# The Unary Resource with Transition Times

Cyrille Dejemeppe, Sascha Van Cauwelaert, Pierre Schaus

UCLouvain, ICTEAM,
Place Sainte Barbe 2,
1348 Louvain-la-Neuve, Belgium
{firstname.lastname}@uclouvain.be

**Abstract.** Transition time constraints are ubiquitous in scheduling problems. They are said to be sequence-dependent if their durations depend on both activities between which they take place. In this context, we propose to extend the $\Theta$-tree and $\Theta$-$\Lambda$-tree data structures introduced by Vilím in order to strengthen the bound computation of the earliest completion time of a set of activities, by taking into account the sequence dependent transition time constraints. These extended structures can be substituted seamlessly in the state-of-the-art Vilím's filtering algorithms for unary resource constraints (Overload Checking, Detectable Precedences, Not-First/Not-Last and Edge-Finding algorithms) without changing their $\mathcal{O}(n \log(n))$ time complexities. Furthermore, this new propagation procedure is totally independent from additional constraints or the objective function to optimize. The proposed approach is able to reduce the number of nodes by several order of magnitudes on some instances of the job-shop with transition times problem, without introducing too much overhead on other instances for which it is less effective.

**Keywords:** Scheduling, transition times, global constraints, Constraint Programming

## 1 Introduction

This work extends the classic unary/disjunctive resource propagation algorithms to include propagation over *sequence-dependent* transition times between activities. A wide range of real-world scheduling problems from the industry involves transition times between activities. An example is the quay crane scheduling problem in container terminals [22] where the crane is modeled as a unary resource and transition times represent the moves of the crane on the rail to move from one position to another along the vessel to load/unload containers.

We introduce filtering algorithms to tighten the bounds of (non-preemptive) activities while taking into account the transition times between them. These filtering algorithms are extensions of the unary resource propagation algorithms (Overload Checking, Detectable Precedences, Not-First/Not-Last, Edge-Finding) introduced in [18]. All these algorithms rely on an efficient computation of the earliest completion time (ect) of a group of activities using the so-called Theta

tree and Theta-Lambda tree data structures. We demonstrate the efficiency of the filtering on job-shop with transition times problem instances.

In Section 2, we give an overview of the tackled problems and of current state-of-the-art techniques to solve them. In Section 3, we explain the requirements needed to integrate transition times propagation. Section 4 explains how to obtain lower bounds for the time spent by transitions between activities from a set. Then, Section 5 describes how to integrate this bound to efficiently compute the *ect* of a set of activities with extended $\Theta$-tree structures. Section 6 then explains how classic unary algorithms can consider transition times by using the extended $\Theta$-tree structures. Finally, we report results obtained by the new propagation procedure in Section 7.

## 2 Background

In Constraint Programming (CP), a scheduling problem is modeled by associating three variables to each activity $A_i$: $\text{start}_i$, $\text{end}_i$, and $\text{duration}_i$ representing respectively the starting time, ending time and processing time of $A_i$. These variables are linked together by the following relation:

$$\text{start}_i + \text{duration}_i = \text{end}_i$$

Depending on the considered problem, global constraints linking the activity variables are added to the model. In this work, we are interested in the unary resource constraint. A *unary resource*, sometimes referred to as a *machine*, is a resource allowing only a single activity to use it at any point in time. As such, all activities demanding the same unary resource cannot overlap in time:

$$\forall i, j \;\; i \neq j : (\text{end}_i \leq \text{start}_j) \vee (\text{end}_j \leq \text{start}_i)$$

The unary resource can be generalized by requiring transition times between activities. A transition time $tt_{i,j}$ is a minimal amount of time that must occur between two activities $A_i$ and $A_j$ if $A_i \prec A_j$ (precedes). These transition times are described in a matrix $\mathcal{M}$ in which the entry at line $i$ and column $j$ represents the minimum transition time between $A_i$ and $A_j$, $tt_{i,j}$. We assume that transition times respect the triangular inequality. That is, inserting an activity between two activities always increases the time between these activities:

$$\forall i, j, k \;\; i \neq j \neq k : tt_{i,j} \leq tt_{i,k} + tt_{k,j}$$

The unary resource with transition times imposes the following relation:

$$\forall i, j : (\text{end}_i + tt_{i,j} \leq \text{start}_j) \vee (\text{end}_j + tt_{j,i} \leq \text{start}_i) \tag{1}$$

### 2.1 Related Work

As described in a recent survey [2], scheduling problems with transition times can be classified in different categories. First the activities can be in *batch* (i.e.

a machine allows several activities of the same batch to be processed simultaneously) or not. Transition times may exist between successive batches. A CP approach for batch problems with transition times is described in [18]. Secondly the transition times may be *sequence-dependent* or *sequence-independent*. Transition times are said to be sequence-dependent if their durations depend on both activities between which they occur. On the other hand, transition times are sequence-independent if their duration only depend on the activity after which it takes place. The problem category we study in this article is non-batch sequence-dependent transition time problems.

Several methods have been proposed to solve such problems. Ant Colony Optimization (ACO) approaches were used in [9] and [15] while [6], [4], [13] and [10] propose Local Search and Genetic Algorithm based methods. [13] proposes a propagation procedure with the Iterative Flattening Constraint-Based Local Search technique. The existing CP approaches for solving sequence-dependent problems are [8], [3], [21] and [11].

Focacci et al [8] introduce a propagator for job-shop problems involving alternative resources with non-batch sequence-dependent transition times. In this approach a successor model is used to compute lower-bounds on the total transition time. The filtering procedures are based on a minimum assignment algorithm (a well known lower bound for the Travelling Salesman Problem). In this approach the total transition time is a constrained variable involved in the objective function (the makespan).

