Horizontally Elastic Edge-Finder Algorithm for Cumulative Resource Constraint Revisited

Sévérine Fetgo Betmbe, Clémentin Tayou Djamegni

Horizontally Elastic Edge-Finder Algorithm for Cumulative Resource Constraint Revisited

Sévérine Fetgo Betmbe, Clémentin Tayou Djamegni

Department of Mathematics and Computer Science
Faculty of Sciences
University of Dschang
Cameroon
severine.fetgo@gmail.com
dtayou@gmail.com

ABSTRACT. The success of the constraint programming on scheduling problems comes from the low complexity and power of propagators. The *Profile* data structure recently introduced by Gingras and Quimper in [1] when applied to the edge finder rule for cumulative resource constraint (which we call *horizontally elastic edge finder*) has improved the filtering power of this rule. In this paper, the algorithm proposed by Gingras and Quimper [1] is revisited . It is proved that the detection phase partially use the data structure *Profile* and a new formulation of the horizontally elastic edge finder rule is proposed. Similar to [1], a $\mathcal{O}(kn^2)$ algorithm (where $k \leq n$ represents the number of distinct capacities required by tasks and n the number of tasks sharing the resource) is proposed for the new rule. Experimental results on cumulative instances of resource constrained project scheduling problems (RCPSPs) from suites benchmarks highlight that using this new algorithm reduces the number of backtracks for a majority of instances with a marginal augmentation of the running time.

RÉSUMÉ. Le succès de la programmation par contraintes sur les problèmes d'ordonnancement vient de la faible complexité et de la puissance des propagateurs. La structure de données *Profile* récemment introduite par Gingras et Quimper[1] appliquée à la règle edge finder pour la contrainte de ressource en ordonnancement cumulatif (que nous appelons *horizontally elastic edge finder*) a amélioré la puissance de filtrage de cette règle. Dans cet article, l'algorithme proposé par Gingras et Quimper [1] est revisité. Il est prouvé que la phase de détection de cet algorithme utilise partiellement la structure de donnée et une nouvelle reformulation de la règle "horizontally elastic edge finder" est proposée. De manière similaire à [1], un algorithme de complexité $\mathcal{O}(kn^2)$ (où $k \leq n$ représente le nombre de différentes demandes en ressource et n le nombre de tâches partageant la ressource) est proposé pour la nouvelle règle. Les résultats expérimentaux sur les instances fortement cumulatives des problèmes d'ordonnancement de projet à moyen limité (RCPSPs) de la littérature montrent qu'en utilisant le nouvel algorithme, il y a réduction du nombre de backtracks sur la large majorité des instances pour une faible augmentation du temps d'exécution.

KEYWORDS : Constraint programming, Cumulative scheduling, Edge finder, "Profile" data structure, Horizontally elastic scheduling, RCPSP

MOTS-CLÉS : Programmation par contraintes, Ordonnancement cumulatif, Edge finder, Structure de donnée "Profile", Ordonnancement horizontallement élastique, RCPSP

1. Introduction

The CUMULATIVE [2] constraint models the problems where a limited number of tasks can executed simultaneously. Many filtering algorithms are embedded in this constraint among which edge-finder [3] and timetabling [4] are generally the most used. There exists many others such as not-first/not-last [5], energetic reasoning [6]. A constraint CUMULATIVE is used to solve the Cumulative Scheduling Problems (CuSP). It is defined by the given of a set of tasks T to be executed on a resource of capacity C. Each task $i \in T$ is executed without interruption during p_i time units and used $c_i \leq C$ units of resource. For a task $i \in T$, the earliest starting time est_i and the latest completion time lct_i are specified. Each task $i \in T$ has an energy $e_i = c_i \cdot p_i$, an earliest completion time $\text{ect}_i = \text{est}_i + p_i$ and a latest starting time $\text{lst}_i = \text{lct}_i - p_i$. A solution of a CuSP instance is an assignment of valid starting time $s_i \in [\text{est}_i, \text{lst}_i]$ to each task $i \in T$ such that the resource constraint is satisfied i.e.,

$$\forall \tau, \quad \sum_{i \in T,\ s_i \leq \tau < s_i + p_i} c_i \leq C \tag{1}$$

