American Journal of Operations Research, **2013, 3, 589-600**
Published Online November 2013 (http://www.scirp.org/journal/ajor)
http://dx.doi.org/10.4236/ajor.2013.36056

Scientific Research

Energetic Extended Edge Finding Filtering Algorithm for Cumulative Resource Constraints

Roger Kameugne[1,2], Sévérine Betmbe Fetgo[3], Laure Pauline Fotso[3]
[1]Department of Mathematics, Higher Teachers' Training College, University of Maroua, Maroua, Cameroon
[2]Department of Mathematics, Faculty of Sciences, University of Yaounde I, Yaoundé, Cameroon
[3]Department of Computer Sciences, Faculty of Sciences, University of Yaounde I, Yaoundé, Cameroon
Email: rkameugne@yahoo.fr, rkameugne@gmail.com, betmbe2000@yahoo.fr, lpfotso@ballstate.bsu.edu

Received September 28, 2013; revised October 28, 2013; accepted November 5, 2013

ABSTRACT

Edge-finding and energetic reasoning are well known filtering rules used in constraint based disjunctive and cumulative scheduling during the propagation of the resource constraint. In practice, however, edge-finding is most used (because it has a low running time complexity) than the energetic reasoning which needs $\mathcal{O}(n^2)$ time-intervals to be considered (where n is the number of tasks). In order to reduce the number of time-intervals in the energetic reasoning, the maximum density and the minimum slack notions are used as criteria to select the time-intervals. The paper proposes a new filtering algorithm for cumulative resource constraint, titled energetic extended edge finder of complexity $\mathcal{O}(n^3)$. The new algorithm is a hybridization of extended edge-finding and energetic reasoning: more powerful than the extended edge-finding and faster than the energetic reasoning. It is proven that the new algorithm subsumes the extended edge-finding algorithm. Results on Resource Constrained Project Scheduling Problems (RCPSP) from BL set and PSPLib librairies are reported. These results show that in practice the new algorithm is a good trade-off between the filtering power and the running time on instances where the number of tasks is less than 30.

Keywords: Constraint-Based Scheduling; Global Constraint; Cumulative Resource; Energetic Reasoning; Edge-Finding; Extended Edge-Finding; Maximum Density; Minimum Slack

1. Introduction

Scheduling is the process of assigning resources to tasks or activities over the time. There exist many types of scheduling problems following the tasks properties (preemptive or non-preemptive), the type of resources (disjunctive or cumulative) and the objective function (makespan, time late...). When a unique cumulative resource with non-preemptive tasks is considered, the problem is called cumulative scheduling problem (CuSP).

In a CuSP, a set of tasks T has to be executed on a resource of fixed capacity C. Each task i requires a fixed and constant amount of resource c_i, and has to be executed during a fixed amount of time p_i without interruption between an earliest start time est_i (release date) and a latest completion time lct_i (deadline). A solution of a CuSP instance is an assignment of valid start time s_i to each task i in such a way that resource constraints are satisfied *i.e.*,

$$\forall i \in T : \text{est}_i \le s_i \le s_i + p_i \le \text{lct}_i \quad (1)$$

$$\forall \tau : \sum_{i \in T,\, s_i \le \tau < s_i + p_i} c_i \le C \quad (2)$$

The inequalities in (1) ensure that each task is assigned a feasible start and end time, while (2) enforces the resource constraint. An example of CuSP is given in **Figure 1**.

The energy of a task i, is defined as $e_i = c_i \cdot p_i$, its earliest completion time as $\text{ect}_i = \text{est}_i + p_i$ and its latest start time as $\text{lst}_i = \text{lct}_i - p_i$. The energy notation along with that of earliest start and latest completion time may be extended to non-empty sets of tasks as follows:

$$e_\Omega = \sum_{j \in \Omega} e_j,\ \text{est}_\Omega = \min_{j \in \Omega} \text{est}_j,\ \text{lct}_\Omega = \max_{j \in \Omega} \text{lct}_j \quad (3)$$

where Ω is a non-empty set of tasks. By convention, if Ω is the empty set, $\text{est}_\Omega = +\infty$, $\text{lct}_\Omega = -\infty$, and $e_\Omega = 0$. Throughout the paper, it is assumed that for any

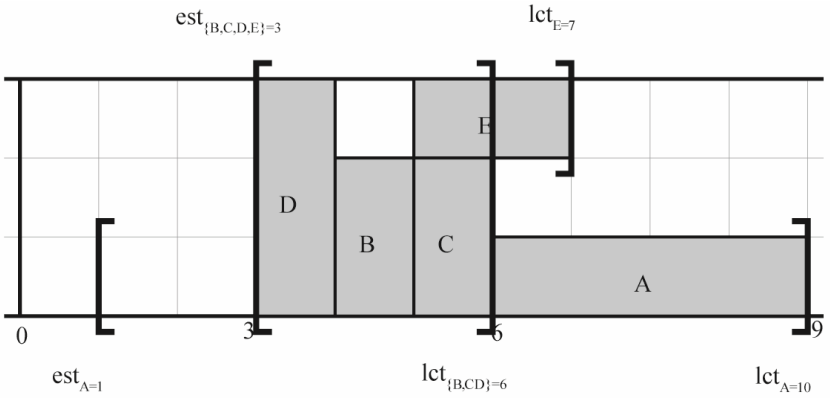

Figure 1. A scheduling problem of 5 tasks sharing a resource of capacity $C = 3$.

task $i \in T$, $\text{est}_i + p_i \leq \text{lct}_i$ and $c_i \leq C$, otherwise the problem has no solution.

The CuSP is a NP-complete problem [1]. Therefore, only relaxation of the problem, for which it is possible to implement a polynomial time algorithm exists. In [2], the cumulative resource constraint is modelized by the global constraint CUMULATIVE in a constraint programming approach. The global constraint CUMULATIVE embeds many filtering algorithms. Among these algorithms, energetic reasoning, edge-finding and timetabling are the most used.

1.1. Related Works

To our knowledge, the word "energetic edge-finder" was firstly used in [3] where the author incorporates the energy-based deduction rule to edge-finder algorithm for disjunctive (unary) resource. The idea of hybridization of the edge-finding rule and the energetic reasoning for cumulative resource was suggested in [4]. Indeed, in [4], Mercier and Van Hentenryck propose a two phase edge-finding algorithm where in the first phase, the potential adjustment values are computed. They found that many of these potential update values are unused and can be used inside an energetic-based second phase. After this suggestion, many filtering algorithms, hybridization of edge-finding rule and energetic reasoning have been proposed [5,6]. In [6], the authors use in the first phase, the edge-finding algorithm of [7,8] to compute the potential update values and identify the corresponding time bounds of task intervals which provide the maximum update values. To each task, the time bounds used in the second phase is either the task intervals of maximum density or the task intervals of minimum slack. The resulting algorithm runs in $\mathcal{O}(n^2)$ since both phases have this complexity. This algorithm was later improved in [5] by choosing for each task, the time bounds of task intervals of both maximum density and minimum slack which provide the maximum update values. Experimental results on RCPSP of the PSPLib [9] library and BL set [10] show that this variant is a good trade-off between the filtering power and the complexity but it does not dominate the edge-finding algorithm.

Energetic reasoning is one of the most powerful filtering algorithm in cumulative scheduling problems [1,10] since it dominates all the other rules (edge-finding [4,7, 8,21], extended edge-finding [4,11], timetable [1], timetable edge-finding [12], timetable extended edge-finding [13]) except the not-first/not-last rule [14,15]. However, it is not commonly used because it has a high running time ($\mathcal{O}(n^3)$ time complexity), needs a high number of time-intervals to be considered and its success highly depends on the tightness of the variable bounds (highly cumulative problems). Recently, in [16], the authors proposed an approximative criterion for the potential of the energetic reasoning which allows the decrease of the running time with more nodes to explore in the search tree. The combination of a solver of the CP and SAT techniques for conflict analysis played recently an important role to solve cumulative scheduling problems efficiently [17,18].