In [3], a Travelling Salesman Problem with Time Window (TSPTW) relaxation is associated to each resource. The activities used by a resource are represented as vertices in a graph and edges between vertices are weighted with corresponding transition times. The TSPTW obtained by adding time windows to vertices from bounds of corresponding activities is then resolved. If one of the TSPTW is found un-satisfiable, then the corresponding node of the search tree is pruned. A similar technique is used in [5] with additional propagation.

In [21], an equivalent model of multi-resource scheduling problem is proposed to solve sequence-dependent transition times problems. Finally, in [11], a model with a reified constraint for transition times is associated to a specific search to solve job-shop with sequence-dependent transition times problems.

To the best of our knowledge, the CP filtering introduced in this article is the first one proposing to extend all the classic filtering algorithms for unary resources (Overload Checking [7], Detectable Precedences [17], Not-First/Not-Last [19] and Edge Finding [19]) by integrating transition times, independently of the objective function of the problem. This filtering can be used in any problem involving a unary resource with sequence-dependent transition times.

### 2.2 Unary Resource Propagators in CP

The earliest starting time of an activity $A_i$ denoted $est_i$, is the time before which $A_i$ cannot start. The latest starting time of $A_i$, $lst_i$, is the time after which $A_i$ cannot start. The domain of $\text{start}_i$ is thus the interval $[est_i; lst_i]$. Similarly the earliest completion time of $A_i$, $ect_i$, is the time before which $A_i$ cannot end and

the latest completion time of $A_i$, $lct_i$, is the time after which $A_i$ cannot end. The domain of $\text{end}_i$ is thus the interval $[ect_i; lct_i]$. These definitions can be extended to a set of activity $\Omega$. For example, $est_\Omega$ is defined as follows:

$$est_\Omega = \min\{est_j | j \in \Omega\}$$

The propagation procedure for the unary resource constraint introduced in [18] contains four different propagation algorithms all running with time complexity in $\mathcal{O}(n\log(n))$ : Overload Checking (OC), Detectable Precedences (DP), Not-First/Not-Last (NF/NL) and Edge Finding (EF). These propagation algorithms all rely on an efficient computation of the earliest completion time of a set of activities $\Omega$ using data structures called *Theta Tree* and *Theta-Lambda Tree* introduced in [18]. Our contribution is a tighter computation of the lower bound $ect_\Omega$ taking into account the transition times between activities.

## 3 Transition Times Extension Requirements

The propagation procedure we introduce in this article relies on the computation of $ect_\Omega$, the earliest completion time of a set of activities. This value depends on the transition times between activities inside $\Omega$. Let $\Pi_\Omega$ be the set of all possible permutations of activities in $\Omega$. For a given permutation $\pi \in \Pi_\Omega$ (where $\pi(i)$ is the activity taking place at position $i$), we can define the total time spent by transition times, $tt_\pi$, as follows:

$$tt_\pi = \sum_{i=1}^{|\Omega|-1} tt_{\pi(i),\pi(i+1)}$$

A lower bound for the earliest completion time of $\Omega$ can then defined as:

$$ect_\Omega^{NP} = \max_{\Omega' \subseteq \Omega} \left\{ est_{\Omega'} + p_{\Omega'} + \min_{\pi \in \Pi_{\Omega'}} tt_\pi \right\} \tag{2}$$

Unfortunately, computing this value is NP-hard. Indeed, computing the optimal permutation $\pi \in \Pi$ minimizing $tt_\pi$ is equivalent to solving a TSP. Since embedding an exponential algorithm in a propagator is generally impractical, a looser lower bound can be used instead:

$$ect_\Omega = \max_{\Omega' \subseteq \Omega} \{est_{\Omega'} + p_{\Omega'} + \underline{tt}(\Omega')\}$$

where $\underline{tt}(\Omega')$ is a lower bound of the total time consumed by transition times between activities in $\Omega'$:

$$\underline{tt}(\Omega') \leq \min_{\pi \in \Pi_{\Omega'}} tt_\pi$$

Our goal is to keep the overall $\mathcal{O}(n\log(n))$ time complexity of Vilím's algorithms. The lower bound $\underline{tt}(\Omega')$ must therefore be available in constant time for a given set $\Omega'$. Our approach to obtain constant time lower-bounds for a

given set $\Omega'$ during search is to base its computation solely on the cardinality $|\Omega'|$. More precisely, for each possible subset of cardinality $k \in \{1, \ldots, n\}$, we pre-compute the smallest transition time permutation of size $k$ on $\Omega$:

$$\underline{tt}(k) = \min_{\{\Omega' \subseteq \Omega:\ |\Omega'|=k\}} \left\{ \min_{\pi \in \Pi_{\Omega'}} tt_\pi \right\}$$

For each $k$, the lower bound computation thus requires to solve a resource constrained shortest path problem (also NP-hard) with a fixed number of edges $k$ and with a free origin and destination. The next section proposes several ways of pre-computing efficient lower bounds $\underline{tt}(k)$ for $k \in \{1, \ldots, n\}$. Our formula to compute a lower bound for the earliest completion time of a set of activities (making use of pre-computed lower-bounds of transition times) becomes:

$$ect^{\diamond}_{\Omega} = \max_{\Omega' \subseteq \Omega} \{est_{\Omega'} + p_{\Omega'} + \underline{tt}(|\Omega'|)\} \tag{3}$$

## 4 Lower Bound of Transitions Times

The computation of $\underline{tt}(k)$ for all $k \in \{1, \ldots, n\}$ is NP-hard. This is a constrained shortest path problem (for $k = n$ it amounts to solving a TSP) in a graph where each node corresponds to an activity and directed edges between nodes represent the transition time between corresponding activities. Although these computations are achieved at the initialization of the constraint, we propose to use polynomial lower bounding procedures instead. Several approaches are used and since none of them is dominated any other one, we simply take the maximum of the computed lower bounds.

