Extensions of the COMPLETION Constraint

András Kovács
Projet Contraintes, INRIA Rocquencourt, France
and
Computer and Automation Research Institute, Hungary
akovacs@sztaki.hu

J. Christopher Beck
Department of Mechanical & Industrial Engineering
University of Toronto
jcb@mie.utoronto.ca

Abstract

The COMPLETION global constraint has been proposed for single-machine, unary-resource, total weighted completion time scheduling problems where it has shown good performance. In this paper, we look at extending the constraint in two ways. First, we apply the constraint to multiple machine scheduling problems, in the form of job shop scheduling. It is shown that under the right allocation of weights to activities, the COMPLETION constraint results in significantly better scheduling performance compared to the standard expression of the weighted completion time. Second, we extend the constraint from the unary to discrete resources. Empirically this extension results in an orders of magnitude improvement in the number of nodes required to find a solution though a somewhat more mixed result on run-time.

Introduction

The COMPLETION constraint is a recently proposed global constraint to propagate the total weighted completion time of activities on a single unary capacity resource (Kovács & Beck 2007). It employs a lower bound based on a preemptive relaxation that is computable in polynomial time and a recomputation of the lower bound to prune values from the start time domains of the activities. Empirical results have shown strong performance compared to existing CP approaches on single machine problems with release times. On the other hand, state-of-the-art techniques developed for the specific single machine problem, including a branch-and-bound search with powerful dominance rules and a sophisticated dynamic programming approach, are still significantly better than the COMPLETION constraint with basic CP tree search. The strength of the COMPLETION constraint is its applicability to problems with various side constraints, which are not tractable by the above dedicated methods.

In this paper, we investigate two extensions of the COMPLETION constraint:

1. Applying COMPLETION to multiple machine scheduling;
2. Extending COMPLETION to discrete resource scheduling.

In the next section, we provide background on the COMPLETION constraint and a short discussion of related work. We then present sections on either of the extensions individually.

Previous Work

A critical component of the success of pure constraint programming (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 optimization function. Models that exhibit a high degree of back propagation will tend to be successful as new (sub-optimal) solutions will result in 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 any better performance than any other search technique.

The significance of cost-based global constraints for strong back propagation has been emphasized by Focacci et al. (Focacci, Lodi, & Milano 2002b). Cost-based constraints with effective propagation algorithms include the global cardinality constraint with costs (Régin 1999), the minimum weight all different constraint (Sellmann 2002), the path cost constraint for traveling salesman problems (Focacci, Lodi, & Milano 2002a), and the cost-regular constraint (Demassey, Pesant, & Rousseau 2006). The COMPLETION constraint is a cost constraint for total weighted completion time. This objective and other "sum-type" objectives are common in scheduling applications.

The single-machine, unary capacity total weighted completion time problem when activities have different release dates is NP-hard (Chen, Potts, & Woeginger 1998). Using the classical $\alpha|\beta|\gamma$ scheduling notation (Graham *et al.* 1979) with w_i being the weight of job A_i, r_i its release time, and C_i being its completion time in the schedule, this problem can be expressed as $1|r_i|\sum w_i C_i$. However, a preemptive version of the problem with a slightly modified objective function is computable in polynomial time and serves as a lower bound. Let M_i be the mean busy time of activity A_i. That is, M_i is 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 (Goemans *et al.* 2002).

We can, therefore, implement a global constraint that 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 relaxation is greater than the current best known solution, then t is removed from the domain of the start time variable of A_i.

In practice, we do not naively re-solve the relaxation for each start time in the domain of each activity. Instead, a much faster algorithm allows us to transform an initial relaxed solution into preemptive schedules with given start time assignments. For a detailed presentation of this algorithm and the COMPLETION constraint, in general, readers are referred to (Kovács & Beck 2007).

Formally, the COMPLETION constraint is defined as follows:

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

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. The total weighted completion time of the activities will be denoted by C. We assume that all data are integral. The constraint enforces $C = \sum_i w_i(S_i + p_i)$.

