) \mid 0 \le q < |\Delta(r_l, d_i)|\}$
7       for $(j, q) \in U_i$ do $A \leftarrow A \cup I_{r_j, d_i, q}$
8   for all deadlines $d_i$ in non-decreasing order do
9       $r'_i \leftarrow \min\{t \notin A \mid t \ge r_i\}$
        if $d_i \in D$ then
10          for $(j, q) \in U_i$ do $A \leftarrow A \cup E_{r_j, d_i, q}$
    for $i \in [1, n]$ do $r_i \leftarrow r'_i$
```

Algorithm 2: Enforcing bounds 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 $r_i \le r_{i+1}$ and $r_i = r_{i+1} \Rightarrow d_i \le d_{i+1}$.

Lemma 12 *Algorithm 2 encodes in the data structure U_i a sequence $(r_{j_1}, d_i) \prec (r_{j_2}, d_i) \prec \ldots \prec (r_{j_k}, d_i)$ generating all internal adjustment intervals associated to the deadline d_i.*

Proof The *for* loop on line 2 processes each release time r_j such that $d_j \le d_i$ and $r_j \le r_i$. Other release times can be safely ignored as they correspond to the cases 1 and 2 stated above.

Line 3 tests whether $ect(F, r_l, b - a) \le r_j$. The first time Line 3 is executed, the test is positive since $l = j = \min(P)$ and $a = b$. We therefore include $(r_{\min(P)}, d_i)$ in the sequence. Subsequent tests are positive only if the pair (r_j, d_i) dominates the last pair (r_l, d_i) tested positive as proved below.

$$\begin{aligned} ect(F, r_l, b-a) \le r_j &\implies ect(F, r_l, b-a) \le ect(F, r_j, 0) && \text{by definition of } ect \quad (10) \\ &\implies ect(F, r_l, b-q) \le ect(F, r_j, a-q)\ \forall\ 0 \le q < a && \text{by Lemma 6} \quad (11) \\ &\implies (r_l, d_i) \prec (r_j, d_i) && \text{by def. of dominance} \quad (12) \end{aligned}$$

Similarly, one can prove that a negative test implies that $(r_l, d_i) \succ (r_j, d_i)$ and that the pair (r_j, d_i) should not belong to the sequence. The sequence $r_1 \prec r_2 \prec \ldots \prec r_k$ is therefore not missing any release time.

Each time the relation $(r_l, d_i) \prec (r_j, d_i)$ is discovered, the algorithm appends to U_i the tuples (j, q) for $|\Delta(r_j, d_i)| \le q < |\Delta(r_l, d_i)|$. Each tuple $(j, q) \in U_i$ will be used later to create the internal internal adjustment intervals $I_{r_j, d_i, q}$. According to Equation 9, these intervals are sufficient. ■

Following Artiouchine and Baptiste (2005), the second *for* loop on line 8 processes the deadlines in non-decreasing order. The external adjustment intervals are therefore computed in an order that ensures that the processed variable is not contained in any $\Delta(r_i, d_j)$ considered so far.

Algorithm 2 only prunes the release times. Following Puget (1998), one can prune the deadlines by creating the symmetric problem where task T'_i has release time $r'_i = -d_i$ and deadline $d'_i = -r_i$. Algorithm 2 can then prune the release times in the symmetric problem, which prunes the deadlines in the original problem.

The data structures P and U_i can be implemented using some linked lists. However, to obtain an algorithm running in quadratic time, we need to craft a special data structure to store the adjustment intervals in A. This data structure should allow the execution of lines 7, 9, and 10 in no more than $O(n)$ time even if A contains up to $O(n^2)$ intervals. The next section describes how the adjustment data structure A can meet these requirements.

4.2 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 7 of Algorithm 2 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 3 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 7 we have $\min(nextQ(I_{r_a,d_b,q})) \leq \min(nextQ(I_1))$ and therefore $\min(I_{r_a,d_b,q+1}) \leq \min(I_2)$.

We show that Algorithm 3 inserts a sequence of $O(n)$ intervals in the linked list A in $O(n)$ steps.

Lemma 13 *Algorithm 3 inserts on line 7 a sequence of $O(n)$ internal adjustment intervals in the linked list A in $O(n)$ time.*

Proof 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 7, 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 7 can be implemented in $O(n)$ steps. ■

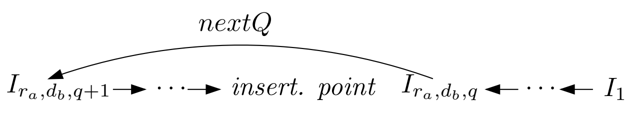
$$\begin{tikzcd} I_{r_a,d_b,q+1} \arrow[r] & \cdots \arrow[r] & insert.\ point & I_{r_a,d_b,q} \arrow[lll, bend right, "nextQ"'] & \cdots \arrow[l] & I_1 \arrow[l] \end{tikzcd}$$

Figure 2: Nodes in the doubly linked list that Algorithm 3 visits to find the insertion point of an adjustment interval.

