The Multi-Inter-Distance Constraint

Pierre Ouellet
Université Laval
Département d'informatique et de génie logiciel
pierre.ouellet.4@ulaval.ca

Claude-Guy Quimper
Université Laval
Département d'informatique et de génie logiciel
claude-guy.quimper@ift.ulaval.ca

Abstract

We introduce the MULTI-INTER-DISTANCE constraint that ensures no more than m variables are assigned to values lying in a window of p consecutive values. This constraint is useful for modeling scheduling problems where tasks of processing time p compete for m identical resources. We present a propagator that achieves bounds consistency in cubic time. Experiments show that this new constraint offers a much stronger filtering than an edge-finder and that it allows to solve larger instances of the runway scheduling problem.

Introduction

Constraint solvers offer many ways to model scheduling problems. The constraint ALL-DIFFERENT (Régin 1994) ensures that at most one task starts at a given time. When there are m resources, the GCC (Régin 1996) limits the number of tasks starting at a given time to m. These two constraints are useful when tasks have a unit processing time. When processing times are greater than one unit, the INTER-DISTANCE (Artiouchine & Baptiste 2005; Quimper, López-Ortiz, & Pesant 2006) constraint makes sure that the starting times are at least p units of time apart. When processing times differ for each task, the CUMULATIVE (Mercier & Hentenryck 2008; Vilím 2009) constraint allows to model this situation. All constraints, except the CUMULATIVE constraint, offer a complete filtering of the release times and deadlines of the task in polynomial time. This filtering is NP-Hard for the CUMULATIVE constraint. However, there exists no filtering algorithm that prunes the release times and deadlines when the processing time of each task is p and when there are $m > 1$ resources. We introduce the MULTI-INTER-DISTANCE constraint and its first filtering algorithm achieving bounds consistency. This constraint complements the offering of constraint solvers. We show that this constraint is very efficient to solve problems such as the runway scheduling problem.

The MULTI-INTER-DISTANCE Constraint

The MULTI-INTER-DISTANCE constraint limits the number of variables assigned in a window of p consecutive values to m. Definition 1 formally presents the constraint.

Definition 1 MULTI-INTER-DISTANCE$([X_1, \ldots, X_n], m, p)$ *is satisfied if and only if at most m variables X_i are assigned to a value in any window of p consecutive values. More formally, we have the following logical identity.*

$$\text{MULTI-INTER-DISTANCE}([X_1, \ldots, X_n], m, p)$$
$$\iff$$
$$\forall v \, |\{i \mid X_i \in [v, v+p)\}| \leq m$$

The MULTI-INTER-DISTANCE is a generalization of well known constraints. When $p = 1$ and $m = 1$, the MULTI-INTER-DISTANCE constraint specializes into an ALL-DIFFERENT constraint. When $p = 1$ and $m > 1$, the MULTI-INTER-DISTANCE encodes a GCC where each value can be assigned to at most m variables. Finally, with $m = 1$ and $p > 1$, the MULTI-INTER-DISTANCE becomes an INTER-DISTANCE constraint. On the other hand, the MULTI-INTER-DISTANCE constraint is a specialization of the CUMULATIVE constraint. Each variable can be seen as the starting time of a task that consumes one unit of resource during a period of time p where the cumulative capacity of the resources is m.

Two results emerge from these relations among the constraints. Since it is NP-Hard to enforce domain consistency on the INTER-DISTANCE constraint (Artiouchine, Baptiste, & Dürr 2004), it is necessarily NP-Hard to enforce domain consistency on the MULTI-INTER-DISTANCE. Because the edge-finder (Mercier & Hentenryck 2008; Vilím 2009) does not enforce bounds consistency on the INTER-DISTANCE (Artiouchine & Baptiste 2005), it does not enforce bounds consistency on the MULTI-INTER-DISTANCE either.

We show how to detect feasibility of the MULTI-INTER-DISTANCE constraint in quadratic time. We then show how bounds consistency can be enforced in cubic time.

General Background

We consider constraint satisfaction problems where variables are assigned to integer values. The domain of a variable X_i is denoted $\text{dom}(X_i)$. We assume that the domain of a variable $\text{dom}(X_i) = [l_i, u_i)$ is a semi-open interval where l_i and u_i are called the lower bound and upper bound of the domain. The value of the variable X_i must be greater than or equal to l_i and must be strictly smaller than u_i.