*Minimum Weight Forest* A lower bound for $\underline{tt}(k)$ is a minimal subset of edges of size $k$ taken from this graph. We propose to strengthen this bound by using Kruskal's algorithm [12] to avoid selecting edges forming a cycle. We stop this algorithm as soon as we have collected $k$ edges. The result is a set of edges forming a minimum weight forest (i.e. a set of trees) with exactly $k$ edges.

*Dynamic Programming* We can build the layered graph with exactly $k$ layers and each layer containing all the activities. Arcs are only defined between two successive layers with the weights corresponding to the transition times. A shortest path on this graph between the first and last layer can be obtain with Dijkstra. By construction this shortest path will use exactly $k$ transitions, the relaxation being that a same activity or transition can be used several times.

*Minimum Cost Flow* Another relaxation is to keep the degree constraint but relax the fact that selected edges must form a contiguous connected path. This relaxation reduces to solving a minimum cost flow problem of exactly $k$ units on the complete bipartite graph formed by the transitions.

*Lagrangian Relaxation* As explained in Chapter 16 of [1], the Lagrangian relaxation (a single Lagrangian multiplier is necessary for the exactly $k$ transitions constraint) combined with a sub-gradient optimization technique can easily be applied to compute a lower bound on the constrained shortest path problem.

## 5 Extending the $\Theta$-tree with Transition Times

As introduced in [20], the $\mathcal{O}(n \log n)$ propagation algorithms for unary resource use the so-called $\Theta$-tree data structure. We propose to extend it in order to integrate transition times while keeping the same time complexities for all its operations.

A $\Theta$-tree is a balanced binary tree in which each leaf represents an activity from a set $\Theta$ and internal nodes gather information about the set of (leaf) activities under this node. For an internal node $v$, we denote by Leaves($v$), the leaf activities under $v$. Leaves are ordered in non-decreasing order of the *est* of the activities. That is, for two activities $i$ and $j$, if $est_i < est_j$, then $i$ is represented by a leaf node that is at the left of the leaf node representing $j$. This ensures the property :

$$\forall i \in \mathit{Left}(v), \forall j \in \mathit{Right}(v) : est_i \leq est_j$$

where left($v$) and right($v$) are respectively the left and right children of $v$, and Left($v$) and Right($v$) denote Leaves(left($v$)) and Leaves(right($v$)).

A node $v$ contains precomputed values about Leaves($v$): $\Sigma\mathrm{P}_v$ represents the sum of the durations of activities in Leaves($v$) and $ect_v$ is the *ect* of Leaves($v$). More formally, the values maintained in an internal node $v$ are defined as follows:

$$\Sigma\mathrm{P}_v = \sum_{j \in \mathrm{Leaves}(v)} p_j$$
$$ect_v = ect_{\mathrm{Leaves}(v)} = \max_{\Theta' \subseteq \mathrm{Leaves}(v)} \{est_{\Theta'} + p_{\Theta'}\}$$

For a given leaf $l$ representing an activity $i$, the values of $\Sigma\mathrm{P}_l$ and $ect_l$ are $p_i$ and $ect_i$, respectively. In [18] Vilím has shown that for a node $v$ these values only depends on the values defined in both its $\mathit{left}(v)$ and $\mathit{right}(v)$ child. The incremental update rules introduced in [18] are:

$$\Sigma\mathrm{P}_v = \Sigma\mathrm{P}_{\mathit{left}(v)} + \Sigma\mathrm{P}_{\mathit{right}(v)}$$
$$ect_v = \max\left\{ect_{\mathit{right}(v)}, ect_{\mathit{left}(v)} + \Sigma\mathrm{P}_{\mathit{right}(v)}\right\}$$

An example of a classic $\Theta$-tree is given in Figure 1.

When transition times are considered, the $ect_v$ value computed in the internal nodes of the $\Theta$-tree may only be a loose lower-bound since it is only based on the earliest start times and the processing times. We strengthen the estimation of the earliest computation times (denoted $ect^*$) by also considering transition times. We add another value inside the nodes: $n_v$ is the cardinality of Leaves($v$) ($n_v = |\mathrm{Leaves}(v)|$). The new update rules for the internal nodes of a $\Theta$-tree are:

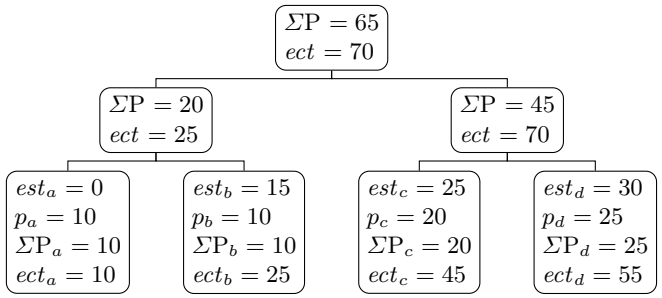



Fig. 1: Classic $\Theta$-tree as described in [18].

$$
\begin{aligned}
\Sigma\mathrm{P}_v &= \Sigma\mathrm{P}_{left(v)} + \Sigma\mathrm{P}_{right(v)} \\
n_v &= n_{left(v)} + n_{right(v)} \\
ect^*_v &= \begin{cases} \max\{ect^*_{right(v)}, ect^*_{left(v)} + \Sigma\mathrm{P}_{right(v)} + \underline{tt}(n_{right(v)} + 1)\} & : v \text{ internal} \\ ect_v & : v \text{ leaf} \end{cases}
\end{aligned}
$$

As an example, let us consider the set of four activities used in the $\Theta$-tree example of Figure 1. Let us assume that the associated transition times are as defined in the matrix $\mathcal{M}$ of Figure 2. The lower bounds for set of activities of different cardinality are reported next to the matrix. With the new update rules

$$
\mathcal{M} = \begin{pmatrix} 0 & 10 & 13 & 18 \\ 12 & 0 & 15 & 15 \\ 10 & 18 & 0 & 20 \\ 19 & 11 & 16 & 0 \end{pmatrix}
$$

| Lower Bound | $k=1$ | $k=2$ | $k=3$ |
|---|---|---|---|
| Min Weight Forest | 10 | 20 | 31 |
| Dynamic Programming | 10 | 20 | 32 |
| Min Cost Flow | 10 | 20 | 33 |
| Lagrangian Relaxation | 10 | 20 | 32 |
| $\underline{tt}(k)$ | 10 | 20 | 33 |

Fig. 2: Example of transition time matrix and associated lower bounds of transition times permutations.

defined above, we obtain the extended $\Theta$-tree presented in Figure 3. Note that the values of $ect^*$ in the internal nodes are larger than the values of $ect$ reported in the classic $\Theta$-tree (Figure 1).

