

# A Global Constraint for Total Weighted Completion Time for Cumulative Resources

András Kovács<sup>a</sup>, J. Christopher Beck<sup>b</sup>

<sup>a</sup>*Computer and Automation Research Institute  
Hungarian Academy of Sciences  
Budapest, Hungary  
akovacs@sztaki.hu*

<sup>b</sup>*Department of Mechanical & Industrial Engineering  
University of Toronto  
Toronto, Canada  
jcb@mie.utoronto.ca*

---

## Abstract

The criterion of total weighted completion time occurs as a sub-problem of combinatorial optimization problems in such diverse areas as scheduling, container loading and storage assignment in warehouses. These applications often necessitate considering a rich set of requirements and preferences, which makes constraint programming (CP) an effective modeling and solving approach. On the other hand, basic CP techniques can be inefficient in solving models that require inference over sum type expressions. In this paper, we address increasing the solution efficiency of constraint-based approaches to cumulative resource scheduling with the above criterion. Extending previous results for unary capacity resources, we define the  $\text{COMPLETION}_m$  global constraint for propagating the total weighted completion time of activities that require the same cumulative resource. We present empirical results in two different problem domains: scheduling a single cumulative resource, and container loading with constraints on the location of the center of gravity. In both domains, the proposed constraint propagation algorithm out-performs existing propagation techniques.

*Key words:* constraint programming, scheduling, total weighted completion time, global constraint, container loading

---

## 1 Introduction

Much of the success of constraint programming (CP) arises for the ability to declaratively model and solve a given problem with a combination of global constraints.

A global constraint captures a recurrent and important relation amongst a set of variables [19] and, importantly, has an associated algorithm for efficiently removing inconsistent values from the domains of the variables. The classic example of a global constraint is the all-different constraint which imposes that a set of variables must each take a unique value and which has an inference algorithm based on matchings in bi-partite graph [14]. As CP has been increasingly used to address combinatorial optimization problems, the need to explicitly incorporate reasoning about costs within global constraints has been recognized [8]. In this paper, we continue this line of work by addressing the objective function of the total weighted completion time of a set of jobs that must share a single cumulative resource. While such a constraint is directly applicable to and, indeed, most clearly explained in the context of, scheduling problems, there are a wide variety of combinatorial optimization problems in other domains for which this constraint is useful. In particular, in this paper, after validating the constraint on scheduling problems, we use it to model and solve container loading problems. Our empirical results demonstrate significant improvement in problem solving in both domains as compared to the standard weighted-sum constraint.

In the next section, we provide background on the unary capacity COMPLETION constraint and a short discussion of related work. We then present the details of the cumulative COMPLETION<sub>m</sub> constraint including our proposed propagation algorithm. Section 4 empirically validates the global constraint by applying it to single machine scheduling problems. In Section 5, we turn to the container loading problem and demonstrate how such a problem can be modeled and solved with the use of the COMPLETION<sub>m</sub> constraint. We then conclude in Section 6.

## 2 Previous Work

A critical component of the success of CP techniques to optimization problems is the ability to design a model that exhibits significant *back propagation*. Back propagation is the reduction in search space through pruning of the domains of decision variables as a result of a new bound on the objective function. Models that exhibit a high degree of back propagation tend to be successful as each new (sub-optimal) solution will result in a tighter bound on the cost and, in turn, a smaller subsequent search space. In contrast, without back propagation, the full search space will need to be explored, suggesting that CP will not result in better performance than any other solution technique.

The significance of cost-based global constraints for strong back propagation has been emphasized by Focacci et al. [8]. The same authors define a global constraint for the path cost in traveling salesman problems in [7,8]. Further examples of cost-based constraints with efficient propagation algorithms are the global cardinality constraint with costs [15], the minimum weight all-different constraint [18], the

cost-regular constraint [5], and the inequality-sum constraint [16]. The latter constraint propagates objective functions of the form  $\sum_i x_i$  where variables  $x_i$  are subject to binary inequality constraints of the form  $x_j - x_i \leq c$ . The inequality-sum constraint can be applied to propagate the total (non-weighted) completion time or total (non-weighted) tardiness objective functions in a special family of scheduling problems, the Simple Temporal Problems (STP) [4]. An STP involves finding start times for activities subject to minimum and maximum time lags between activities, but it provides no means for representing limited capacity resources.

The COMPLETION constraint is a recently proposed global constraint to propagate the total weighted completion time of activities on a single unary capacity resource [10]. Formally, the COMPLETION constraint is defined as follows:

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

where there are  $n$  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  $A_i$  is denoted by  $S_i$  with  $C_i = S_i + p_i$  being the activity's completion time in the schedule. The total weighted completion time of the activities will be denoted by  $C$ . We assume that all data are integral. The constraint enforces  $C = \sum_i w_i(S_i + p_i)$ .

The COMPLETION constraint can be viewed as requiring a solution to a scheduling problem denoted as  $1|r_i, d_i|\sum w_i C_i$  in the classical scheduling notation. Though this problem is NP-hard, a pre-emptive variant with a slightly modified objective function is solvable in polynomial time and can serve as a lower bound. Let  $M_i$  be the mean busy time of activity  $A_i$ , i.e., the mean point in time at which the machine is busy processing activity  $A_i$ . The problem  $1|r_i, pmtn|\sum w_i M_i$  can be solved in  $O(n \log n)$  where  $n$  is the number of activities [9]. The COMPLETION constraint filters the domains of the start time variables by computing the cost of the optimal preemptive mean-busy time relaxation for each activity  $A_i$  and each possible start time  $t$  of activity  $A_i$ , with the added constraint that activity  $A_i$  must start at time  $t$ . If the cost of the relaxed solution is greater than the upper bound on the cost, then  $t$  is removed from the domain of  $S_i$ . In practice, the relaxation is not actually re-solved for each start time in the domain of each activity. Instead, a much faster algorithm transforms an initial relaxed solution into a series of preemptive schedules with given start time assignments. For a detailed presentation of this algorithm and the COMPLETION constraint readers are referred to [10].