An *interval support* for a value $v \in \text{dom}(X_i)$ with respect to a constraint $C(X_1, \ldots, X_n)$ is a tuple t satisfying three conditions for $j \in 1..n$: 1) $l_j \leq t[j] < u_j$, 2) $t[i] = v$ and 3) the tuple $(t[1], \ldots, t[n])$ satisfies the constraint C. A constraint is bounds consistent if l_i and $u_i - 1$ have an interval support for all $i = 1..n$. Enforcing bounds consistency on a constraint consists of increasing the lower bounds and decreasing the upper bounds of the domains until all bounds have an interval support. Example 1 shows the result of enforcing bounds consistency on the MULTI-INTER-DISTANCE constraint.

Example 1 *We consider the constraint* MULTI-INTER-DISTANCE$([X_1, X_2, X_3, X_4, X_5], m, p)$ *with* $m = 2$*,* $p = 3$*, and the following domains.*

$$\text{dom}(X_1) = [7, 9) \quad \text{dom}(X_2) = [2, 4) \quad \text{dom}(X_3) = [4, 7)$$
$$\text{dom}(X_4) = [2, 7) \quad \text{dom}(X_5) = [3, 5)$$

The domains are semi-open intervals which implies that the value 1 belongs to the domain of X_1 *but the value* 9 *does not belong to the domain of* X_1*.*

Enforcing bounds consistency on the MULTI-INTER-DISTANCE *shrinks the domains as follows.*

$$\text{dom}(X_1) = [8, 9) \quad \text{dom}(X_2) = [2, 3) \quad \text{dom}(X_3) = [5, 7)$$
$$\text{dom}(X_4) = [5, 7) \quad \text{dom}(X_5) = [3, 4)$$

Testing for Satisfaction in Polynomial Time

The MULTI-INTER-DISTANCE has a strong connection to scheduling problems. Consider the constraint MULTI-INTER-DISTANCE$([X_1, \ldots, X_n], m, p)$ where the domain of the variable X_i is the semi-open interval $[l_i, u_i)$. The variable X_i can be considered as the starting time of a task whose processing time is p, whose release time is l_i, and whose deadline is $u_i + p - 1$. A total of m identical resources execute the tasks concurrently without preemption. The MULTI-INTER-DISTANCE constraint is satisfiable if and only if there exists a schedule.

This scheduling problem can be solved in polynomial time. The best algorithm (López-Ortiz & Quimper 2011) so far has a worst-case time complexity of $O(n^2 \min(1, \frac{p}{m}))$. The algorithm finds a valid schedule by computing a shortest path in a *scheduling graph* (Dürr & Hurand 2009). This section provides a description of this graph and its properties. The scheduling graph is the key concept on which the filtering algorithm is based.

Let $l_{\min} = \min_i l_i$ and $u_{\max} = \max_i u_i$ be the smallest lower bound and the greatest upper bound of a variable domain. The *scheduling graph* G has a node v for every integer $l_{\min} \leq v \leq u_{\max}$. There is a *forward edge* of weight m from node v to node $v + p$ for all $l_{\min} \leq v \leq u_{\max} - p$. For every pair of lower bound and upper bound $l_i < u_j$, there is a *backward edge* (u_j, l_i) of weight $-|\{k \mid l_i \leq l_k \wedge u_k \leq u_j\}|$. The absolute value of the weight of the backward edge (u_j, l_i) is the number of variables whose domain is contained in the interval $[l_i, u_j)$. A backward edge can have a null weight. There is a *null edge* from any node $v + 1$ to node v of weight 0 for all $v \in [l_{\min}, u_{\max})$. Finally, there is an edge $(l_{\min}, u_{\max})$ of weight n where n is the arity of the MULTI-INTER-DISTANCE constraint.

$\alpha \backslash \beta$	2	3	4	5	6	7	8	9
2				2				
3	0				2			
4	-1	0				2		
5	**-2**	**-1**	**0**				2	
6				0				2
7	-4	-2	-1		0			
8						0		
9	-5	-3	-2			-1	0	

Table 1: The weight matrix of the scheduling graph of Example 1. Empty entries indicate the lack of an edge between nodes α and β.