**Lemma 1.** $ect_v \leq ect^*_v \leq ect^\diamond_{Leaves(v)} = \max_{\Theta' \subseteq Leaves(v)} \{est_{\Theta'} + p_{\Theta'} + \underline{tt}(|\Theta'|)\}$

*Proof.* The proof is similar to the proof of Proposition 7 in [18], by also integrating the inequality $\underline{tt}(|\Theta'|) \geq \underline{tt}(|\Theta'| \cap Left(v)|) + \underline{tt}(|\Theta'| \cap Right(v)|)$, which is itself a direct consequence of the fact that $\underline{tt}(k)$ is monotonic in $k$.

Since the new update rules are also executed in constant time for one node, we keep the time complexities of the initial $\Theta$-tree structure from [18] which are at worst $\mathcal{O}(n \log(n))$ for the insertion of all activities inside the tree.

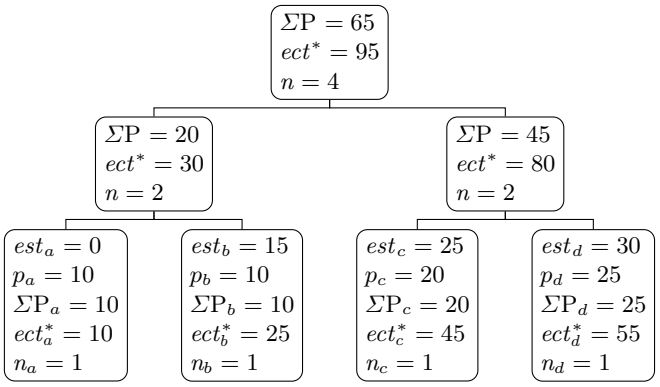



Fig. 3: Extended $\Theta$-tree for transition times. The $ect^*$ values reported in the internal nodes have been computed using the update rule of the extended $\Theta$-tree.

### Extending the $\Theta$-$\Lambda$-tree with Transition Times

The Edge-Finding (EF) algorithm requires an extension of the original $\Theta$-tree, called $\Theta$-$\Lambda$-tree [18]. This extension is used to obtain an efficient EF algorithm. In this extension, in addition to the activities included in a $\Theta$-tree, activities can be marked as *gray nodes*. Gray nodes represent activities that are not really in the set $\Theta$. However, they allow to easily compute $ect_\Theta$ if *one* of the gray activities were included in $\Theta$. If we consider the set of gray activities $\Lambda$ such that $\Lambda \cap \Theta = \emptyset$, we are interested in computing the largest $ect$ obtained by including *one* activity from $\Lambda$ into $\Theta$:

$$\overline{ect}_{(\Theta,\Lambda)} = \max_{i \in \Lambda} \ ect_{\Theta \cup \{i\}}$$

In addition to $\Sigma\mathrm{P}_v$, $ect_v$, the $\Theta$-$\Lambda$-tree structure also maintains $\overline{\Sigma\mathrm{P}}_v$ and $\overline{ect}_v$, respectively corresponding to $\Sigma\mathrm{P}_v$ and $ect_v$, *if* the single gray activity in the sub-tree rooted by $v$ maximizing $ect_v$ *were* included:

$$\overline{ect}^*_{(\Theta,\Lambda)} = \max\left\{ect^*_\Theta,\ \max_{i \in \Lambda}\left\{ect^*_{\Theta\cup\{i\}}\right\}\right\}$$

The update rule for $\overline{\Sigma\mathrm{P}}_v$ remains the same as the one described in [18]. However, following a similar reasoning as the one used for the extended $\Theta$-tree, we add

the $n_v$ value, and update rules are modified for $\overline{ect}_v$ and $\overline{n}_v$. The rules become:

$$\overline{\Sigma \mathrm{P}}_v = \max\left\{\overline{\Sigma \mathrm{P}}_{left(v)} + \Sigma \mathrm{P}_{right(v)}, \Sigma \mathrm{P}_{left(v)} + \overline{\Sigma \mathrm{P}}_{right(v)}\right\}$$

$$\overline{ect}^*_v = \max\left\{\begin{array}{l} \overline{ect}^*_{right(v)}, \\ \overline{ect}^*_{left(v)} + \Sigma \mathrm{P}_{right(v)} + \underline{tt}(n_{right(v)} + 1), \\ ect^*_{left(v)} + \overline{\Sigma \mathrm{P}}_{right(v)} + \underline{tt}(\overline{n}_{right(v)} + 1) \end{array}\right\}$$

$$\overline{n}_v = \begin{cases} n_v + 1 & \text{if the subtree rooted in } v \text{ contains a gray node} \\ n_v & \text{otherwise} \end{cases}$$

This extended $\Theta$-$\Lambda$-tree allows us to efficiently observe how the $ect^*$ of a set of activities is impacted if a single activity is added to this set. This information allows the EF algorithm to perform propagation efficiently[1]. An example of $\Theta$-$\Lambda$-tree based on the example from Figure 3 and Figure 2 is displayed in Figure 4.