```
$I \leftarrow previous(I_{r_j,d_i,q})$
$I_2 \leftarrow I_{r_j,d_i,q+1}$
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 3: Compute the insertion point of $I_{r_j,d_i,q+1}$ provided that $I_{r_j,d_i,q}$ has already been inserted.

Line 10 inserts in A a sequence of $O(n)$ external adjustment intervals. Notice that at this point, A already contains the internal adjustment intervals and that by definition, the lower bound of $E_{r_i,d_j,q}$ is equal to the lower bound of $I_{r_k,d_j,q+1}$. Line 10 can be implemented by simply changing $I_{r_k,d_j,q+1}$ in A by $E_{r_i,d_j,q}$. If $I_{r_k,d_j,q+1}$ does not exist in A, it can be added using Algorithm 3.

Lemma 14 *A sequence of $O(n)$ external adjustment intervals can be inserted in $O(n)$ time.*

Proof For every pair $(j, q) \in U_i$, one can keep a pointer on its associated interval $I_{r_j,d_i,q}$ in the data structure A. Following the $nextQ$ pointer to reach $I_{r_k,d_i,q+1}$ takes constant time. Setting the new upper bound of $I_{r_k,d_i,q+1}$ also takes constant time. The $nextQ$ pointer is undefined for the last interval to insert, Algorithm 3 can insert the interval $I_{r_k,d_i,q+1}$ in $O(n)$ steps. The total running time complexity is therefore $O(n)$. ■

Line 9 of Algorithm 2 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 change t_i to $\max(I) + 1$. We then follow the $next$ pointers from I to check if other intervals intersect t_i and increase t_i for each intersecting interval. 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 not to check twice a same interval. When executing Line 9 of Algorithm 2, we simply retrieve from S the set s containing i and return t_j where $j = \max(s)$.

Lemma 15 *Updating and requesting the variables t_i is performed in $O(n^2)$ steps.*

Proof There are up to $O(n^2)$ intervals in the data structure A and each of them can be visited at most once. Indeed, the pointers p_i make sure that the search is always resumed from the last visited position in the list A. The union-find data structure ensures that if an interval $I \in A$ increases more than one release time, this operation is done in constant time since all release times are grouped in the same set in S and only the representative t_i of this set is updated. Merging the elements in S and requesting the representive elements takes $O(n\alpha(n))$. The total complexity is therefore $O(n^2)$. ■

4.3 Running Time Analysis

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 to evaluate $ect(F, r, q)$. The function $|\Delta(r_1, d_i)|$ can trivially be computed in $O(n)$ steps. Since the release times are sorted in non-decreasing order, one can compute $|\Delta(r_j, d_i)|$ using the following recursion.

$$|\Delta(r_j, d_i)| = \begin{cases} |\Delta(r_{j-1}, d_i)| & \text{if } d_{j-1} > d_i \\ |\Delta(r_{j-1}, d_i)| - 1 & \text{otherwise} \end{cases} \tag{13}$$

The function $|\Delta(r_i, d_j)|$ can be implemented with a table D such that $D[i][j] = |\Delta(r_i, d_j)|$. Initializing the table using the recursion above requires $O(n^2)$ steps. Each function call then takes constant time.

