A Resource Cost Aware Cumulative

Helmut Simonis and Tarik Hadzic*

Cork Constraint Computation Centre
Department of Computer Science, University College Cork, Ireland
{h.simonis,t.hadzic}@4c.ucc.ie

Abstract. We motivate and introduce an extension of the well-known cumulative constraint which deals with time and volume dependent cost of resources. Our research is primarily interested in scheduling problems under time and volume variable electricity costs, but the constraint equally applies to manpower scheduling when hourly rates differ over time and/or extra personnel incur higher hourly rates. We present a number of possible lower bounds on the cost, including a min-cost flow, different LP and MIP models, as well as greedy algorithms, and provide a theoretical and experimental comparison of the different methods.

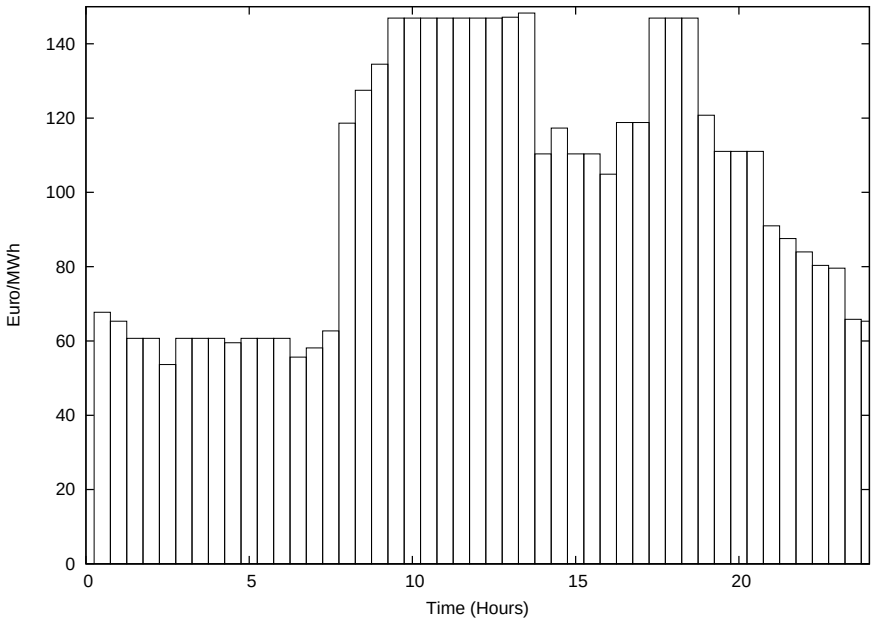
1 Introduction

The cumulative constraint [1, 2] has long been a key global constraint allowing the effective modeling and resolution of complex scheduling problems with constraint programming. However, it is not adequate to handle problems where resource costs change with time and use, and thus must be considered as part of the scheduling. This problem increasingly arises with electricity cost, where time variable tariffs become more and more widespread. With the prices for electricity rising globally, the contribution of energy costs to total manufacturing cost is steadily increasing, thus making an effective solution of this problem more and more important. Figure 1 gives an example of the whole-sale price in the Irish electricity market for a sample day, in half hour intervals. Note that the range of prices varies by more than a factor of three, and the largest difference between two consecutive half-hour time periods is more than 50 units. Hence, large cost savings might be possible if the schedule of electricity-consuming activities takes that cost into account.

The problem of time and volume dependent resource cost equally arises in manpower scheduling, where hourly rates can depend on time, and extra personnel typically incurs higher costs. We suggest extending the standard CP scheduling framework to variable resource cost scenarios by developing a cost-aware extension of the cumulative constraint, `CumulativeCost`. The new constraint should support both the standard feasibility reasoning as well as reasoning about the cost of the schedule. As a first step in this direction we formally describe the semantics of `CumulativeCost` and discuss a number of algorithms for producing *high quality lower-bounds* (used for bounding or pruning during optimization) for this constraint. The proposed algorithms are compared both theoretically and experimentally. Previous research on the cumulative constraint has focused along three lines: 1) improvement of the reasoning methods

* This work was supported by Science Foundation Ireland (Grant Number 05/IN/I886).

Fig. 1. Irish Electricity Price (Source: `http://allislandmarket.com/`)



used inside the constraint propagation, see [10, 7, 8] for some recent results. 2) applying the constraint to new, sometimes surprising problem types, for example expressing producer/consumer constraints [9] or placement problems [4], and 3) extending the basic model to cover new features, like negative resource height, or non-rectangular task shapes [3, 5]. The focus of this paper belongs to the last category.

2 The `CumulativeCost` Constraint

We start by formally defining our new constraint. It is an extension of the classical cumulative constraint [1]

$$\texttt{Cumulative}([s_1, s_2, ...s_n], [d_1, d_2, ...d_n], [r_1, r_2, ...r_n], l, p),$$

describing n tasks with start s_i, fixed duration d_i and resource use r_i, with an overall resource limit l and a scheduling period end p. Our new constraint is denoted as

$$\texttt{CumulativeCost}(\texttt{Areas}, \texttt{Tasks}, l, p, \texttt{cost}).$$

