A Quadratic Extended Edge-Finding Filtering Algorithm for Cumulative Resource Constraints*

Roger Kameugne[1], Laure Pauline Fotso[2], and Joseph D. Scott[3]

[1]University of Maroua, Higher Teachers' Training College, Dept. of Mathematics, Maroua, Cameroon, and University of Yaoundé I, Faculty of Sciences, Dept. of Mathematics, Yaoundé, Cameroon
[2]University of Yaoundé I, Faculty of Sciences, Dept. of Computer Sciences, Yaoundé, Cameroon
[3]Uppsala University, Dept. of Information Technology, Uppsala, Sweden

June 5, 2013

Abstract

Edge-finding, extended edge-finding, not-first/not-last and energetic reasoning are well-known filtering rules used in constraint-based scheduling problems for propagating constraints over disjunctive and cumulative resources. In practice, these filtering algorithms frequently form part of a sequence to form a more powerful propagator, thereby helping to reduce search tree size. In this paper, we propose a sound $\mathcal{O}(n^2)$ extended edge-finding algorithm for cumulative resources, where n is the number of tasks sharing the resource. This algorithm uses the notion of minimum slack to detect when extended edge-finding justifies a strengthening of a domain, and it is more efficacious when executed on a domain already at the fix point of standard edge-finding. Previously, the best known complexity for filtering extended edge-finding on cumulative resources was $\mathcal{O}(kn^2)$ (where k is the number of distinct capacity requirements). Experimental results on resource constrained scheduling benchmarks confirm that the new algorithm outperforms previous extended edge-finding algorithms, and sometimes results in better performance than standard edge-finding alone. Furthermore, we show that our method is competitive with the current state-of-the-art in edge-finding based algorithms.

1 Introduction

Many real-world scheduling problems may be modeled as cumulative scheduling problems (CuSP). These problems arise for example in industry, where resources might represent workers and machines, or in computer science, where

*Accepted for publication in the *International Journal of Planning and Scheduling*

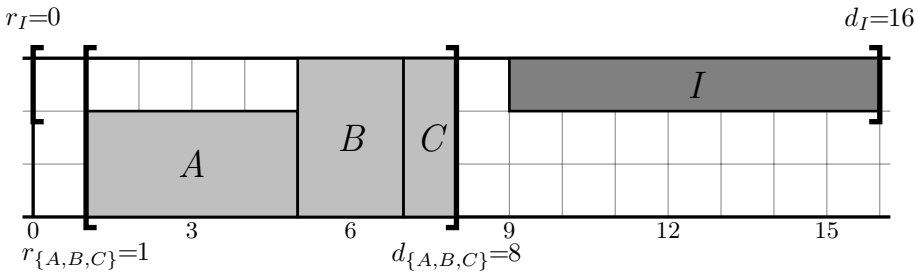


Figure 1: A scheduling problem of 4 tasks sharing a resource of capacity $C = 3$.

resources could be processors or network bandwidth. A CuSP models these sorts of problems as a set T of tasks, which must be executed on a renewable resource of fixed capacity C. Each task $i \in T$ requires a fixed, constant amount of the resource, denoted c_i. Execution of a task requires a fixed amount of time p_i, which must occur without interruption, and execution must occur wholly within a time frame between an earliest start time r_i (release date) and a latest end time d_i (deadline). A solution of a CuSP instance is an assignment of valid start time s_i to each task i, such that the resource constraint is satisfied at all times; i.e.,

$$\forall i \in T\colon r_i \leq s_i \leq s_i + p_i \leq d_i \tag{1}$$

$$\forall \tau\colon \sum_{i \in T,\ s_i \leq \tau < s_i + p_i} c_i \leq C \tag{2}$$

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 a CuSP is given in Figure 1.