The inequality in (1) enforces the resource constraint. The notation energy, earliest starting and latest completion time, can be extended to non-empty sets of tasks as follows:

$$e_\Omega = \sum_{j \in \Omega} e_j, \quad \text{est}_\Omega = \min_{j \in \Omega} \text{est}_j, \quad \text{lct}_\Omega = \max_{j \in \Omega} \text{lct}_j. \tag{2}$$

By convention, for an empty set we have : $\text{est}_\emptyset = +\infty$, $\text{lct}_\emptyset = -\infty$, and $e_\emptyset = 0$. Throughout the paper, we assume that for any task $i \in T$, $\text{ect}_i \leq \text{lct}_i$ and $c_i \leq C$, otherwise the problem has no solution. We let $n = |T|$ denotes the number of tasks, $k = |\{c_i, i \in T\}|$ denotes the number of distinct capacity requirements. The global constraint CUMULATIVE removes inconsistent values from the domain of starting time variables $s_i \in [\text{est}_i, \text{lst}_i]$. Since the CuSP is a NP-Hard problem [7], it is NP-Hard to remove all such values. Polynomial time algorithms only exist for relaxations of the problem.

In this paper, we revisit the two phases *horizontally elastic edge finder* algorithm proposed by Gingras and Quimper [1] for cumulative resource constraint. We prove that the detection phase of this algorithm partially uses the *Profile* data structure. We propose a new formulation of the rule. A new algorithm (two phases algorithm for detection and adjustment) for the new rule of similar complexity ($\mathcal{O}(kn^2)$) is proposed. Experimental results on cumulative instances of resource constrained project scheduling problems (RCPSPs) from suites benchmarks highlight that using this new algorithm reduces the number of backtracks for a majority of instances with a marginal augmentation of the running time.

The rest of the paper is organized as follows. Section 2 presents useful notions used in the paper and in Section 3, we show that the detection of the horizontally elastic edge finder presented in [1] can be strengthened for a fully utilization of the Profile data structure and we propose a new formulation of the rule. We prove that the new formulation subsumes the previous one and we extend the attributes of the Profile data structure to be able in Section 4 to present a two phases detection and adjustment algorithm for the new horizontally elastic edge finder rule. In Section 5, the empirical evaluation of the new algorithm on cumulative instances of RCPSP is presented while Section 6 concludes the paper.

2. Backgrounds

In this section, we recall useful notions for the paper. After reviewing notations of attributes used in the *Profile*, we provide the rule used in [1] for the detection. For more details on the data structure see [1].

A stronger relaxation for the computation of the earliest completion time of a set of tasks (it is NP-hard to compute the earliest completion time of a set of tasks [7]) noted ect^H_Ω described in [1] consumes between 0 and c_i units of resource any time and exactly e_i units of energy are consumed in the interval $[\text{est}_i, \text{lct}_i)$. The Profile is initialized with a time point of capacity C for every distinct value of est, ect and lct. A sufficiently large time point is added to act as a sentinel. While initializing the data structure, pointers are kept so that $t.\text{est}_i$, $t.\text{ect}_i$ and $t.\text{lct}_i$ return the time point associated to est_i, ect_i, and lct_i. The algorithm *ScheduleTasks* computes the functions $t.c_{req} = c_{req}(t)$, $t.c_{max} = c_{max}(t)$, $t.c_{cons} = c_{cons}(t)$ and $t.ov = ov(t)$ (the capacity required, available, consume and the overflow at time t respectively) to schedule a set of tasks Θ on the profile P. The interesting properties of this data structure come from the number of time points and the linearity of algorithm *ScheduleTasks*.

Proposition 1 *[1]*

The Profile contains at most $4n+1$ time points and the algorithm ScheduleTasks runs in $\mathcal{O}(n)$ time where n is the number of tasks sharing the resource.

The algorithms proposed in [1] uses the left cut of the set T by a task $j \in T$.