.

Table 1 shows the weight matrix of the scheduling graph of the problem depicted in Example 1. The scheduling graph has important properties that we state here. Theorem 1 provides the sufficient and necessary condition to test whether the scheduling problem has a solution.

Theorem 1 ((López-Ortiz & Quimper 2011)) *The scheduling problem has a solution if and only if the scheduling graph has no negative cycles.*

Theorem 2 ((López-Ortiz & Quimper 2011)) *Testing whether there is a negative cycle in the scheduling graph can be done in* $O(n^2 \min(1, \frac{p}{m}))$ *time.*

Theorems 1 and 2 allow to test the satisfiability of the constraint in quadratic time. In the next sections, we reuse this result to design an efficient filtering algorithm.

Conditions to Maintain Bounds Consistency

Our filtering algorithm detects forbidden sets of values that cannot be assigned to a set of variables. For the ALL-DIFFERENT constraint, these sets are called Hall intervals (Leconte 1996). For the MULTI-INTER-DISTANCE, we call them *forbidden regions*. We present lemmas that detect these forbidden regions.

To test whether an interval $[a, b)$ is a forbidden region to which a variable X_i is subject, one can modify the domain of X_i to be $[a, b)$ and test the satisfiability of the problem. If the scheduling graph of the transformed problem has a negative cycle, then $[a, b)$ is a forbidden region for X_i.

We define the *altered scheduling graph* G_i^v like the scheduling graph with one exception, we alter the upper bound of the domain of X_i such that the new domain becomes $[l_i, v)$. If G_i^v has a negative cycle, the constraint has no solution and we conclude that $X_i \geq v$. Table 1 gives the weight matrix of the graph G. Decrementing bold entries by one results in the altered scheduling graph G_3^5. In the graph G_3^5, the edges $(5, 2)$ and$(2, 5)$ form a cycle of weight -1 which proves that $X_3 \geq 5$. The following theorem shows how a forbidden region for one variable can also be a forbidden region for another variable.

Theorem 3 *Let X_i and X_j be two variables whose domains satisfy $u_i \leq u_j$. If the values in $[l_i, v)$ have no interval support in* $\text{dom}(X_i)$*, then they do not have an interval support in* $\text{dom}(X_j)$ *either.*

Proof: We prove the contraposition: if there exists a value $a \in [l_i, v)$ that has an interval support in the domain of X_j then there exists a value $b \in [l_i, v)$ that has an interval support in the domain of X_i. Suppose there exists an interval support t such that $t[j] \in [l_i, v)$, we want to prove that there exists an interval support t' such that $t'[i] \in [l_i, v)$. Two cases can occur. If $t[i] \leq t[j]$ then the inequalities $l_i \leq t[i] \leq t[j] < v$ hold and $t[i] \in [l_i, v)$. The theorem holds with $t' = t$. However, if $t[i] > t[j]$ the inequalities $l_j \leq t[j] < t[i] \leq u_i \leq u_j$ hold. We can permute the values $t[i]$ and $t[j]$ to obtain the support t'. We have $t'[i] = t[j] \in [l_i, v)$ and $t'[j] = t[i] \in [l_j, u_j)$ hence t' is a valid support. □

In Example 1, the graph G_3^5 has a negative cycle which proves that $[4, 5)$ is a forbidden region for X_3. Theorem 3 also makes $[4, 5)$ a forbidden region for the variable X_4.

The distance matrix D_G of a graph G is a square matrix such that $D_G[a, b]$ is the shortest distance between node a and node b in graph G. This matrix exists only if there are no negative cycles in the graph G. When comparing the scheduling graph G of Example 1 and the altered scheduling graph G_3^5, one sees that the weights of a subset of the edges are decremented by one. This can only lead to shorter paths in G_3^5 than in G. Lemma 1 captures this property.