Application to Multiple Machine Scheduling

An $n \times m$ job shop scheduling problem (JSP) has n jobs each composed of m completely ordered activities. Each activity requires exclusive use of one resource during its execution. The duration and the resource for each activity are given and may be different from that of other activities: often, as in the problems studied here, a different resource is specified for each activity in a job. An activity cannot start until the activity immediately preceding it in the same job has finished. The standard JSP decision problem asks if, for a given makespan, D, all activities can finish by D. This is a well-known NP-complete problem (Garey & Johnson 1979). It is not uncommon to solve the optimization version of the JSP with the goal of minimizing makespan or some other metric such as sum of earliness and tardiness (Beck & Refalo 2003) or, in the present case, the sum of the weighted completion time of all jobs. More formally, given a set of jobs J and a weight, $w_j, j \in J$, our goal is to find a start time for each activity such that:

- no resource executes more than one activity at a time
- each activity starts after its preceding activity in the job-order ends
- $\sum_{j \in J} w_j C_{Ej}$ is minimized, where E_j is the last activity in job j.

We study square JSPs (i.e., $n = m$) where each job has exactly one activity on each resource.

From Job Weights to Activity Weights

Applying the COMPLETION constraint to the JSP is straightforward as the resources have unary capacity. Therefore, we can apply the constraint to each resource, individually. The only complication is that the constraint uses a weight on each activity and the JSP has weights on each job. Therefore, we need to define a mapping from job weight to activity weight.

The obvious approach, which we refer to as *last*, is to assign the job weight to the last activity in each job and to assign all other activities a weight of zero. We then place a COMPLETION constraint on each resource that has a non-zero weight activity and the total weighted completion time is the sum of the C values of each COMPLETION constraint.

The *last* approach has two main drawbacks. First, a computationally expensive COMPLETION constraint is placed on each resource. Second, the COMPLETION constraint makes inferences based on a relaxation that focuses on the interaction among activities on the same resource. Clearly, this interaction is not captured when the weighted activities are on different resources. In the extreme, the last activity in each job may be the only weighted activity on a resource. Under such circumstances, the COMPLETION constraint is not able to make any inferences stronger than the simple weighted sum constraint.

Therefore, we propose a different weight mapping, called *busy*. Before solving, we identify the most loaded resource, i.e., the "busy" resource, by summing the durations of the activities on each resource and selecting the resource with highest sum. The weight of each job is assigned to the last activity of the job that is processed on the busy resource. All other activities have a weight of zero. A single COMPLETION constraint can then be posted on the busy resource. To calculate the total weighted completion time, we need to correct for the fact that the weighted activity is not necessarily the last activity in the job.

Formally, as above, let E_j be the last activity in job j and let B_j be the single weighted activity in job j. Our optimization function is then: $C + \sum_{j \in J} w_j(C_{E_j} - C_{B_j})$ where C is the cost variable associated with the COMPLETION constraint.

Experimental Details

To test the effectiveness of the COMPLETION constraint, we compare it against the standard weighted sum, *WS*, form of the optimization function. For completeness, we also run WS with *last* and *busy* weight allocations.

We experiment with two styles of search: chronological backtracking and randomized restart. For chronological backtracking (i.e., depth-first search) we use a customized version of the SetTimes heuristic available in ILOG Scheduler 6.3. The heuristic selects all activities with minimum start time and breaks ties by choosing the activity with the highest ratio of weight to duration. The selected activity is scheduled at its earliest start time. Upon backtracking, the scheduled activity is postponed, meaning that it will not be considered for selection again until constraint propagation

has increased its minimum start time. When all minimum start time activities are postponed, the search backtracks further as no better schedule exists in the subtree.

For randomized restart, the limit on the number of backtracks before restarting evolves according to the universal limit developed by Luby et al. (Luby, Sinclair, & Zuckerman 1993). The heuristic is a randomized version of the customized SetTimes heuristic used above. Again, the set of activities with minimum start time are selected. One activity from this set is randomly chosen by a biased roulette wheel weighted by the ratio of activity weight to duration. Higher weight, lower duration activities have a higher probability of being selected.

Two sets of 10×10 JSP problems are used. Initially 10 makespan instances were generated with an existing generator (Watson *et al.* 1999). The machine routings were randomly generated and the durations were uniformly drawn from [1, 99]. These instances were transformed into two sets of total weighted completion time problems with the only difference being the range of job weights: the first set has job weights uniformly drawn from the interval [1, 9] and the second set has job weights uniformly drawn from the interval [1, 99].