Theorem 16 *The running time complexity of Algorithm 2 is $O(n^2)$.*

Proof We assume that the forbidden regions F have already been computed in $O(n \log n)$ time as explained by Garey et al. (1981). The data structures P and U_i are implemented using linked lists and the functions lst, ect, and $|\Delta(r_i, d_j)|$ are implemented using tables. The data structure A is implemented with the data structure described in Section 4.2. Release times and deadlines are sorted in $O(n \log n)$ time. Initializing the algorithm therefore requires $O(n^2)$ time.

For a fixed i, every pair (l, q) added to U_i on line 4 and line 6 have a distinct component q ranging from 0 to n exclusively. There are therefore at most $O(n)$ elements in U_i that were appended in $O(1)$ time. The complexity to build the list U_i is therefore $O(n)$. Line 7 takes $O(n)$ time to execute as stated by Lemma 13. The total running time for the *for* loop on line 1 is therefore $O(n^2)$.

By Lemma 15, the cummulative running time complexity of the line 9 is $O(n^2)$. Line 10 takes $O(n)$ time to execute as stated in Lemma 14. The *for* loop on line 8 therefore runs in $O(n^2)$.

Since the initialization and the *for* loops on lines 1 and 8 all run in $O(n^2)$ time, the total running time complexity of Algorithm 2 is also $O(n^2)$. ■

5. Achieving Bounds Consistency on the Distance Variable

We now consider the general form of the INTER-DISTANCE constraint i.e., when the distance variable P is not fixed. The constraint INTER-DISTANCE$(X_1, \ldots, X_n, P)$ is satisfied if and only if

$$i \neq j \implies |X_i - X_j| \geq P \tag{14}$$

Clearly, if there is an assignment $\langle X_1, ..., X_n \rangle$ that satisfies the case when $P = p$, then this assignment is also a support for $P = p - 1$. Therefore, any value that has a support in the domain of X_i for $P = p$ has a support in the same domain for $P < p$. To prune the X_i variables, one only needs to find the values in the domains that have a support for $P = \min(D(P))$. The algorithm presented in Section 4, when used with $p = \min(D(P))$, can therefore prune the X_i variables.

To prune the domain of P, we rely on the following observation. If the constraint is unsatisfiable with $P = p$, then it is unsatisfiable for $P > p$. Therefore, to achieve bounds consistency on the variable P, one only needs to prune the value $\max(D(P))$.

The algorithm by Garey et al. allows testing, in $O(n \log n)$ steps, whether there exists a solution for $P = p$. Using a one-sided binary search, we can find the largest value in $D(P)$ such that the INTER-DISTANCE constraint is satisfiable. A one-sided binary search returning value p whose test requires $O(n \log n)$ time has a running time complexity of $O(n \log n \log p)$. We can enforce bounds consistency on the INTER-DISTANCE constraint when the distance variable is not fixed in $O(n^2 + n \log n \log p)$ steps.

6. Experiments

We implemented our algorithm using the ILOG Solver C++ library, (ILOG (1998))[1]. The library already provides a propagator for the INTER-DISTANCE constraint called *IlcAllMinDistance* and offers two levels

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

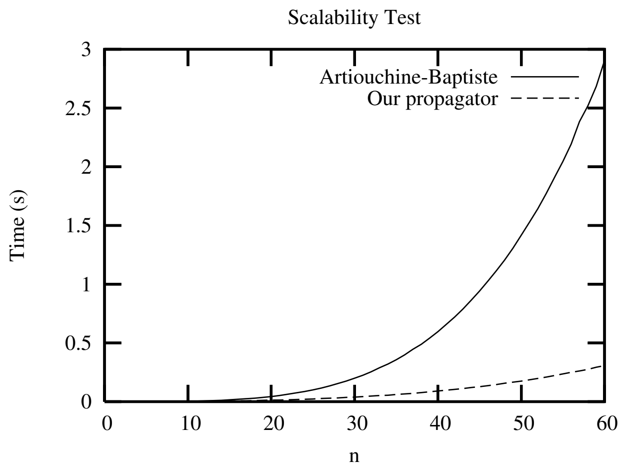

Figure 3: Running time of the Artiouchine-Baptiste propagator ($O(n^3)$) and our propagator ($O(n^2)$) as a function of the number of tasks. For this scalability test, we set all release times to $r_i = 0$ and deadlines to $d_i = 6n$.