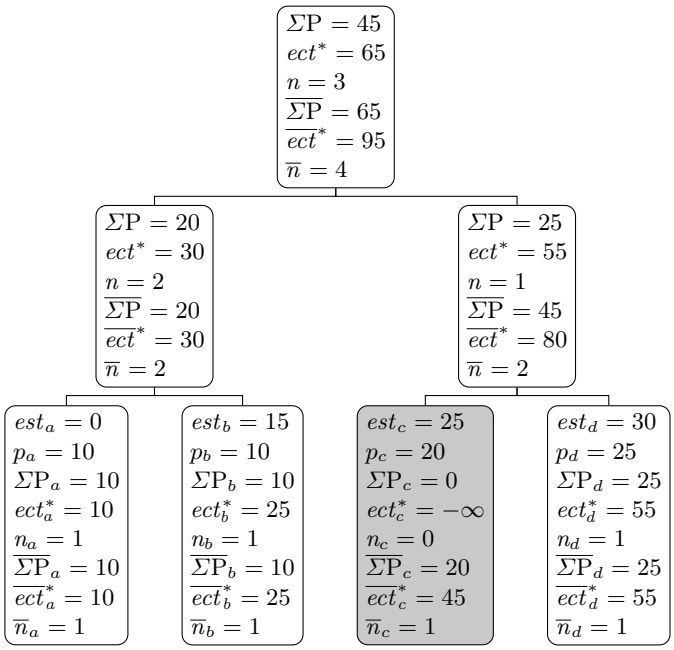



Fig. 4: Extended $\Theta$-$\Lambda$-tree with modified update rules.

[1] Finding the "responsible" activity $\arg\max_i ect_{\Theta \cup \{i\}}$ (required by EF) is done similarly to [18].

## 6 Disjunctive Propagation Algorithms with Transition Times

In [18], a propagation procedure for the unary resource constraint is defined. This propagation procedure consists of a propagation loop including Overload Checking (OC), Detectable Precedences (DP), Not-First/Not-Last (NF/NL) and Edge Finding (EF) propagation algorithms. The first three rely on the $\Theta$-tree while the latter employs the $\Theta$-$\Lambda$-tree. Some small modifications can be done to these algorithms to obtain an efficient propagation procedure making use of knowledge about transition times.

The four mentioned propagation algorithms use a $\Theta$-tree or a $\Theta$-$\Lambda$-tree to compute $ect_\Theta$ on a set of activities $\Theta$. OC checks if $ect_\Theta > lct_\Theta$. DP, NF/NL and EF rely on a set of rules that potentially allow to update the *est* or *lct* of an activity. They all incrementally add/remove activities to a set of activities $\Theta$ while maintaining the value $ect_\Theta$. When a rule is triggered by the consideration of a given activity, the *est* or *lct* of this activity can be updated according to the current value of $ect_\Theta$.

These four propagation algorithms can be used for the propagation of the transition time constraints. To do so, we propose to substitute in the filtering algorithms the $\Theta$-tree and the $\Theta$-$\Lambda$-tree structures by their extended versions. In the presence of transition times, $ect^*/\overline{ect}^*$ is indeed a stronger bound than $ect/\overline{ect}$. Furthermore, the update rules can be slightly modified to obtain an even stronger propagation. When one of these algorithms detects that an activity $i$ is after all activities in a set $\Theta$, the following update rule can be applied:

$$est_i \leftarrow \max\{est_i, ect^*_\Theta\}$$

In addition to all the transitions between activities of $\Theta$ - already taken into account in $ect^*_\Theta$ - there must be a transition between one activity and $i$ (not necessarily from $\Theta$, as we do not know which activity will be just before $i$ in the final schedule). It is therefore correct to additionally consider the minimal transition from any activity to $i$. The update rule becomes:

$$est_i \leftarrow \max\left\{est_i, ect^*_\Theta + \min_{j \neq i} tt_{j,i}\right\}$$

An analogous reasoning can be applied to the update rule of the *lct* of an activity.

Similarly to the fix point propagation loop proposed in [18] for the unary resource constraint, the four extended propagation algorithms are combined to achieve an efficient propagation on transition time constraints. This allows to obtain a global propagation procedure instead of the conjunction of pairwise transition constraints described by Equation 1. The approach has however the disadvantage that the computation of $ect^*_\Theta$ integrates a lower bound. This prevents having the guarantee that sufficient propagation is achieved. The loop must thus also integrate the conjunction of pairwise transition constraints given in Equation 1. However, experimental results provided in Section 7 exhibits that the supplementary global constraint reasoning can provide a substantial filtering gain.

### 6.1 Detectable Precedences Propagation Example

Let us consider a small example (inspired from an example of [18]) with 3 activities, $A$, $B$ and $C$ whose domains are illustrated in Figure 5. The corresponding transition matrix and lower bounds are given in Figure 6.

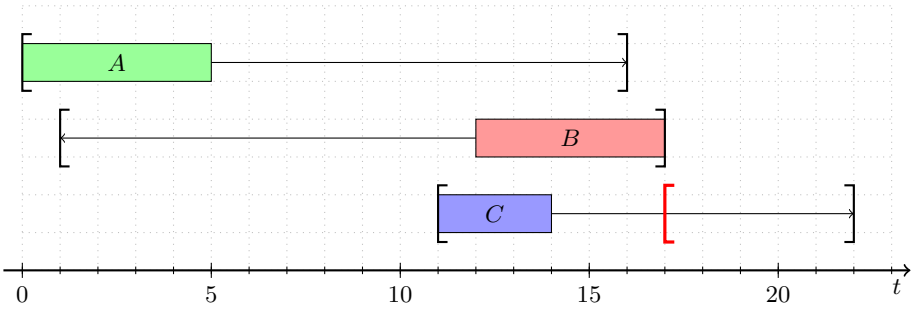



Fig. 5: Example of extended *Detectable Precedences* with transition times. The extended version updates $est_C$ from 11 to 17, while the joint use of transition time binary constraints with the original unary constraint is not able to make this deduction.

$$\mathcal{M} = \begin{pmatrix} 0 & 4 & 6 \\ 2 & 0 & 5 \\ 4 & 3 & 0 \end{pmatrix}$$

| Lower Bound | $k=1$ | $k=2$ |
|---|---|---|
| Min Weight Forest | 2 | 5 |
| Dynamic Programming | 2 | 5 |
| Min Cost Flow | 2 | 5 |
| Lagrangian Relaxation | 2 | 5 |
| $\underline{tt}(k)$ | 2 | 5 |

Fig. 6: Transition times for activities from Figure 5

From this example, the *Detectable Precedences* algorithm will eventually build a $\Theta$-tree containing activities $A$ and $B$. Figures 7a and 7b respectively show the classic and the extended $\Theta$-trees.