In this paper, it is extended the energetic edge finder of [5] by adding the guarantee that the final algorithm deduces more than edge-finding and extended edge-finding. The main idea of our energetic extended edge finder is the combination of the best properties of (extended) edge-finding rule (interesting time bounds and good running time) and energetic reasoning (powerful filtering rule). The paper starts with the hypothesis that, the time bounds of task intervals used in the (extended) edge-finding algorithm can be interesting in an energetic reasoning. Based on this hypothesis, the number of time intervals to be considered in an energetic reasoning is reduced to those of (extended) edge-finding.

1.2. Contribution

This paper uses the time-bounds of task intervals considered in the computation of the edge-finding algorithm [7,8] and the extended edge-finding algorithm [11] in an

energetic reasoning. Indeed, in [7,8], a complete edge-finding algorithm is proposed based on maximum density and minimum slack. The minimum slack is used in [11] to perform the extended edge-finding algorithm. The algorithm presented in this paper is an energetic version of the edge-finding and the extended edge-finding algorithms of [7,8,11] respectively. The complexity of the corresponding algorithm is $\mathcal{O}(n^3)$ where n is the number of tasks since each of the algorithm of [7,8] and [11] are quadratic.

It is obvious that the new algorithm subsumes the edge-finding and the extended edge-finding algorithms. The filtering power of this algorithm is less than the one of the energetic reasoning, but in practice, it is a good trade-off between the filtering power and the running time. Empirical evaluation of the algorithm on the RCPSP instances of well known library prove that, the new algorithm performs in most of the cases better in term of reduction of number of nodes of the search tree than the quadratic extended edge-finding [11], but needs more time to do so, for instances of more than 30 tasks.

The rest of the paper is organized as follows: Section 2 is devoted to the specification of the edge-finding rule and the energetic reasoning. In Section 3, the new energetic extended edge-finder is described and some of its properties are deduced. Experimental results are provided in the last Section.

2. Edge-Finding Rule and Energetic Reasoning Rule

In this section, it is specified the (extended) edge-finding rule as well as the energetic reasoning.

2.1. Edge-Finding and Extended Edge-Finding Rules

Let T be the set of tasks of a CuSP. If the energy e_Ω of a set of tasks $\Omega \subset T$ is larger than the available energy $C(\mathrm{lct}_\Omega - \mathrm{est}_\Omega)$, then the problem has no feasible solution. *Overload checking* algorithms typically enforce the following relaxation of this feasibility condition, which may be computed in $\mathcal{O}(n \log n)$ time [19,20].

Definition 1 (E-Feasibility) ([4]) *A CuSP problem is E-feasible if*:

$$\forall \Omega \subseteq T, \quad \Omega \neq \varnothing, \quad C(\mathrm{lct}_\Omega - \mathrm{est}_\Omega) \geq e_\Omega. \tag{4}$$

For a given CuSP, an edge-finding rule identifies a set of tasks $\Omega \subset T$ and a task $i \notin \Omega$ such that, in any solution, all tasks of Ω end before the end of i. More precisely, if the scheduling of task i as early as possible (*i.e.*, starting at est_i) induces an overload in the interval $[\mathrm{est}_\Omega, \mathrm{lct}_\Omega)$ then, all the tasks in Ω end before the end of i noted here $\Omega \ll i$ as in [21]. When all tasks of a set Ω end before the end of a task i, then the release date of task i is updated to:

$$\Omega \ll i \Rightarrow \mathrm{est}_i \geq \mathrm{est}_\Theta + \left\lceil \frac{1}{c_i} \mathrm{rest}(\Theta, c_i) \right\rceil \tag{5}$$

for all $\Theta \subseteq \Omega$ such that $\mathrm{rest}(\Theta, c_i) > 0$,

$$\mathrm{rest}(\Theta, c_i) = \begin{cases} e_\Theta - (C - c_i)(\mathrm{lct}_\Theta - \mathrm{est}_\Theta) & \text{if } \Theta \neq \varnothing \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

Proposition 1 provides conditions under which all tasks of a set Ω of a CuSP end before the end of a task i.

Proposition 1 [4,7,8,11,21] *Let $\Omega \subseteq T$ be a set of tasks of a CuSP of capacity C and $i \in T \setminus \Omega$ be a task.*

$$e_{\Omega \cup \{i\}} > C(\mathrm{lct}_\Omega - \mathrm{est}_{\Omega \cup \{i\}}) \Rightarrow \Omega \ll i; \tag{EF}$$

$$\mathrm{ect}_i \geq \mathrm{lct}_\Omega \Rightarrow \Omega \ll i; \tag{EF1}$$

$$\begin{cases} \mathrm{est}_i \leq \mathrm{est}_\Omega < \mathrm{ect}_i \\ e_\Omega + c_i(\mathrm{ect}_i - \mathrm{est}_\Omega) > C(\mathrm{lct}_\Omega - \mathrm{est}_\Omega) \end{cases} \Rightarrow \Omega \ll i. \tag{EEF}$$

And if $\Omega \ll i$ then the earliest start time of task i is updated to

$$\mathrm{est}_i = \max\left(\mathrm{est}_i, \max_{\substack{\Theta \subseteq \Omega \\ \mathrm{rest}(\Theta, c_i) > 0}} \mathrm{est}_\Theta + \left\lceil \frac{1}{c_i} \mathrm{rest}(\Theta, c_i) \right\rceil \right) \tag{Upd}$$

with

$$\mathrm{rest}(\Theta, c_i) = \begin{cases} e_\Theta - (C - c_i)(lct_\Theta - est_\Theta) & \text{if } \Theta \neq \varnothing \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

Rules (EF) and (EF1) are known as edge-finding detection rules while (EEF) is the extended edge-finding detection rule. It is proved in [4] that a complete edge-finding algorithm that only considers sets $\Omega \subset T$ and $\Theta \subseteq \Omega$, which are task intervals can be implemented. The definition of the task intervals is given in Definition 2.

Definition 2 (Task Intervals) (*After* [22]) *Let $j, k \in T$. The task intervals $\Omega_{j,k}$ is the set of tasks*

$$\Omega_{j,k} = \{ s \in T \mid \mathrm{est}_j \leq \mathrm{est}_s \wedge \mathrm{lct}_s \leq \mathrm{lct}_k \}. \tag{8}$$

In [7,8], the authors propose a quadratic edge-finding algorithm based on maximum density and minimum slack notions.

Definition 3 [7,8] *Let Ω be a task set of an E-feasible CuSP. The slack of the task set Ω, denoted SL_Ω, is given by*:

$$SL_\Omega = C(\mathrm{lct}_\Omega - \mathrm{est}_\Omega) - e_\Omega.$$

Definition 4 [7,8] *Let i and k be two tasks of an E-feasible CuSP. $\tau(k,i)$ is a task depending on the tasks k and i, where $\mathrm{est}_{\tau(k,i)} \leq \mathrm{est}_i$ and that defines the task intervals with the minimum slack: for all $j \in T$*

such that $\text{est}_j \le \text{est}_i$,

$$C\left(\text{lct}_k - \text{est}_{\tau(k,i)}\right) - e_{\Omega_{\tau(k,i),k}} \le C\left(\text{lct}_k - \text{est}_j\right) - e_{\Omega_{j,k}}. \quad (9)$$

The detection of the classic edge-finding rule (EF) is done with the task intervals of minimum slack. For adjustment, as it is proved in [7,8], the task intervals of minimum slack and the one of maximum density are considered.