Lemma 1 *Let G be the scheduling graph of a problem where the domains are given by the intervals $[l_i, u_i)$ for $1 \leq i \leq n$. Let G' be the scheduling graph of the same problem except that the domain of X_i is changed to $[l'_i, u'_i)$ where $l_i \leq l'_i < u'_i \leq u_i$ holds. Thus $D_G[a, b] \geq D_{G'}[a, b]$ holds for all pairs of nodes (a, b).*

Proof: The weight of a backward edge (u_j, l_k) is minus the number of variable domains contained in the interval $[l_k, u_j)$. If the domain of a variable is shrunk, the weight of the backward edges can be decremented by one or remain unchanged. These changes to the scheduling graph can only favor shorter paths.

However, the backward edges outgoing from u_i or incoming to l_i may not exist in G' as u_i and l_i may not be domain bounds after shrinking the domain of X_i. Let (u_f, l_e) be a backward edge in G such that $f = i \vee e = i$. We prove that the edge (u_f, l_e) in G can be replaced by an equivalent path in G'_i. Let $u'_f = \max(\{u_k \mid u_k \leq u_f \wedge k \neq i\} \cup \{u'_i\})$ and let $l'_e = \min(\{l_k \geq l_e \wedge k \neq i\} \cup \{l'_i\})$. The weight of the backward edge (u'_f, l'_e) in G'_i is the same as the weight of the backward edge (u_f, l_e) in G. Moreover, we have $l_e \leq l'_e$ and $u'_f \leq u_f$. Any path in G passing by the edge (u_f, l_e) can be replaced by a sequence of null edges from u_f to u'_f, then passing by the edge (u'_f, l'_e), and completing with a sequence of null edges from l'_e to l_e. This path in G'_i has the same weight as the edge (u_f, l_e) in G. □

Let $U = \{u_i \mid i \in 1..n\}$ be the set of the domain upper bounds and let $u^* = \min\{u \in U \mid u > l_i\}$ be the smallest upper bound that does not make the interval $[l_i, u^*)$ empty. The altered scheduling graph $G_i^{u^*}$ gives information about how the lower bound of X_i can be filtered. If $G_i^{u^*}$ has a negative cycle, then $X_i \geq u^*$ and the lower bound l_i can be safely increased to u^*. If $G_i^{u^*}$ has no negative cycle, then Theorem 4 and Theorem 5 indicates how to prune the lower bound of variable X_i.

Theorem 4 *Let X_i be a variable subject to a* MULTI-INTER-DISTANCE *constraint. Let u^* be the smallest upper bound that is greater than l_i, and let $G_i^{u^*}$ be its associated altered scheduling graph. Suppose this graph has no negative cycle. Let t be the largest value such that the distance $D_{G_i^{u^*}}[l_i, t]$ is null. Values smaller than t in the domain of X_i have no interval support.*

Proof: Since $G_i^{u^*}$ has no negative cycles, there exists at least one solution with $X_i \in [l_i, u^*)$ where $u^* = \min\{u_k \mid u_k > l_i\}$. Let t be the largest value for which the shortest distance in $G_i^{u^*}$ from l_i to t is null. We prove by contradiction that $t < u^*$. Suppose $t \geq u^*$. The weight of the edge (u^*, l_i) is at most -1 since it fully contains the altered domain of X_i. Contradictorily, there is a negative cycle in $G_i^{u^*}$ starting from l_i, going to t through a path of weight zero (by definition of t), passing by the null edges $(t, t-1), (t-1, t-2), \ldots, (u^*+1, u^*)$ with weight zero, and then going back to l_i using the edge (u^*, l_i) with weight at most -1. Therefore, $t < u^*$.

Knowing that $t < u^*$, we construct the altered scheduling graph G_i^t using the original variable domains except for X_i for which we use the domain $\text{dom}(X_i) = [l_i, t)$. The edge (t, l_i) has a negative weight since the interval $[l_i, t)$ contains the domain of X_i. The distance from l_i to t is null in $G_i^{u^*}$ and, by Lemma 1, no greater than 0 in G_i^t. Therefore, there is a negative cycle in G_i^t passing by the edge (t, l_i) which implies that no values in $[l_i, t)$ have an interval support in $\text{dom}(X_i)$. □

To illustrate Theorem 4, consider the variable X_1 in Example 1. The smallest upper bound that is greater than the lower bound 7 is $u^* = 9$. The altered scheduling graph G_1^9 is identical to the scheduling graph G. The path $(7, 2), (2, 5), (5, 8)$ has a null weight in G_1^9. By Theorem 4, there is no valid assignment with $X_1 < 8$.

