# A Global Constraint for Total Weighted Completion Time

András Kovács[1,3] and J. Christopher Beck[2]

[1] Projet Contraintes, INRIA Rocquencourt, France
[2] Dept. of Mechanical and Industrial Engineering, University of Toronto, Canada
[3] Computer and Automation Research Institute, Hungarian Academy of Sciences
akovacs@sztaki.hu, jcb@mie.utoronto.ca

**Abstract.** We introduce a novel global constraint for the total weighted completion time of activities on a single unary capacity resource. For propagating the constraint, an $O(n^4)$ algorithm is proposed, which makes use of the preemptive mean busy time relaxation of the scheduling problem. The solution to this problem is used to test if an activity can start at each start time in its domain in solutions that respect the upper bound on the cost of the schedule. Empirical results show that the proposed global constraint significantly improves the performance of constraint-based approaches to single-machine scheduling for minimizing the total weighted completion time. Since our eventual goal is to use the global constraint as part of a larger optimization problem, we view this performance as very promising. We also sketch the application of the global constraint to cumulative resources and to problems with multiple machines.

## 1 Introduction

Many successful applications of constraint programming (CP) to optimization problems exhibit a "maximum type" optimization criteria, characterized by minimizing the maximum value of a set of variables (e.g., makespan, maximum tardiness, or peak resource usage in scheduling). Such criteria exhibit strong back propagation: placing an upper bound on the cost variable results in the pruning of the domain (i.e., the reduction of the maximum value) of the constituent variables. CP has not been as successful for other practically important optimization criteria such as "sum type" objective functions characterized by the minimization of the sum of a set of variables. Examples in the scheduling domain include total weighted completion time, weighted tardiness, weighted earliness and tardiness, and the number of late jobs. Nearly all CP-based approaches to scheduling with these criteria use only the basic sum constraint to propagate the objective function. However, back propagation of the sum constraint is weak because it is often the case that the maximum value of each decision variable is supported by the minimum values of all the other decision variables. The significance of more efficient global constraints for back propagation has been emphasized by Focacci et al. in [10,11].

P. Van Hentenryck and L. Wolsey (Eds.): CPAIOR 2007, LNCS 4510, pp. 112–126, 2007.

Our purpose is to develop algorithms for propagating "sum type" objective functions in constraint-based scheduling. In this paper, we address the total weighted completion time criterion on a single unary resource. The total weighted completion time criterion has equivalents in a wide range of applications. In container loading problems, it is a frequent requirement that the center of gravity of the loaded container has to be situated as low as possible and, depending on the means of transport, either above the axes of the vehicle or in the center of the container. Along each axis of the coordinate system, the location of the center of gravity of box-shaped goods corresponds to the average weighted completion time of the activities in a schedule. The weight of the activities equals the physical weight of the goods, while their duration corresponds to the length, and their resource requirement to the cross section of the loaded goods. In lot-sizing problems, different items are produced on a single machine, with specific deadlines. The cost of a solution is composed of a holding cost and a setup or ordering cost. The holding cost is computed as the total weighted difference of deadlines and actual production times. Apart from a constant factor, this is equivalent to the weighted distance of the activities from a remote point in time, which corresponds to the weighted completion time in a reversed schedule. In all these applications, the total weighted completion time constraint appears as only one component of a complex satisfaction or optimization problem, in conjunction with various other constraints. This justifies our ambition to develop a generic constraint propagation algorithm, instead of customized search algorithms for specific problems.

The remainder of this paper is organized as follows. In the next section, we introduce the notation used in the paper. In Section 3 we review the related literature. This is followed by the presentation of the proposed constraint propagation algorithm (Section 4). In Section 5, we evaluate the performance of our algorithm on a set of benchmark problems from the literature. In Section 6, we sketch extensions of this work to cumulative resources and multiple resource problems. Finally, conclusions are drawn and directions of future research are outlined.

## 2 Definitions and Notations

While the proposed constraint has potential applications in various fields, we present this work in the context of a single, unary capacity resource scheduling problem where the optimization criterion is the minimization of total weighted completion time.

This scheduling problem involves $n$ activities, $A_i$, to be executed without preemption on a single, unary resource. Each activity is characterized by its processing time, $p_i$, and a non-negative weight, $w_i$. The start time variable of $A_i$ will be denoted by $S_i$. When appropriate, we call the current lower bound on a start time variable $S_i$ the *release time* of the activity, and denote it by $r_i$. The total weighted completion time of the activities will be denoted by $C$. We assume

that all data are integral. Thus, the constraint that enforces $C = \sum_i w_i(S_i + p_i)$ on activities takes the following form.

$$\text{COMPLETION}([S_1, ..., S_n], [p_1, ..., p_n], [w_1, ..., w_n], C)$$

Throughout this paper we assume that $p_i$ and $w_i$ are constants. In applications where this assumption is restrictive, the lower bounds can be used during the propagation. Our algorithm filters the domain of the $S_i$ variables, while it tightens only the lower bound of $C$. The minimum and maximum values in the current domain of a variable $X$ will be denoted by $\underline{X}$ and $\bar{X}$, respectively.