The models and algorithms were implemented in ILOG Scheduler 6.3. Experiments were run on 2GHz Dual Core AMD Opteron 270 with 2Gb RAM running Red Hat Enterprise Linux 4. We used an overall time-out of 1200 CPU seconds for each run. The randomized restart results are the mean over 10 independent runs.

Results

For this experiment, the dependent variable is mean relative error (MRE) relative to the best solution known for the problem instance. The MRE is the arithmetic mean of the relative error over each run of each problem instance:

$$MRE(a, K, R) = \frac{1}{|R||K|} \sum_{r \in R} \sum_{k \in K} \frac{c(a, k, r) - c^*(k)}{c^*(k)} \quad (1)$$

where K is a set of problem instances, R is a set of independent runs with different random seeds, $c(a, k, r)$ is the lowest cost found by algorithm a on instance k in run r, and $c^*(k)$ is the lowest cost known for k. As these problem instances were generated for this experiment, the best-known solution was found either by the algorithms tested here or by variations used in preliminary experiments.

Figures 1 and 2 display the results for the two problem sets. There are two main comparisons of interest:

1. COMPLETION vs. WS back propagation, with the search held constant. With either search technique and on both problem sets, the COMPLETION constraint with *busy* weight allocation (COMP-BUSY) significantly outperforms the other three variations (WS-BUSY, WS-LAST, and COMP-LAST). The difference is larger with chronological backtracking than with randomized restart.
2. Randomized restart (RR) vs. chronological backtracking (Chron) with the propagation held constant. The chronological search significantly out-performs randomized restart. For the problems with narrower weight range, even chronological backtracking with the weaker propagation out-performs randomized restart with the stronger propagation.

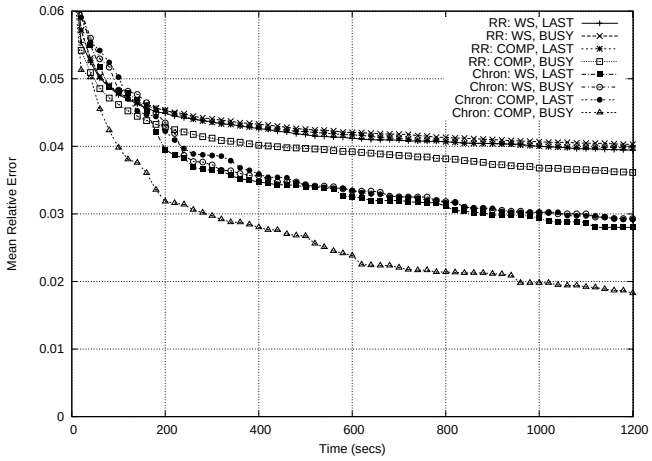

Figure 1: The mean relative error for different propagation techniques with different search for the problems with job weight uniformly drawn from [1, 9].

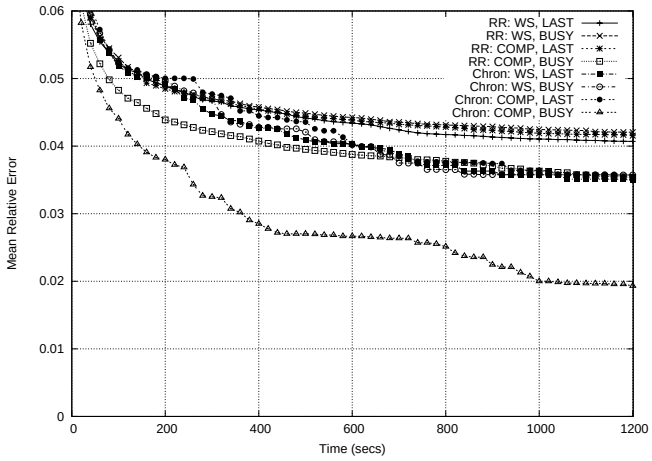

Figure 2: The mean relative error for different propagation techniques with different search for the problems with job weight uniformly drawn from [1, 99].