Definition 5 [7,8] *Let* Θ *be a task set of an E-feasible CuSP. The density of the task set* Θ, *denoted* $Dens_\Theta$, *is given by*:

$$Dens_\Theta = \frac{e_\Theta}{\text{lct}_\Theta - \text{est}_\Theta}.$$

Definition 6 [7,8] *Let* i, k *be two tasks of an E-feasible CuSP.* $\rho(k,i)$ *is a task depending on the tasks* k *and* i, *where* $\text{est}_i < \text{est}_{\rho(k,i)}$ *and that defines the task intervals with the maximum density*: *for all tasks* $j \in T$ *such that* $\text{est}_i < \text{est}_j$,

$$\frac{e_{\Theta_{j,k}}}{\text{lct}_k - \text{est}_j} \le \frac{e_{\Theta_{\rho(k,i),k}}}{\text{lct}_k - \text{est}_{\rho(k,i)}}. \quad (10)$$

The main idea of the edge-finding algorithm of [7,8] is that once the relation $\Omega \ll i$ is discovered, then it is not necessary to iterate over all subsets Θ of Ω. It is enough to consider only (1) subset with minimum slack and $\text{est}_\Theta \le \text{est}_i$ and (2) subset with maximum density and $\text{est}_\Theta > \text{est}_i$. Using those two subsets, if est_i can be improved, then it will be updated, although not necessarily immediately to the best value. More iterations of the algorithm may be needed for that.

The extended edge-finding rule (EEF) detects additional updates missing by the edge-finding rule. In [11], the authors propose a quadratic extended edge-finding algorithm based on minimum slack notion. This algorithm supposes that the fix point of the edge-finding is reached and looks for the upper bound of the task intervals of minimal slack instead of the lower bound as it is the case for the edge-finding algorithm.

Definition 7 [11] *Let* i *and* j *be two tasks of an E-feasible CuSP with* $\text{est}_i < \text{est}_j$. $\delta(j,i)$ *is a task depending on the tasks* j *and* i, *where* $\text{lct}_{\delta(j,i)} \le \text{lct}_i$ *and that defines the largest task intervals with the minimum slack*: *for all* $k \in T$ *such that* $\text{lct}_k \le \text{lct}_i$

$$C\left(\text{lct}_{\delta(j,i)} - \text{est}_j\right) - e_{\Omega_{j,\delta(j,i)}} \le C\left(\text{lct}_k - \text{est}_j\right) - e_{\Omega_{j,k}}. \quad (11)$$

2.2. Energetic Reasoning

In the edge-finding rule, the energy required by a non-empty set of tasks Ω only considers tasks which are completely processed within the time window $[\text{est}_\Omega, \text{lct}_\Omega)$ while, the partial contribution of each task is taken into account in the energetic reasoning. There exists many varieties of energy consumption required by a task depending on the type of tasks, Fully Elastic energy and Partial Elastic energy for preemptive tasks and Left-shift/Right-shift energy for non-preemptive tasks [10].

Let i be a task and $[a,b)$ be a time interval with $a < b$. The left-shift/right-shift energy consumption required by i over $[a,b)$ noted $W(a,b,i)$ is the non-negative minimum of 1) the volume in the interval $[a,b)$, 2) the energy of task i, 3) the left shifted energy, and 4) the right shifted energy *i.e.*,

$$\begin{aligned} &W(a,b,i) \\ &= c_i \cdot \max\left(0, \min\left(b-a, p_i, \text{ect}_i - a, b - \text{lst}_i\right)\right) \end{aligned} \quad (12)$$

The overall energy consumption required by all tasks noted $W(a,b)$ over an interval $[a,b)$ is defined as

$$W(a,b) = \sum_{i \in T} W(a,b,i). \quad (13)$$

For a given CuSP, it is obvious that if there exists a time interval $[a,b)$ with $a < b$ such that its overall required energy consumption is more than available energy, then the problem is infeasible. This necessary condition is provided in the following proposition.

Proposition 2 [1] *Let* $[a,b)$ *be a time interval with* $a < b$. *If*

$$C(b-a) < W(a,b) \quad (14)$$

then the problem is infeasible.

In [1,10], the authors give a precise characterization of time bounds for which the necessary condition of the existence of a feasible scheduling should be guaranteed. This necessary condition is more powerful than the E-feasible one defined earlier. There exist $\mathcal{O}(n^2)$ relevant intervals to be considered to detect infeasibility. These intervals correspond to start and completion times of tasks. If

$$\begin{aligned} &O_1 := \bigcup_{i \in T} \{\text{est}_i, \text{ect}_i, \text{lst}_i\}, \\ &O_2 := \bigcup_{i \in T} \{\text{lct}_i, \text{ect}_i, \text{lst}_i\} \\ &\text{and } O(t) := \bigcup_{i \in T} \{\text{est}_i + \text{lct}_i - t\} \end{aligned}$$

then the relevant intervals are given by $(a,b) \in O_1 \times O_2$, for a fixed $a \in O_1$: $(a,b) \in O_1 \times O(a)$, and for a fixed $b \in O_2$: $(a,b) \in O(b) \times O_2$, with $a < b$. The authors propose a quadratic algorithm for testing this necessary condition in [1,10].

The left-shift energy required of a task i over a time interval $[a,b)$ defined by

$$\begin{aligned} &W_l(a,b,i) \\ &= c_i \cdot \min\left(b-a, p_i, \min\left(\text{ect}_i, b\right) - \max\left(\text{est}_i, a\right)\right) \end{aligned} \quad (15)$$

is c_i times the number of time units during which i executes after time a if i is left-shifted, *i.e.*, scheduled as soon as possible. If scheduling task i as early as possible (*i.e.*, starting at est_i) induces an overload in the interval $[a,b)$ then task i ends after b and est_i can be updated according to Proposition 3.

Proposition 3 [1] *Let* $[a,b)$ *be an interval with* $a<b$ *and* i *be a task such that*

$$[a,b)\cap[\text{est}_i,\text{ect}_i)\neq\varnothing.$$

If

$$W(a,b)-W(a,b,i)+W_l(a,b,i)>C\cdot(b-a) \quad \text{(ER)}$$

holds, then the earliest start time of task i is updated to:

$$\text{est}_i=\max\left(\text{est}_i,a+\left\lceil\frac{1}{c_i}\left(W(a,b)-W(a,b,i)-(C-c_i)\cdot(b-a)\right)\right\rceil\right). \quad \text{(ERupd)}$$

No specific characterization of relevant intervals for the adjustment was proposed so far. With the $\mathcal{O}(n^2)$ relevant intervals used for infeasibility test, it is derived a $\mathcal{O}(n^3)$ algorithm for adjustment in [1,10]. In practice, the energetic reasoning adjustment is too time-consuming for producing any useful result [1]. The paper tries to determine a better trade off between the filtering power and running time by reducing the number of intervals to examine. It is used the time bounds of task intervals of edge-finding and extended edge-finding algorithm in an energetic reasoning. The corresponding algorithm runs in $\mathcal{O}(n^3)$ as pure energetic reasoning adjustment.

3. The Energetic Extended Edge-Finder

3.1. The Rule

The energetic extended edge-finding rule is obtained by substituting e_Ω by $W(\text{est}_\Omega,\text{lct}_\Omega)$ in the edge-finding detection rule (EF) and the extended edge-finding detection rule (EEF). It can be observed that the energetic reasoning rule (ER) is a generalization of rules (EF) and (EEF) with more strong energy consumption required by tasks over interval $[\text{est}_\Omega,\text{lct}_\Omega)$. The energetic extended edge-finding combines the techniques of edge-finding, extended edge-finding and partially the energetic reasoning. The new adjustment rule is obtained by substituting the energy e_Θ by $W(\text{est}_\Theta,\text{lct}_\Theta)$ in the (extended) edge-finding adjustment rule.