### 3 Propagating Total Weighted Completion Time on a Cumulative Resource

Extending the COMPLETION constraint to a cumulative resource, we introduce the global constraint  $\text{COMPLETION}_m$ , which states that given a set of non-preemptive activities  $\{A_1, \dots, A_n\}$  that require the same cumulative resource with capacity  $R$ , the total weighted completion time of these activities is  $C$ . The constraint takes the form

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

where finite-domain variables  $S_i$  stand for the start time of  $A_i$ , while  $p_i$ ,  $\varrho_i$ , and  $w_i$  denote the duration, the capacity requirement, and the weight of  $A_i$ , respectively. The cost variable  $C$  is also a finite-domain variable. We assume that  $p_i$ ,  $\varrho_i$ ,  $w_i$ , and  $R$  are non-negative integer constants, however our approach can be easily adapted to reasoning with the lower bounds of  $p_i$ ,  $\varrho_i$ , and  $w_i$ , and the upper bound of  $R$ . The minimum and maximum values in the current domain of a variable  $X$  will be denoted by  $\check{X}$  and  $\hat{X}$ , respectively. When appropriate, we call the current lower bound of a start time variable,  $\check{S}_i$ , the *release time* of the activity, and denote it by  $r_i$ . For brevity, we denote the relative weight of an activity by  $\mu_i = w_i/p_i\varrho_i$ .

The  $\text{COMPLETION}_m$  constraint tightens the lower and the upper bounds of variables  $S_i$ , but only the lower bound of  $C$ . The latter is sufficient in problems where the cost is to be minimized. In other applications, the upper bound of  $C$  can be tightened by posting a  $\text{COMPLETION}_m$  constraint on a flipped schedule.

#### 3.1 The Relaxed Problem

A standard approach to the effective propagation of a global constraint is to embed a polynomially solvable relaxation into the constraint. For the  $\text{COMPLETION}_m$  constraint, we seek a relaxation of the single cumulative resource total weighted completion time problem that simultaneously considers capacity constraints, resource requirements, and release times. Unlike in the unary case, such a relaxation does not appear to have been presented in the literature. We therefore propose a novel relaxation of this scheduling problem.

In the relaxed problem, defined in Fig.1, we assume that activities  $A_i$  can be executed with a varying intensity over time. That is, in each time period  $[t, t + 1)$ ,  $t = 0, \dots, T - 1$ , an intensity  $x_t^i$  of  $A_i$  can be chosen from  $[0, R]$ . As an extremity of varying intensity, preemption is allowed. The sum of the intensities over time has

Minimize

$$\sum_{i,t} t \mu_i x_t^i + \frac{1}{2} \sum_i w_i \quad (1)$$

s.t.

$$\forall i, t \quad x_t^i \geq 0 \quad (2)$$

$$\forall i \quad \sum_t x_t^i = p_i \rho_i \quad (3)$$

$$\forall t \quad \sum_i x_t^i \leq R \quad (4)$$

$$\forall i, t \quad \sum_{t'=0}^t x_{t'}^i \leq \begin{cases} 0 & \text{if } t < r_i \\ (t - r_i + 1) \rho_i & \text{if } t \geq r_i \end{cases} \quad (5)$$

Fig. 1. The variable-intensity relaxation of the cumulative resource scheduling problem.

to match the original volume of the activity (3), and the capacity constraint must be respected (4). The release time constraint (5) now states that  $A_i$  cannot be processed before  $r_i$ , and for  $t \geq r_i$ , the maximum volume of  $A_i$  processed in  $[0, t]$  grows linearly with  $t$ . The objective is to minimize the total weighted mean busy time of the activities ( $\sum_{i,t} t \mu_i x_t^i$ ), shifted by a constant ( $\frac{1}{2} \sum_i w_i$ ) that compensates for the difference between the mean busy time and the total weighted completion time criteria. Below, we differentiate between the original problem and the variable-intensity relaxation by denoting the former as  $\Pi$ , and the latter as  $\Pi'$ .  $C'$  will stand for the cost of the optimal relaxed solution to  $\Pi'$ .

**Proposition 1**  $C'$  is a valid lower bound on the original problem  $\Pi$ .

**Proof:** Let us denote the optimal schedule for  $\Pi$  by  $\sigma$  and its total weighted completion time by  $C$ . Since each activity is processed without preemption in  $\sigma$ , the difference between the total weighted completion time (weighted sum of the end times,  $\sum_i w_i C_i$ ) and the mean busy time (weighted sum of the mid-points,  $\sum_i w_i M_i$ ) equals  $\frac{1}{2} \sum_i w_i p_i$ . In the variable intensity representation, the mean busy time of  $\sigma$  can be calculated as  $\sum_{i,t} t \mu_i x_t^i$ . Note that  $\sigma$  is a feasible solution of the relaxed problem  $\Pi'$  as well, and its relaxed solution cost equals

$$\sum_{i,t} t \mu_i x_t^i + \frac{1}{2} \sum_i w_i p_i = \sum_i w_i M_i + \frac{1}{2} \sum_i w_i p_i = \sum_i w_i C_i = C$$

Since  $\sigma$  is a possibly sub-optimal relaxed solution, its cost,  $C$ , is greater or equal to the cost of the optimal relaxed solution  $C'$ . Hence,  $C'$  is a lower bound on  $C$ .  $\square$

The optimal solution of the variable-intensity mean busy time relaxation can be computed using the following procedure, called `PrepareRelaxed()`, which constructs the schedule chronologically. At each point in time  $t$  when a schedul-

ing decision has to be made, the algorithm assigns intensities to activities in the order of non-increasing  $\mu_i$ . The intensity of  $A_i$  will be the minimum of

- the intensity allowed by constraint (5),  $\max(0, (t - r_i + 1)\varrho_i) - \sum_{t'=0}^{t-1} x_t^i$ ;
- the remaining volume of  $A_i$ ,  $p_i\varrho_i - \sum_{t'=0}^{t-1} x_t^i$ ;
- the remaining capacity for the subsequent time period.