As one can see, $ect^*$ is larger than $ect$ as it is not agnostic about the transition time constraints. Furthermore, the update rule of $est_C$ also includes the minimal transition time from any activity to $C$. This leads to the following update of $est_C$:

$$\begin{aligned} est_C &= \max\left\{est_C, ect^*_\Theta + \min_{i \neq C} tt_{i,C}\right\} \\ &= \max\{11, 12+5\} = 17 \end{aligned}$$

We finally obtain an updated $est_C$, as shown by the red bold bracket in Figure 5. Notice that the joint use of the constraints given in Equation 1 with the original unary constraint of [18] would not make this deduction.


(a) Classic $\Theta$-tree

(b) Extended $\Theta$-tree

Fig. 7: Comparison of classic and extended $\Theta$-tree on the example described in Figures 5 and 6.

## 7 Evaluation

To evaluate our constraint, we used the OscaR solver [14] and ran instances on AMD Opteron processors (2.7 GHz). For each considered instance, we used the 3 following filterings for the unary constraint with transition times :

1. Binary constraints[2] ($\phi_b$) given in Equation 1.
2. Binary constraints given in Equation 1 with the Unary global constraint of [18] ($\phi_{b+u}$).
3. The constraint introduced in this article ($\phi_{uTT}$). Based on our experience, we slightly changed the propagation loop order : Edge-finding is put in first position.

*Considered benchmarks* We constructed instances considering transition times from famous JobShop benchmarks. For a given benchmark $\mathcal{B}$, in each instance, we added generated transition times between activities, while ensuring that triangular inequality always hold. From $\mathcal{B}$, we generated new benchmarks $\mathcal{B}_{(a,b)}$ inside which the instances are expanded by transition times uniformly picked between $a\%$ and $b\%$ of $\overline{D}$, where $\overline{D}$ is the average duration of all activities in the original instance.

We generated instances from the well-known Taillard's instances[3]. From each instance, we generated 2 instances for a given pair $(a, b)$, where the following pairs were used : $(50, 100)$, $(50, 150)$, $(50, 200)$, $(100, 150)$, $(100, 200)$ and $(150, 200)$. This allowed us to create 960 new instances[4].

[2] For efficiency reason, dedicated propagators have been implemented instead of posting reified constraint.

[3] Available at http://mistic.heig-vd.ch/taillard/problemes.dir/ordonnancement.dir/ordonnancement.html.

[4] Available at http://becool.info.ucl.ac.be/resources/benchmarks-unary-resource-transition-times

**Comparison of the 3 models**

In order to present fair results regarding the benefits that are only provided by our constraint, we first followed the methodology introduced in [16]. Afterwards, we made measurements using a static search strategy, as it cannot be influenced by the additional pruning provided by our constraint.

*Potential of the constraint* In brief, the approach presented in [16] proposes to pre-compute a search tree using the filtering that prunes the less - the *baseline* propagator - and then to *replay* this search tree using the different studied filtering procedures. The point is to only measure the time gain provided by the propagation, by decoupling the gain provided by the search strategy (while still being able to use dynamic ones) from the one provided by the propagation. We used $\phi_b$ as the baseline filtering, and the *SetTimes* (st) search strategy to construct the search tree, as this strategy is recognized to provide good performances in Scheduling. The search tree construction time was limited to 600 seconds. We then constructed *performance profiles* as described in [16]. Basically, those are cumulative distribution functions of a performance metric $\tau$. Here, $\tau$ is the ratio between the solution time (or number of backtracks) of a target approach (i.e. $\phi_{b+u}$ or $\phi_{uTT}$) and that of the baseline (i.e. $\phi_b$). For time (similar for number of backtracks), the function is defined as:

$$F_{\phi_i}(\tau) = \frac{1}{|\mathcal{M}|} \left| \left\{ M \in \mathcal{M} : \frac{t(replay(\text{st}), M \cup \phi_i)}{t(replay(\text{st}), M)} \leq \tau \right\} \right| \tag{4}$$

where $\mathcal{M}$ is the set of considered instances while $t(replay(\text{st}), M \cup \phi_i)$ and $t(replay(\text{st}), M)$ are respectively the time required to replay the generated search tree with the studied model (model using $\phi_i$, i.e. $\phi_{b+u}$ or $\phi_{uTT}$) and with the baseline model.

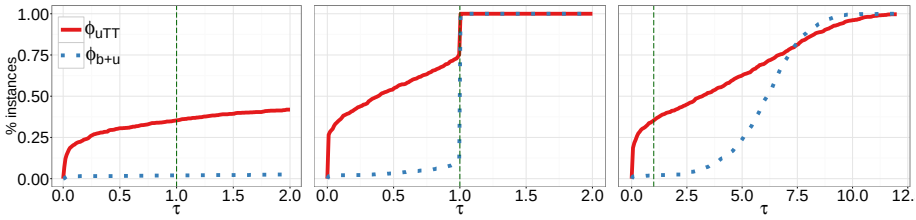
Figures 8a and 8b respectively provide the profiles for time and backtrack for all the 960 instances[5]. Figure 8c provides a "long-term" view of Figure 8a.

From Figure 8a, we can first conclude that $\phi_{b+u}$ is clearly worse than $\phi_{uTT}$ and $\phi_b$ from a time perspective. Moreover, Figure 8b shows that $\phi_{b+u}$ rarely offers more pruning than $\phi_b$.