3.2. Algorithm

The algorithm proposed in this Section is an energetic version of the edge-finding and the extended edge-finding algorithm of [7,8,11] respectively. The time bounds of task intervals used in the edge-finding (resp. the extended edge-finding) for detection and adjustment are used in an energetic reasoning. The new algorithm runs in $\mathcal{O}(n^3)$ since each of the edge-finding and extended edge-finding algorithms are quadratic [7,8,11].

The algorithm is broken into two parts to make it more digest. The first part is an energetic variant of the edge-finding algorithm of [7,8] while the second part is the one of the extended edge-finding algorithm of [11] (adjustments missing by the edge-finding and detect with the extended edge-finding).

3.2.1. First Part

This part consists only in adding an inner loop in the edge-finding algorithm of [7,8] after detection of time bounds, to recompute the energy of each task intervals Ω plus the partial contribution of the rest of tasks $T\setminus\Omega$. It is presented in **Algorithm 1** where:

1) The first outer loop (line 3) selects, in the order of non-decreasing deadlines, the tasks $k\in T$ which form the possible upper bounds of the task intervals.

2) The first inner loop (line 5) selects the tasks $i\in T$ that includes the possible lower bounds for the task intervals, in non-increasing order by release date. If $\text{lct}_i\le\text{lct}_k$, then the energy and density of $\Omega_{i,k}$ are calculated; if the new density is higher than $\Omega_{\rho(k,i),k}$ where the task $\rho(k,i)$ is specified in Definition 6, then $\rho(k,i)$ becomes i (line 9). If $\text{lct}_i>\text{lct}_k$, then if the current $\rho(k,i)$ satisfies $\text{est}_{\rho(k,i)}<\text{lct}_k$ and $\text{ect}_i>\text{est}_{\rho(k,i)}$ (line 12), then the energy required by interval $[\text{est}_{\rho(k,i)},\text{lct}_k)$ is computed in the inner loop of line 13. The condition of line 15 checks if the interval $[\text{est}_{\rho(k,i)},\text{lct}_k)$ is overloaded (see inequality (12)) and if the condition of line 17 is fulfilled, then the potential update $Dupd_i$ of the release date of i is calculated (line 18), based on the current interval $[\text{est}_{\rho(k,i)},\text{lct}_k)$. This potential update is stored only if it is greater than the previous potential update value calculated for this task using the maximum density time bounds. The release date of task i is updated at line 20 when the energetic reasoning condition of line 19 is fulfilled (see inequality (ER)).

3) The second inner loop (line 23) selects i in non-decreasing order by release date. The energies stored in the previous loop (line 21) are used to compute the slack of the current interval $\Omega_{i,k}$. If the slack is lower than that of $\Omega_{\tau(k,i),k}$ where $\tau(k,i)$ is specified in Definition 4, then $\tau(k,i)$ becomes i (see line 25). For any task i with a deadline greater than lct_k, if the current $\tau(k,i)$ satisfies $\text{est}_{\tau(k,i)}<\text{lct}_k$ (line 28), then the energy required by interval $[\text{est}_{\tau(k,i)},\text{lct}_k)$ is computed in the inner loop of line 29. The condition of line

Require: T is an array of tasks
Ensure: A lower bound est'_i is computed for the release date of each task

1 **for** $i \in T$ **do**
2 $est'_i := est_i$, $Dupd_i := -\infty$, $SLupd_i := -\infty$
3 **for** $k \in T$ by non-decreasing deadline **do**
4 $Energy := 0$, $\max Energy := 0$, $est_\rho := -\infty$
5 **for** $i \in T$ by non-increasing release dates **do**
6 **if** $lct_i \le lct_k$ **then**
7 $Energy := Energy + e_i$
8 **if** $\left(\frac{Energy}{lct_k - est_i} > \frac{\max Energy}{lct_k - est_\rho}\right)$ **then**
9 $\max Energy := Energy$, $est_\rho := est_i$
10 **else**
11 $a := est_\rho$, $b := lct_k$, $W(a,b) := 0$
12 **if** $(b > a \wedge ect_i \ge a)$ **then**
13 **for** $j \in T$ **do**
14 $W(a,b) := W(a,b) + W(a,b,j)$
15 **if** $(C(b-a) < W(a,b))$ **then**
16 **fail** (No solution exists)
17 **if** $(W(a,b) - W(a,b,i) - (C - c_i)\cdot(b-a) > 0)$ **then**
18 $Dupd_i := \max\left(Dupd_i, a + \left\lceil \frac{W(a,b) - W(a,b,i) - (C-c_i)\cdot(b-a)}{c_i} \right\rceil\right)$
19 **if** $(W(a,b) - W(a,b,i) + W_i(a,b,i) > C(b-a))$ **then**
20 $est'_i := \max(est'_i, Dupd_i)$
21 $E_i := Energy$
22 $\min SL := +\infty$, $est_\tau := lct_k$
23 **for** $i \in T$ by non-decreasing release date **do**
24 **if** $(C(lct_k - est_i) - E_i < \min SL)$ **then**
25 $est_\tau := est_i$, $\min SL := C(lct_k - est_\tau) - E_i$
26 **if** $(lct_i > lct_k)$ **then**
27 $a := est_\tau$, $b := lct_k$, $W(a,b) := 0$
28 **if** $(b > a)$ **then**
29 **for** $j \in T$ **do**
30 $W(a,b) := W(a,b) + W(a,b,j)$
31 **if** $(C(b-a) < W(a,b))$ **then**
32 **fail** (No solution exists)
33 **if** $(W(a,b) - W(a,b,i) - (C - c_i)\cdot(b-a) > 0)$ **then**
34 $SLupd_i := \max\left(SLupd_i, a + \left\lceil \frac{W(a,b) - W(a,b,i) - (C-c_i)\cdot(b-a)}{c_i} \right\rceil\right)$
35 **if** $(W(a,b) - W(a,b,i) + W_i(a,b,i) > C(b-a))$ **then**
36 $est'_i := \max(est'_i, SLupd_i, Dupd_i)$

Algorithm 1. Energetic extended edge finding algorithm in $\mathcal{O}(n^3)$ time.

31 checks if the interval $\left[\text{est}_{\tau(k,i)}, \text{lct}_k\right)$ is overloaded (see inequality (12)). If the condition of line 33 is fulfilled, then the potential update $SLupd_i$ of the release date of i is calculated (line 34), based on the current interval $\left[\text{est}_{\tau(k,i)}, \text{lct}_k\right)$. This potential update is stored only if it is greater than the previous potential update value calculated for this task using the minimum slack time bounds. If the energetic reasoning condition is fulfilled at line 35 (see inequality (ER)), then the release date of task i is updated at line 36.

4) At the next iteration of this first outer loop, $\rho(k,i)$ and $\tau(k,i)$ are re-initialized.

This first algorithm (**Algorithm 1**) corresponds to the energetic edge-finding algorithm.

3.2.2. Second Part

This part is the energetic version of the extended edge-finding algorithm of [11]. It consist in adding an inner loop in the extended edge-finding algorithm of [11] after detection of time bounds, to recompute the energy of each task intervals Ω plus the partial contribution of the rest of tasks $T \setminus \Omega$. The corresponding algorithm is presented in **Algorithm 2** where:

1) The outer loop (line 37) iterates through the tasks $j \in T$ forming the possible lower bounds of the task intervals.