Definition 1 *Let $j \in T$ be a task. The left cut of T by task j is the set of tasks*

$$\mathit{LCut}(T, j) = \{k, k \in T \wedge \mathit{lct}_k \leq \mathit{lct}_j\}.$$

In [1], to detect that the set of tasks $\Omega = \text{LCut}(T, j)$ precedes a task $i \notin \Omega$ noted $\text{LCut}(T, j) \lessdot i$, the algorithm checks the following rule:

$$\text{ect}^H_{\text{LCut}(T,j)\cup\{i\}} > \text{lct}_j \Rightarrow \text{LCut}(T, j) \lessdot i \qquad \text{(old-HE-EF)}$$

The detection proceeds by batching and detecting all precedence in $\mathcal{O}(kn^2)$ where $k \leq n$ is the number of distinct capacities required by tasks and n the number of tasks sharing the resource. When the relation $\text{LCut}(T, j) \lessdot i$ is detected, a $\mathcal{O}(n^2)$ algorithm is used to adjust the earliest start time of task i.

3. Strengthened the use of Profile in Edge finder and new Attributes for the Profile

Consider the CuSP instance of Figure 1 where three tasks $T = \{a, b, c\}$ share a resource of capacity 2.

When scheduling the set of tasks $\text{LCut}(T, b) = \{b, c\}$, we have enough free energy to schedule task a before $\text{lct}_b = 7$. Therefore, no detection is found by the algorithm of [1]. On the other hand, when task a starts at est_a with non-preemption, there is no possibility to schedule tasks c in his time window. Therefore, the precedence $\text{LCut}(T, b) \lessdot a$ is missed by the Gingras and Quimper's algorithm [1]. The problem comes from the fact that the algorithm checks whether

$$\text{ect}^H_{\text{LCut}(T,b)\cup\{a\}} > \text{lct}_b$$

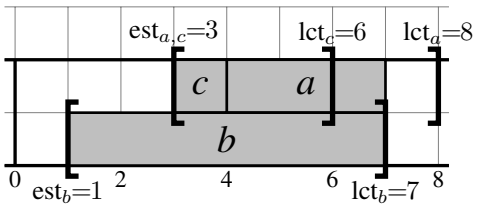


Figure 1. *(a A CuSP problem of 3 tasks sharing a resource of capacity $C = 2$.*

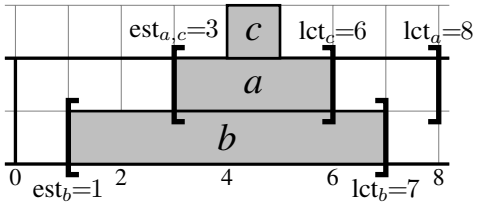


Figure 2. *(b) When task a is scheduled at est$_a$, no possibility to schedule task c in his time window*

were the non-preemption of task a is not taken into account.

The new horizontally elastic edge finder rule proposed in this paper is given by the formula:
For all $i, j \in T$ with $\text{lct}_i > \text{lct}_j$,

$$\text{ect}^H_{\text{LCut}(T,j)\cup\{i'\}} > \text{lct}_j \Rightarrow \text{LCut}(T, j) \ll i \qquad \text{(new-HE-EF)}$$

where i' is a task derived from task i whose parameters $\langle \text{est}_{i'}, \text{lct}_{i'}, p_{i'}, c_{i'} \rangle$ are

$$\langle \text{est}_i, \min(\text{ect}_i, \text{lct}_j), \min(\text{ect}_i, \text{lct}_j) - \text{est}_i, c_i \rangle.$$

Theorem 1 *The horizontally elastic edge finder rule* (new-HE-EF) *subsumes the old horizontally elastic edge finder rule* (old-HE-EF)

Proof. Since the set of tasks $\text{LCut}(T, j) \cup \{i'\}$ is most constrained than $\text{LCut}(T, j) \cup \{i\}$, it follows that $\text{ect}^H_{\text{LCut}(T,j)\cup\{i'\}} \geq \text{ect}^H_{\text{LCut}(T,j)\cup\{i\}}$. Therefore, all detections performed by the rule (old-HE-EF) are also done by the rule (new-HE-EF).