The *Areas* argument is a collection of m areas $\{A_1, \ldots, A_m\}$, which do not overlap and partition the entire available resource area $[0, p] \times [0, l]$. Each area A_j has a fixed position x_j, y_j, fixed width w_j and height h_j, and fixed per-unit cost c_j. Consider the running

example in Fig. 2 (left). It has 5 areas, each of width 1 and height 3. Our definition allows that an area A_j could start above the bottom level ($y_j > 0$). This reflects the fact that the unit-cost does not only depend on the time of resource consumption but also on its volume. In our electricity example, in some environments, a limited amount of renewable energy may be available at low marginal cost, generated by wind-power or reclaimed process heat. Depending on the tariff, the electricity price may also be linked to the current consumption, enforcing higher values if an agreed limit is exceeded. We choose the numbering of the areas so that they are ordered by non-decreasing cost ($i \leq j \Rightarrow c_i \leq c_j$); in our example costs are $0, 1, 2, 3, 4$. There could be more than one area defined over the same time slot t (possibly spanning over other time-slots as well). If that is the case, we require that the area "above" has a higher cost. The electricity consumed over a certain volume threshold might cost more. The *Tasks* argument is a collection of n tasks. Each task T_i is described by its start s_i, and fixed duration d_i and resource use r_i. In our example, we have three tasks with durations $1, 2, 1$ and resource use $2, 2, 3$. The initial start times are $s_1 \in [2, 5]$, $s_2 \in [1, 5]$, $s_3 \in [0, 5]$. For a given task allocation, variable a_j states how many resource units of area A_j are used. For the optimal solution in our example we have $a_1 = 2, a_2 = 3, a_3 = 2, a_4 = 0, a_5 = 2$. Finally, we can define our constraint:

Definition 1. *Constraint* `CumulativeCost` *expresses the following relationships:*

$$\forall\, 0 \leq t < p: \quad pr_t := \sum_{\{i \mid s_i \leq t < s_i + d_i\}} r_i \leq l \tag{1}$$

$$\forall\, 1 \leq i \leq n: \quad 0 \leq s_i < s_i + d_i \leq p \tag{2}$$

$$\mathtt{ov}(t, pr_t, A_j) := \begin{cases} \max(0, \min(y_j + h_j, pr_t) - y_j) & x_j \leq t < x_j + w_j \\ 0 & otherwise \end{cases} \tag{3}$$

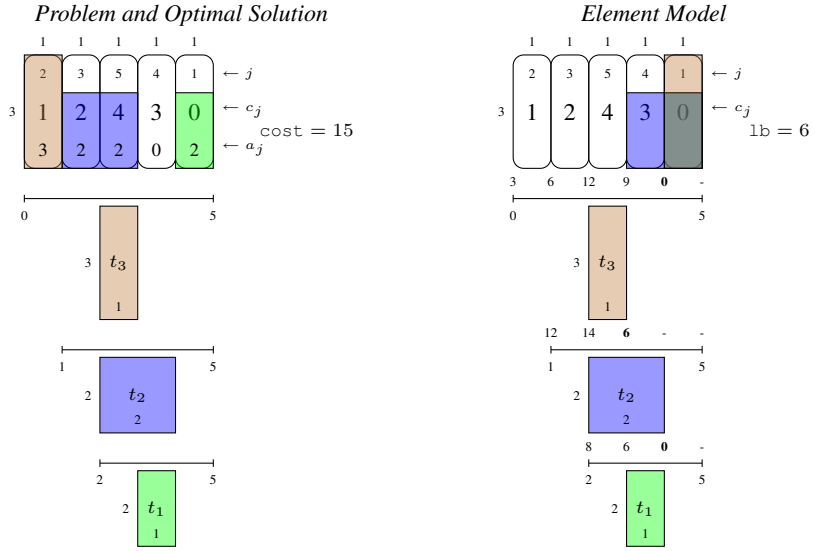
$$\forall\, 1 \leq j \leq m: \quad a_j = \sum_{0 \leq t < p} \mathtt{ov}(t, pr_t, A_j) \tag{4}$$

$$\mathtt{cost} = \sum_{j=1}^{m} a_j c_j \tag{5}$$

For each time point t we first define the resource profile pr_t (the amount of resource consumed at time t). That profile must be below the overall resource limit l, as in the standard cumulative. The term $\mathtt{ov}(t, pr_t, A_j)$ denotes the intersection of the profile at time t with area A_j, and the sum of all such intersections is the total resource usage a_j. The cost is computed by weighting each intersection a_j with the per-unit cost c_j of the area.

Note that our constraint is a strict generalization of the standard cumulative. Since enforcing generalized arc consistency (GAC) for `Cumulative` is NP-hard [6], the complexity of enforcing GAC over `CumulativeCost` is NP-hard as well. In the remainder of the paper we study the ability of `CumulativeCost` to reason about the cost through computation of lower bounds.

Fig. 2. An example with 3 tasks and 5 areas. Areas are drawn as rounded rectangles at the top, the tasks to be placed below, each with a line indicating its earliest start and latest end. The optimal placement of tasks has cost 15 (left). The `Element` model produces a lower bound 6 (right).



3 Decomposition With `Cumulative`

We first present a number of models where we decompose the `CumulativeCost` into a standard `Cumulative` and some other constraint(s) which allow the computation of a lower bound for the `cost` variable. This cost component exchanges information with the `Cumulative` constraint only through domain restrictions on the s_i variables.

3.1 Element Model

The first approach is to decompose the constraint into a `Cumulative` constraint and a set of `Element` constraints. For a variable $x \in \{1, \ldots, n\}$ and a variable $y \in \{v_1, \ldots, v_n\}$, the element constraint $\texttt{element}(x, [v_1, v_2, ..., v_n], y)$ denotes that $y = v_x$. For each task i and start time t we precompute a cost of having a task T_i starting at time t in the cheapest area overlapping the time slot t (the bottom area). This value, denoted as v_{it}, is only a lower bound, and is incapable of expressing the cost if the task starts at t but in a higher (more expensive) area. A lower bound can then be expressed as

$$\texttt{lb} = \min \sum_{i=1}^{n} u_i$$

$$\forall\, 1 \leq i \leq n: \quad \texttt{element}(s_i, [v_{i1}, v_{i2}, ..., v_{ip}], u_i)$$

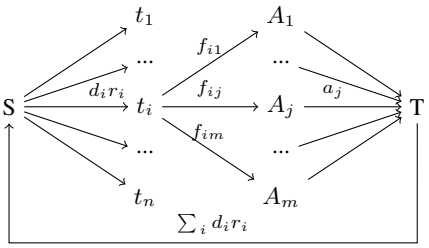
This lower bound can significantly underestimate the optimal cost since each task is assumed to be placed at its "cheapest" start time, even if the overall available volume

would be exceeded. In our example, the v_{it} values for each start time are displayed above the time-lines for each task in Figure 2 (right). The lowest cost values for the tasks are 0, 6 and 0, resp., leading to a total lower bound estimate of 6.