of consistency, namely *IlcBasic* and *IlcExtended*. We also implemented the Artiouchine-Baptiste propagator (Artiouchine and Baptiste (2005)). The scalability test presented in Section 6.1 was run on a Pentium III 900 MHz with 256 Mb of memory and ILOG Solver 4.2. The experiments on the runway scheduling problem presented in Section 6.2 were run on a AMD64 Opteron 250 with a 2.4 GHz dual processor (only one processor was used) and 3 GB of RAM. We used on this computer the library ILOG Solver 6.1. All reported times are averaged over 10 runs.

6.1 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 3 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.

6.2 Runway Scheduling Problem

We then study a runway scheduling problem (Artiouchine et al. (2004)). In this problem, n airplanes have certain time intervals in which they can land. Airplane number i has s_i time intervals $[r_i^1, d_i^1], \dots, [r_i^{s_i}, d_i^{s_i}]$. Following Artiouchine and 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 have the constraint INTER-DISTANCE$([t_1, \dots, t_n], P)$ that ensures a gap of P between each landing. For security reasons, we want to maximize the time P between each landing.

In order to fairly compare both propagators, we enhanced the Artiouchine-Baptiste propagator with the algorithm presented in Section 5 to prune the variable P. With ILOG Solver, we set the goal of performing a binary search on P. We also set the objective of minimizing $-P$. We use the default heuristics and parameters proposed by ILOG Solver.

We used the same benchmark as Artiouchine and Baptiste (2005) on random runway scheduling problems where the sizes of intervals and the gap between intervals may vary. Figure 4 shows the number of problems solved in the benchmark in a given period of time. Our propagator has consistently solved the problems at greater speed than the Artiouchine-Baptiste propagator. The two levels of consistency provided in ILOG for the *IlcAllMinDistance* constraint were not able to compete with the Artiouchine-Baptiste propagator nor with ours.

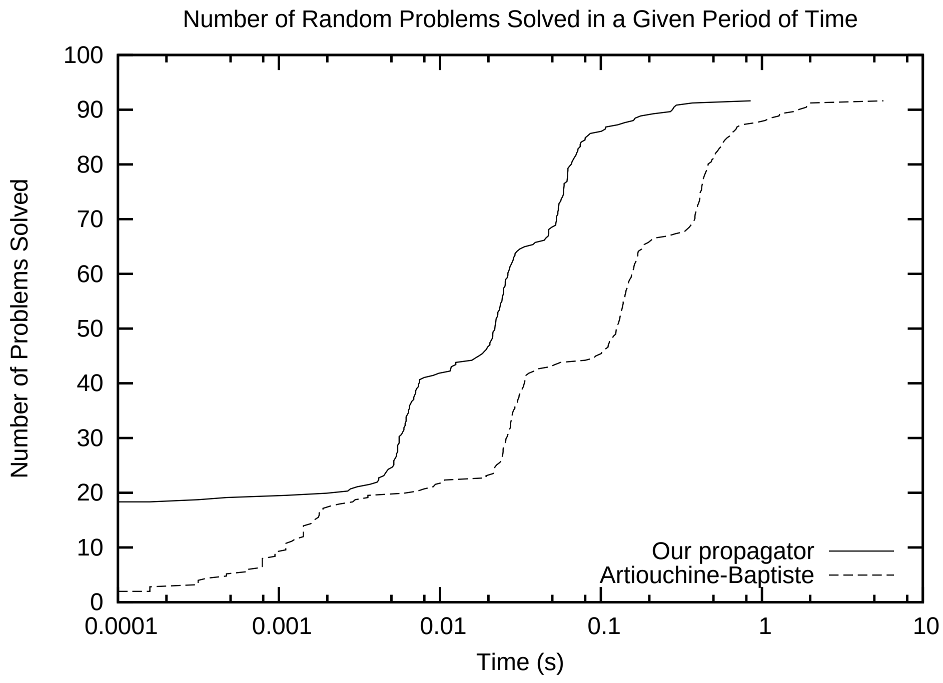

Figure 4: Number of random problems from the benchmark that were solved in a given period of time.