## 3 Related Literature

The complexity, approximability, and algorithmic aspects of total weighted completion time scheduling problems have been studied extensively. The most widely discussed problem variants are the single and parallel machine versions with release dates. The classical scheduling notations for these problems are $1|r_i|\sum w_iC_i$ and $P|r_i|\sum w_iC_i$, respectively. Both variants are known to be NP-hard in the strong sense, even with uniform weights. Various polynomially solvable cases have been identified: without release dates, ordering the activities according to the *Weighted Shortest Processing Time* (WSPT) rule, i.e., by non-decreasing $p_i/w_i$ yields an optimal solution. The preemptive version of the single machine problem with release dates and unit weights ($1|r_i, pmtn|\sum C_i$) is polynomially solvable using *Shortest Remaining Processing Time* rule, but adding non-uniform weights renders it NP-hard. A comprehensive overview of the complexity of related scheduling problems is presented in [7].

Linear programming (LP) and combinatorial lower bounds for the single machine problem have been studied and compared by Goemans et al. [12] and Dyer & Wolsey [9]. The *preemptive time-indexed relaxation* corresponds to an assignment problem in which variables indicate whether activity $A_i$ is processed at time $t$. In an alternative LP relaxation, the *non-preemptive time-indexed formulation*, variables express if activity $A_i$ is completed at time $t$. Dyer & Wolsey [9] have shown that the latter is strictly stronger than the former. Since these LP formulations only include continuous variables, but their size depends both on the number of activities and the number of time units, they can be solved in pseudo-polynomial time.

A different LP relaxation has been proposed by Schulz [18], using completion time variables. Subsequently, Goemans et al. [12] proved that this relaxation is equivalent to the preemptive time-indexed formulation, by showing that a preemptive schedule that minimizes the mean busy time (see Section 4) yields the optimal solution for both relaxations. Moreover, this preemptive schedule can be found in $O(n \log n)$ time, where $n$ is the number of activities. The authors also propose two randomized algorithms (and their de-randomized counterparts) to convert the preemptive schedule into a feasible solution of the original problem, and prove that these algorithms lead to 1.69 and 1.75-approximations, respectively. These results also imply a guarantee on the quality of the lower bound.

Polynomial time approximation schemes for the single and parallel machines case, as well as for some other variants are presented in Afrati et al. [1]. The time complexity of the algorithm to achieve a $(1+\varepsilon)$-approximation for a fixed $\varepsilon$ is $O(n \log n)$, but the complexity increases super-exponentially with $\varepsilon$.

Papers presenting complete solution methods for different versions of the total weighted completion time problem include a classical work of Belouadah et al. [6] and more recent papers by Jouglet et al. [13] and Pan & Shi [17] for a single machine, Nessah et al. [15] for identical machines, and Della Croce et al. [8], as well as Akkan and Karabatı [2] for the two-machine flowshop problem. Most of these algorithms make use of lower bounds similar to the ones discussed above, as well as various dominance rules and customized branching strategies.

The literature of global constraint propagation algorithms for "sum type" objective functions in scheduling is scarce. Notable exceptions are the works of Focacci et al. [10,11] on embedding relaxations of the *Traveling Salesman Problem* into global constraints. Baptiste et al. [4] proposed a branch-and-bound method for minimizing the total tardiness on a single machine. While building the schedule chronologically, the algorithm makes use of constraint propagation to filter the set of possible next activities by examining how a given choice affects the value of the lower bound. Baptiste et al. [5] address the minimization of the number of late activities on a single resource, and generalize some well-known resource constraint propagation techniques for the case where there are some activities that complete after their due dates. The authors also propose propagation rules to infer if activities are on time or late, but the applicability of these inference techniques is restricted by the fact that they incorporate dominance rules that might be invalid in more general contexts. For propagating the weighted earliness/tardiness cost function in general resource constrained project scheduling problems, Kéri & Kis [14] defined a simple method for tightening time windows of activities by eliminating values that would lead to solutions with a cost higher than the current upper bound.

## 4 Propagating Total Weighted Completion Time on a Unary Resource

Our propagation algorithm relies on solving the *preemptive mean busy time* relaxation [12] of the scheduling problem. This relaxed problem minimizes $\sum_i w_i M_i$, where $M_i$ denotes the mean busy time of activity $A_i$, i.e., the average point in time at which the machine is busy processing $A_i$. This is easily calculated by finding the mean of each time point at which activity $A_i$ is executed.

The underlying idea of our constraint propagator is to exploit the above relaxation to obtain a lower bound on the solution value of the original problem. However, instead of computing only one lower bound, we recompute the lower bound for restricted versions of the problem in which the value of a start time variable $S_i$ is bound to a given value $t$. We denote such restricted problems by $\Pi\langle S_i = t\rangle$, and the resulting lower bound on $C$ by $\underline{C}\langle S_i = t\rangle$. Our domain filtering mechanism will rely on the following proposition.

**Proposition 1.** *If* $\bar{C} < \underline{C}\langle S_i = t\rangle$*, then* $t$ *can be removed from the domain of* $S_i$*.*

In what follows, we first present an algorithm to compute the optimal relaxed schedule, and then show how this relaxed schedule can be quickly recomputed for the restricted problems. These algorithms will be illustrated using the sample problem introduced in Fig. 1.

| | $r_i$ | $p_i$ | $w_i$ | $w_i/p_i$ |
|---|---|---|---|---|
| $A_0$ | 0 | 5 | 10 | 2 |
| $A_1$ | 4 | 2 | 10 | 5 |
| $A_2$ | 5 | 4 | 12 | 3 |
| $A_3$ | 13 | 2 | 8 | 4 |

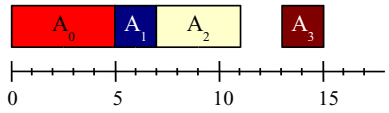


**Fig. 1.** Left: The input data for the sample problem. Right: The optimal solution of the sample problem. The total weighted completion time is 372.