3.2 Flow Model

We can also describe our problem as a min-cost flow model, where the flow cost provides a lower bound to the cost variable of our constraint. We need to move the flow $\sum_i d_i r_i$ from tasks to areas. Figure 3 shows the flow graph used, where the tasks T_i are linked to the areas A_j through flow variables f_{ij} which indicate how much volume of task i is contained in area j. The sum of all flows a_j through all areas must be equal to the total task volume. Only the links from A_j to the terminal T have non-zero cost c_j. The lower bound estimate is a min-cost flow.

Fig. 3. Flow Graph



The model can also be expressed by a set of equations and inequalities.

$$\texttt{lb} = \min \sum_{j=1}^{m} a_j c_j \tag{6}$$

$$\forall\, 1 \le j \le m: \quad a_j = \sum_{i=1}^{n} f_{ij} \tag{7}$$

$$\forall\, 1 \le i \le n, \forall\, 1 \le j \le m: \quad \underline{f_{ij}} \le f_{ij} \le \overline{f_{ij}} \tag{8}$$

$$\forall\, 1 \le j \le m: \quad 0 \le \underline{a_j} \le a_j \le \overline{a_j} \le w_j h_j \tag{9}$$

$$\forall\, 1 \le i \le n: \quad \sum_{j=1}^{m} f_{ij} = d_i r_i \tag{10}$$

$$\forall\, 1 \le i \le n: \quad \sum_{i=1}^{n} d_i r_i = \sum_{j=1}^{m} a_j \tag{11}$$

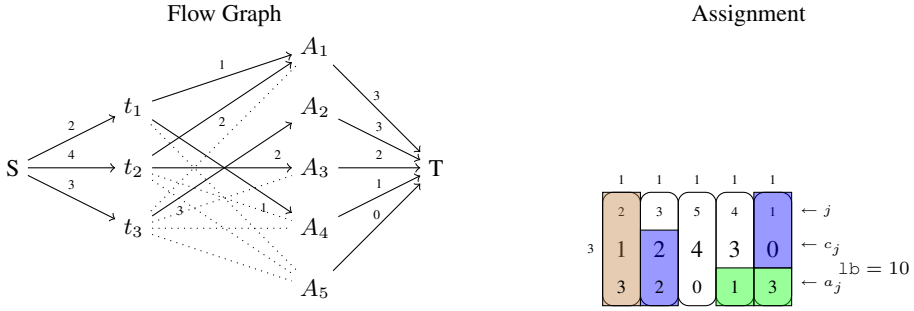
While the flow model avoids placing too much work in the cheapest areas, it does allow splitting tasks over several, even non-contiguous areas to reduce overall cost, i.e. it ignores resource use and non-preemptiveness of tasks (Fig. 4, right).

The quality of the bound lb can be improved by computing tight upper bounds $\overline{f_{ij}}$. Given a task i with domain $d(s_i)$, we can compute the maximal overlap between the task and the area j as:

$$\overline{f_{ij}} = \max_{t \in d(s_i)} \max(0, (\min(x_j + w_j, t + d_i) - \max(x_j, t))) * \min(h_j, r_i) \qquad (12)$$

For the running example, the computed $\overline{f_{ij}}$ values are shown in Table 1 and the minimal cost flow is shown in Fig. 4. Its cost is $3*0+3*1+2*2+1*3 = 10$. Note how both tasks 1 and 2 are split into multiple pieces.

Fig. 4. Example Flow Bound



3.3 LP Models

The quality of the lower bound can be further improved by adding to the flow model some linear constraints, which limit how much of some task can be placed into low-cost areas. These models are no longer flows, but general LP models. We consider two variations denoted as *LP1* and *LP2*. We obtain *LP1* by adding equations (13) to the LP formulation of the flow model (Constraints 6-11).

$$\forall\, 1 \leq j \leq m: \quad \sum_{i=1}^{n} \sum_{k=1}^{j} f_{ik} = \sum_{k=1}^{j} a_k \leq \overline{B_j} = \sum_{i=1}^{n} \overline{b_{ij}} \qquad (13)$$

The $\overline{b_{ij}}$ values tell us how much of task i can be placed into the combination of cheaper areas $\{A_1, A_2, ...A_j\}$. Note that this is a stronger estimate, since $\overline{b_{ij}} \leq \sum_{k=1}^{j} \overline{f_{ij}}$. Therefore, the LP models dominate the flow model, but require a more expensive LP optimization at each step. $\overline{B_j}$ denotes the total amount of resources (from all tasks) that is possible to place into the first j areas. We can compute the $\overline{b_{ij}}$ values with a sweep along the time axis.