In the example of Figure 1, the rule (new-HE-EF) was able to detect that $\{b, c\} \ll a$. This detection was not found by the rule (old-HE-EF). ■

Let j be a task and c be a capacity required by a task. Before scheduling tasks of $\text{LCut}(T, j)$, the profile is divided into two: a upper part of capacity c and a bottom part of capacity $C - c$. We extend the function *ScheduleTask*($\text{LCut}(T, j), C$) of [1] to *ScheduleTasks*($\text{LCut}(T, j), C, c$) by specifying at each time point t the overlap energy $t.over$, the slack under $t.sUnder$ and over $t.sOver$ the line $C - c$ accumulated through the interval $[0, t.time]$ and the contact point $t.cont$ which represents the nearest time point were consumption is greater than $C - c$ and from there to other contact point, the overlap is greater than the slack under. At the first time point $t.first$ the values $t.over$, $t.sUnder$ and $t.sOver$ are initialized to 0 and they are updated at any time point with the relations:

$$t.over = t.previuos.over + \max(t.c_{cons} - (C - c), 0) \cdot \ell \qquad (3)$$

$$t.sUnder = t.previous.sUnder + \max(\min(C - c, t.c_{max}) - t.c_{cons}), 0) \cdot \ell \qquad (4)$$

$$t.sOver = t.previous.sOver + \max(t.c_{max} - \max(C - c, t.c_{cons}), 0) \cdot \ell \qquad (5)$$

where $\ell = t.time - t.previous.time$.

$$t.cont = \underset{A}{\text{argmin}}\, t \qquad (6)$$

where $A = \{t \in P \mid t.c_{cons} > C - c \wedge \forall t' \in P$ with $t'.time > t.time$ and $t'.c_{cons} > C - c,\ t'.over - t.over > t'.sUnder - t.sUnder\}$. An illustration of the above relations are shown in the following example.

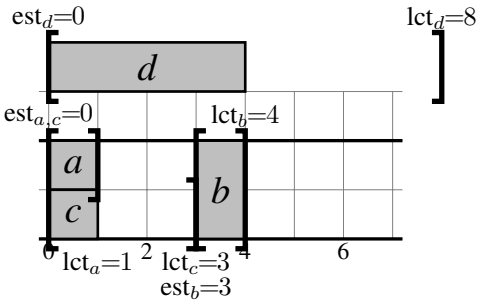


Figure 3. *(A CuSP problem of 4 tasks sharing a resource of capacity $C = 2$.*

Consider the CuSP instance of Figure 3 where four tasks $T = \{a, b, c, d\}$ share a resource of capacity 2. When the tasks $\{a, b, c\}$ are scheduled and $C - c = 1$, between the time point 0 and 3, we have $sOver(0, 3) = 0$ since $c_{\max} = 1$ between 1 and 3. $sUnder(0, 3) = 2$ since task c can be executed till time 3 and $over(0, 3) = 1$. The contact point at time point 0 is then 3 since $over(0, 3) < sUnder(0, 3)$.

4. New Edge Finder Algorithm

4.1. Edge Finder Detection

In Algorithm 1, we iterate through the set of tasks sorted in increasing order of lct (line 1) and we consider all the capacity required by tasks of set $\Delta = \{c_i \mid i \in \Lambda\}$ where $\Lambda = T \setminus \Theta$ and $\Theta = \text{LCut}(T, j)$ (line 2). By this way, we proceed by batching tasks of the same capacity. The function *ScheduleTasks*$(\text{LCut}(T, j), C, c)$ is used at line 3 to compute the earliest completion time of the set Θ and when the current set of tasks is e-feasible (line 4), the function *Detection* is called to detect all the tasks of $\Delta_c = \{i \in \Lambda \mid c_i = c\}$ which satisfy the edge finding rule.

Algorithm 1: EdgeFinderDetection in $\mathcal{O}(kn^2)$ time.