### 4.1 Computing a Lower Bound

The optimal solution of the preemptive mean busy time relaxation can be computed in $O(n \log n)$ time [12]. The algorithm maintains a priority queue of the activities sorted by non-increasing $w_i/p_i$. At each point of time, $t$, the queue contains the activities $A_i$ with $r_i \leq t$ that have not yet been completely processed. Scheduling decisions must be made each time a new activity is released or an activity is completely processed. In either case, the queue is updated and a *fragment* of the first activity in the queue, lasting until the next decision point, is inserted into the schedule. If the queue is empty, but there are activities not yet released, a gap is created. Technically, gaps are represented as fragments of a zero-weight, zero-release-time activity, and will be called *empty fragments*. We assume that the schedule ends with a sufficiently long empty fragment. Since there are at most $2n$ release time and activity completion events, and updating the queue requires $O(\log n)$ time, the algorithm runs in $O(n \log n)$ time.

The optimal relaxed schedule for the sample problem is presented in Fig. 2. The objective value of this relaxed solution of 362. The fragments of activity $A_i$ are denoted by $\alpha_i, \alpha_i', \alpha_i''$, etc. Empty fragments are named $\varepsilon, \varepsilon', \varepsilon''$, etc.

### 4.2 Incrementally Recomputing the Lower Bound

The above algorithm can easily be modified to compute optimal relaxed solutions for restricted problems $\Pi\langle S_i = t\rangle$, by assigning $r_i = t$ and $w_i = \infty$. This gives activity $A_i$ the largest $w_i/p_i$ ratio among all the activities, ensuring that it starts at $t$ and is not preempted. Relaxed solutions for various restrictions on the sample problems are presented in Fig. 3.

We apply the above method only once for each activity $A_i$, restricting it to start exactly at its release time, i.e., for $\Pi\langle S_i = r_i\rangle$. For other possible start

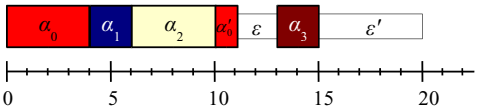


**Fig. 2.** The solution of the relaxed problem, with objective value 362. This is a lower bound on the original problem.

times, we incrementally convert the relaxed solution of $\Pi\langle S_i = t\rangle$ into a solution of $\Pi\langle S_i = t+1\rangle$ (or even directly for $\Pi\langle S_i = t+\Delta\rangle$ with $\Delta \geq 1$). This recomputation is based on the observation that one can represent the transformation of the relaxed solutions of $\Pi\langle S_i = t\rangle$ to that for $\Pi\langle S_i = t+1\rangle$ by a permutation of unit-duration sections of activities as follows.

**Definition 1.** *The permutation $\pi = (\alpha_0, \alpha_1, ..., \alpha_K)$ transforms a preemptive schedule by moving the first unit of each activity fragment $\alpha_k$ to the place of the first unit of $\alpha_{(k+1) \bmod (K+1)}$. If the moved unit is placed next to a fragment of the same activity then they are merged, otherwise a new fragment is created.*

In Fig. 3, the corresponding permutations are displayed next to each relaxed solution. For example, moving from $\Pi\langle S_i = 0\rangle$ to $\Pi\langle S_i = 1\rangle$ requires the movements of the first unit of the fragments as follows: $\alpha_0 \to \alpha_1, \alpha_1 \to \alpha_2, \alpha_2 \to \varepsilon, \varepsilon \to \alpha_0$. The final move creates a new empty fragment, $\varepsilon''$.

**Lemma 1.** *The permutation from $\Pi\langle S_i = t\rangle$ to $\Pi\langle S_i = t+1\rangle$ starts with $\alpha_0 = A_i$, while its further elements can be computed as*

$$\alpha_{k+1} = \textit{the leftmost fragment with } S(\alpha_{k+1}) > S(\alpha_k) \textit{ and } \frac{w(\alpha_{k+1})}{p(\alpha_{k+1})} < \frac{w(\alpha_k)}{p(\alpha_k)}.$$

*The permutation ends when it reaches a fragment $\alpha_K$ with $r(\alpha_K) \leq t$. Recall that the empty fragment at the end of the schedule always satisfies this condition, and $w(A_i) = \infty$ is assumed.*

Applying permutation $\pi = (\alpha_0, ..., \alpha_K)$ increases the total mean busy time of the preemptive schedule by

$$C(\pi) = \sum_{k=0}^{K-1} (S(\alpha_{k+1}) - S(\alpha_k)) \frac{w(\alpha_k)}{p(\alpha_k)} \; - \; (S(\alpha_K) - S(\alpha_0)) \frac{w(\alpha_K)}{p(\alpha_K)},$$

where $S(\alpha_k)$ denotes the start time of fragment $\alpha_k$. Thus, the cost of the new relaxed solution is $\underline{C}\langle S_i = t+1\rangle = \underline{C}\langle S_i = t\rangle + C(\pi)$.