2) The inner loop (line 39) selects the tasks $i \in T$ that comprise the possible upper bounds for the task intervals, in non-decreasing order of deadlines. If $\text{est}_j \le \text{est}_i$, then the energy and the slack of $\Omega_{j,i}$ are calculated. The slack is then compared to the slack of $\Omega_{j,\delta(j,i)}$ (see line 42) where $\delta(j,i)$ is specified in Definition 7. If the new slack is higher, $\delta(j,i)$ becomes i (line 43). If $\text{est}_j > \text{est}_i$ (line 44), then if the current $\delta(j,i)$ satisfies

37 **for** $j \in T$ **do**
38 $Energy := 0$, $\max Energy := 0$, $lct_\delta := est_j$
39 **for** $i \in T$ (T sorted by non-decreasing deadlines) **do**
40 **if** $(est_j \le est_i)$ **then**
41 $Energy := Energy + e_i$
42 **if** $(C(lct_i - est_j) - Energy \le C(lct_\delta - est_j) - \max Energy)$ **then**
43 $\max Energy := Energy$, $lct_\delta := lct_i$
44 **else**
45 $a := est_j$, $b := lct_\delta$, $W(a,b) := 0$
46 **if** $(b > a \wedge ect_i \ge a)$ **then**
47 **for** $k \in T$ **do**
48 $W(a,b) := W(a,b) + W(a,b,k)$
49 **if** $(C(b-a) < W(a,b))$ **then**
50 **fail** (No solution exists)
51 **if** $(W(a,b) - W(a,b,i) + W_i(a,b,i) > C(b-a))$ **then**
52 $est'_i := \max\left(est'_i, a + \left\lceil \frac{W(a,b) - W(a,b,i) - (C-c_i)\cdot(b-a)}{c_i} \right\rceil\right)$
53 **for** $i \in T$ **do**
54 $est_i := est'_i$

Algorithm 2. Energetic extended edge finding algorithm in $\mathcal{O}(n^3)$ time.

$\text{lct}_{\delta(j,i)} > \text{est}_j$ and $\text{ect}_i > \text{est}_j$ (line 46), then the energy required by interval $\left[\text{est}_j, \text{lct}_{\delta(j,i)}\right)$ is computed in the inner loop of line 47. The condition of line 49 checks if the interval $\left[\text{est}_{\tau(k,i)}, \text{lct}_k\right)$ is overloaded (see inequality (12)). If the energetic reasoning condition is fulfilled at line 51 (see inequality (ER)), then the release date of task i is updated at line 52 based on the current potential update value determined by interval $\left[\text{est}_j, \text{lct}_{\delta(j,i)}\right)$.

3) At the next iteration of this outer loop, $\delta(j,i)$ is re-initialized.

Merging **Algorithms 1** and **2**, the corresponding algorithm title "EnEEF" is an energetic extended edge-finding algorithm.

3.3. Some Properties of EnEEF

The energetic extended edge-finding algorithm EnEEF is compared to the conjunction of the edge-finding and the extended edge-finding. It is proved that, this algorithm subsumes the conjunction of edge-finding and extended edge-finding algorithm.

3.3.1. The Energetic Extended Edge-Finder EnEEF Performs Some Additional Adjustments Missing by (Extended) Edge-Finding

Using the CuSP instance of **Figure 1**, it is found that the energetic extended edge-finder EnEEF performs additional adjustments missing by (extended) edge-finding. Indeed, the application of the (extended) edge-finding algorithm on the CuSP instance of **Figure 1** doesn't produce any adjustment whereas the application of our energetic extended edge finder EnEEF permits to update the release date of task *A* from 1 to 5. When the task B (resp. C or D) is considered at the outer loop of line 3, the bounds of the task intervals of maximum density are $[3,6)$ and the condition of line 19 holds for $a=3$ and $b=6$. It follows that

$$\begin{aligned} &W(a,b)-W(a,b,A)+W_l(a,b,A) \\ &=8+2=10>9=3\times(6-3)=C\cdot(b-a) \end{aligned}$$

and

$$\begin{aligned} &W(a,b)-W(a,b,A)-(C-c_A)(b-a) \\ &=8-0-(3-1)(6-3)=2. \end{aligned}$$

Therefore, the release date est_A is updated to

$$\begin{aligned} \text{est}_A &\geq a+\left\lceil \frac{W(a,b)-W(a,b,A)-(C-c_A)(b-a)}{c_A} \right\rceil \\ &=3+2=5. \end{aligned}$$

The reader can check that no propagation is performed by the timetable edge-finding rule of [12] and the timetable extended edge-finding rule of [13] on the CuSP instance of **Figure 1**.

3.3.2. The Energetic Extended Edge-Finder EnEEF Subsumes the Edge-Finding Algorithm

Theorem 1 *The energetic extended edge-finder EnEEF subsumes the edge-finding algorithm of* [7,8].

Proof. It is important to prove that the edge-finding algorithm can not propagate anything when the energetic extended edge-finder EnEEF reaches the fix point.

By contradiction: assume that the energetic extended edge-finder EnEEF reaches the fix point, and that however, the edge-finding algorithm can propagate *i.e.*, there are i, Ω and Θ, $\Theta \subseteq \Omega$ such that (EF) or (EF1) holds and (Upd) improves est_i using Θ. It will be prove (by contradiction) that the energetic extended edge-finding algorithm EnEEF can update the time bounds of task i.

1) The rule (EF1) holds. In this case, two subcases are considered:

(a) If $\text{est}_\Theta \leq \text{est}_i$, then the energy contribution of task i in the time interval $\left[\text{est}_\Theta, \text{lct}_\Theta\right)$ is $c_i\left(\text{lct}_\Theta - \text{est}_i\right)$ since $\text{ect}_i \geq \text{lct}_\Theta$. The inequality

$$\text{est}_\Theta + \frac{1}{c_i}\text{rest}(\Theta, c_i) > \text{est}_i \tag{16}$$

holds since the release date of task i is updated using the rule (Upd) and it is algebraically equivalent to

$$e_\Theta + c_i\left(\text{lct}_\Theta - \text{est}_i\right) > C\left(\text{lct}_\Theta - \text{est}_\Theta\right). \tag{17}$$

Let $k \in T$ be a task such that $lct_k := lct_\Theta$. According to [7,8], Θ is the task intervals of minimum slack (Definition 4) and it follows

$$\begin{aligned} C\left(\text{lct}_k - \text{est}_{\tau(k,i)}\right) - e_{\Omega_{\tau(k,i),k}} &\leq C\left(\text{lct}_\Theta - \text{est}_\Theta\right) - e_\Theta \\ &< c_i\left(\text{lct}_\Theta - \text{est}_i\right). \end{aligned} \tag{18}$$

When the task k is considered in the outer loop of line 3, in the second inner loop of line 23 the condition

$$W(a,b) - W(a,b,i) + W_l(a,b,i) > C\cdot(b-a) \tag{19}$$

is detected at line 35 since $W(a,b) - W(a,b,i) \geq e_{\Omega_{\tau(k,i),k}}$ and $W_l(a,b,i) = c_i\left(\text{lct}_\Theta - \text{est}_i\right)$ where $a := \text{est}_{\tau(k,i)}$ and $b := \text{lct}_k$ and the adjustment follows.