These intensity values are applied until an activity can no longer be processed at this rate or the algorithm reaches the release time of another activity. The algorithm finishes when all activities are completely processed. Since the number of scheduling decisions is at most  $O(n^2)$  and intensities can be assigned in  $O(n)$  time, the overall time complexity of the algorithm is  $O(n^3)$ . Note that the complexity is independent of the length of the scheduling horizon. The pseudo-code of the algorithm is presented in Fig. 2.

**Proposition 2** *The above algorithm builds an optimal schedule for the variable-intensity mean busy time problem.*

**Proof:** Let  $\sigma$  be an arbitrary feasible schedule that differs from schedule  $\sigma^*$  built by our algorithm, such that the difference cannot be characterized by an interchange of intensities between activities with identical relative weights. Let  $t_1$  be the earliest point in time and  $A_{i1}$  be the activity with the highest  $\mu_{i1}$  such that  $x_{t_1}^{i1}[\sigma] \neq x_{t_1}^{i1}[\sigma^*]$ . The construction of the algorithm ensures that  $x_{t_1}^{i1}[\sigma] < x_{t_1}^{i1}[\sigma^*]$ . Then, there exists a time  $t_2$  and activity  $A_{i2}$  with  $t_1 < t_2$  and  $\mu_{i1} > \mu_{i2}$  such that increasing  $x_{t_1}^{i1}[\sigma]$  and  $x_{t_2}^{i2}[\sigma]$  and decreasing  $x_{t_1}^{i2}[\sigma]$  and  $x_{t_2}^{i1}[\sigma]$  preserves feasibility and improves the objective value of  $\sigma$ . Therefore a schedule that differs essentially from the one built by the algorithm cannot be optimal.  $\square$

### 3.2 From Relaxed Solutions to Bounds Tightening

The above presented algorithm can easily be modified to `PrepareRelaxed( $A_i, t$ )`, which computes optimal relaxed solutions with the additional constraint that activity  $A_i$  must start at  $t$ . This can be achieved by assigning  $r_i = t$  and  $\mu_i = \infty$ , which gives activity  $A_i$  the largest relative weight among all the activities. This ensures that  $A_i$  starts at  $t$  and it is processed at intensity  $\varrho_i$  throughout its duration. This restricted relaxed problem will be denoted by  $\Pi'\langle S_i, t \rangle$ , and the value of its optimal solution by  $C'\langle S_i, t \rangle$ . Our approach to tightening the bounds of the start time variables  $S_i$  exploits the following proposition.

**Proposition 3** *If  $C'\langle S_i, t \rangle > \hat{C}$ , then  $t$  can be removed from the domain of  $S_i$ .*

Unlike in the unary case, we are not able to define methods for computing  $C'\langle S_i, t \rangle$  for *all* values of  $i$  and  $t$  in low-degree polynomial time. Instead, we apply `PrepareRelaxed` to compute two relaxed solutions for each activity, for the situations

```

PROCEDURE PrepareRelaxed()
   $\sigma :=$  an empty schedule
   $t := \min_i r_i$ 
   $\forall i \ v_i := 0$            % Volume of activity not yet processed
  WHILE there is an activity not completely processed
     $R'_t := R$            % Remaining capacity at time  $t$ 
    FORALL  $A_i$  ordered by non-decreasing  $\mu_i$ 
       $x_t^i := \min(\max(0, (t - r_i + 1)\varrho_i) - \sum_{t'=0}^{t-1} x_{t'}^i,$ 
         $p_i\varrho_i - \sum_{t'=0}^{t-1} x_{t'}^i,$ 
         $R'_t)$ 
      IF  $r_i > t$  THEN
         $\Delta_i = r_i - t$ 
      ELSE IF  $x_t^i > \varrho_i$  THEN
         $\Delta_i := \min(\frac{v_i}{x_t^i}, \frac{(t-r_i)\varrho_i - v_i}{x_t^i - \varrho_i})$ 
      ELSE
         $\Delta_i := \frac{v_i}{x_t^i}$ 
       $R'_t := R'_t - x_t^i$ 
     $\Delta = \min_i \Delta_i$ 
    Add to  $\sigma$  a fragment  $[t, t + \Delta]$  with intensities  $x_t^i$ 
     $t := t + \Delta$ 
  RETURN  $\sigma$ 

```

Fig. 2. Algorithm for solving the variable-intensity relaxed problem.

when  $A_i$  starts at  $\check{S}_i$  and when it starts at  $\hat{S}_i$ . If either of these relaxed solutions violate the current upper bound on the cost, then we estimate how the current bounds of  $S_i$  must be modified to achieve consistency. Since our methods can under-estimate the change that is necessary, this procedure must be iterated until consistent bounds are found for  $S_i$  (see Fig. 3). We note that in some uncommon cases the number of adjustment cycles needed to achieve consistent bounds can be large, which is computationally costly. For this reason, we limited the number of cycles to 5.