Transformations of consecutive relaxed solutions $\Pi\langle S_i = t\rangle$ to $\Pi\langle S_i = t+1\rangle$ and $\Pi\langle S_i = t+1\rangle$ to $\Pi\langle S_i = t+2\rangle$ may be identical. Such is the case for $i = 0$ and $t = 0$ in the sample problem shown in Fig. 3. Note that the $\Delta$-fold application of permutation $\pi$ shifts fragments $\alpha_k$ that satisfy $S(\alpha_{k+1}) = S(\alpha_k) + p(\alpha_k)$ with $\Delta$ units to the right, while it relocates a $\Delta$-long portion of every other fragment

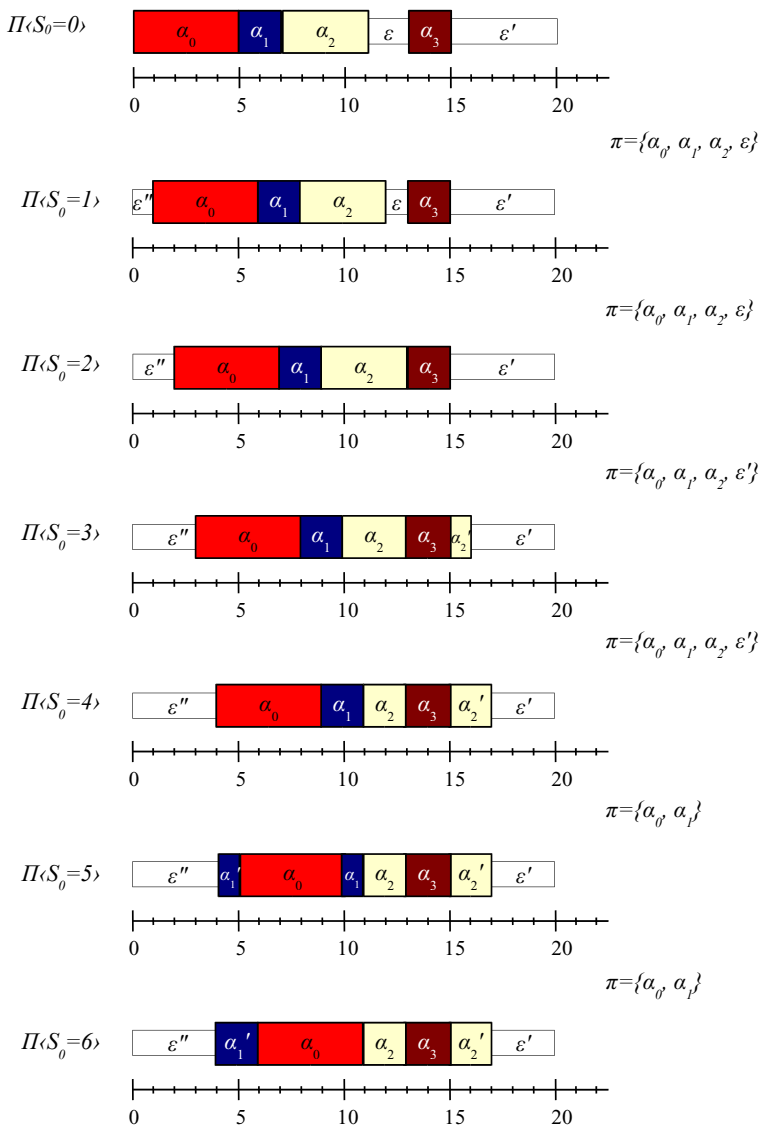


**Fig. 3.** Relaxed solutions of various restricted problems. The corresponding restrictions are displayed on the left, the permutations applied are shown on the left of the schedule.

into newly created fragments. Thus, a permutation $\pi$ on an optimal preemptive schedule results in optimal relaxed solutions for subsequent problems exactly $\Delta$ times, where $\Delta$ is the minimum of the following values:

(1) $p(\alpha_k)$ of fragments $\alpha_k \in \pi$ with $S(\alpha_{(k+1) \bmod (K+1)}) \neq S(\alpha_k) + p(\alpha_k)$;
(2) $r(\alpha_k) - S(\alpha_0)$ for fragments $\alpha_k \in \pi$ with $1 \leq k < K$.

Condition (1) must be respected because the $\Delta$-long, relocated portion of a fragment cannot be longer than the fragment itself. On the other hand, violating condition (2) would lead to sub-optimality: after the $(r(\alpha_{k^*}) - S(\alpha_0))$-fold application of permutation $\pi$, it is a different permutation $\pi' = (\alpha_0, ..., \alpha_{k^*})$ that results in optimal solutions for the subsequent relaxed problems.

The lower bound cost for the new restricted problem can be calculated as $\underline{C}\langle S_i = t + \Delta\rangle = \underline{C}\langle S_i = t\rangle + \Delta \cdot C(\pi)$, and linear interpolation can be applied for determining $\underline{C}\langle S_i = t'\rangle$ for $t' \in (t, t + \Delta)$.

### 4.3 From a Lower Bound to Domain Filtering

If both $\underline{C}\langle S_i = t\rangle > \bar{C}$ and $\underline{C}\langle S_i = t + \Delta\rangle > \bar{C}$ hold then the complete interval $[t, t + \Delta]$ can be removed from the domain of $S_i$. If only the first (second) condition holds, then the first (second), proportional part of the interval is removed. No removal is made otherwise. The new lower bound on the total weighted completion time is computed as $\underline{C} = \max_i \min_t C\langle S_i = t\rangle$.

Thus, each recomputation step consist of determining the permutation to apply and the corresponding $\Delta$, followed by filtering the appropriate variable domains. These steps are iterated until activity $A_i$ reaches the end of its time window, and the same procedure is repeated on each activity. The pseudo-code of the proposed constraint propagation algorithm is presented in Fig. 4.