Generalizing to Discrete Resources

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

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

where finite-domain variables S_i stand for the start time of A_i, while p_i, ϱ_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 in the CSP. We assume that p_i, ϱ_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, ϱ_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 Relaxed Problem

Effective propagation requires embedding into the constraint a polynomially solvable relaxation of the single discrete resource total weighted completion time problem that considers capacity constraints, resource requirements, and release times at the same time. As such a relaxation does not appear to have been presented in the literature, below we propose a novel relaxation of this scheduling problem, see Fig.3.

Minimize

$$\sum_{i,t} t\ \mu_i\ x_t^i \tag{2}$$

s.t.

$$\forall i, t \qquad x_t^i \geq 0 \tag{3}$$

$$\forall i \qquad \sum_t x_t^i = p_i\varrho_i \tag{4}$$

$$\forall t \qquad \sum_i x_t^i \leq R \tag{5}$$

$$\forall i, t \qquad \sum_{t'=0}^{t} x_{t'}^i \leq \begin{cases} 0 & \text{if } t < r_i \\ (t - r_i + 1)\varrho_i & \text{if } r_i \leq t < r_i + p \\ p_i\varrho_i & \text{if } t \geq r_i + p \end{cases} \tag{6}$$

Figure 3: The variable-intensity relaxation of the discrete resource scheduling problem.

In the relaxed problem, we assume that activities A_i can be executed with a varying intensity over time. That is, in each time period $[t, t+1)$, $t = 0, ..., T-1$, an intensity x_t^i of A_i can be chosen from $[0, R]$. As an extremity of varying intensity, preemption is also allowed. The sum of the intensities over time has to match the original volume of the activity (4), and the capacity constraint must be respected (5). The release time constraint now states that A_i cannot be processed before r_i and its volume is released gradually afterwards (6). The objective is to minimize the total weighted mean busy time of the activities. In the sequel, we differentiate between the original problem and the variable-intensity relaxation by denoting the first as Π, and the second as Π'. C' will stand for the cost of the optimal relaxed solution.

Proposition 1 *$C' + \frac{1}{2}\sum_i w_i(p_i+1)$ is a valid lower bound on the original problem Π.*

Proof: The optimal solution of Π with cost C^* is a feasible solution of Π' as well, and it total weighted mean busy time is $C^* - \frac{1}{2}\sum_i w_i(p_i+1) \geq C'$. □

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 scheduling decision has to be made, the algorithm assigns intensities to activities in the order of non-increasing μ_i. The intensity of A_i will be the minimum of

- the volume of A_i that has already been released but not yet processed, $min(p_i\varrho_i, (t - r_i + 1)\varrho_i) - \sum_{t'=0}^{t-1} x_t^i$;
- the remaining capacity for the subsequent time period.

These intensity values are applied until the scheduled volume of an activity exceeds its released volume or the release time of another activity is reached. 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 $O(n^3)$, independently of the length of the scheduling horizon.

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

Proof: Let σ be an arbitrary feasible schedule that differs from schedule σ^* 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 μ_{i1} such that $x_{t1}^{i1}[\sigma] \neq x_{t1}^{i1}[\sigma^*]$. The construction of the algorithm ensures that $x_{t1}^{i1}[\sigma] < x_{t1}^{i1}[\sigma^*]$. Then, there exist a time t_2 and activity A_{i2} with $t_1 < t_2$ and $\mu_{i1} > \mu_{i2}$ such that increasing $x_{t1}^{i1}[\sigma]$ and $x_{t2}^{i2}[\sigma]$ and decreasing $x_{t1}^{i2}[\sigma]$ and $x_{t2}^{i1}[\sigma]$ preserves feasibility and improves the objective value. Therefore a schedule that differs essentially from the one built by the algorithm cannot be optimal. □

Observe that the above procedure can easily be modified to `PrepareRelaxed(`A_i`, t)`, which computes optimal relaxed solutions for restricted problems where the start time of activity A_i is bound to 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 and ensures that it starts at t and processed at intensity ϱ_i throughout its duration.