Table 1. $\overline{f_{ij}}$, $\overline{b_{ij}}$ and $\overline{B_j}$ Values for Example

$\overline{f_{ij}}$	1	2	3	4	5
1	2	0	0	2	2
2	2	0	2	2	2
3	3	3	3	3	3

$\overline{b_{ij}}$	1	2	3	4	5
1	2	2	2	2	2
2	2	2	2	4	4
3	3	3	3	3	3
$\overline{B_j}$	7	7	7	9	9

Table 1 shows the $\overline{f_{ij}}$, $\overline{b_{ij}}$ and $\overline{B_j}$ values for the running example. The flow solution in Fig. 4 does not satisfy equation (13) for $j = 3$, as $a_1 + a_2 + a_3 = 3 + 3 + 2 = 8 > b_{13} + b_{23} + b_{33} = 2 + 2 + 3 = 7$. Solving the LP model finds an improved bound of 11.

The *LP2 model* produces potentially even stronger bounds by replacing constraints (13) with more detailed constraints on individual $\overline{b_{ij}}$:

$$\forall\, 1 \le i \le n, \forall\, 1 \le j \le m: \quad \sum_{k=1}^{j} f_{ik} \le \overline{b_{ij}} \tag{14}$$

In our example though, this model produces the same bound as model *LP1*.

4 Coarse Models and Greedy Algorithms

If we want to avoid running an LP solver inside our constraint propagator, we can derive other lower bounds based on greedy algorithms.

4.1 Model A

We can compute a lower bound through a coarser model (denoted as *Model A*) which has m variables u_j, denoting the total amount of work going into A_j.

$$\texttt{lb} = \min \sum_{j=1}^{m} u_j c_j \tag{15}$$

$$\forall\, 1 \le j \le m: \quad u_j \le w_j h_j \tag{16}$$

$$\forall\, 1 \le j \le m: \quad u_j \le \sum_{i=1}^{n} \overline{f_{ij}} \tag{17}$$

$$\sum_{j=1}^{m} u_j = \sum_{i=1}^{n} d_i r_i \tag{18}$$

It can be shown that this bound can be achieved by Algorithm A, which can be described by the recursive equations:

$$\texttt{lb} = \sum_{j=1}^{m} u_j c_j \tag{19}$$

$$\forall\, 1 \leq j \leq m: \quad u_j = \min(\sum_{i=1}^{n} d_i r_i - \sum_{k=1}^{j-1} u_k, \sum_{i=1}^{n} \overline{f_{ij}}, w_j h_j) \tag{20}$$

The algorithm tries to fill the cheapest areas first as far as possible. For each area we compute the minimum of the remaining unallocated task area, and the maximum amount that can be placed into the area based on the $\overline{f_{ij}}$ bounds and the overall area size.

4.2 Model B

We can form a stronger model (*Model B*) by extending Model A with the constraints:

$$\forall\, 1 \leq j \leq m: \quad \sum_{k=1}^{j} u_k \leq \overline{B_j} = \sum_{i=1}^{n} \overline{b_{ij}} \tag{21}$$

It can be shown that the lower-bound can be computed through Algorithm B, which extends Algorithm A by also considering the $\overline{B_j}$ bounds (constraints (13)) and is recursively defined as:

$$\texttt{lb} = \sum_{j=1}^{m} u_j c_j \tag{22}$$

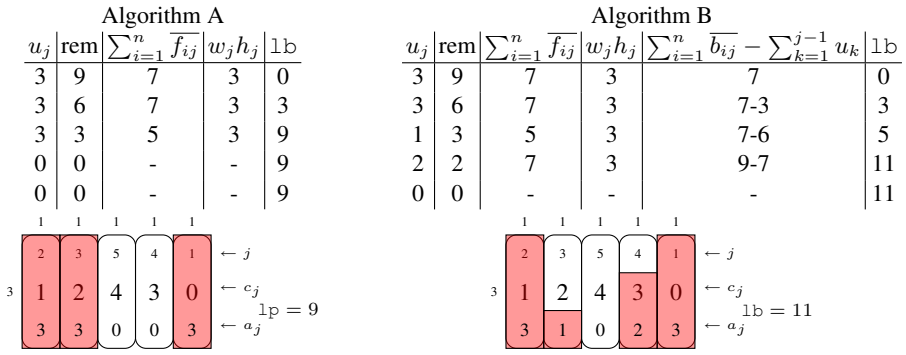
$$\forall\, 1 \leq j \leq m: \quad u_j = \min(\sum_{i=1}^{n} d_i r_i - \sum_{k=1}^{j-1} u_k, \sum_{i=1}^{n} \overline{f_{ij}}, w_j h_j, \sum_{i=1}^{n} \overline{b_{ij}} - \sum_{k=1}^{j-1} u_k) \tag{23}$$

Both algorithms A and B only compute a bound on the cost, and do not produce a complete assignment of tasks to areas. On the other hand, they only require a single iteration over the areas, provided the $\overline{f_{ij}}$ and $\overline{b_{ij}}$ bounds are known. Figure 5 compares the execution of algorithms A and B on the running example (top) and shows the resulting area utilization (bottom). Algorithm A computes a bound of 9, which is weaker than the Flow Model ($\texttt{lb} = 10$). Algorithm B produces 11, the same as models LP1 and LP2. Note that we can not easily determine which tasks are used to fill which area.

5 Direct Model

All previous models relied on a decomposition using a standard cumulative constraint to enforce overall resource limits. We now extend the model of the cumulative constraint given in [6] to handle the cost component directly (equations (24)-(33)).

Fig. 5. Execution of algorithms A and B on the running example.