### 4.4 Computational Complexity

In order to determine the computational complexity of the propagation algorithm, we first give a bound on the number of recomputation steps during the filtering of the domain of one start time variable $S_i$. We distinguish between two kinds of recomputation steps, depending on whether condition (1) or condition (2) bounds $\Delta$, and call these (1)-type and (2)-type steps, respectively. Recomputation steps where (1) and (2) are equally bounding are considered to be of the (1)-type.

Now, let us define the number of inversions $I(\sigma)$ as the number of fragment pairs $(\alpha_1, \alpha_2)$ in the preemptive schedule $\sigma$ such that $S(A_i) \leq S(\alpha_1) < S(\alpha_2)$ and $w(\alpha_1)/p(\alpha_1) < w(\alpha_2)/p(\alpha_2)$. Since there are at most $2n$ fragments in $\sigma$, $I(\sigma)$ is at most $O(n^2)$. Observe that $I(\sigma)$ is strictly decreased by (1)-type recomputation steps, while it is not affected by (2)-type steps. Therefore, the number of (1)-type steps is at most $O(n^2)$, while the number of (2)-type steps is bounded by the number of different activity release times, which is not greater than $n$.

Thus, the complete run of the constraint propagation algorithm takes at most $O(n^4)$ time: for each of the $n$ activities, there are at worst $O(n^2)$ recomputation steps and each step is carried out in at worst $O(n)$ time.

### 4.5 Implementation Details

While the pseudo-code depicted in Fig. 4 captures the underlying ideas of the proposed propagation algorithm, its performance can be increased in various

```
PROCEDURE RecomputeSchedule(σ - schedule, A_i - activity, t - time)
  LET permutation π = (A_i)
  WHILE r(last(π)) > t OR size(π) = 1
      α := leftmost fragment in σ fragments with S(α) > S(last(π))
           and w(α)/p(α) < w(last(π))/p(last(π))
      Append α to π
  LET Δ := min(min(p(α_k) | α_k ∈ π : S(α_{k+1 mod K+1}) ≠ S(α_k) + p(α_k)),
               min(r(α_k) − S(α_0) | α_k ∈ π, 1 ≤ k < K))
  σ' := Schedule obtained by performing π^Δ on σ
  RETURN ⟨σ', Δ⟩


PROCEDURE Propagate()
  FORALL activity A_i
      LET t := r_i
      σ_t := schedule computed by the lower bounding procedure for Π⟨S_i = t⟩
      WHILE t < S̄_i
           ⟨σ_{t+Δ}, Δ⟩ := RecomputeSchedule(σ, A_i, t)
           IF C⟨S_i = t⟩ > C̄ and C⟨S_i = t + Δ⟩ > C̄ THEN
               Remove [t, t + Δ] from domain(S_i)
           ELSE IF C⟨S_i = t⟩ > C̄ THEN
               Remove [t, t + ⌈(Δ (C̄ − C⟨S_i=t⟩)/(C⟨S_i=t+Δ⟩ − C⟨S_i=t⟩))⌉ − 1] from domain(S_i)
           ELSE IF C⟨S_i = t + Δ⟩ > C̄ THEN
               Remove [t + ⌊Δ (C̄ − C⟨S_i=t⟩)/(C⟨S_i=t+Δ⟩ − C⟨S_i=t⟩)⌋ + 1, t + Δ] from domain(S_i)
           t := t + Δ
  C ≥ max_i min_t C⟨S_i = t⟩
```

**Fig. 4.** Pseudo-code of the constraint propagation algorithm

ways. We have improved the average complexity of our implementation of the algorithm with the following changes:

- Common branching strategies bind activity start times in a chronological order, which results in a bound head of the schedule. This bound head is ignored when building the relaxed solutions, only its cost is taken into account;
- Relaxed solutions are saved during each run of the propagator; filtering the domain of $S_i$ is attempted again only after $\underline{S}_j$, $j \neq i$ have increased, or $\bar{C}$ decreased sufficiently to modify the relaxed solutions.

## 5 Computational Experiments

We ran computational experiments to measure the efficiency of the proposed propagation algorithm on the single-machine total weighted completion time problem with release times. The propagator has been implemented in C++ and embedded it into ILOG Solver and Scheduler versions 6.1. For propagating the

resource constraint, we used the edge-finding algorithm. We applied an adapted version of the *SetTimes* branching heuristic: in each search node from the set of not yet scheduled (and not postponed) activities, the heuristic selects the activity that has the smallest earliest start time (EST) and then breaks ties by choosing the activity with the highest $w/p$ ratio. Two branches are created according to whether the start time of this activity is bound to its EST or it is postponed.

We compared the performance of three different models. The first used the standard weighted sum constraint for propagating the optimization criterion (WS). The second calculated the lower bound presented in Section 4.1 (WS+LB) at each node and used it for bounding. The third model made use of the proposed COMPLETION constraint.

These algorithms were tested on benchmark instances from the online repository [16]. The same instances or the some problem generation method have been used in various previous works [3,6,13,17]. The repository contains 10 single-machine problem instances for each combination of parameters $n$ and $R$, where $n$ denotes the number of activities and takes values between 20 and 200 in increments of 10, while $R$ is the relative range of the release time, chosen from $\{0.2, 0.4, 0.6, 0.8, 1.0, 1.25, 1.5, 1.75, 2.0, 3.0\}$. Activity durations are randomly chosen from $U[1, 100]$, weights from $U[1, 10]$, and release times from $U[0, 50.5nR]$, where $U[a, b]$ denotes the integer uniform distribution over interval $[a, b]$. Out of these 1900 instances in total, we ran experiments on the 300 instances with $n \leq 70$ and every second value of parameter $R$. The experiments were run on a 1.86 GHz Pentium M computer with 1 GB of RAM, with a time limit of 120 seconds imposed.