(b) If $\text{est}_i < \text{est}_\Theta$, then the energy contribution of task i in the time interval $\left[\text{est}_\Theta, \text{lct}_\Theta\right)$ is $c_i\left(\text{lct}_\Theta - \text{est}_\Theta\right)$ since $\text{ect}_i \geq \text{lct}_\Theta$. The inequality

$$\text{rest}(\Theta, c_i) > 0 \tag{20}$$

holds since the release date of task i is updated using the set Θ and the rule (Upd) and it is algebraically equivalent to

$$e_\Theta + c_i\left(\text{lct}_\Theta - \text{est}_\Theta\right) > C\left(\text{lct}_\Theta - \text{est}_\Theta\right). \tag{21}$$

Let $k \in T$ be a task such that $lct_k := lct_\Theta$. According to [7,8], Θ is the task intervals of maximum density

(Definition 6) and it follows

$$\frac{e_{\Omega_{\rho(k,i),k}}}{\mathrm{lct}_k - \mathrm{est}_{\rho(k,i)}} \geq \frac{e_\Theta}{\mathrm{lct}_\Theta - \mathrm{est}_\Theta} > C - c_i. \quad (22)$$

Therefore, it appears that

$$e_{\Omega_{\rho(k,i),k}} + c_i\left(\mathrm{lct}_k - \mathrm{est}_{\rho(k,i)}\right) > C\left(\mathrm{lct}_k - \mathrm{est}_{\rho(k,i)}\right). \quad (23)$$

When the task k is considered in the outer loop of line 3, in the first inner loop of line 5 the condition

$$W(a,b) - W(a,b,i) + W_l(a,b,i) > C\cdot(b-a) \quad (24)$$

is detected at line 19 since $W(a,b) - W(a,b,i) \geq e_{\Omega_{\rho(k,i),k}}$ and $W_l(a,b,i) = c_i\left(\mathrm{lct}_k - \mathrm{est}_{\rho(k,i)}\right)$ where $a := \mathrm{est}_{\rho(k,i)}$ and $b := \mathrm{lct}_k$ and the adjustment follows.

2) The rule (EF) holds: Let $k \in T$ be a task such that $\mathrm{lct}_k := \mathrm{lct}_\Omega$. According to [7,8], Ω is the task intervals of minimum slack (Definition 4) and it follows

$$C\left(\mathrm{lct}_k - \mathrm{est}_{\tau(k,i)}\right) - e_{\Omega_{\tau(k,i),k}} < e_i. \quad (25)$$

When the task k is considered in the outer loop of line 3, in the second inner loop of line 23 the condition

$$W(a,b) - W(a,b,i) + W_l(a,b,i) > C\cdot(b-a) \quad (26)$$

is detected at line 35 since $W(a,b) - W(a,b,i) \geq e_{\Omega_{\tau(k,i),k}}$ and $W_l(a,b,i) = e_i$ where $a := \mathrm{est}_{\tau(k,i)}$ and $b := \mathrm{lct}_k$ and the adjustment follows using the potential update values previously computed.■

According to this theorem, the first outer loop of line 3 ensures us that the fix point of the edge-finding rule will always be reached. This result is used to demonstrate that our algorithm dominates the extended edge-finding rule. In the following theorem, it is proved that, at the fix point of the edge-finding rule, if the extended edge-finding rule detects the relation $\Omega \ll i$ for a set of tasks Ω and a task i then the set Ω can help to compute the potential updated value of est_i.

3.3.3. The Energetic Extended Edge-Finder EnEEF Subsumes the Extended Edge-Finding Algorithm

Theorem 2 [11] *Let $\Omega \subset T$ be a set of tasks and $i \in T \setminus \Omega$ be a task of an E-feasible CuSP. At the fix point of the edge-finding rule, if the extended edge-finding rule detects $\Omega \ll i$ then*

$$\mathrm{rest}(\Omega, c_i) > 0 \text{ and } \mathrm{est}_\Omega + \left\lceil \frac{1}{c_i}\mathrm{rest}(\Omega, c_i) \right\rceil > \mathrm{est}_i.$$

Using theorem 2, it is derived the following theorem:

Theorem 3 *The energetic extended edge-finder EnEEF subsumes the extended edge-finding algorithm of* [11].

Proof. As in Theorem 1, it is prove that the extended edge-finding algorithm can not propagate anything when the energetic extended edge-finder EnEEF reaches the fix point.

The contradiction is used: assume that the energetic extended edge-finder EnEEF reaches the fix point, and that however, the extended edge-finding algorithm can propagate *i.e.*, there are i, Ω and Θ, $\Theta \subseteq \Omega$ such that (EEF) holds and (Upd) improves est_i using Θ. It will be prove (by contradiction) that the energetic extended edge-finding algorithm EnEEF can update the time bounds of task i.

Let $j \in T$ be a task such that $\mathrm{est}_j := \mathrm{est}_\Omega$. According to [11], Ω is the task intervals of minimum slack (Definition 7) and it follows

$$\begin{aligned} C\left(\mathrm{lct}_{\delta(j,i)} - \mathrm{est}_j\right) - e_{\Omega_{j,\delta(j,i)}} &\leq C\left(\mathrm{lct}_\Omega - \mathrm{est}_\Omega\right) - e_\Omega \\ &< c_i\left(\mathrm{ect}_i - \mathrm{est}_\Omega\right). \end{aligned} \quad (27)$$

When the task j is considered in the outer loop of line 37, in the inner loop of line 39 the condition

$$W(a,b) - W(a,b,i) + W_l(a,b,i) > C\cdot(b-a) \quad (28)$$

is detected at line 51 since $W(a,b) - W(a,b,i) \geq e_{\Omega_{j,\delta(j,i)}}$ and $W_l(a,b,i) = c_i\left(\mathrm{ect}_i - \mathrm{est}_\Omega\right)$ where $a := \mathrm{est}_j$ and $b := lct_{\delta(j,i)}$. According to Theorem 1, the fix point of the edge-finding rule is reached and the Theorem 3 shows that the task intervals used for detection can also be used to perform the adjustment. Therefore, an adjustment is performed at line 52 since this adjustment is justified by the condition of line 51.■

4. Experimental Results

For our experiments, it is consided the resource-constrained project scheduling problems (RCPSP). A RCPSP consists of a set of resources of finite capacities, a set of tasks of given processing times, an acyclic network of precedence constraints between tasks, and a horizon (a deadline for all tasks). Each task requires a fixed amount of each resource over its execution time. The problem is to find a start time assignment for every task satisfying the precedence and resource capacity constraints, with a makespan (*i.e.*, the time at which all tasks are completed) at most equal to the horizon. The cumulative scheduling problem (CuSP) is a sub-problem of the RCPSP, where precedence constraints are relaxed and a single resource is considered at a time; both problems are NP-complete [10].

Tests were performed on the RCPSP single-mode J30, J60 and J90 test sets of the well-established benchmark library PSPLib [9] as well as on the library of [10] (BL). The data sets J30, J60 and J90 consist of 480 instances of 30, 60 and 90 tasks respectively, while BL consists of 40

instances of 20 and 25 tasks respectively. Each instance from the PSPLib sets includes tasks to be scheduled over 4 resources, while instances from the BL suite share 3 resources.

Starting with the provided horizon as an upper bound, each instance of problem is modeled as an instance of Constraint Satisfaction Problem (CSP); variables are start times of tasks and they are constrained by precedence graph (*i.e*., precedence relations between pairs of tasks were enforced with linear constraints) and resource limitation (*i.e*., each resource was modeled with a single CUMULATIVE constraint [2]).

Dynamic branching schemes are the most used branching strategy in CP, as they typically result in smaller search trees. However, when comparing filtering algorithms of differing pruning strengths, dynamic branching can be misleading: in some cases the domain resulting from weaker pruning may result in a choice point yielding a smaller subtree, and hence a faster solution. In order to minimize the effect of differing pruning strength on the shape of the search trees, it is consided two branching models:

1) For dynamic branching, variable selection was based on the minimum of domain size, divided by degree (*i.e*., the number of propagators depending on the variable). Ties were broken by selecting the task with the minimum latest start time; values were taken from the smallest range for domains with multiple ranges, or the lesser half of the domain when only one range existed [23].