### 3.2.1 Adjusting the Earliest Start Time

In order to adjust  $\check{S}_i$ , procedure `AdjustEST`( $\sigma, A_i$ ) departs from the relaxed solution  $\sigma$  prepared for  $\Pi' \langle S_i = \check{S}_i \rangle$ . It investigates how the relaxed cost changes as the start of  $A_i$  is increased from  $\check{S}_i$  to  $t > \check{S}_i$ .

Since the exact computation of the relaxed cost for all possible values of  $t$  is computationally expensive, our procedure exploits a further relaxation. We take release times  $r_j$  such that  $j \neq i$  and  $\check{S}_i < r_j < t + p_i$ , and relax  $r_j$  to  $r'_j = \check{S}_i$ . This leads to relaxed solutions in which intensities  $x_{t'}^j$  with  $t' < \check{S}_i$  or  $t' \geq t + p_i$  equal the corresponding intensities in  $\sigma$ . Activity  $A_i$  is processed from  $t$  until  $t + p_i$  at intensity  $\varrho_i$ . However, in interval  $[\check{S}_i, t + p_i]$  activities other than  $A_i$  will be processed in the non-increasing order of  $\mu_j$ .

```

PROCEDURE TightenBounds ()
  FORALL activity  $A_i$ 
     $\sigma := \text{PrepareRelaxed}(A_i, \check{S}_i)$ 
    WHILE  $\text{cost}(\sigma) > \hat{C}$ 
      UPDATE  $\check{S}_i \leftarrow \text{AdjustEST}(\sigma, A_i)$ 
       $\sigma := \text{PrepareRelaxed}(A_i, \check{S}_i)$ 
     $\sigma := \text{PrepareRelaxed}(A_i, \hat{S}_i)$ 
    WHILE  $\text{cost}(\sigma) > \hat{C}$ 
      UPDATE  $\hat{S}'_i \leftarrow \text{AdjustLST}(\sigma, A_i)$ 
       $\sigma := \text{PrepareRelaxed}(A_i, \hat{S}'_i)$ 

```

Fig. 3. Algorithm for tightening the bounds of the start time variables.

Solutions to this further relaxed problem can be easily found for all relevant values of  $t$ . Only a small section of the schedule, namely the section in interval  $[\check{S}_i, t + p_i]$  varies over different values of  $t$ . However, this section can be represented as a queue. The queue, as well as the cost of the schedule, is updated incrementally for subsequent values of  $t$  at time  $O(n)$  for each step. This step is iterated until the cost of  $\sigma_t$  decreases below the current upper bound cost  $\hat{C}$ . The earliest start time of  $A_i$  is then updated to this value of  $t$ .

### 3.2.2 Adjusting the Latest Start Time

Given an optimal relaxed solution  $\sigma$  with cost  $C'$  for  $\Pi' \langle S_i = \hat{S}_i \rangle$ , let  $t^*$  denote the earliest point in time with  $t^* \geq \hat{S}_i + p_i$  in this relaxed solution such that all the volume of the activities that has been released before  $t^*$  is processed before  $t^*$ :

$$\forall j \sum_{t=0}^{t^*-1} x_t^j = \min(p_j \varrho_j, (t^* - r_j) \varrho_j).$$

Furthermore, let  $W$  denote the total weight of activity fragments processed between  $\hat{S}_i$  and  $t^*$ , including activity  $A_i$ :

$$W = \sum_{j=1}^n \sum_{t'=\hat{S}_i}^{t^*-1} \mu_j x_{t'}^j.$$

Procedure  $\text{AdjustLST}(\sigma, A_i)$  computes these values  $t^*$  and  $W$ , and adjust the latest start time of  $A_i$  according to the following proposition.

**Proposition 4** *Activity  $A_i$  cannot start later than  $\hat{S}'_i = \hat{S}_i - \lceil \frac{C' - \hat{C}}{W} \rceil$ .*

**Proof:** Let us divide the scheduling horizon to four intervals,  $I_1 : [0, \hat{S}'_i)$ ,  $I_2 :$



$[\hat{S}'_i, \hat{S}_i)$ ,  $I_3 : [\hat{S}_i, t^*)$ , and  $I_4 : [t^*, \infty)$ . The construction of the `PrepareRelaxed` algorithm ensures that decreasing  $r_i$  from  $\hat{S}_i$  to  $\hat{S}'_i$  does not affect activity intensities in intervals  $I_1$  and  $I_4$ . Activity fragments in  $I_2$  may be swapped later, while activity fragments in  $I_3$  will be swapped earlier by at most  $\hat{S}_i - \hat{S}'_i$ . Since the total weight of activities in  $I_3$  is exactly  $W$ , this operation decreases the cost of the relaxed solution by at most  $W(\hat{S}_i - \hat{S}'_i) \leq C' - \hat{C}$ .  $\square$

### 3.2.3 Computational Complexity

For a given activity  $A_i$ , the time and space complexity of computing optimal relaxed solutions for  $\Pi' \langle S_i = \check{S}_i \rangle$  and  $\Pi' \langle S_i = \hat{S}_i \rangle$  is  $O(n^3)$ , which equals the maximum size of a relaxed solution. A single run of the functions `AdjustEST(.)` and `AdjustLST(.)` takes at most the same amount of time. However, we cannot give an estimation of the number of recomputation cycles that are necessary to achieve consistent bounds. With a constant limit on the number of cycles (as it is done in our implementation), the time complexity of the propagation algorithm is  $O(n^3)$  for a single activity, and  $O(n^4)$  for a complete run on  $n$  activities.