The experimental results are presented in Table 1, where each row contains combined results for the 10 instances with the given number of activities $n$ and release time range $R$. For each of the three models, the table displays the number of the instances that could be solved to optimality (column *Solved*), the average number of search nodes (*Nodes*), and average search time in seconds (*Time*). The average is computed only on the instances that the algorithm solved. From these results we can conclude that the COMPLETION constraint adds significant pruning strength to the constraint-based approach. This pruning not only paid off in the means of the number of search nodes, but also decreased solution time on *every instance*, compared to both other models. While the classical WS model fails on some of the 20-activity instances, the COMPLETION constraint enabled us to solve – with one exception – all problems with at most 40 activities, and also performed well on the 50-activity instances.

The results also illustrate that instances with release time range $R \in \{0.6, 1.0\}$ are significantly more complicated for models WS+LB and COMPLETION than other instances. This is explained by the fact that with $R \ll 1$, activities in the second half of the schedule can simply be ordered by non-increasing $w_i/p_i$. On this section of the schedule, the lower bound is exact and our propagator achieves completeness. On the other hand, $R \gg 1$ leads to problems where only a few activities can be chosen for scheduling at any point in time, which makes the instance easily solvable as well.

**Table 1.** Experimental results: number of instances solved, average number of search nodes and average search time for the different versions of the branch and bound. Dash '-' means that none of the instances could be solved within 120 CPU seconds.

| $n$ | $R$ | WS | | | WS+LB | | | COMPLETION | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | Solved | Nodes | Time | Solved | Nodes | Time | Solved | Nodes | Time |
| 20 | 20 | - | - | - | 10 | 591 | 0.00 | 10 | 46 | 0.00 |
| | 60 | 2 | 1842024 | 62.50 | 10 | 1872 | 0.00 | 10 | 95 | 0.00 |
| | 100 | 10 | 114359 | 3.60 | 10 | 1756 | 0.10 | 10 | 109 | 0.00 |
| | 150 | 10 | 2518 | 0.00 | 10 | 223 | 0.00 | 10 | 67 | 0.00 |
| | 200 | 10 | 140 | 0.00 | 10 | 102 | 0.00 | 10 | 51 | 0.00 |
| 30 | 20 | - | - | - | 10 | 1718 | 0.10 | 10 | 114 | 0.00 |
| | 60 | - | - | - | 10 | 20674 | 2.90 | 10 | 417 | 0.00 |
| | 100 | 5 | 845422 | 58.60 | 9 | 113523 | 17.00 | 10 | 7046 | 2.90 |
| | 150 | 10 | 21555 | 1.30 | 10 | 1912 | 0.10 | 10 | 189 | 0.00 |
| | 200 | 10 | 2633 | 0.10 | 10 | 857 | 0.00 | 10 | 159 | 0.00 |
| 40 | 20 | - | - | - | 10 | 8434 | 1.30 | 10 | 241 | 0.00 |
| | 60 | - | - | - | 8 | 290209 | 57.12 | 9 | 4815 | 4.77 |
| | 100 | 2 | 164455 | 29.00 | 4 | 55760 | 12.00 | 10 | 27366 | 15.60 |
| | 150 | 10 | 40160 | 2.80 | 10 | 8943 | 1.30 | 10 | 592 | 0.20 |
| | 200 | 10 | 60602 | 3.70 | 10 | 3685 | 0.20 | 10 | 374 | 0.00 |
| 50 | 20 | - | - | - | 8 | 89148 | 22.75 | 10 | 1557 | 2.30 |
| | 60 | - | - | - | - | - | - | 8 | 31486 | 55.25 |
| | 100 | - | - | - | - | - | - | 2 | 26807 | 41.00 |
| | 150 | 3 | 92954 | 8.33 | 7 | 113198 | 26.71 | 10 | 12180 | 15.10 |
| | 200 | 8 | 36056 | 3.25 | 9 | 6898 | 1.22 | 10 | 1498 | 0.80 |
| 60 | 20 | - | - | - | 3 | 161594 | 49.66 | 10 | 16159 | 33.20 |
| | 60 | - | - | - | - | - | - | - | - | |
| | 100 | - | - | - | - | - | - | - | - | |
| | 150 | - | - | - | 4 | 194053 | 60.25 | 8 | 43353 | 32.75 |
| | 200 | 4 | 120345 | 12.75 | 9 | 48066 | 11.55 | 10 | 5290 | 4.30 |
| 70 | 20 | - | - | - | 2 | 72540 | 30.00 | 6 | 2591 | 10.83 |
| | 60 | - | - | - | - | - | - | - | - | |
| | 100 | - | - | - | - | - | - | 1 | 12125 | 46.00 |
| | 150 | - | - | - | 2 | 184557 | 64.50 | 3 | 8940 | 14.33 |
| | 200 | 4 | 228762 | 43.75 | 7 | 108701 | 34.14 | 9 | 14448 | 12.22 |

Our goal in this work is to develop a widely applicable constraint rather than to solve the single machine weighted completion time problem. However, it is instructive to compare these results directly against state-of-the-art, dedicated techniques for solving the single machine problem. Our algorithms comparable favorably to existing LP-based methods [3] that are able to solve instances with at most 30–35 activities, and earlier branch-and-bound methods [6], which solve problems with up to 40–50 activities. On the other hand, our approach is outperformed by two different, recent solution methods. One is a branch-and-bound algorithm combined with powerful dominance rules, constraint propagation, and no-good recording by Jouglet et al. [13], which has originally been developed for solving the more general total weighted tardiness problem. The other is a

dynamic programming approach enhanced with dominance rules and constraint propagation by Pan and Shi [17]. These two approaches are able to solve instances with up to 100 and 200 activities, respectively. It should be noted that a part of the contributions of these works, especially the strong dominance rules are orthogonal and complementary to the COMPLETION constraint. We expect that combining such approaches with the COMPLETION constraint would lead to further performance improvements.