From Relaxed Solutions to Bounds Tightening

Below we propose algorithms that tighten the bounds of the start time variable domains by exploiting the above presented polynomially solvable relaxation. Similarly to the unary resource COMPLETION constraint, propagation is

based on computing (or approximating) the cost of the optimal relaxed solutions for restricted problems where an activity A_i must start at time t. This restricted relaxed problem will be denoted by $\Pi'\langle S_i, t\rangle$, and the value of its optimal solution by $C'\langle S_i, t\rangle$. Formally, we exploit 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.*

However, in contrast to the unary case, we are not able to define efficient recomputation methods that determine in a low-degree polynomial time *all* restricted relaxed solution costs from *one* relaxed solution computed by `PrepareRelaxed`. Instead, we apply the `PrepareRelaxed` procedure to compute two relaxed solutions for each activity, one for $\Pi'\langle S_i = \check{S}_i\rangle$ and another for $\Pi'\langle S_i = \hat{S}_i\rangle$. If either of these relaxed solutions violate the current upper bound on the cost, then we estimate how the current lower/upper bounds of S_i have to be modified to achieve consistency. Since estimations are not exact, this procedure has to be iterated until bounds consistency is reached. The algorithm is presented in Figure 4, while the two different earliest and latest start time approximation methods are presented in detail afterwards.

```
PROCEDURE TightenBounds()
  FORALL activity A_i
    LOOP
      σ := PrepareRelaxed(A_i, Š_i)
      IF cost(σ) > Ĉ THEN
        Š_i' := RecomputeEarliestStart(σ, A_i)
    LOOP
      σ := PrepareRelaxed(A_i, Ŝ_i)
      IF cost(σ) > Ĉ THEN
        Ŝ_i' := RecomputeLatestStart(σ, A_i)
```

Figure 4: Algorithm for tightening the bounds of the start time variables.

Recomputing the Earliest Start Time The earliest start time of activity A_i is adjusted based on the optimal relaxed solution σ for $\Pi'\langle S_i = \check{S}_i\rangle$. In order to obtain lower bounds for $\Pi'\langle S_i = t\rangle$ with $t > \check{S}_i$, we introduce procedure `RecomputeEarliestStart(`σ`,` A_i`)` and a further relaxation as follows.

For the situation where A_i starts at t, we consider a schedule σ_t in which intensities $x^j_{t'}$ with $t' < \check{S}_i$ or $t' \geq t + p_i$ equal the corresponding intensities in σ. Activity A_i is processed from t until $t + p_i$ at intensity ϱ_i. Otherwise, in interval $[\check{S}_i, t + p_i]$ we assume that the release times of all activities A_j with $j \neq i$ equal $\check{S}_i$. Hence, those activities will be processed in non-increasing order of μ_j.

This further relaxation has two advantages: firstly, only a small section of the schedule, namely the section in interval $[\check{S}_i, t + p_i]$ varies over different values of t. Secondly, this section of the schedule 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 σ_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.

Recomputing the Latest Start Time Given an optimal relaxed solution σ 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^*:

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

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

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

Procedure `RecomputeLatestStart(`σ`,` 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 $t = \hat{S}_i - \lceil \frac{C' - \hat{C}}{W} \rceil$.*

Proof: Consider a further relaxation of the relaxed problem where, in interval $[t, t^*]$, resource capacity is increased from R to $2R$, and the released volume of each activity A_j is increased with $\sum_{t'=\hat{S}_i}^{t^*} x^j_{t'}$. The optimal solution of this further relaxed problem is a schedule with all intensities in the interval $[\hat{S}_i, t^*]$ in σ moved earlier by $(\hat{S}_i - t)$. The cost of this schedule is exactly $C' - (\hat{S}_i - t)W$, from which the above proposition follows. □

Computational Experiments

We ran computational experiments to test the efficiency of the proposed COMPLETION$_m$ constraint on a set of single discrete resource scheduling problems with release times for minimizing total weighted completion time. We compared the performance of two models, one of them using weighted sum (WS) back propagation, the other using the COMPLETION$_m$ constraint. The proposed propagation algorithms have been implemented in C++ and embedded into ILOG Solver and Scheduler versions 6.1. In the experiments, we applied the same adapted *SetTimes* branching heuristic with chronological backtracking as for the job shop case.