Input: T set of tasks sorted in non-decreasing lct.
Output: Prec precedence relation and MaxOver maximum upper overload energy at the contact point for each task $i \in \Lambda$

1 **forall** $j \in T$ *with* $lct_j < lct_{j+1}$ **do**
2 **forall** $c \in \Delta$ *where* $\Delta = \{c_i \mid i \notin \Theta\}$ *and* $\Theta = LCut(T, j)$ **do**
3 $ect^H \leftarrow ScheduleTasks(\Theta, C, c)$
4 **if** $ect^H \leq lct_j$ **then**
5 (Prec, MaxOver) $\leftarrow Detection(\Delta_c)$ where $\Delta_c = \{i \in \Lambda \mid c_i = c\}$
6 **return** est′

The *Detection* function iterates over the set Δ_c (line 1) and consider three particular time points: $t_1 = t.\text{est}_i.cont$, $t_2 = t.\text{ect}_i$ and $t_3 = t.\text{lct}_j$ for each case. In the case of line 3, $(\text{ect}_i \leq \text{lct}_j)$, we use the relation $over(t_1, t_2) > sUnder(t_1, t_3) + sOver(t_2, t_3)$ at line 4 to detect the precedence at line 5 and compute the maximum overload energy of the upper part of the profile at the contact point of est_i (line 6). In the other case, $(\text{ect}_i > \text{lct}_j)$ line 8, the relation $over(t_1, t_3) > sUnder(t_1, t_3)$ is used at line 9 to detect the precedence and compute $MaxOver[i]$.

Algorithm 2: *Detection*(Δ_c) in $\mathcal{O}(n)$ time

Input: Δ_c

Output: Prec[] and $MaxOver[]$ vector of precedence and the maximum upper overload energy after the contact point

```
1  forall i ∈ Δ_c do
2      t_1 = t.est_i.cont, t_2 = t.ect_i, t_3 = t.lct_j
3      if t_1.time < t_2.time ∧ t_2.time ≤ t_3.time then
4          if over(t_1, t_2) > sUnder(t_1, t_3) + sOver(t_2, t_3) then
5              Prec[i] ← j
6              MaxOver[i] ← overlap(t_1, t_2) − sUnder(t_1, t_3)
7              Λ ← Λ \ {i}
8      if t_1.time < t_2.time ∧ t_2.time > t_3.time then
9          if over(t_1, t_3) > sUnder(t_1, t_3) then
10             Prec[i] ← j
11             MaxOver[i] ← overlap(t_1, t_3) − sUnder(t_1, t_3)
12             Λ ← Λ \ {i}
```

Proposition 2 *Detection runs in $\mathcal{O}(n)$ time.*

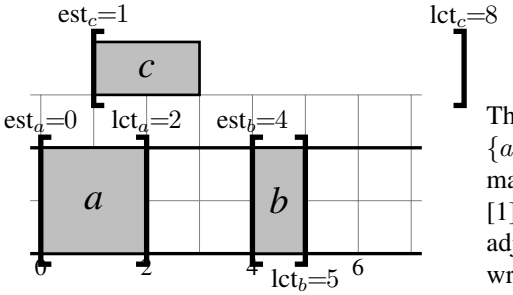
Proof. The algorithm iterates over the set Δ_c of cadinality n in the worse case and the remaining operations are made in $\mathcal{O}(1)$. ■

Proposition 3 *EdgeFinderDetection runs in $\mathcal{O}(kn^2)$ time where k is the different capacities required by tasks.*

Proof. For each task and for each capacity required, *EdgeFinderDetection* calls *ScheduleTasks* and *Detection*. Therefore, the complexity of *EdgeFinderDetection* is $\mathcal{O}(kn(n + n) = \mathcal{O}(kn^2)$. ■

4.2. Edge Finder Adjustment

Let's consider the CuSP instance of three tasks $T = \{a, b, c\}$ sharing a resource of capacity 2 presented in the following figure. In Algorithm 3, the vector est′ of new earliest



The new rule will detect $\{a\} \ll c$ and $\{a, b\} \ll c$ while only the detection $\{a\} \ll c$ is made by Gingras and Quimper algorithm's [1]. Therefore, we cannot used the same adjustment algorithm since it can deduced wrong values with the detection $\{a, b\} \ll c$.

Figure 4. *(A CuSP problem of 3 tasks sharing a resource of capacity $C = 2$.*

starting time of task is initialised with the current value of earliest starting time of tasks (line 2). For each detection, we schedule tasks of $\Theta = \text{LCut}(T, j)$ with parameters c_i for the upper part of the profile (line 4). The function *ComputeBound* is then used to compute the new earliest completion time of task i (line 5).