Theorem 4 shows there is no interval support $X_i \in [l_i, t)$. Theorem 5 shows that there is an interval support for $X_i = t$.

Theorem 5 *Let X_i be a variable subject to a* MULTI-INTER-DISTANCE *constraint. Let u^* be the smallest upper bound that is greater than l_i, and let $G_i^{u^*}$ be its associated altered scheduling graph. Suppose this graph has no negative cycle. Let t be the largest value such that the distance $D_{G_i^{u^*}}[l_i, t]$ is null. The value t has an interval support in the domain of X_i.*

Proof: We construct the scheduling graph G_i^{t+1} using the original variable domains except for X_i for which we use the domain $[l_i, t+1)$. If one proves that G_i^{t+1} has

no negative cycle, then one proves there is a solution with $X_i \in [l_i, t+1)$. Theorem 4 ensures that $X_i \geq t$ leaving $X_i = t$ as the unique valid assignment for the variable X_i. The graph G_i^{t+1} has no negative cycles when $t+1 = u^*$. Indeed, in that specific case, the graphs G_i^{t+1} and $G_i^{u^*}$ are by construction identical. From the proof of Theorem 4, we know that $t \leq u^*$. We therefore need to prove that G_i^{t+1} has no negative cycles when $t+1 < u^*$.

Let u_q be the greatest upper bound that is strictly smaller than u^*. By construction, there is no upper bounds greater than l_i and smaller than u^*, thus $u_q < l_i$. The null edges form a path from l_i to u_q which implies $D_{G_i^{t+1}}[l_i, u_q] \leq 0$. Consider any lower bound l_p that is smaller than or equal to l_i. The domain $\text{dom}(X_i) = [l_i, t+1)$ is the only one that is not contained in the interval $[l_p, u_q)$ but is contained in the interval $[l_p, t+1)$, thus $-w(u_q, l_p) = -w(t+1, l_p) - 1$.

The graphs $G_i^{u^*}$ and G_i^{t+1} are very similar. They only differ by the additions of backward edges $(t+1, l_q)$ for all lower bound $l_q \leq l_i$. It is therefore sufficient to prove that any cycle passing by these edges has a non-negative weight.

Since the addition of the edges leaving $t+1$ does not affect the paths leading to $t+1$, we have $D_{G_i^{t+1}}[l_i, t+1] = D_{G_i^{u^*}}[l_i, t+1]$. By the definition of t, we obtain $0 < D_{G_i^{u^*}}[l_i, t+1] = D_{G_i^{t+1}}[l_i, t+1]$. The consequence of this inequality is that any path going from l_i to $t+1$ has a positive weight and in particular, a path going from l_i to $t+1$ passing by the edge (u_q, l_p) has positive weight.

$$0 < D_{G_i^{t+1}}[l_i, u_q] + w(u_q, l_p) + D_{G_i^{t+1}}[l_p, t+1] \quad (1)$$

By substituting $-w(u_q, l_p) = -w(t+1, l_p) - 1$ and $D_{G_i^{t+1}}[l_i, u_q] \leq 0$ in (1) we prove that all cycles passing by the edge $(t+1, l_p)$ are non-negative.

$$0 < 0 + w(t+1, l_p) + 1 + D_{G_i^{t+1}}[l_p, t+1] \quad (2)$$

$$0 \leq w(t+1, l_p) + D_{G_i^{t+1}}[l_p, t+1] \quad (3)$$

□

In Example 1, the path $(7,2), (2,5), (5,8)$ in G_1^9 has a null weight which certifies that there exists an interval support for 8 in the domain of X_1. This interval support is $t = [8, 2, 5, 6, 3]$.