We introduce binary variables y_{it} which state whether task i starts at time t. For each task, exactly one of these variables will be one (constraint (30)). Equations (29) connect the s_i and y_{it} variables. Continuous variables pr_t describe the resource profile at each time point t, all values must be below the resource limit l. The profile is used in two ways: In (31), the profile is built by cumulating all active tasks at each time-point. In (32), the profile overlaps all areas active at a time-point, where the contribution of area j at time-point t is called z_{jt} (a continuous variable ranging between zero and h_j). Adding all contributions of an area leads to the resource use a_j for area j. This model combines the start-time based model of the cumulative with a standard LP formulation of the convex, piece-wise linear cost of the resource profile at each time point. Note that this model relies on the objective function to fill up cheaper areas to capacity before using more expensive ones. Enforcing the integrality in (26) leads to a mixed integer programming model *DMIP*, relaxing the integrality constraint leads to the LP model *DLP*. The MIP model solves the cumulative-cost constraint to optimality, thus providing an exact bound for the constraint. We can ignore the actual solution if we want to use the constraint in a larger constraint problem.

$$\mathtt{lb} = \min \sum_{j=1}^{m} a_j c_j \tag{24}$$

$$\forall\, 0 \le t < p: \quad pr_t \in [0, l] \tag{25}$$

$$\forall\, 1 \le i \le n, 0 \le t < p: \quad y_{it} \in \{0, 1\} \tag{26}$$

$$\forall\, 1 \le j \le m, \forall\, x_j \le t < x_j + w_j: \quad z_{jt} \in [0, h_j] \tag{27}$$

$$\forall\, 1 \le j \le m: \quad 0 \le \underline{a_j} \le a_j \le \overline{a_j} \le w_j h_j \tag{28}$$

$$\forall\, 1 \le i \le n: \quad s_i = \sum_{t=0}^{p-1} t y_{it} \tag{29}$$

$$\forall\, 1 \le i \le n: \quad \sum_{t=0}^{p-1} y_{it} = 1 \tag{30}$$

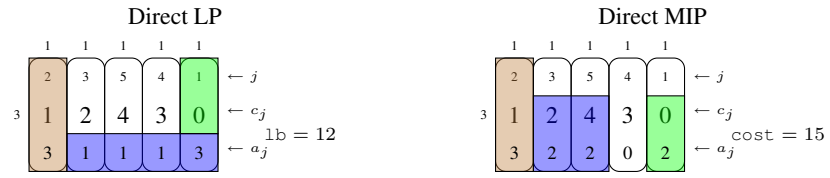
$$\forall\, 0 \le t < p: \quad pr_t = \sum_{t' \le t < t'+d_i} y_{it'} r_i \tag{31}$$

$$\forall\, 0 \le t < p: \quad pr_t = \sum_{j=1}^{m} z_{jt} \tag{32}$$

$$\forall\, 1 \le j \le m: \quad a_j = \sum_{t=x_j}^{x_j+w_j-1} z_{jt} \tag{33}$$

Example Figure 6 shows the solutions of *DLP* and *DMIP* on the running example. The linear model on the left uses fractional assignments for task 2, splitting it into two segments. This leads to a bound of 12, while the *DMIP* model computes a bound of 15.

Fig. 6. Direct Model Example

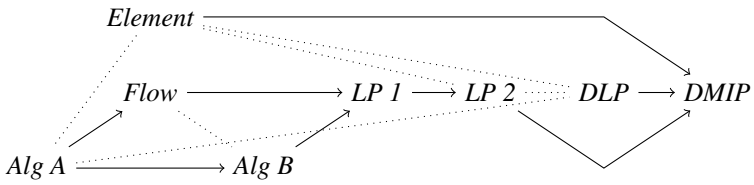


6 Comparison

We now want to compare the relative strength of the different algorithms. We first define the concept of strength in our context.

Definition 2. *We call algorithm q `stronger than` algorithm p if its computed lower bound $\texttt{lb}_q$ is always greater than or equal to $\texttt{lb}_p$, and for some instances is strictly greater than.*

Theorem 1. *The relative strength of the algorithms can be shown as a diagram, where a line $p \longrightarrow q$ indicates that algorithm q is stronger than algorithm p, and a dotted line between two algorithms states that they are incomparable.*



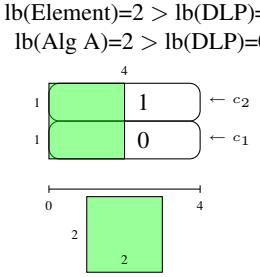
Proof. We need two lemmas, which are presented without proof:

Lemma 1. *Every solution of the Flow Model is a solution of Model A.*

Lemma 2. *Every solution of the model LP1 is a solution of Model B.*

This suffices to show that the Flow Model is stronger than Model A, and Model LP1 is stronger than Model B. The other "stronger" relations are immediate. All "incomparable" results follow from experimental outcomes. Figure 7 shows as example where the Element Model and Algorithm A outperform the model DLP. Table 4 below shows cases where the Element Model is stronger than Model A and Model LP2, and where the Flow Model outperforms Algorithm B.

Fig. 7. Example: Element and Flow Models stronger than DLP



Why are so many of the models presented incomparable? They all provide different relaxations of the basic constraints which make up the `CumulativeCost` constraint. In Table 2 we compare our methods based on four main concepts, whether the algorithms respect the capacity limits for each single area, or for (some of the) multiple areas, whether the earliest start/latest end limits for the start times are respected, and whether the tasks are placed as rectangles with the given width and height. Each method respects a different subset of the constraints. Methods are comparable if for all concepts one model is stronger than the other, but incomparable if they relax different concepts.

Without space for a detailed complexity analysis, we just list on the right side of Table 2 the main worst-case results for each method. We differentiate the time required to preprocess and generate the constraints, and the number of variables and constraints for the LP and MIP models. Recall that n is the number of tasks, m the number of areas and p the length of the scheduling period. Note that these numbers do not estimate the time for computing the lower bounds but for setting-up the models.