Algorithm 3: *EdgeFinderAdjustment*(Prec[], MaxOver[]) in $\mathcal{O}(n^2)$ time

Input: Prec[] and MaxOver[]
Output: est'_i the new bounds of all tasks $i \in T$

```
1 forall i ∈ T do
2     est'_i ← est_i
3     if Prec[i] = j ∧ MaxOver[i] > 0 then
4         ScheduleTasks(Θ, C, c_i)
5         est'_i = max(est'_i, ComputeBound(i, j, MaxOver[i]))
```

In Algorithm 4, we iterate over the profile starting at the contact point of the est_i of task i. From that point, we consume the maximum overload energy in the upper part of the profile (line 6) and the new bound correspond to the end of this energy.

Algorithm 4: *ComputeBound*$(i, j, \text{MaxOver}[i])$ in $\mathcal{O}(n)$ time

Input: i, j and MaxOver$[i]$
Output: est'_i the new bounds of tasks i

```
1 t ← t.est_i.cont
2 maxOv ← MaxOver[i]
3 while t.next ≠ null do
4     over ← t.next.over − t.over
5     if maxOv > over then
6         maxOv ← maxOv − over
7         t ← t.next
8     else
9         est'_i = min(t.next.time, t.time + ⌈maxOv / (t.cons − (C + c_i))⌉
```

Proposition 4 *ComputeBound runs in $\mathcal{O}(n)$ time.*

Proof. Direct consequence of Proposition 1. ■

Proposition 5 *EdgeFinderAdjustment runs in $\mathcal{O}(n^2)$ time.*

Proof. For each task, the function *ScheduleTasks* and *ComputeBound* are called which lead to a global complexity of $\mathcal{O}(n(n+n)) = \mathcal{O}(n^2)$. ■

5. Experimental Results

Empirical evaluation of the different versions of the algorithm was carried out on resource-constrained project scheduling problems (RCPSP) to compare the new algorithms with the state-of-the art algorithm. 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 starting time assignment for all tasks satisfying the precedence and resource capacity constraints, with the least makespan (i.e., the time at which all tasks are completed) at most equal to the horizon.

Tests were performed on benchmark suites of RCPSP known to be highly cumulative [6]. On highly cumulative scheduling instances, many tasks can be scheduled simultaneously as contrary to the highly disjunctive ones. We use the libraries BL [6], Pack [10], KSD15_D [11]. The data set BL consists of 40 instances of 20 and 25 tasks sharing three

resources, Pack consists of 55 instances of 15-33 tasks sharing a resource of capacity 2-5 while the set KSD15_D consists of 480 instances of 15 tasks sharing a resource of capacity 4.

Starting with the provided horizon as an upper bound, we modeled each problem as an instance of Constraint Satisfaction Problem (CSP); variables are starting times of tasks and they are constrained by the precedence graph (i.e., precedence relations between pairs of tasks were enforced with linear constraints) and resource limitations (i.e., each resource was modeled with a single CUMULATIVE constraint [2]). We used a branch and bound search to minimize the makespan.

The implementation was done in Java using Choco solver 4.10.1 [12]. Two filtering algorithms for different configurations of the global constraint CUMULATIVE were considered.

1) The first CUMULATIVE propagator noted **old-HE-EF** (for Gingras and Quimper horizontally elastic edge finder) is a sequence of two filtering algorithms: the $\mathcal{O}(kn^2)$ horizontally elastic edge finder from [1] and timetabling algorithm from [13].

2) The second propagator noted **new-HE-EF** is obtained when replacing in the first propagator, the Gingras and Quimper's algorithm [1] by the algorithm presented in Section 4.