We then consider the runway scheduling problem where all intervals have the same length. Over both series of problems available in the benchmark Artiouchine and Baptiste (2005), we obtain an improvement over the Artiouchine-Baptiste propagator proportional to n. This observation is compatible with the running time complexities of the algorithms. Figure 5 shows the number of problems solved in a given period of time. Again, the two levels of consistency provided in ILOG for the *IlcAllMinDistance* constraint were not competitive.

7. Conclusion

We presented a new propagator achieving bounds 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 bounds consistency. It would also be interesting to study how the constraint could be generalized for the cumulative scheduling problem.

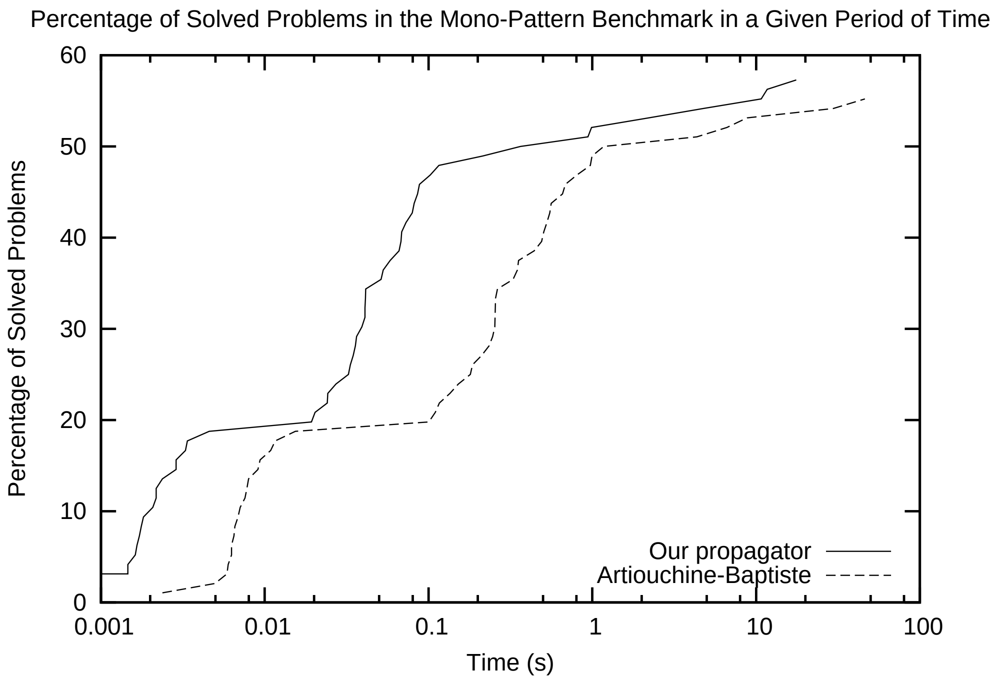

Figure 5: Number of problems with equal intervals from the benchmark that were solved in a given period of time.

References

K. Artiouchine and P. Baptiste. 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*, pages pp. 62–76, 2005.

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

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

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.

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

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

K. Mehlhorn and S. Thiel. 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*, pages pp. 306–319, 2000.

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

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

C.-G. Quimper, A. López-Ortiz, and G. Pesant. A quadratic propagator for the inter-distance constraint. In *Proceedings of the 21rst National Conference on Artificial Intelligence (AAAI 06)*, pages pp. 123–128, 2006.

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

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