A Filtering Algorithm Achieving Bounds Consistency

The filtering algorithm for the MULTI-INTER-DISTANCE is a direct application of Theorem 3, Theorem 4, and Theorem 5. The algorithm on Figure 1 processes the variables in non-decreasing order of domain upper bounds. When processing X_i, the algorithm updates the lower bound l_i according to previously discovered fobidden regions. Then it computes the value u^* that is the smallest domain upper bound greater than l_i. Based on this value, the algorithm tests whether the scheduling graph $G_i^{u^*}$ has a negative cycle. If so, the algorithm stores in the data structure F the newly detected forbidden region $[l_i, u^*)$. Because the algorithm processes variables in non-decreasing order of upper bounds, Theorem 3 ensures that the interval $[l_i, u^*)$ is a forbidden region for all unprocessed variables. On its next iteration, the lower bound l_i is increased to avoid all forbidden regions, including the newly discovered one. This process continues until the scheduling graph $G_i^{u^*}$ contains no negative cycles.

When the scheduling graph $G_i^{u^*}$ contains no negative cycles, Theorem 4 and Theorem 5 apply. The algorithm computes the shortest path from node l_i to all the other nodes and finds the node t which is the largest one whose distance from node l_i is null. From Theorem 4, we know that values smaller than t have no interval support in the domain of the current variable. The algorithm removes them from the domain by increasing the value of l_i. Finally, Theorem 5 guarantees that the lower bound of each variable domain has an interval support upon completion of the algorithm.

Algorithm 1: PruneLowerBounds($[X_1, \ldots, X_n]$)

```
Sort the variables such that u_i ≤ u_{i+1}
U ← {u_1, ..., u_n}
F ← ∅
for i ∈ 1..n do
    repeat
        l_i ← min([l_i, u_i) \ F)
        u* ← min((l_i, u_i] ∩ U)
        Let G_i^{u*} be the altered scheduling graph with
        dom(X_i) = [l_i, u*)
        Compute the shortest distances D_{G_i^{u*}}[l_i, t] from
        node l_i to all the other nodes t
        if there is a negative cycle in G_i^{u*} then
            F ← F ∪ [l_i, u*)
    until there is no negative cycles in G_i^{u*}
    l_i ← max({t | D_{G_i^{u*}}[l_i, t] = 0})
```

Computing the single-source shortest path in the scheduling graph should be done with care. Indeed, the scheduling graph has as many nodes as the number of values in the domains. These domains can be very large especially in scheduling problems where domains represent time intervals. (López-Ortiz & Quimper 2011) showed how to compute the shortest path from the right-most node to all the other nodes in $O(n^2 \min(1, \frac{p}{m}))$ steps which is strongly polynomial and invariant in the cardinality of the variable domains. The algorithm is an adaptation of the Bellman-Ford algorithm (Cormen *et al.* 2001) which initializes a distance vector that keeps track of the distances from the node l_i to all the other nodes. Since the right-most node can reach all the other nodes via the null edges, the distance vector is initialized to zero and as new paths are explored, the distance vector is updated with smaller values. To compute the shortest path from l_i to all the other nodes, the distance vector should be initialized to zero for all the nodes smaller than or equal to l_i and n for all the nodes that are greater than l_i. Indeed, any node t can be reached by going to $l_{\min}$ through the null edges, then going to $u_{\max}$ using the edge $(l_{\min}, u_{\max})$, and finally using the null edges to go to t. This

simple adaptation to the shortest path algorithm is sufficient to compute the shortest path from the source node l_i to all the other nodes or to detect a negative cycle. The running time complexity remains $O(n^2 \min(1, \frac{p}{m}))$.

The computation of shortest paths dominates the algorithms's complexity.

Lemma 2 *The algorithm PruneLowerBounds performs at least n and at most $3n-1$ shortest path computations where n is the number of variables.*

Proof: Computing the shortest path of an altered scheduling graph leads to two possible outcomes: the graph has negative cycles or it does not. If the graph has no negative cycles, the *repeat* loop terminates. This occurs exactly n times, once for each variable. If the graph has negative cycles, the algorithm loops and create the forbidden region $[l_i, u^*)$. The value u^* is a domain upper bound. Each time a new forbidden region is created, one of two events can occur: 1) a domain upper bound becomes for the first time the open upper bound of a forbidden region 2) a domain upper bound gets included in forbidden region. These two events can occur only once for each upper bound at the exception of the largest upper bound $u_{\max}$ for which only the first event can occur. Therefore, the algorithm creates at most $2n-1$ forbidden regions and performs at most $3n-1$ shortest path computations. □

The shortest path problem is solved in $O(n^2 \min(1, \frac{p}{m}))$ steps when using the algorithm of (López-Ortiz & Quimper 2011) and modifying the initial state of the distance vector as described above. The total running time complexity of the filtering algorithm is therefore $O(n^3 \min(1, \frac{p}{m}))$ which is never more than $O(n^3)$.