Static branching scheme is the best way to compare two propagators of different filtering power. We use the lexicographic static branching scheme which consists of selecting unscheduled tasks in lexicographic order and assign it to its lower bound. We also evaluate our propagator on dynamic branching scheme which is a heuristic used to select tasks and values during the resolution process. We combine the conflict-ordering search heuristic [14] with the default search heuristic $domOverWDegSearch$ from Choco. During the search, the solver records conflicting tasks and at the backtrack, the last one is selected in priority until they are all instantiated without causing any failure. When no conflicting tasks is recorded, the heuristic $domOverWDegSearch$ is used. Tests were performed on Intel(R) Core(TM) i5, 1.6 GHz CPU with 4 GB memory, using a single core. Any search taking more than 5 minutes was counted as a failure.

	old-HE-EF			**new-HE-EF**			Speedup (%)
	solve	time	backts	solve	time	backts	speedup
	Static branching						
BL	26	26.12	399600	**27**	**23.22**	**232623**	**112.51**
PACK	10	**18.2**	88853	10	22.08	**74063**	82.47
KSD15_D	398	**0.85**	7692	**399**	1.43	7692	59.08
	Dynamic branching						
BL	38	**8.39**	64637	38	10.01	**53680**	83.86
PACK	**11**	**13.54**	**73554**	9	25.54	101853	52.67
KSD15_D	**450**	**0.99**	7372	444	1.83	**7137**	54.15

Table 1. *For each propagator, we report the number of instances solved (solve), the average number of backtracks (backts), the average time (time) and the average speed up factor required to solve all instances that are commonly solved by the two propagators on set BL, Pack, KSD15 respectively.*

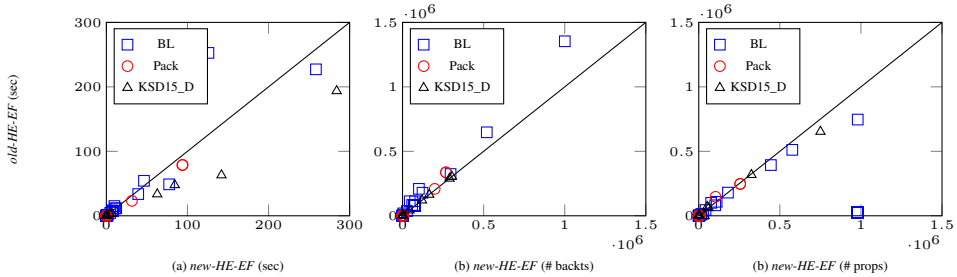


Figure 5. *Static branching scheme: (a) Runtime old-HE-EF vs. new-HE-EF, (b) Comparison of the number of Backtracks old-HE-EF vs. new-HE-EF , (c) Comparison of the number of propagation old-HE-EF vs. new-HE-EF on instances of BL, Pack and KDS15_D where the two propagators found the best solution.*

In Table 1, we notice that in static branching scheme, we do less backtracks but we need slight time and more propagation to do so. Therefore, we perform better than the state of the art algorithm. In dynamic branching scheme, we most of the time do less backtracks and we need little more time and propagation to achieve it. This is due to the type of the heuristic used.

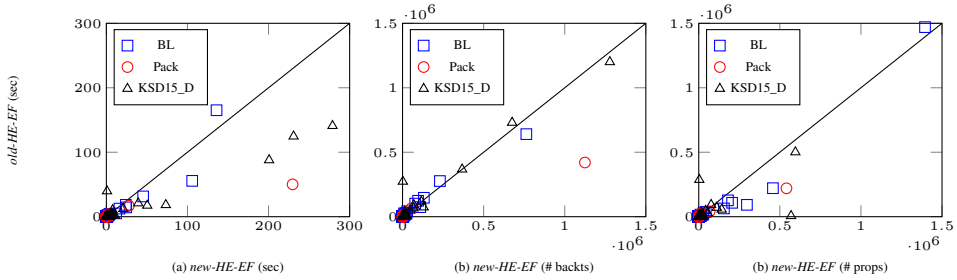


Figure 6. *Dynamic branching scheme: (a) Runtime old-HE-EF vs. new-HE-EF, (b) Comparison of the number of Backtracks old-HE-EF vs. new-HE-EF , (c) Comparison of the number of propagation old-HE-EF vs. new-HE-EF on instances of BL, Pack and KDS15_D where the two propagators found the best solution.*

6. Conclusion

In this paper, we revisited the two phases *horizontally elastic edge finder* algorithm proposed by Gingras and Quimper [1] for cumulative resource constraint. We proved that the detection phase partially uses the *Profile* data structure and proposed a new formulation of the rule. A new detection algorithm of complexity $\mathcal{O}(kn^2)$ was then presented and combined with the $\mathcal{O}(n^2)$ adjustment phase for an overall algorithm of complexity $\mathcal{O}(kn^2)$. Experimental results on highly cumulative instances of resource constrained project scheduling problems (RCPSPs) from suites benchmarks highlight that using this new algorithm reduces the number of backtracks with a slight increase of the running time. Future work will focus on finding how to improve the complexity of this algorithm from $\mathcal{O}(kn^2)$ to $\mathcal{O}(n^2)$ and design a branching scheme more suitable for the new rule.