## 4 Scheduling a Single Cumulative Resource

To evaluate the `COMPLETIONm` constraint in isolation, we ran computational experiments on a set of single cumulative resource scheduling problems with release times, with the objective of minimizing total weighted completion time. The variables in the constraint-based model of the problem are the start times of the activities. They are subject to inequality constraints expressing release times, and a single cumulative resource constraint. We compared the performance of the `SUM` and the `COMPLETIONm` models. The former uses the standard weighted-sum constraint to propagate the objective function, the latter uses the `COMPLETIONm` constraint. The proposed propagation algorithms have been implemented in C++ and embedded into `ILOG Solver` and `Scheduler` versions 6.1. The resource capacity constraint is enforced using the timetable and the disjunctive constraints [17].

We applied a depth-first search with a customized version of the *SetTimes* branching heuristic [11,17]: in each search node we identify the set of activities that has not been either scheduled or postponed. We then select the activity with the minimum start time, breaking ties by choosing the activity with greatest  $\mu_i$ . If any postponed activity can be scheduled before the selected activity, search backtracks immediately as such a decision would lead to a state that we have already searched over. Otherwise a choice point is created assigning the selected activity to its minimum start time and, on the other branch, postponing the activity. An activity is no longer postponed when its minimum start time is modified via constraint propagation.

Our set of benchmark instances consisted of problems with the number of activities,  $n$ , taken from  $\{15, 20, 25, 30\}$ , the resource requirement range  $\alpha$  from  $\{0.5, 1.0\}$ , and the release time range  $\beta$  from  $\{0, 0.2, 0.6, 1.0\}$ . For each combination of the above values we generated 10 instances giving 320 problem instances in total. The capacity of the resource was fixed to  $R = 10$ . Activity durations,  $p_i$ , were randomized from  $[1, 100]$  with a discrete uniform distribution, weights,  $w_i$ , from  $[1, 10]$ , and resource requirements,  $\varrho_i$ , from  $[1, \alpha R]$ . This leads to instances where approximately  $k = \frac{2}{\alpha} - \frac{1}{2}$  activities are processed in parallel on the resource. Hence, the release times were randomized from  $[0, 50.5n\beta/k]$ . The experiments were run on a 1.86 GHz Intel Xeon computer with 2 GB of RAM under a MS Windows Server 2003, with a time limit of 1200 seconds.

The experimental results are presented in Table 1. Each row of the table contains combined results for given values of  $n$  and  $\beta$ , achieved with the SUM and the COMPLETION <sub>$m$</sub>  models. The results do not depend significantly on the value of  $\alpha$ . For each model, *Opt* is the number of instances out of 20 for which the optimal solution is found and proved. Columns *MRE* and *XRE* contain the mean and maximum relative error compared to the best solution known. Column *Nodes* shows the average number of search nodes, while *Time* presents the average search time, including the proof of optimality, or 1200 seconds where the solver hit the time limit.

Problem instances with few activities ( $n = 15$ ) or release times varying substantially ( $\beta = 1.0$ ) were easily solved by both models. In contrast, neither model could find and prove optimal solutions for the larger and tighter problems. The results confirm that the COMPLETION <sub>$m$</sub>  constraint reduced the search space efficiently for all the instances: it helped find optimal solutions quicker for the easy problems, prove optimality more often, and construct better solutions for the hard instances. The latter is the most significant advantage of the COMPLETION <sub>$m$</sub>  model, since the difference between the solutions found reached up to 19% in favor of the COMPLETION <sub>$m$</sub>  model. Notably, the COMPLETION <sub>$m$</sub>  model always found at least as good solution as the SUM model for each problem, except for a single problem instance (with  $n = 30$  and  $\beta = 0.6$ , the difference was 0.74%).

## 5 Using COMPLETION <sub>$m$</sub> to Solve the Container Loading Problem

Our primary interest in developing the COMPLETION <sub>$m$</sub>  constraint is to use it as a component in a variety of combinatorial optimization problems. In order to evaluate its wider applicability, in this section we use the COMPLETION <sub>$m$</sub>  constraint to model and solve the container loading problem.

The container loading problem involves the placement of a set of items in a container. While the core of the problem corresponds to bin packing with the objec-

$n$	$\beta$	SUM					COMPLETION <sub><math>m</math></sub>				
		Opt	MRE	XRE	Nodes	Time	Opt	MRE	XRE	Nodes	Time
15	0.0	20	0.00	0.00	1221819	37.8	20	0.00	0.00	41022	16.2
	0.2	20	0.00	0.00	80897	2.2	20	0.00	0.00	3359	1.6
	0.6	20	0.00	0.00	5175	0.1	20	0.00	0.00	413	0.0
	1.0	20	0.00	0.00	5714	0.2	20	0.00	0.00	198	0.1
20	0.0	0	0.25	1.35	33724243	1200.0	15	0.00	0.00	583550	589.9
	0.2	15	0.91	13.84	14223048	494.5	20	0.00	0.00	48554	50.0
	0.6	20	0.00	0.00	813644	36.0	20	0.00	0.00	6654	7.0
	1.0	20	0.00	0.00	3225	0.0	20	0.00	0.00	549	0.2
25	0.0	0	2.40	8.61	30768029	1200.0	2	0.00	0.00	1136717	1149.0
	0.2	0	3.11	11.30	31261661	1200.0	11	0.00	0.00	942258	763.2
	0.6	14	1.03	10.48	10424189	401.1	20	0.00	0.00	77950	124.1
	1.0	20	0.00	0.00	410177	14.8	20	0.00	0.00	4058	4.5
30	0.0	0	4.01	14.71	31718440	1200.0	0	0.00	0.00	1550706	1200.0
	0.2	0	3.71	18.72	27453645	1200.0	3	0.00	0.00	1273294	1093.9
	0.6	6	0.42	3.81	21736970	946.8	15	0.04	0.74	249753	457.0
	1.0	20	0.00	0.00	3132597	157.1	20	0.00	0.00	50870	77.1