2) For static branching, it is selected the first unassigned variable, and the smallest value in the domain.

Tests were performed on a Pentium(R) Dual-Core processor, CPU 2.70 ghz, 1.96 GB of RAM. The implementation was done in c++ using the Gecode 3.7.3 [23] constraint solver. For each benchmark instance, it is used branch and bound search to minimize the makespan, stopping only when the optimum solution was found. Each test was run three times, with the best result reported; any search taking more than 300 seconds was counted as a failure.

Two filtering algorithms for different configurations of the global constraint CUMULATIVE have been considered.

1) The first CUMULATIVE propagator noted "EEF" for tasks of fixed duration is a sequence of three filters: the $\mathcal{O}(n^2)$ extended edge-finding algorithm from [11], the overload checking from [19] and timetabling algorithm from [1].

2) The second propagator noted "EnEEF" is a modified version of "EEF" that substitutes the extended edge-finding algorithm from [11] for the $\mathcal{O}(n^3)$ energetic extended edge-finder of this paper (EnEEF).

4.1. Dynamic Branching

Table 1 reports the results for all instances from the test sets BL, J30, J60 and J90 that were solved by at least one propagator using dynamic branching. There were 40, 392, 341 and 324 for BL, J30, J60 and J90 respectively. In this table, the line "solve" reports the number of instances in which each algorithm found the optimal solution, did so in the fastest time "time", and generated the smallest search tree "nodes", using dynamic branching. Line "Av.time" reports the average CPU time (in second) used to reach the optimal solution while line "Av.node" denotes the average number of nodes reported on instances solved by both propagators. Both propagators solve the same number of instances in each test sets. As can be observed form **Figure 2**, using dynamic branching EnEEF performs the strongest among the two algorithms on instances less than 30 tasks. Unfortunately, the use of a dynamic branching scheme appears to hide the domination of EnEEF on EEF. On instances with more than 30 tasks, EnEEF requires more time for small reduction of tree search.

Here, on BL set (reputed to be highly cumulative [10]), 87.5% of instances were solved by EnEEF with better running time and 85% of instances were solved with smallest search tree. On J30, J60 and J90 instances (reputed to be highly disjunctive [10]), the performance of the EnEEF is reduced. This result confirms that of Baptiste *et al*. [1] concerning the usage of the energetic reasoning on tightness instances.

Table 1. Number of instances in which each algorithm found the optimal solution (solve), did so in the fastest time (time), and generated the smallest search tree (nodes), using dynamic branching. Average runtime (Av.time) and nodes (Av.node) count on instances where both solvers can found the optimal solution are considered.

	BL20		BL25		J30		J60		J90	
	EEF	EnEEF	EEF	EnEEF	EEF	EnEEF	EEF	EnEEF	EEF	EnEEF
solve	20	20	20	20	392	392	341	341	324	324
time	5	15	0	20	289	101	319	21	319	5
node	0	14	0	20	5	59	3	21	4	10
Av.time	4.04	3.44	35.60	15.00	5.64	6.59	1.93	2.10	1.08	2.94
Av.node	27073	20410	166389	86930	19462	17302	3792	3161	2031	2050

4.2. Static Branching

In **Table 2**, lines "solve", "time", "node", "Av.time" and "Av.node" have the same meaning as in **Table 1**. For static branching scheme, an instance of BL25 was solved only by the propagator EnEEF in the time available. Two other instances for J30 and J60 respectively was soled by EEF only in the time available. The tests show that the EEF propagator is faster in a large majority of test instances but the EnEEF remains the best on BL test set. As in **Table 1**, the average running time of EEF is more than two times the one of EnEEF on the BL25 test set. The same remark can be observed on the average of nodes count of the different propagators on BL25 for both static and dynamic branching scheme. **Figure 3** compares (left) the proportional runtime of EnEEF over

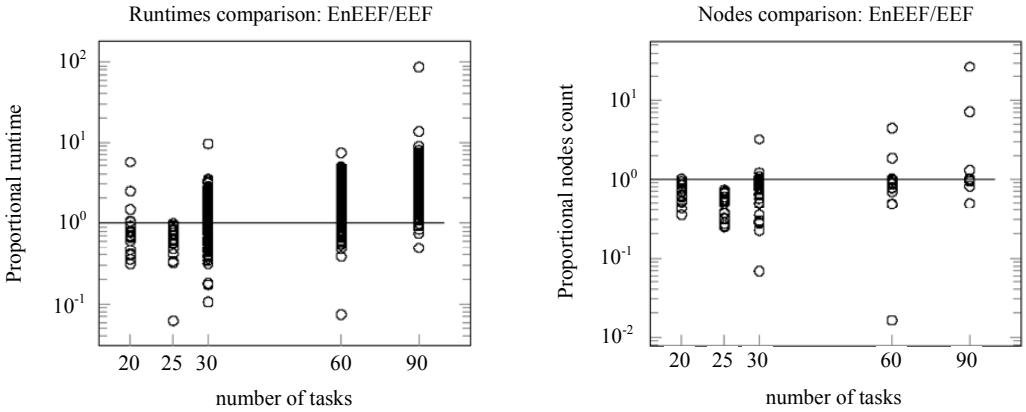

Figure 2. Comparison of (left) proportional runtimes of EnEEF over EEF and (right) proportional nodes count when using dynamic branching, sorted by number of tasks of instances.

Table 2. Number of instances in which each algorithm found the optimal solution (solve), did so in the fastest time (time), and generated the smallest search tree (nodes), using static branching. Average runtime (Av.time) and nodes (Av.node) count on instances were both solvers can found the optimal solutions are considered.

	BL20		BL25		J30		J60		J90	
	EEF	EnEEF	EEF	EnEEF	EEF	EnEEF	EEF	EnEEF	EEF	EnEEF
solve	18	18	17	18	359	358	326	325	325	325
time	3	15	3	14	280	78	304	20	322	3
node	0	17	0	17	0	52	0	24	0	11
Av.time	3.69	2.50	28.86	14.11	4.31	5.56	1.37	1.56	0.43	1.36
Av.node	28818	13291	149298	53120	12465	11150	2616	2313	195	190

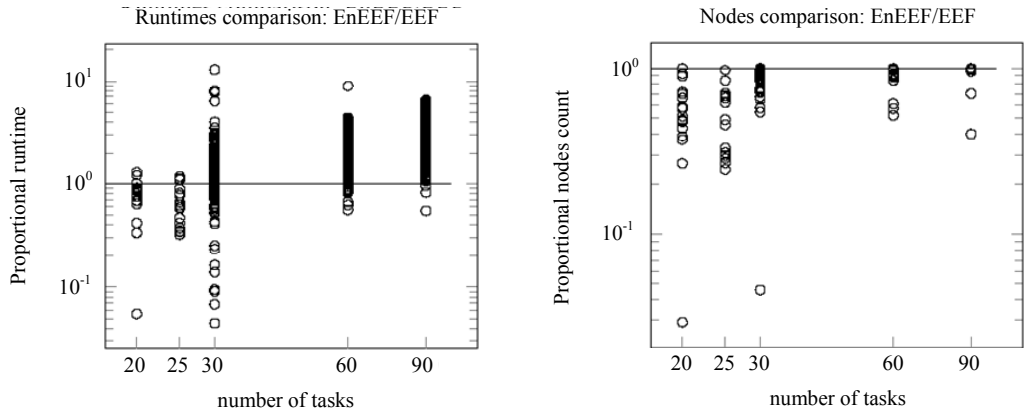

Figure 3. Comparison of (left) proportional runtimes of EnEEF over EEF and (right) proportional nodes count when using static branching, sorted by number of tasks of instances.