In comparison, we can see from Figure 8a that for $\sim 20\%$ of the instances, $\phi_{uTT}$ is about 10 times faster than $\phi_b$, and that we solve $\sim 35\%$ of the instances faster (see $F_{\phi_{uTT}}(1)$). Moreover, it offers more pruning for $\sim 75\%$ of the instances (see Figure 8b).

From Figure 8c, we can see that the constraint does not have too much overhead, as $\phi_{uTT}$ is at worst about 7.5 times slower than $\phi_b$ for $\sim 45\%$ percent of the instances ($F_{\phi_{uTT}}(7.5) - F_{\phi_{uTT}}(1)$). It is a bit slower for the remaining $\sim 20\%$, which roughly corresponds to the percentage of instances for which $\phi_{uTT}$ provides no extra pruning (see $F_{\phi_{uTT}}(1)$ in Figure 8b).

[5] When instances were separated by number of jobs, the profiles had similar shapes.




(a) $\tau$ is a time ratio. (b) $\tau$ is a backtrack ratio. (c) "Long-term" profile ($\tau$ is a time ratio).

Fig. 8: Performance profiles for the 960 generated instances.

*Evaluation over a static search strategy* We here present results in a more "traditional" fashion. We compute the best makespan $m$ that can be obtained with $\phi_b$ within 600 seconds, using the following binary static search strategy : fixed variable order, left branch assigns $\text{start}_i$ to $est_i$, right branch removes $est_i$ from the domain of $\text{start}_i$. Then, the time and number of failures required by each model to find this solution are computed. We filtered out instances for which the solution was found by $\phi_b$ in less than 1 seconds and we computed the time ratio between $\phi_{uTT}$ and $\phi_b$. From this perspective, the 10 best and worst results are reported in tables 1 and 2, respectively. On the 10 best instances, the gains (the number of failures and time) are significant (sometimes two orders of magnitude). On the 10 worst instances, the times obtained with $\phi_{uTT}$ are similar to the results using the classical unary resource (i.e. $\phi_{b+u}$), while they are at worst around 6.4 times slower than the simple binary decomposition (i.e. $\phi_b$).

| Instance | $m$ | $\phi_{uTT}$ | | $\phi_b$ | | $\phi_{b+u}$ | |
|---|---|---|---|---|---|---|---|
| | | Time | #Fails | Time | #Fails | Time | #Fails |
| 15_15-3_225_50_100-1 | 2,344 | **1.12** | **2,442** | 117.92 | 980,330 | 432.07 | 911,894 |
| 50_15-8_750_50_100-2 | 6,682 | **2.11** | **744** | 182.27 | 1,127,272 | 999.79 | 1,127,272 |
| 20_15-7_300_150_200-2 | 4,784 | **0.24** | **449** | 17.63 | 168,466 | 62.27 | 168,466 |
| 15_15-6_225_50_100-1 | 2,398 | **3.90** | **5,593** | 187.93 | 889,079 | 534.20 | 602,591 |
| 50_20-3_1000_50_150-2 | 7,387 | **2.96** | **1,709** | 126.61 | 584,407 | 829.25 | 584,407 |
| 100_20-4_2000_150_200-1 | 18,595 | **11.59** | **885** | 340.32 | 332,412 | 1225.44 | 206,470 |
| 30_15-3_450_50_200-1 | 4,643 | **1.97** | **1,178** | 39.23 | 226,700 | 314.34 | 226,700 |
| 15_15-5_225_100_150-2 | 3,320 | **0.91** | **2,048** | 16.40 | 119,657 | 63.38 | 119,657 |
| 50_20-2_1000_50_100-1 | 6,979 | **3.79** | **1,680** | 63.16 | 878,162 | 4.63 | 1,695 |
| 30_15-10_450_100_200-1 | 5,586 | **0.74** | **687** | 9.24 | 106,683 | 41.25 | 106,683 |

Table 1: Best time results for $\phi_{uTT}$ compared to $\phi_b$. The problem is to find the given makespan $m$ using a binary static search strategy. Time is in seconds.

| Instance | $m$ | $\phi_{uTT}$ | | $\phi_b$ | | $\phi_{b+u}$ | |
|---|---|---|---|---|---|---|---|
| | | Time | #Fails | Time | #Fails | Time | #Fails |
| 15_15-10_225_50_200-2 | 2,804 | 645.26 | **546,803** | **127.38** | **546,803** | 572.81 | **546,803** |
| 50_15-9_750_50_200-1 | 6,699 | 954.77 | **164,404** | **174.63** | 164,437 | 1,108.43 | 164,437 |
| 20_20-5_400_100_150-2 | 4,542 | 213.54 | **78,782** | **38.26** | 78,968 | 180.20 | 78,968 |
| 20_20-8_400_100_150-2 | 4,598 | 147.55 | **164,546** | **26.42** | 164,576 | 175.69 | 164,576 |
| 15_15-2_225_50_100-2 | 2,195 | 178.37 | **96,821** | **31.23** | **96,821** | 139.84 | **96,821** |
| 20_20-6_400_100_200-1 | 4,962 | 11.15 | **8,708** | **1.94** | 8,745 | 11.87 | 8,745 |
| 30_20-8_600_50_200-1 | 5,312 | 18.63 | **6,665** | **3.15** | 6,687 | 19.93 | 6,687 |
| 20_15-10_300_50_200-2 | 3,571 | 85.84 | **61,185** | **14.24** | **61,185** | 65.12 | **61,185** |
| 50_20-8_1000_100_200-1 | 9,186 | 286.61 | **88,340** | **46.17** | **88,340** | 180.23 | **88,340** |
| 20_15-1_300_100_150-1 | 3,557 | 189.37 | **208,003** | **29.55** | 209,885 | 157.33 | 209,885 |

Table 2: Worst time results for $\phi_{uTT}$ compared to $\phi_b$. The problem is to find the given makespan $m$ using a binary static search strategy. Time is in seconds.