7 Experiments

We implemented all algorithms discussed in the paper, referring to them as {*DMIP, Element, Algo A, Algo B, Flow, LP1, LP2, DLP*}. For convenience we denote them also

Table 2. Model Satisfies Basic Constraints/Complexity

Method	Single Area Capacity	Multi Area Capacity	Earliest Start Latest End	Task Profile	Model Setup	Variables	Constraints
Element	no	no	yes	yes	$\mathcal{O}(npm)$	-	-
Flow	yes	no	yes	$\leq$ height	$\mathcal{O}(nm)$	$\mathcal{O}(nm)$	$\mathcal{O}(n+m)$
LP 1	yes	$\overline{B_j}$	yes	$\leq$ height	$\mathcal{O}(nm^2)$	$\mathcal{O}(nm)$	$\mathcal{O}(n+m)$
LP 2	yes	$\overline{b_{ij}}$	yes	$\leq$ height	$\mathcal{O}(nm^2)$	$\mathcal{O}(nm)$	$\mathcal{O}(nm)$
Alg A	yes	no	no	no	$\mathcal{O}(nm)$	-	-
Alg B	yes	$\overline{B_j}$	no	no	$\mathcal{O}(nm^2)$	-	-
DLP	yes	yes	yes	width		$\mathcal{O}(np+mp)$	$\mathcal{O}(n+m+p)$
DMIP	yes	yes	yes	yes		$\mathcal{O}(np+mp)$	$\mathcal{O}(n+m+p)$

as $Algo_0, \ldots, Algo_7$. We evaluated their performance over a number of instances, denoted E. Each instance $e \in E$ is defined by the areas $A_1, \ldots, A_m$ and tasks $T_1, \ldots, T_n$. In all experiments we use a day-long time horizon divided into 48 half-hour areas and two cost settings. In the first setting for each half-hour period we fix a price to given values from a real-world price distribution (Fig. 1) over all the task specifications. We denote this as a *fixed* cost distribution ($cost = fixed$). Alternatively, for each area we choose a random cost from interval $[0, 100]$, for each task specification. We refer to this as a *random* cost distribution ($cost = random$). In both cases the costs are normalized by subtracting the minimal cost $c_{min} = min_{j=1}^{m} c_j$ from each area cost. For a given $n, d_{max}, r_{max}, \Delta$ we generate tasks $T_1, \ldots, T_n$ with randomly selected durations $d_i \in [1, d_{max}]$ and resource consumptions $r_i \in [1, r_{max}]$, $i = 1, \ldots, n$. Δ denotes the maximal distance between the earliest and latest start time. For a given Δ we randomly select the interval $s_i \in [\underline{s_i}, \overline{s_i}]$ so that $0 \leq \overline{s_i} - \underline{s_i} \leq \Delta$. One of the major parameters we consider is *utilization* defined as a portion of the total available area that must be covered by the tasks: $util = \sum_{i=1}^{n} d_i r_i / (l \cdot p)$. In the experiments we first decide the desired level of utilization, $util$, and then select a corresponding capacity limit $l = \lceil \frac{\sum_{i=1}^{n} d_i r_i}{util \cdot p} \rceil$. Hence, each instance scenario is obtained by specifying $(n, d_{max}, r_{max}, \Delta, util, cost)$.

We used *Choco v2.1.1*[1] as the underlying constraint solver, and *CPLEX v12.1* [2] as a MIP/LP solver. The experiments ran as a single thread on a Dual Quad Core Xeon CPU, 2.66GHz with 12MB of L2 cache per processor and 16GB of RAM overall, running Linux 2.6.25 x64. All the reported times are in milliseconds.

General Comparison In the first set of experiments we compare the quality of the bounds for all algorithms presented. Let val_i denote a value (lower bound) returned by $Algo_i$. The *DMIP* algorithm, $Algo_0$ solves the problem exactly, yielding the value val_0 equal to the optimum. For other algorithms i we define its quality q_i as val_i / val_0.

For a given instance we say that $Algo_i$ is *optimal* if $q_i = 100\%$. For all algorithms except *DMIP* (i.e. $i > 0$), we say that they are the *best* if they have the maximal lower

[1] `http://choco-solver.net`

[2] `http://www-01.ibm.com/software/integration/optimization/cplex/`

bound over the non-exact algorithms, i.e. $q_i = max_{j>0} q_j$. Note that the algorithm might be best but not optimal. For a given scenario $(n, d_{max}, r_{max}, \Delta, util, cost)$ we generate 100 instances, and compute the following values: 1) number of times that the algorithm was optimal or best 2) average and minimal quality q_i, 3) average and maximal execution time. We evaluate the algorithms over scenarios with $n = 100$ tasks, $d_{max} = 8$, $r_{max} = 8$, $\Delta = 10$, $util \in \{30\%, 50\%, 70\%, 80\%\}$ and $cost \in \{fixed, random\}$. This leads in total to eight instance scenarios presented in Table 3. The first column indicates the scenario (utilization and cost distribution). The last eight columns indicate the values described above for each algorithm. First note that the bound estimates for algorithms *B, LP1, LP2* are on average consistently higher than 95% over all scenarios, but that *DLP* provides exceptionally high bounds, on average over 99.6% for each scenario. While algorithms *A* and *Flow* can provide weaker bounds for lower utilization, the bounds improve for higher utilization. Algorithm *Element* on the other hand performs better for lower utilization (since its bound ignores capacity l) and deteriorates with higher utilization.

Table 3. Evaluation of algorithms for $d_{max} = r_{max} = 8$ and $\Delta = 10$ with varying *util* and *cost*.