The problem instances were generated with a modified version of a previous benchmark generator for the single unary machine total weighted completion time problem (Pan 2007). The parameters of the generator are the number of activities, n, which we took from $\{15, 20, 25, 30\}$, the resource requirement range $\alpha \in \{0.5, 1.0\}$, and the relative

release time range $\beta \in \{0, 0.2, 0.6, 1.0\}$. For each combination of the above values we generated 10 instances, which resulted in 320 different problem instances.

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 ϱ_i from $[1, \alpha R]$. This leads to instances where approximately $k = \frac{2R}{\alpha R+1} - \frac{1}{2}$ 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 Pentium M computer with 1 GB of RAM, with a time limit of 600 seconds imposed.

The experimental results are presented in Table 1. Each row of the table contains combined results for given values of n and β, achieved with the WS and the COMPLETION$_m$ models. The results do not depend significantly on the value of α. For either of the models WS and COMPLETION$_m$, column *Opt* displays the number of instances out of 20 that could be solved and the optimality of the solution has been proven. Columns *Mean RE* and *Max RE* contain the mean and maximum relative error compared to the best solutions known. Column *Nodes* shows the average number of search nodes, while *Time* presents the average search time, including the proof of optimality, or 600 seconds where the solver hit the time limit.

Problem instances with a low number of activities, $n = 15$, or with a high β were solvable easily for both models, while instances with a greater n and a lower β often proved hard for both models. The COMPLETION$_m$ constraint reduced the number of search nodes for all the instances, often by two orders of magnitude. On the other hand, the computational effort invested in this pruning paid off only for the hard instances, i.e., for $n \geq 20$ and $\beta \leq 60$. For such instances, the COMPLETION$_m$ model found better solutions and proved optimality for more instances. Easier problems were solved more quickly with the simple weighted sum constraint.

From the low number of search nodes with the COMPLETION$_m$ constraint we conclude that the applied variable-intensity relaxation is sufficiently tight. On the other hand, the reason of the high computational cost was the high number of recomputation cycles within the `TightenBounds()` procedure: on average, the recomputation of the earliest (latest) start times required 2–4 (4–10) cycles, while in a few extreme cases, up to 100 cycles were necessary to achieve consistency. This suggests that more accurate earliest and latest start time recomputation techniques are required. Further experiments are necessary to investigate whether a better trade-off between pruning strength and computational effort can be achieved by aborting recomputations before consistency is achieved. Also, the performance of the model with the COMPLETION$_m$ constraint has to be compared to a model in which the same relaxation is exploited in the form of a lower bound.

Conclusions

We investigated applications and extensions of the earlier defined COMPLETION constraint. In multiple-machine project scheduling problems, where activities linked by precedence constraints constitute jobs, weights and performance measures are often related to jobs. Since CP solution techniques infer over individual activities, the assignment of job weights to activities is a crucial issue. We defined a weight assignment heuristic, which allocates weights to the activities on the most loaded resource. For the criterion of total weighted completion time in job shop problems, we showed in computational experiments that the COMPLETION constraint with this weight assignment outperforms standard representations of the cost function.

We introduced the COMPLETION$_m$ constraint for the total weighted completion time of activities on a discrete resource. The proposed propagation algorithms exploit a variable-intensity relaxation of the discrete-resource scheduling problem. The new constraint achieved significant pruning in single discrete resource scheduling problems, though, due to its high computational complexity, this did not always result in a reduction in overall solution time. Our future work will focus on the improvement of earliest/latest start time recomputation methods for the COMPLETION$_m$ constraint, and investigating the possibility of developing a generic framework for cost constraints in scheduling.

Acknowledgments This research was supported in part by the Natural Sciences and Engineering Research Council, ILOG, S.A, the EU FP6 Net-WMS, and the NKFP 2/010/2004 projects. A. Kovács acknowledges the support of the ERCIM Alain Bensoussan fellowship programme and the János Bolyai scholarship of the Hungarian Academy of Sciences.

References

Beck, J. C., and Refalo, P. 2003. A hybrid approach to scheduling with earliness and tardiness costs. *Annals of Operations Research* 118:49–71.