EEF and (right) the proportional nodes count when using static branching, sorted by number of tasks of instances. It appears that the propagator EnEEF subsumes the EEF in almost all test instances. On tests set J30, J60 and J90, EnEEF need more time for a weak reduction of the tree search on instances solved by the propagators.

5. Conclusion

In this paper, it is presented a new filtering algorithm for cumulative resource, hybridization of the extended edge-finding rule and the energetic reasoning. The new algorithm is stronger than the extended edge-finding algorithm, but weaker than the energetic reasoning and runs in $\mathcal{O}(n^3)$ where n is the number of tasks sharing the resource. In practice, it is a good trade-off between the filtering power and the running time. Experimental results demonstrate that on a standard benchmark suite, our new algorithm reduces substantially more number of nodes—thus the tree search—than the extended edge-finding algorithm on instances where the number of tasks is less than 30. The time complexity of this algorithm remains too high. Our future work will focus on the reduction of the complexity of this algorithm from $\mathcal{O}(n^3)$ to $\mathcal{O}(n^2 \log n)$ using a Θ-tree data structures.

6. Acknowledgements

The authors would like to thank Pierre Ouellet and Claude-Guy Quimper for providing their paper. We also thank anonymous referees sincerely for their constructive and insightful feedback.

REFERENCES

[1] P. Baptiste, C. Le Pape and W. P. M. Nuijten, "Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems," Springer, Berlin, 2001. http://dx.doi.org/10.1007/978-1-4615-1479-4

[2] A. Aggoun and N. Beldiceanu, "Extending CHIP in Order to Solve Complex Scheduling and Placement Problems," *Mathematical and Computer Modelling*, Vol. 17, No. 7, 1993, pp. 57-73. http://dx.doi.org/10.1016/0895-7177(93)90068-A

[3] P. Baptiste, "Resource Constraints for Preemptive and Non-Preemptive Scheduling," Master 2 in Computer Science and Operation Research, University of Paris VI, Institut Blaise Pascal, 1995.

[4] L. Mercier and P. Van Hentenryck, "Edge Finding for Cumulative Scheduling," *INFORMS Journal on Computing*, Vol. 20, No. 1, 2008, pp. 143-153. http://dx.doi.org/10.1287/ijoc.1070.0226

[5] S. F. Betmbe, "Energetic Edge Finder: Propagator of cUmulative Resource Constraints," Master 2 in Computer Science, University of Yaounde 1, 2012.

[6] R. Kameugne and L. P. Fotso, "Energetic Edge-Finder for Cumulative Resource Constraint," *Proceeding of CPDP* 2009 *Doctoral Program*, Lisbone, 2009, pp. 54-63.

[7] R. Kameugne, L. P. Fotso, J. Scott and Y. Ngo-Kateu, "A Quadratic Edge-Finding Filtering Algorithm for Cumulative Resource Constraints," In: J. H. M. Lee, Ed., *CP* 2011—*Principles and Practice of Constraint Programming*, Springer, Berlin, 2011, pp. 478-492. http://dx.doi.org/10.1007/978-3-642-23786-7_37

[8] R. Kameugne, L. P. Fotso, J. Scott and Y. Ngo-Kateu, "A Quadratic Edge-Finding Filtering Algorithm for Cumulative Resource Constraints," Extended Version of the CP 2011 Paper, Constraints, Forthcoming, 2013.

[9] "PSPLib—Project Scheduling Problem Library," Online, 2012. http://129.187.106.231/psplib/

[10] P. Baptiste and C. Le Pape, "Constraint Propagation and Decomposition Techniques for Highly Disjunctive and Highly Cumulative Project Scheduling Problems," *Constraints*, Vol. 5, No. 1, 2000, pp. 119-139. http://dx.doi.org/10.1023/A:1009822502231

[11] R. Kameugne, L. P. Fotso and J. Scott, "A Quadratic Extended Edge-Finding Filtering Algorithm for Cumulative Resource Constraints," *International Journal of Planning and Scheduling*, Forthcoming, 2013.

[12] P. Vilm, "Timetable Edge Finding Filtering Algorithm for Discrete Cumulative Resources," In: T. Achterberg and J. C. Beck, Eds., *CPAIOR* 2011—*Integration of AI and OR Techniques in Constraint Programming*, Springer, Berlin, 2011, pp. 230-245. http://dx.doi.org/10.1007/978-3-642-21311-3_22

[13] P. Ouellet and C.-G. Quimper, "Time-Table Extended-Edge-Finding for the Cumulative Constraint," *Proceeding of the* 19*th International Conference on Principles and Practice of Constraint Programming*, *LNCS*, 2013, Springer, Berlin, Vol. 8124, pp. 562-577.

[14] R. Kameugne and L. P. Fotso, "A Cumulative Not-First/Not-Last Filtering Algorithm in $O(n^2 \log n)$," *Indian Journal of Pure and Applied Mathematics*, Vol. 44, No. 1, 2013, pp. 95-115. http://dx.doi.org/10.1007/s13226-013-0005-z

[15] A. Schutt and A. Wolf, "A New $O(n^2 \log n)$ Not-First/Not-Last Pruning Algorithm for Cumulative Resource Constraints," In: D. Cohen, Ed., *CP* 2010—*Principles and Practice of Constraint Programming*, Springer, Berlin, 2010, pp. 445-459. http://dx.doi.org/10.1007/978-3-642-15396-9_36

[16] T. Berthold, S. Heinz and J. Schulz, "An Approximative Criterion for the Potential of Energetic Reasoning," In: A. Marchetti-Spaccamela and M. Segal, Eds., *Theory and Practice of Algorithms in* (*Computer*) *Systems*, Springer, Berlin, 2011, pp. 229-239. http://dx.doi.org/10.1007/978-3-642-19754-3_23

[17] S. Heinz and J. Schulz, "Explanations for the Cumulative Constraint: An Experimental Study," In: P. M. Pardalos and S. Rebennack, Eds., *Experimental Algorithms*, Springer, Berlin, 2011, pp. 400-409. http://dx.doi.org/10.1007/978-3-642-20662-7_34

[18] A. Schutt, T. Feydy and P. J. Stuckey, "Explaining Time-Table-Edge-Finding Propagation for the Cumulative Resource Constraint," CPAIOR 2011—Integration of AI

and OR Techniques in Constraint Programming, 2013, pp. 234-250.

[19] P. Vilm, "Max Energy Filtering Algorithm for Discrete Cumulative Resources," In: W. J. van Hoeve and J. N. Hooker, Eds., *CPAIOR* 2009—*Integration of AI and OR Techniques in Constraint Programming*, Springer, Berlin, 2009, pp. 294-308. http://dx.doi.org/10.1007/978-3-642-01929-6_22

[20] A. Wolf and G. Schrader, "*O*(*n*log*n*) Overload Checking for the Cumulative Constraint and Its Application," In: M. Umeda, A. Wolf, O. Bartenstein, U. Geske, D. Seipel and O. Takata, Eds., *INAP* 2005—*Applications of Declarative Programming for Knowledge Management*, Springer, Berlin, 2006, pp. 88-101. http://dx.doi.org/10.1007/11963578_8

[21] P. Vilm, "Edge Finding Filtering Algorithm for Discrete Cumulative Resources in *O(kn*log*n)*," In: I. P. Gent, Ed., *CP* 2009—*Principles and Practice of Constraint Programming*, Springer, Berlin, 2009, pp. 802-816. http://dx.doi.org/10.1007/978-3-642-04244-7_62

[22] Y. Caseau and F. Laburthe, "Improved CLP Scheduling with Task Intervals," In: P. Van Hentenryck, Ed., *ICLP* 1994—*Logic Programming*, MIT Press, Cambridge, 1994, pp. 369-383.

[23] Gecode, 2012. http://www.gecode.org