Table 1

Experimental results: number of instances (out of 20) solved to optimality (Opt), mean and maximum relative error in percent (MRE and XRE), average number of search nodes (Nodes) and average search time in seconds (Time) with the SUM and the COMPLETION <sub>$m$</sub>  model.

tive of high volumetric utilization, in practical applications there are various further requirements. These include stacking conditions, cargo stability, and visibility and accessibility considerations. The rich set of requirements makes CP an attractive modeling approach in this domain [12]. Among the additional requirements, Davies & Bischoff [3] highlight the importance of the weight distribution of the loaded container, focusing on the location of the center of gravity (COG). The exact requirements depend on the specific application, especially on the means of transport. For example, in aircraft loading, or when loading a container that will be lifted by a crane, the COG of the cargo has to be located in the center of the container. Since the length of the container (or the aircraft hull) is much greater than its width or height, the longitudinal balance is the most important issue. In contrast, in road transport, it is often preferred to have the COG above the axles of the vehicle.

For container loading with weight distribution considerations, Davies & Bischoff [3] proposed a heuristic that achieves high space utilization combined with an even weight distribution, and claim that the latter leads to a COG located near to the center of the container. Wodziak & Fadel [20] apply a genetic algorithm to minimize the distance of the COG from the desired location in one, two, and two and a half dimensions. Various authors have focused on the longitudinal balance, and

approached the problem from a one-dimensional perspective (e.g. [13]). Fasano [6] proposed a mixed-integer programming approach to solving the 3D single bin packing problem with orthogonal rotation allowed and the center of gravity location constrained in all three dimensions.

We address the problem of loading box-shaped items into a rectangular container, with rotation disallowed. The location of the COG can be constrained to an arbitrary rectangular region of the container. For simplicity, we present our results on the two-dimensional variant of the problem, although it is straightforward to extend the model to  $k$  dimensions.

### 5.1 Modeling the Problem

Let there be given a set of two-dimensional boxes  $B_1, \dots, B_n$  that have to be placed in a rectangular container of length  $L$  and width  $W$ . Box  $B_i$  is characterized by its length  $a_i$ , width  $b_i$ , and weight  $w_i$ . The placement of  $B_i$  is described by two integer variables  $x_i$  and  $y_i$  that stand for the horizontal and vertical origin of the box, respectively. Assuming homogeneous boxes, the COG of the cargo is then located at

$$(x^{COG}, y^{COG}) = \left( \frac{\sum_i w_i (x_i + \frac{a_i}{2})}{\sum_i w_i}, \frac{\sum_i w_i (y_i + \frac{b_i}{2})}{\sum_i w_i} \right).$$

The objective is to find a placement of the boxes in the container such that the COG of the cargo is situated in the rectangular area defined by  $x_{min}^{COG}$ ,  $x_{max}^{COG}$ ,  $y_{min}^{COG}$ , and  $y_{max}^{COG}$ . In order to avoid fractional variables and facilitate simpler computation, our model works with scaled variables  $x^*$  and  $y^*$  as follows.

$$\begin{aligned} x^* &= \sum_i w_i (x_i + a_i) = \alpha x^{COG} + \beta_x \\ y^* &= \sum_i w_i (y_i + b_i) = \alpha y^{COG} + \beta_y, \end{aligned}$$

where  $\alpha = \sum_i w_i$ ,  $\beta_x = \frac{1}{2} \sum_i w_i a_i$  and  $\beta_y = \frac{1}{2} \sum_i w_i b_i$ . The bounds on the COG location in the scaled representation can be described by the following constants.

$$\begin{aligned} x_{min}^* &= \alpha x_{min}^{COG} + \beta_x & y_{min}^* &= \alpha y_{min}^{COG} + \beta_y \\ x_{max}^* &= \alpha x_{max}^{COG} + \beta_x & y_{max}^* &= \alpha y_{max}^{COG} + \beta_y \end{aligned}$$

Then, the container loading problem can be stated as displayed in Fig. 4.

$$\forall i \quad 0 \leq x_i \leq L - a_i \quad (6)$$

$$\forall i \quad 0 \leq y_i \leq W - b_i \quad (7)$$

$$\forall i, j \neq i \quad (x_i + a_i \leq x_j) \vee (x_j + a_j \leq x_i) \vee (y_i + b_i \leq y_j) \vee (y_j + b_j \leq y_i) \quad (8)$$

$$x^* = \sum_i w_i(x_i + a_i) \quad (9)$$

$$y^* = \sum_i w_i(y_i + b_i) \quad (10)$$

$$x_{min}^* \leq x^* \leq x_{max}^* \quad (11)$$

$$y_{min}^* \leq y^* \leq y_{max}^* \quad (12)$$

$$(13)$$

Fig. 4. The SUM model of container loading problem.

Constraints (6) and (7) state that the boxes have to be located inside the container. The boxes must not overlap (8). Finally, inequalities (9-12) determine the location of the COG and constrain it in both dimensions. Strong propagation is trivial for the inequality constraint, and several current constraint solvers offer efficient global constraints for propagating the non-overlapping relation of a set of boxes [1,2]. The weakness of this model is the propagation of the weighted-sum constraints in equalities (9) and (10). This model will be referred to as the SUM model. In the sequel we demonstrate how the  $COMPLETION_m$  constraint can be used to strengthen the propagation of the constraints on the COG location.