## 6 Extensions to Other Scheduling Models

Our future work will focus on the application and extension of these results to more complex scheduling models and other application domains, such as constraining the location of the center of gravity in container loading. Below we sketch two possible extensions.

### 6.1 Extension to Cumulative Resources

To extend the COMPLETION constraint to cumulative resources (i.e., resources with a non-unary capacity), we define a variation:

$$\text{COMPLETION}_m([S_1, ..., S_n], [p_1, ..., p_n], [\varrho_1, ..., \varrho_n], [w_1, ..., w_n], R, C)$$

As with the unary case, the scheduling problem involves $n$ activities $A_i$ to be executed without preemption on a single, cumulative resource. Each activity is characterized by its processing time $p_i$, a non-negative weight factor $w_i$, and its resource requirement $\varrho_i$. $R$ represents the capacity of the resource. The total weighted completion time of the activities will be denoted by $C$. We assume that $\varrho_i$ and $R$ are constants, however our approach is easily adapted by reasoning with the lower bound of $\varrho_i$ and the upper bound of $R$.

As above, we use the mean busy time relaxation to obtain a lower bound on $C$. A preemptive schedule is prepared chronologically, by choosing at each decision point the $k$ available activities that have the highest $w_i/p_i$ ratio, so that $\sum_{i=1}^{k-1} \varrho_i < R \leq \sum_{i=1}^{k} \varrho_i$ holds. A schedule fragment is created in which activities $A_1, ..., A_{k-1}$ are processed at rates $\varrho_1, ..., \varrho_{k-1}$, and $A_k$ is processed at rate $R - \sum_{i=1}^{k-1} \varrho_i$. This fragment lasts until the next decision point, which corresponds to a release time, an activity completion, or a point in time where the remaining volume of an activity decreases below its previous processing rate. The complexity of the lower bounding algorithm is $O(n^2)$.

However, the cumulative extension of the COMPLETION constraint is more challenging than the unary version, because recomputing the mean busy time relaxation for each relevant value of the start time variables imposes an extensive computational burden in the cumulative case. We are currently investigating ways of partially relaxing the release times or resource requirements in order to facilitate quicker computations, at the price of reduced pruning strength.

### 6.2 Extension to Multiple Resource Problems

In a simple multiple resource problem, each activity requires one or more resources and has a weight. The obvious approach therefore is simply to have one COMPLETION constraint on each resource and represent the sum of completion times criterion as the sum of the sum of completion times on each resource, correcting for activities that require more than one resource.

More interesting is the extension to multiple resource project scheduling problems with more complex temporal relations amongst activities. For example, in a job shop scheduling problem, a job is made up of a sequence of activities linked by precedence constraints. The standard weighted completion model associates the weight and the completion time of a job to the final activity in the job, assigning zero weight to all other activities. We propose a more generic approach that allows us to use the COMPLETION constraint as defined above, and leads to more efficient pruning.

In our approach, activities can be assigned arbitrary weights, under the condition that the sum of activity weights within a job must equal the weight of the job. One COMPLETION constraint is placed on each resource that processes weighted activities, and the overall cost of the solution is the sum of total weighted completion times on each resource. The difference between activity and job completion times is compensated by a bias computed as the scalar product of activity weights and the temporal distance of activity and job completions. This way, weighted activities contribute to the overall cost of the solution both directly by their weighted completion time, and indirectly by delaying other weighted activities that have a lower $w_i/p_i$ ratio. On the other hand, zero-weight activities do not affect the cost in either way, unless they satisfy $\bar{S}_i < \underline{S}_i + p_i$, in which case they must use the resource in the interval $[\bar{S}_i,\ \underline{S}_i + p_i]$. This can be represented by a fragment $\alpha$ of appropriate duration, infinite weight, and $r(\alpha) = \bar{S}_i$.

Intuitively, the above considerations imply that the relaxed cost on each resource increases super-linearly with the total activity weight on the resource. Therefore, the strongest pruning is achieved when weighted activities are concentrated on a small number of resources. Such a distribution of the weights can be attained by a preprocessing procedure to assign weights. The procedure selects the most utilized resource $R_1$ and the set of activities $\mathscr{A}_1$ that require $R_1$, but such that none of their job-successors require $R_1$. To each activity in $\mathscr{A}_1$, we assign the weight of the corresponding job. The procedure continues with repeating these steps on the second, third, etc., most utilized resource while there are jobs not yet covered. Finally, all the remaining activities receive zero weights.

This approach does the opposite of what the classical weight-on-finals approach does in the case where job-final activities are uniformly spread over the resources. The superiority of the proposed approach can be best demonstrated on such problems: in the extreme, where there is at most one job-final activity on each resource, the classical approach results in no propagation at all. In contrast, the proposed method can still tighten variable domains by considering various weighted activities on the same resource. Our future work will focus on

elaborating the details of the assignment of weights to activities in the above sketched framework.