## 8 Conclusion

In this paper, we proposed to extend classic unary resource propagation algorithms such that they consider transition times. We first stated that a lower bound of the time taken by transitions between activities from a set $\Omega$ is required to have a tighter bound of $ect_\Omega$. We described several possible methods to compute these lower bounds. We then proposed to extend the $\Theta$-tree and $\Theta$-$\Lambda$-tree structures to integrate these lower bounds. These extended structures can then be used in unary propagation algorithms: OC, DP, NF/NL and EF. The new obtained propagation procedure has the advantage that it can be used conjointly with any other constraint and that it is completely independent from the objective to optimize. We have demonstrated that the additional pruning achieved by this propagation can dramatically reduce the number of nodes (and thus the time taken to solve the problem) on a wide range of instances.

Future work would analyze the possibility to integrate tighter incremental lower bounds in $\Theta$-tree and $\Theta$-$\Lambda$-tree structures. The order and real usefulness of the propagators (OC, DP, NF/NL, EF) should also be studied in order to acquire the most efficient fixpoint propagation loop. Finally, we would like to experiment on a new update rule in $\Theta$-tree and $\Theta$-$\Lambda$-tree to be able to obtain tighter lower bounds for $ect_\Omega$.

## References


1. Ahuja, R.K., Magnanti, T.L., Orlin, J.B.: Network Flows: Theory, Algorithms, and Applications. Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1993)
2. Allahverdi, A., Ng, C., Cheng, T.E., Kovalyov, M.Y.: A survey of scheduling problems with setup times or costs. European Journal of Operational Research 187(3), 985–1032 (2008)
3. Artigues, C., Belmokhtar, S., Feillet, D.: A new exact solution algorithm for the job shop problem with sequence-dependent setup times. In: Integration of ai and or techniques in constraint programming for combinatorial optimization problems, pp. 37–49. Springer (2004)
4. Artigues, C., Buscaylet, F., Feillet, D.: Lower and upper bound for the job shop scheduling problem with sequence-dependent setup times. In: Proceedings of the second multidisciplinary international conference on scheduling: theory and applications (MISTA'2005) (2005)
5. Artigues, C., Feillet, D.: A branch and bound method for the job-shop problem with sequence-dependent setup times. Annals of Operations Research 159(1), 135–159 (2008)
6. Balas, E., Simonetti, N., Vazacopoulos, A.: Job shop scheduling with setup times, deadlines and precedence constraints. Journal of Scheduling 11(4), 253–262 (2008)
7. Baptiste, P., Le Pape, C., Nuijten, W.: Constraint-based scheduling: applying constraint programming to scheduling problems, vol. 39. Springer Science & Business Media (2001)
8. Focacci, F., Laborie, P., Nuijten, W.: Solving scheduling problems with setup times and alternative resources. In: AIPS. pp. 92–101 (2000)
9. Gagné, C., Price, W.L., Gravel, M.: Scheduling a single machine with sequence dependent setup time using ant colony optimization. Faculté des sciences de l'administration de l'Université Laval, Direction de la recherche (2001)
10. González, M.A., Vela, C.R., Varela, R.: A new hybrid genetic algorithm for the job shop scheduling problem with setup times. In: ICAPS. pp. 116–123 (2008)
11. Grimes, D., Hebrard, E.: Job shop scheduling with setup times and maximal time-lags: A simple constraint programming approach. In: Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, pp. 147–161. Springer (2010)
12. Kruskal, J.B.: On the shortest spanning subtree of a graph and the traveling salesman problem. Proceedings of the American Mathematical society 7(1), 48–50 (1956)
13. Oddi, A., Rasconi, R., Cesta, A., Smith, S.F.: Exploiting iterative flattening search to solve job shop scheduling problems with setup times. PlanSIG2010 p. 133 (2010)
14. OscaR Team: OscaR: Scala in OR (2012), available from `https://bitbucket.org/oscarlib/oscar`
15. Tahar, D.N., Yalaoui, F., Amodeo, L., Chu, C.: An ant colony system minimizing total tardiness for hybrid job shop scheduling problem with sequence dependent setup times and release dates. In: Proceedings of the International Conference on Industrial Engineering and Systems Management. pp. 469–478 (2005)
16. Van Cauwelaert, S., Lombardi, M., Schaus, P.: Understanding the potential of propagators. In: Proceedings of the Twelfth International Conference on Integration of Artificial Intelligence and Operations Research techniques in Constraint Programming (2015)

17. Vilím, P.: O (nlog n) filtering algorithms for unary resource constraint. In: Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, pp. 335–347. Springer (2004)
18. Vilım, P.: Global constraints in scheduling. Ph.D. thesis, PhD thesis, Charles University in Prague, Faculty of Mathematics and Physics, Department of Theoretical Computer Science and Mathematical Logic, KTIML MFF, Universita Karlova, Malostranské námestı 2/25, 118 00 Praha 1, Czech Republic (2007)
19. Vilím, P., Barták, R., Čepek, O.: Extension of o (n log n) filtering algorithms for the unary resource constraint to optional activities. Constraints 10(4), 403–425 (2005)
20. Vilím, P.: O(nlogn) filtering algorithms for unary resource constraint. In: Régin, J.C., Rueher, M. (eds.) Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Lecture Notes in Computer Science, vol. 3011, pp. 335–347. Springer Berlin Heidelberg (2004)
21. Wolf, A.: Constraint-based task scheduling with sequence dependent setup times, time windows and breaks. GI Jahrestagung 154, 3205–3219 (2009)
22. Zampelli, S., Vergados, Y., Van Schaeren, R., Dullaert, W., Raa, B.: The berth allocation and quay crane assignment problem using a cp approach. In: Principles and Practice of Constraint Programming. pp. 880–896. Springer (2013)