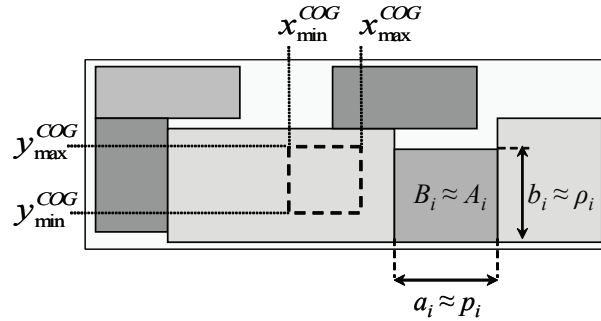


Fig. 5. Similarities of a container loading and a cumulative resource scheduling problem.

Observe that  $x^*$  equals the total weighted completion time in a single cumulative resource scheduling problem, where the container corresponds to the hull of the schedule, defined by the scheduling horizon (horizontal axis) and the resource capacity (vertical axis). Boxes correspond to activities, with box length standing for activity duration and box width for the resource requirement of the activity. The physical weight of the box corresponds to the weight assigned to activity  $A_i$ .<sup>1</sup>

<sup>1</sup> Cumulative resource scheduling is a relaxation of container loading, rather than an equivalent problem. The difference is that in scheduling it is allowed to assign any two units of

Container loading	Scheduling
Container length	Scheduling horizon
Container width	Resource capacity
Box	Activity
Box length	Activity duration
Box width	Activity's resource requirement
COG location	Total weighted completion time (scaled)

Table 2

Corresponding notions in container loading and scheduling.

Fig. 5 illustrates the similarities of the container loading and the scheduling problems, while the corresponding notions in the two problem domains are summarized in Table 2. The same relation holds between  $y^*$  and the total weighted completion time in the schedule that is received by rotating Fig. 5 by 90 degrees. Furthermore, since the  $\text{COMPLETION}_m$  constraint propagates only the lower bound of the cost but we need both lower and upper bounds on  $x^*$  and  $y^*$ , we post the  $\text{COMPLETION}_m$  constraint on flipped schedules as well. Hence, we define the  $\text{COMPLETION}_m$  model of the container loading problem by adding the following four global constraints to the basic SUM model.

$$\begin{aligned}
& \text{COMPLETION}_m([x_1, \dots, x_n], [a_1, \dots, a_n], [b_1, \dots, b_n], [w_1, \dots, w_n], W, x^*) \\
& \text{COMPLETION}_m([L - x_1 - a_1, \dots, L - x_n - a_n], [a_1, \dots, a_n], [b_1, \dots, b_n], \\
& \quad [w_1, \dots, w_n], W, x^* + \sum_i w_i(L + 2a_i)) \\
& \text{COMPLETION}_m([y_1, \dots, y_n], [b_1, \dots, b_n], [a_1, \dots, a_n], [w_1, \dots, w_n], L, y^*) \\
& \text{COMPLETION}_m([W - y_1 - b_1, \dots, W - y_n - b_n], [b_1, \dots, b_n], [a_1, \dots, a_n], \\
& \quad [w_1, \dots, w_n], L, y^* + \sum_i w_i(W + 2b_i))
\end{aligned}$$

## 5.2 Computational Experiments

The performance of the  $\text{COMPLETION}_m$  and the SUM models of the container loading problem was compared in computational experiments. We addressed the satisfiability problem presented above, hence the outcome of the solution process was either a feasible solution or a proof that the boxes could not be loaded with an appropriate weight distribution. A depth-first search was implemented for solving the problem using a simple *labeling* heuristic (called IloGenerate in Ilog) that creates two children of a search node:  $(x_i = \check{x}_i)$  vs.  $(x_i > \check{x}_i)$  or  $(y_i = \check{y}_i)$  vs.  $(y_i > \check{y}_i)$ .

the resource to an activity, while in container loading, adjacent units of the container surface must be assigned to a box. For instance, it is impossible to place one half of the box to the lowermost, the other half to the uppermost region of the container.

We experimented with more sophisticated branching heuristics as well that made decisions on the relative positions of two boxes, but the simple strategy proved to be the best with both models.

Problem instances have been generated with the number of boxes,  $n$ , varying between 10 and 45 with increments of 5, and the container length,  $L$ , varying between 20 and 60 with increments of 10. The container width was fixed to 8. Box sizes  $a_i$  and  $b_i$  were taken randomly from  $[1, 5]$ , while weights  $w_i$  from  $[1, 10]$  with a discrete uniform distribution. We focused on the longitudinal balance, and imposed  $x_{min}^{COG} = \frac{L}{2} - 1$  and  $x_{max}^{COG} = \frac{L}{2} + 1$ . For each value of  $n$  and  $L$ , we generated 10 instances, which resulted in 400 instances altogether.

The models were implemented in ILOG Solver and Scheduler 6.1, using the earlier presented  $COMPLETION_m$  constraint, written in C++. The experiments were run on a 1.86 GHz Intel Xeon computer with 2 GB of RAM under a MS Windows Server 2003 operating system, with a time limit of 1200 CPU seconds.

The experimental results are displayed in Table 3. Each row contains results for a given number of boxes. Columns *Solved*, *Nodes*, and *Time* show the percentage of solved instances, the average number of search nodes, and the average search time for either of the SUM and the  $COMPLETION_m$  models. *Nodes* and *Time* also contain the elapsed effort and time for instances where the time limit was reached. Instances in which the boxes did not fit into the container, independent of the COG location constraints, have been excluded from the experimental results.

The results illustrate that propagation by the  $COMPLETION_m$  constraint efficiently reduced the search space in the container loading problem, resulting in an order of magnitude reduction in solution times. Furthermore, the  $COMPLETION_m$  model scaled much better with  $n$ . It solved 75% of the largest instances experimented as well, whereas those were practically intractable with the SUM model.