Pruning the upper bounds can be done by pruning the lower bounds in a symmetric problem where $\text{dom}(X'_i) = [-u_i, -l_i)$. When pruning the lower bounds, the value $-u_i$ is increased which is equivalent to decrease the upper bound u_i in the original problem.

Experiments

We experiment the MULTI-INTER-DISTANCE on the runway scheduling problem as presented in (Artiouchine & Baptiste 2005). This problem consists of determining the order in which n airplanes should land on a runway. We consider an airport with m runways allowing simultaneous landings. Each airplane labeled from 1 to n has multiple disjoint time windows $[a^i_1, b^i_1], \ldots, [a^i_{k_i}, b^i_{k_i}]$ in which it is in position to land. The goal is to make each airplane land within one of its time windows while maximizing the time gap p between each landing on a same runway.

We model the problem with two variables per airplane. The variable S_i defines at what time the airplane i lands. The variable $W_i \in [1, k_i]$ defines in which time window the airplane i lands. The two variables are connected with the constraints $W_i \leq w \iff S_i \leq b^i_w$ and $W_i \geq w \iff S_i \geq a^i_w$. Finally, there is a MULTI-INTER-DISTANCE posted on the vector of variables S with fixed parameter p and m. We solve the problem for $p = 1, 2, 4, 8, \ldots$ until the problem becomes infeasible. We then perform a binary search between the largest p for which there exists a solution and the smallest p for which we detect infeasibility. We branch on the variables W_i followed by the variables S_i.

For $m = 1$, we use the benchmark described as the second set in (Artiouchine & Baptiste 2005). There are 64 instances for each $n \in \{15, 30, 45, 60, 75, 90\}$. The number of time windows per airplane randomly varies from 1 to 5.

We generate instances for $m = 2$ and $m = 3$ by merging instances from the original benchmark. As a rule, we merge instances having the same size of time windows or the same gap between the time windows. Merged instances also have a similar number of airplanes. As m increases, we obtain instances with more airplanes. We implement our solution (denoted *MID*) using the Choco Solver and compare it against two implementations of the cumulative constraint that the solver provides. The first implementation performs an overload checking (denoted *OC*) in $O(n \log n)$ using a Θ-tree as designed by (Vilím, Barták, & Čepek 2004). The second implementation (denoted *MVH*) is an implementation of the edge-finder (Mercier & Hentenryck 2008) in $O(n^2)$. Experiments are run arbitrarily on an Opteron 275 2.2 GHz and an Opteron 2376 2.3 GHz . Running times may vary depending on which processor executes the program.

Figure 1 shows that MID solves more problems in a given time for $m \in \{1, 2, 3\}$ than MVH and OC. The difference in performance becomes more prominent as m increases. For $m = 3$, MID solves 134 instances in 20 minutes, MVH solves 3 instances, and OC solves none. Table 2 shows that MID solves the problem with significantly fewer backtracks. MVH performs fewer backtracks as n increases indicating that the filtering gets slower and fewer nodes are visited within 20 minutes.

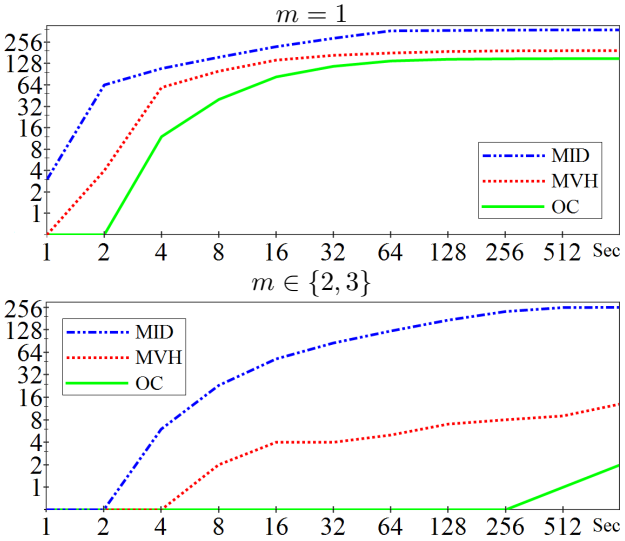

Figure 1: Number of solved problems vs time.