Chen, B.; Potts, C. N.; and Woeginger, G. J. 1998. *A Review of Machine Scheduling: Complexity, Algorithms and Approximation*, volume 3 of *Handbook of Combinatorial Optimization*. Kluwer.

Demassey, S.; Pesant, G.; and Rousseau, L.-M. 2006. A cost-regular based hybrid column generation approach. *Constraints* 11(4):315–333.

Focacci, F.; Lodi, A.; and Milano, M. 2002a. Embedding relaxations in global constraints for solving TSP and TSPTW. *Annals of Mathematics and Artificial Intelligence* 34(4):291–311.

Focacci, F.; Lodi, A.; and Milano, M. 2002b. Optimization-oriented global constraints. *Constraints* 7(3–4):351–365.

Garey, M. R., and Johnson, D. S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York.

Goemans, M. X.; Queyranne, M.; Schulz, A. S.; Skutella, M.; and Wang., Y. 2002. Single machine scheduling

n	β	WS					COMPLETION$_m$				
		Opt	Mean RE	Max RE	Nodes	Time	Opt	Mean RE	Max RE	Nodes	Time
15	0.0	20	0.00	0.00	1221819	38.2	20	0.00	0.00	37394	50.0
	0.2	20	0.00	0.00	80897	2.2	20	0.00	0.00	3073	4.6
	0.6	20	0.00	0.00	5175	0.1	20	0.00	0.00	359	0.2
	1.0	20	0.00	0.00	5714	0.2	20	0.00	0.00	179	0.1
20	0.0	0	0.20	1.35	17184616	600.0	9	0.00	0.00	158712	474.1
	0.2	12	0.89	13.84	9344486	324.1	19	0.00	0.00	40150	108.4
	0.6	20	0.00	0.00	813644	35.0	20	0.00	0.00	5866	19.0
	1.0	20	0.00	0.00	3225	0.0	20	0.00	0.00	485	1.2
25	0.0	0	1.48	5.66	15877777	600.0	0	0.00	0.00	198843	600.0
	0.2	0	1.36	5.96	16227105	600.0	3	0.03	0.62	210544	528.2
	0.6	14	0.69	10.48	5576802	221.7	15	0.00	0.00	42145	217.4
	1.0	20	0.00	0.00	410177	14.5	20	0.00	0.00	3541	13.1
30	0.0	0	2.50	8.02	16402754	600.0	0	0.00	0.00	257192	600.0
	0.2	0	1.60	7.67	15847633	600.0	0	0.00	0.00	230434	600.0
	0.6	5	0.67	4.88	12120552	513.5	9	0.05	1.04	77783	413.5
	1.0	18	0.00	0.00	2315715	117.7	17	0.13	2.64	26355	146.4

Table 1: Experimental results: number of instances solved to optimality (Opt), mean and maximum relative error in percent (Mean/Max RE), average number of search nodes (Nodes) and average search time in seconds (Time) with the weighted sum (WS) constraint and with the COMPLETION$_m$ constraint.

with release dates. *SIAM Journal on Discrete Mathematics* 15(2):165–192.

Graham, R. L.; Lawler, E. L.; Lenstra, J. K.; and Rinnooy Kan, A. H. G. 1979. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics* 4:287–326.

Kovács, A., and Beck, J. C. 2007. 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)*, 112–126.

Luby, M.; Sinclair, A.; and Zuckerman, D. 1993. Optimal speedup of Las Vegas algorithms. *Information Processing Letters* 47:173–180.

Pan, Y. 2007. Test instances for the dynamic single-machine sequencing problem to minimize total weighted completion time. Available at www.cs.wisc.edu/˜yunpeng/test/sm/dwct/instances.htm.

Régin, J. 1999. Arc consistency for global cardinality constraints with costs. In *Proceedings of Principles and Practice of Constraint Programming (LNCS 1713)*, 390–404.

Sellmann, M. 2002. An arc consistency algorithm for the minimum weight all different constraint. In *Proceedings of Principles and Practice of Constraint Programming (LNCS 2470)*, 744–749.

Watson, J.; Barbulescu, L.; Howe, A.; and Whitley, L. 1999. Algorithms performance and problem structure for flow-shop scheduling. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, 688–695.