Scenario	Key	DMIP		Element		A		B		Flow		LP1		LP2		DLP	
util=30	Opt/Best	100	-	92	92	0	0	0	0	0	0	0	0	0	0	93	100
fixed	Avg/Min Quality	100.0	100.0	99.998	99.876	57.268	33.055	99.77	99.337	97.429	94.665	99.77	99.337	99.77	99.337	99.999	99.994
	Avg/Max Time	29	194	185	509	8	73	12	126	34	138	150	277	211	617	111	380
util=50	Opt/Best	100	-	2	2	0	0	0	0	0	0	0	0	0	0	7	100
fixed	Avg/Min Quality	100.0	100.0	99.038	94.641	68.789	54.22	99.131	95.963	97.816	95.89	99.407	97.773	99.435	97.913	99.948	99.358
	Avg/Max Time	89	2,108	176	243	6	12	7	94	33	130	139	274	194	275	96	181
util=70	Opt/Best	100	-	0	0	0	0	0	0	0	1	0	5	0	6	1	100
fixed	Avg/Min Quality	100.0	100.0	93.541	81.603	84.572	69.953	96.495	87.884	99.1	96.994	99.24	97.838	99.346	98.071	99.764	98.992
	Avg/Max Time	2,107	32,666	177	242	7	103	8	72	34	97	136	239	213	1,798	110	1,551
util=80	Opt/Best	100	-	0	0	0	0	0	1	0	4	0	20	0	21	0	100
fixed	Avg/Min Quality	100.0	100.0	88.561	70.901	92.633	81.302	96.163	89.437	99.34	96.728	99.354	96.737	99.392	96.737	99.649	98.528
	Avg/Max Time	13,017	246,762	206	450	10	67	15	96	38	124	156	235	220	426	125	299
util=30	Opt/Best	100	-	94	94	0	0	0	0	0	0	0	0	0	0	97	100
random	Avg/Min Quality	100.0	100.0	99.996	99.872	58.094	41.953	96.965	93.237	73.759	54.641	96.965	93.237	96.966	93.254	99.999	99.977
	Avg/Max Time	29	154	192	427	7	24	8	42	32	94	145	224	203	361	99	274
util=50	Opt/Best	100	-	0	0	2	8	2	8	2	8	2	8	2	8	5	100
random	Avg/Min Quality	100.0	100.0	88.277	30.379	76.457	57.049	96.585	92.563	83.314	69.178	96.619	92.604	96.861	93.242	99.93	99.724
	Avg/Max Time	2,452	62,168	202	814	10	99	13	131	43	177	165	380	238	903	108	327
util=70	Opt/Best	100	-	0	0	0	0	0	0	0	0	0	0	0	0	0	100
random	Avg/Min Quality	100.0	100.0	91.045	72.06	89.784	75.822	95.242	90.496	92.953	84.277	95.953	92.24	96.947	94.012	99.697	99.374
	Avg/Max Time	72,438	2,719,666	226	436	13	74	26	98	70	178	223	543	280	566	152	428
util=80	Opt/Best	100	-	0	0	0	0	0	0	0	0	0	0	0	0	0	100
random	Avg/Min Quality	100.0	100.0	86.377	72.566	94.813	88.039	96.092	89.233	97.231	93.919	97.658	94.917	98.426	96.342	99.626	99.161
	Avg/Max Time	684,660	8,121,775	320	2,148	16	100	31	180	63	370	223	1,586	363	23,91	286	7,242

We also performed experiments over unrestricted initial domains, i.e. where $s_i \in [0, p-1]$. While most of the algorithms improved their bound estimation, *Element* performed much worse for the fixed cost distribution, reaching quality as low as 12% for high utilization. On the other hand, reducing Δ to 5 significantly improved the quality of Element, while slightly weakening the quality of other algorithms. In all the cases, the worst-case quality of *DLP* was still higher than 98.6%.

Aggregate Pairwise Comparison In Table 4 we compare all algorithms over the entire set of instances that were generated under varying scenarios involving 100 tasks. For any two algorithms $Algo_i$ and $Algo_j$ let $E_{ij} = \{e \in E \mid q_{ei} > q_{ej}\}$ denote the set of instances where $Algo_i$ outperforms $Algo_j$. Let $num_{ij} = |E_{ij}|$ denote the number of such instances, while avg_{ij} and max_{ij} denote the average and maximal difference in quality over E_{ij}. In the (i, j)-th cell of Table 4 we show $(num_{ij}, avg_{ij}, max_{ij})$. In approximately 2000 instances *DLP* strictly outperforms all other non-exact algorithms in more than 1700 cases and is never outperformed by another. Algorithm *B* outperforms *Flow* (1107 cases) more often than *Flow* outperforming *B* (333 cases). Furthermore, *LP2* outperforms *LP1* in about 700 cases. Interestingly, even though *Element* produces on average weaker bounds, it is able to outperform all non-exact algorithms except *DLP* on some instances.

Table 4. Comparison of algorithms over all instances generated for experiments with $n = 100$ tasks.