	$m=1$						$m=2$				$m=3$			
n	OC		MVH		MID		MVH		MID		MVH		MID	
15	94%	999194	88%	106325	17%	22								
30	95%	2755211	88%	332083	16%	333	100%	722423	31%	177				
45	97%	2402617	97%	306429	22%	1040	100%	322937	31%	4607	100%	376261	31%	4
60	100%	1583362	97%	238024	22%	1515	100%	215728	15%	543	100%	213391	50%	2840
75	98%	1101707	94%	152017	28%	3111	100%	129688	33%	1436	100%	120684	38%	128
90	98%	927161	98%	106661	36%	238	100%	82496	29%	4781	100%	86733	33%	61
105							100%	66912	56%	1619	100%	60971	19%	1532
120							100%	58777	50%	785	100%	44511	25%	2583
135							100%	46525	25%	1236	100%	38623	44%	451
150							100%	29326	44%	1324	100%	27765	44%	518
165											100%	25181	47%	775

Table 2: Percentage of problems that generated backtracks and average number of backtracks for a period of 20 minutes for each of these problems.

Conclusion

We introduced the MULTI-INTER-DISTANCE constraint. We made the connection between this constraint and the properties of the shortest paths in the scheduling graph. We applied these properties in the design of the first filtering algorithm achieving bounds consistency. This new constraint is very successful at solving large instances of the runway scheduling problem. The theorems we proved about the MULTI-INTER-DISTANCE constraint may lead in the future to the design of better filtering algorithms for related constraints such as the INTER-DISTANCE and the CUMULATIVE constraints.

Acknowledgments

We would like to thank Arnaud Malapert for his support with the Choco solver and the reviewers for helping us simplifying the proof of Theorem 3. This research is supported by a NSERC Discovery Grant.

References

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

Artiouchine, K.; Baptiste, P.; and Dürr, C. 2004. Runway scheduling with holding loop. In *Proc. of 2nd Int. Workshop on Discrete Optimization Methods in Production and Logistics*, 96–101.

Cormen, T. H.; Stein, C.; Rivest, R. L.; and Leiserson, C. E. 2001. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition.

Dürr, C., and Hurand, M. 2009. Finding total unimodularity in optimization problems solved by linear programs. *Algorithmica*. DOI 10.1007/s00453-009-9310-7.

Leconte, M. 1996. A bounds-based reduction scheme for constraints of difference. In *Proc. of the Constraint Int. Workshop on Constraint-Based Reasoning*, 19–28.

López-Ortiz, A., and Quimper, C.-G. 2011. A fast algorithm for multi-machine scheduling problems with jobs of equal processing times. In *In 28th Int. Symp. on Theoretical Aspects of Computer Science (to appear in STACS'11)*.

Mercier, L., and Hentenryck, P. V. 2008. Edge finding for cumulative scheduling. *INFORMS Journal on Computing* 20(1):143–153.

Quimper, C.-G.; López-Ortiz, A.; and Pesant, G. 2006. A quadratic propagator for the inter-distance constraint. In *Proc. of the 21st Nat. Conf. on Artificial Intelligence (AAAI 06)*, 123–128.

Régin, J.-C. 1994. A filtering algorithm for constraints of difference in CSPs. In *Proc. of the 11th National Conference on Artificiel Intelligence (AAAI-94)*, 362–367.

Régin, J.-C. 1996. Generalized arc consistency for global cardinality constraint. In *Proc. of the 8th Annual Conference on Innovative Applications of Artificial Intelligence*, 209–215.

Vilím, P.; Barták, R.; and Čepek, O. 2004. Unary resource constraint with optional activities. In *Proc. of the 10th Int. Conference on Principles and Practice of Constraint Programming*, 62–76.

Vilím, P. 2009. Edge finding filtering algorithm for discrete cumulative resources in o(kn log n). In *Proc. of the 15th Int. Conference on Principles and Practice of Constraint Programming*, 802–816.