$n$	SUM			$COMPLETION_m$		
	Solved (%)	Nodes	Time (sec)	Solved (%)	Nodes	Time (sec)
10	100.0	27	0.0	100.0	16	0.0
15	100.0	71530	30.5	100.0	314	0.1
20	97.6	195948	102.1	97.6	25645	28.8
25	68.6	638110	415.3	100.0	4190	6.3
30	40.0	517817	840.5	90.0	62148	133.7
35	28.6	484507	869.1	92.9	32192	139.6
40	31.6	301704	875.6	78.9	71021	301.5
45	8.3	177966	1154.3	75.0	79416	486.5

Table 3

Experimental results on the container loading problem, for the SUM and the  $COMPLETION_m$  models.

## 6 Conclusions

In this paper we investigated constraint-based scheduling with the total weighted completion time criterion. This scheduling problem is interesting because it appears as a sub-problem of combinatorial optimization problems in diverse areas, such as container loading or storage assignment in warehouses. Extending the earlier defined COMPLETION constraint, we introduced the COMPLETION<sub>m</sub> global constraint for propagating the total weighted completion time of activities on a cumulative resource. Our propagation algorithm is based on a novel variable-intensity relaxation of the single cumulative resource scheduling problem.

To evaluate the applicability of the COMPLETION<sub>m</sub> constraint with different sets of side constraints, we performed computational experiments in two different problem domains: single cumulative resource scheduling and container loading with weight distribution considerations. We showed that the location of the center of gravity of the cargo in a container corresponds to the total weighted completion time of the activities in a cumulative resource scheduling problem, and hence, can be propagated by the COMPLETION<sub>m</sub> constraint. In both domains, the introduction of the novel global constraint led to significant improvement compared to the classical constraint-based model of the problem: better solutions, or the same solutions found often an order of magnitude faster.

**Acknowledgments** This research was supported in part by the Natural Sciences and Engineering Research Council of Canada, ILOG, S.A., and the OTKA grant K 73376. A. Kovács acknowledges the support of the János Bolyai scholarship of the Hungarian Academy of Sciences.

## References

- [1] N. Beldiceanu, M. Carlsson, Sweep as a generic pruning technique applied to the non-overlapping rectangles constraint, in: Proceedings of the 7th International Conference on Principles and Practice of Constraint Programming (CP2001), 2001.
- [2] F. Clautiaux, A. Jouglet, J. Carlier, A. Moukrim, A new constraint programming approach for the orthogonal packing problem, *Computers & Operations Research* 35 (3) (2008) 944–959.
- [3] A. Davies, E. Bischoff, Weight distribution considerations in container loading, *European Journal of Operational Research* 114 (1999) 509–527.
- [4] R. Dechter, I. Meiri, J. Pearl, Temporal constraint networks, *Artificial Intelligence* 49 (1991) 61–95.



- [5] S. Demassey, G. Pesant, L.-M. Rousseau, A cost-regular based hybrid column generation approach, *Constraints* 11 (4) (2006) 315–333.
- [6] G. Fasano, A MIP approach for some practical packing problems: Balancing constraints and tetris-like items, *4OR: A Quarterly Journal of Operations Research* 2 (2) (2004) 161–174.
- [7] F. Focacci, A. Lodi, M. Milano, Embedding relaxations in global constraints for solving TSP and TSPTW, *Annals of Mathematics and Artificial Intelligence* 34 (4) (2002) 291–311.
- [8] F. Focacci, A. Lodi, M. Milano, Optimization-oriented global constraints, *Constraints* 7 (3–4) (2002) 351–365.
- [9] 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) (2002) 165–192.
- [10] A. Kovács, J. C. Beck, A global constraint for total weighted completion time, in: *Proceedings of CPAIOR'07, 4th Int. Conf. on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (LNCS 4510)*, 2007.
- [11] C. Le Pape, P. Couronné, D. Vergamini, V. Gosselin, Time-versus-capacity compromises in project scheduling, in: *Proceedings of the Thirteenth Workshop of the UK Planning Special Interest Group*, 1994.
- [12] J. Martin, F. Fages, From business rules to constraint programs in warehouse management systems, in: *Doctoral Programme of the 13th International Conference on Principles and Practice of Constraint Programming (CP 2007)*, 2007.
- [13] K. Mathur, An integer-programming-based heuristic for the balanced loading problem, *Operations Research Letters* 22 (1) (1998) 19–25.
- [14] J.-C. Régis, A filtering algorithm for constraints of difference in CSPs, in: *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, vol. 1, 1994.
- [15] J.-C. Régis, Arc consistency for global cardinality constraints with costs, in: *Proceedings of Principles and Practice of Constraint Programming (LNCS 1713)*, 1999.
- [16] J.-C. Régis, M. Rueher, Inequality-sum: a global constraint capturing the objective function, *RAIRO Operations Research* 39 (2005) 123–139.
- [17] Scheduler, ILOG Scheduler 6.1 Reference Manual, ILOG, S.A. (2002).
- [18] M. Sellmann, An arc consistency algorithm for the minimum weight all different constraint, in: *Proceedings of Principles and Practice of Constraint Programming (LNCS 2470)*, 2002.
- [19] W.-J. van Hoeve, I. Katriel, Global constraints, in: F. Rossi, P. van Beek, T. Walsh (eds.), *Handbook of Constraint Programming*, chap. 6, Elsevier, 2006, pp. 169–208.
- [20] J. Wodziak, G. Fadel, Packing and optimizing the center of gravity location using a genetic algorithm, unpublished manuscript (1994).