Acknowledgments

The authors would like to thank Vincent Gingras and Claude-Guy Quimper for their code source and Roger Kameugne for his advice and assistance.

7. References

[1] V. GINGRAS AND C.-G. QUIMPER: "Generalizing the Edge-Finder Rule for the Cumulative Constraint.", *In Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI 2016)*, 3103–3109, 2016.

[2] A. AGGOUN, AND N. BELDICEANU: "Extending CHIP in order to solve complex scheduling and placement problems". *Mathematical and Computer Modeling*, vol. 17, num. 7, 57–73, (1993).

[3] R. KAMEUGNE, L. P. FOTSO, J. SCOTT, Y. NGO-KATEU: "A Quadratic Edge-Finding Filtering Algorithm for Cumulative Resource Constraints", *Constraints*, vol. 19, num. 3, pp 243-269. Springer, (2014).

[4] S. GAY, R. HARTERT, AND P. SCHAUS: "Simple and Scalable Time-Table Filtering for the Cumulative Constraint. " *In Proceedings of the 21st International Conference on Principles and Practice of Constraint Programming (CP 2015)*, 149–157, 2015.

[5] R. KAMEUGNE, AND L.P. FOTSO: "A Cumulative Not-First/Not-Last Filtering Algorithm in $\mathcal{O}(n^2 \log(n))$". *Indian Journal of Pure Applied Mathematics*. vol. 44, num. 1, 95–115, 2013.

[6] P. BAPTISTE, C. LE PAPE, AND W. NUIJTEN: "Constraint-based scheduling: applying constraint programming to scheduling problems." *Kluwer*, Boston (2001).

[7] M. R. GAREY AND D. S. JOHNSON: "Computers and Intractability", vol. 29, wh freeman. (2002).

[8] R. KAMEUGNE, L. P. FOTSO, J. SCOTT: "A Quadratic Extended Edge-Finding Filtering Algorithm for Cumulative Resource Constraints", *International Journal of Planning and Scheduling*. vol. 1, num. 4, pp. 264-284, (2013).

[9] R. KAMEUGNE, SEVERINE FETGO BETMBE, V. GINGRAS, Y. OUELLET AND C-G QUIMPER.: "Horizontally Elastic Not-First/Not-Last Filtering Algorithm For Cumulative Resource Constraint.", *In proceeding of CPAIOR, 2018.*, LNCS 10848, pp. 316?332.

[10] J. CARLIER AND E. NÉRON: "On linear lower bounds for the resource constrained project scheduling problem", *European Journal of Operational Research*, vol. 149, num. 2, 314-324, 2003.

[11] O. KONÉ, C. ARTIGUES, P. LOPEZ, AND M. MONGEAU: "Event-based milp models for resource-constrained project scheduling problems". *Computers & Operations Research*, vol. 38, num. 1, 3-13, 2011.

[12] C. PRUD'HOMME, J.-G. FAGES, AND X. LORCA: "Choco Solver Documentation ", *TASC, INRIA Rennes, LINA CNRS UMR 6241*, COSLING S.A.S., 2016, `http://www.choco-solver.org`

[13] A. LETORT, N. BELDICEANU, AND M. CARLSSON: " A scalable sweep algorithm for cumulative constraint", *In Proceedings of the 18th International Conference on Principles and Practice of Constraint Programming (CP 2012)*, 439-454, 2012.

[14] STEVEN GAY, RENAUD HARTERT, CHRISTOPHE LECOUTRE AND PIERRE SCHAUS, "Conflict Ordering Search for Scheduling Problems", *In Proceedings of the 21st International Conference on Principles and Practice of Constraint Programming (CP 2015)*, 140-148, Cork, Ireland, 2015.