## 7 Conclusions

In this paper, we proposed an algorithm for propagating the COMPLETION constraint, which represents the sum of weighted completion times on a single unary capacity resource. The propagation of the constraint exploits a lower bound arising from the optimal solution to the preemptive mean busy time scheduling problem which can be found in polynomial time. Using this lower bound, we propose an algorithm that updates the lower bound on the cost by incrementally recomputing the optimal preemptive mean busy time schedule for a carefully structured subset of the possible start times of each activity. The time complexity of this algorithm is $O(n^4)$.

Empirical results on a set of single resource, minimum weighted completion time benchmarks in the literature show that the COMPLETION constraint significantly improves the performance of constraint-based approaches, which is a considerable result in the field of scheduling with "sum type" objective functions, an area where constraint programming has not yet been especially strong.

Our future work will examine the extension of this approach to cumulative resources, scheduling problems with multiple machines such as the job shop scheduling problem, and other problems with "sum type" cost constraints.

## Acknowledgment

A part of this work was carried out while A. Kovács was with the Cork Constraint Computation Centre, supported by an ERCIM fellowship. The authors acknowledge the support of the EU FP6 Net-WMS project and the Canadian Natural Sciences and Engineering Research Council, and ILOG, S.A.

## References

1. F. Afrati, E. Bampis, C. Chekuri, D. Karger, C. Kenyon, S. Khanna, I. Milis, M. Queyranne, M. Skutella, C. Stein, M. Sviridenko. Approximation Schemes for Minimizing Average Weighted Completion Time with Release Dates. In Proc. of the 40th IEEE Symposium on Foundations of Computer Science, pp. 32–44, 1999.
2. C. Akkan, S. Karabatı. The Two-machine Flowshop Total Completion Time Problem: Improved Lower Bounds and a Branch-and-bound Algorithm. European Journal of Operational Research 159:420–429, 2004.
3. J.M. van den Akker, C.A.J. Hurkens, M.W.P. Savelsbergh. Time-indexed Formulations for Machine Scheduling Problems: Column Generation. INFORMS Journal on Computing 12:111–124, 2000.
4. Ph. Baptiste, J. Carlier, A. Jouglet. A Branch-and-Bound Procedure to Minimize Total Tardiness on One Machine with Arbitrary Release Dates. European Journal of Operational Research 158:595–608, 2004.

5. Ph. Baptiste, L. Peridy, E. Pinson. A Branch and Bound to Mininimze the Number of Late Jobs on a Single Machine with Release Time Constraints. European Journal of Operational Research 144(1):1–11, 2003.
6. H. Belouadah, M.E. Posner, C.N. Potts. Scheduling with Release Dates on a Single Machine to Minimize Total Weighted Completion Time. Discrete Applied Mathematics 36:213–231, 1992.
7. B. Chen, C.N. Potts, G.J. Woeginger. A Review of Machine Scheduling: Complexity, Algorithms and Approximation. In: Handbook of Combinatorial Optimization, Vol. 3, pp. 21–169, Kluwer, 1998.
8. F. Della Croce, M. Ghirardi, R. Tadei. An Improved Branch-and-bound Algorithm for the Two Machine Total Completion Time Flow Shop Problem. European Journal of Operational Research 139:293–301, 2002.
9. M. Dyer, L.A. Wolsey. Formulating the Single Machine Sequencing Problem with Release Dates as Mixed Integer Program. Discrete Applied Mathematics 26:255–270, 1990.
10. F. Focacci, A. Lodi, M. Milano. Embedding Relaxations in Global Constraints for Solving TSP and TSPTW. Annals of Mathematics and Artificial Intelligence 34(4):291–311, 2002.
11. F. Focacci, A. Lodi, M. Milano. Optimization-Oriented Global Constraints. Constraints 7(3-4):351–365, 2002.
12. M.X. Goemans, M. Queyranne, A.S. Schulz, M. Skutella, Y. Wang. Single Machine Scheduling with Release Dates. SIAM Journal on Discrete Mathematics 15(2):165–192, 2002.
13. A. Jouglet, Ph. Baptiste, J. Carlier. Branch-and-Bound Algorithms for Total Weighted Tardiness. In: Handbook of Scheduling: Algorithms, Models, and Performance Analysis, Chapter 13, Chapman & Hall / CRC, 2004.
14. A. Kéri, T. Kis. Primal-dual Combined with Constraint Propagation for Solving RCPSPWET. In Proc. of the 2nd Multidisciplinary International Conference on Scheduling: Theory and Applications, pp. 748–751, 2005.
15. R. Nessah, F. Yalaoui, C. Chu. A Branch-and-bound Algorithm to Minimize Total Weighted Completion Time on Identical Parallel Machines with Job Release Dates. Computers and Operations Research, in print.
16. Y. Pan. Test Instances for the Dynamic Single-machine Sequencing Problem to Minimize Total Weighted Completion Time. Available at `http://www.cs.wisc.edu/~yunpeng/test/sm/dwct/instances.htm`
17. Y. Pan, L. Shi. New Hybrid Optimization Algorithms for Machine Scheduling Problems. IEEE Transactions on Automation Science and Engineering, in print.
18. A.S. Schulz. Scheduling to Minimize Total Weighted Completion Time: Performance Guarantees of LP-Based Heuristics and Lower Bounds. Proc. of the 5th Int. Conf. on Integer Programming and Combinatorial Optimization, pp. 301–315, 1996.