	Element			**A**			**B**			**Flow**			**LP1**			**LP2**			**DLP**		
DMIP	1802	26.39	88.62	1944	17.47	68.55	1944	3.99	30.71	1944	9.85	68.55	1944	3.49	30.71	1944	3.24	30.71	1387	0.18	1.47
EL	-	-	-	1034	25.62	66.95	656	3.0	22.07	856	15.65	65.82	650	3.01	22.07	621	3.04	22.07			
A	1052	38.1	88.61	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-			
B	1429	29.22	88.61	1439	18.22	66.65	-	-	-	1107	11.02	51.86	-	-	-	-	-	-			
FLW	1230	33.97	88.61	1184	12.51	64.35	333	2.39	10.49	-	-	-	-	-	-	-	-	-			
LP1	1436	29.74	88.61	1441	18.86	66.65	726	1.33	10.51	1413	8.75	51.86	-	-	-	-	-	-			
LP2	1465	29.44	88.61	1441	19.19	66.65	846	1.71	10.64	1425	9.02	51.86	690	0.7	5.09	-	-	-			
DLP	1802	26.24	88.61	1752	19.24	68.55	1751	4.28	30.71	1747	10.82	68.55	1727	3.78	30.71	1725	3.51	30.71	-	-	-

Varying Number of Tasks Finally, we evaluated the effect of increasing the number of tasks. We considered scenarios $n \in \{50, 100, 200, 400\}$ where $d_{max} = 8$, $r_{max} = 8$, $\Delta = 10$, $util = 70$ and $cost = random$.The results are reported in Table 5. We can notice that all algorithms improve the quality of their bounds with an increase in the number of tasks. While the time to compute the bounds grows for the non-exact algorithms, interestingly this is not always the case for the *DMIP* model. The average time to compute the optimal value peaks for $n = 100$ (72 seconds on average) and then reduces to 24 and 9 seconds for $n = 200$ and $n = 400$ respectively.

8 Conclusions and Future Work

We have introduced the `CumulativeCost` constraint, a resource cost-aware extension of the standard cumulative constraint, and suggested a number of algorithms to compute lower bounds on its cost. We compared the algorithms both theoretically and experimentally and discovered that while most of the approaches are mutually incomparable, the *DLP* model clearly dominates all other algorithms for the experiments considered. Deriving lower bounds is a necessary first step towards the development of the `CumulativeCost` constraint. We plan to evaluate the performance of the constraint in a number of real-life scheduling problems, which first requires development of domain filtering algorithms and specialized variable/value ordering heuristics.

Table 5. Evaluation of algorithms for random cost distribution, $\Delta = 10$ and $util = 70$.

Scenario	Key	DMIP		Element		A		B		Flow		LP1		LP2		DLP	
n=50	Opt/Best	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100
	Avg/Min Quality	100.0	100.0	89.331	71.865	88.833	70.372	93.991	86.143	92.384	84.589	94.863	87.417	95.899	89.937	98.664	95.523
	Avg/Max Time	1,334	31,426	102	314	8	83	6	36	23	78	93	254	132	298	75	233
n=100	Opt/Best	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100
	Avg/Min Quality	100.0	100.0	91.045	72.06	89.784	75.822	95.242	90.496	92.953	84.277	95.953	92.24	96.947	94.012	99.697	99.374
	Avg/Max Time	72,438	2,719,666	226	436	13	74	26	98	70	178	223	543	280	566	152	428
n=200	Opt/Best	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100
	Avg/Min Quality	100.0	100.0	92.537	84.06	89.819	79.862	95.885	93.01	92.566	83.516	96.48	93.328	97.158	93.885	99.936	99.833
	Avg/Max Time	24,491	233,349	395	700	19	148	22	113	83	208	341	533	468	638	226	456
n=400	Opt/Best	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100
	Avg/Min Quality	100.0	100.0	93.239	86.419	90.417	84.205	96.3	92.721	92.939	86.658	96.716	93.275	97.23	95.013	99.985	99.961
	Avg/Max Time	9,379	158,564	831	1,222	31	164	36	189	181	305	923	3,053	1,214	3,434	484	871

References

1. A. Aggoun and N. Beldiceanu. Extending CHIP in order to solve complex scheduling problems. *Journal of Mathematical and Computer Modelling*, 17(7):57–73, 1993.
2. P. Baptiste, C. Le Pape, and W. Nuijten. *Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems*. Kluwer, Dortrecht, 2001.
3. Nicolas Beldiceanu and Mats Carlsson. A new multi-resource cumulatives constraint with negative heights. In Pascal Van Hentenryck, editor, *CP*, volume 2470 of *Lecture Notes in Computer Science*, pages 63–79. Springer, 2002.
4. Nicolas Beldiceanu, Mats Carlsson, and Emmanuel Poder. New filtering for the cumulative constraint in the context of non-overlapping rectangles. In Laurent Perron and Michael A. Trick, editors, *CPAIOR*, volume 5015 of *Lecture Notes in Computer Science*, pages 21–35. Springer, 2008.
5. Nicolas Beldiceanu and Emmanuel Poder. A continuous multi-resources cumulative constraint with positive-negative resource consumption-production. In Pascal Van Hentenryck and Laurence A. Wolsey, editors, *CPAIOR*, volume 4510 of *Lecture Notes in Computer Science*, pages 214–228. Springer, 2007.
6. John Hooker. *Integrated Methods for Optimization*. Springer, New York, 2007.
7. Luc Mercier and Pascal Van Hentenryck. Edge finding for cumulative scheduling. *INFORMS Journal on Computing*, 20(1):143–153, 2008.
8. Andreas Schutt, Thibaut Feydy, Peter J. Stuckey, and Mark Wallace. Why cumulative decomposition is not as bad as it sounds. In Ian P. Gent, editor, *CP*, volume 5732 of *Lecture Notes in Computer Science*, pages 746–761. Springer, 2009.
9. Helmut Simonis and Trijntje Cornelissens. Modelling producer/consumer constraints. In Ugo Montanari and Francesca Rossi, editors, *CP*, volume 976 of *Lecture Notes in Computer Science*, pages 449–462. Springer, 1995.
10. Petr Vilím. Max energy filtering algorithm for discrete cumulative resources. In Willem Jan van Hoeve and John N. Hooker, editors, *CPAIOR*, volume 5547 of *Lecture Notes in Computer Science*, pages 294–308. Springer, 2009.