The CuSP is an NP-complete problem (Baptiste and Le Pape, 2000), however there are known polynomial time algorithms for solving specific relaxation of CuSP. These relaxations form the basis of filtering algorithms for the CUMULATIVE constraint (Aggoun and Beldiceanu, 1993), a global constraint which models a single cumulative resource constraint. None of these filtering algorithms forms a complete filtering algorithm for CUMULATIVE; however, algorithms based on different relaxations may reveal different ways of strengthening the same domain. Because of this fact, propagators for CUMULATIVE typically embed several filtering algorithms in sequence. Among these relaxations, edge-finding and timetabling are probably the most used, but there exist many others such as not-first/not-last, energetic reasoning, and more recently, timetable edge-finding.

Extended edge-finding is similar to the better known edge-finding filter: by deducing new ordering relations between the tasks, it seeks to reduce the range of possible start times. For a task i, an extended edge-finder searches for a set of tasks Ω that *must* end before the end (or alternately, start before the start)

of i. Based on this newly detected precedence, the earliest start time (or latest completion time) of i is updated.

We define the energy of a task i as $e_i = c_i \cdot p_i$. This notation, along with that of earliest start and latest completion time, may be extended to non-empty sets of tasks as follows:

$$r_\Omega = \min_{j \in \Omega} r_j, \quad d_\Omega = \max_{j \in \Omega} d_j, \quad e_\Omega = \sum_{j \in \Omega} e_j \tag{3}$$

where Ω is a non-empty set of tasks. By convention, if Ω is the empty set, $r_\Omega = +\infty$, $d_\Omega = -\infty$, and $e_\Omega = 0$. Throughout the paper, we assume that for any task $i \in T$, $r_i + p_i \leq d_i$ and $c_i \leq C$, otherwise the problem has no solution. We let $n = |T|$ denote the number of tasks, and $k = |\{c_i, i \in T\}|$ denote the number of distinct capacity requirements. For the remainder of this paper we focus solely on the algorithm for updating release dates, as the deadline algorithm is both symmetric and easily derived.

1.1 Related Work

To our knowledge, extended edge-finding was first proposed in Nuijten (1994), which gave an $\mathcal{O}\left(kn^3\right)$ algorithm, where n is the number of tasks and $k \leq n$ is the number of distinct capacity requirements among tasks. Mercier and Van Hentenryck (2008) proved that the edge-finding algorithm given in Nuijten (1994) is incomplete (missing some adjustments); as a consequence, the extended algorithm is also incomplete, as it relies upon the edge-finding algorithm. Mercier and Van Hentenryck (2008) proposes a two phase algorithm for extended edge-finding, which runs $\mathcal{O}\left(kn^2\right)$ time and uses $\mathcal{O}\left(kn\right)$ space. The first phase uses dynamic programming to precompute potential adjustment values, and the second phase detects the extended edge-finding condition and applies the updates of the first phase when appropriate. (We note that the second phase of the $\mathcal{O}\left(kn^2\right)$ extended edge-finding algorithm presented in Mercier and Van Hentenryck (2008), CalcEEF, is flawed: $CEEF[c, i]$ is computed for all task intervals $\Omega_{X[x],Y[y]}$, yet there is no condition (at line 18) to ensure that $\Omega_{X[x],Y[y]}$ is non-empty, which may result in the extended edge-finding condition being detected when it does not apply. While this oversight is easily corrected, it has not, to our knowledge, been previously reportted.)

Recently, two sound edge-finding algorithms with quadratic time complexity have been proposed. Vilím (2011) introduces timetable edge-finding, which uses the timetabling data structure. It is claimed that this algorithm subsumes the conjunction of the standard and extended edge-finding algorithms, a claim refuted by Kameugne et al. (2013b), which proves that the timetable edge finding alone cannot subsume extended edge-finding. The second quadratic algorithm, from Kameugne et al. (2011), uses a combination of minimum slack and maximum density to detect the standard edge-finding rule.

1.2 Contributions

We know from Baptiste et al. (2001) that the edge-finding rule is not dominated by the extended edge-finding rule. The aim of this paper is to provide an algorithm that performs only those adjustments which are justified by extended edge-finding, but which standard edge-finding misses. We assume that the fix point of edge-finding has already been reached; we prove that, under this condition, a set of tasks detected using the extended edge-finding condition is a valid candidate set for making an adjustment (see Theorem 1).

This result allows us to propose a quadratic algorithm for the remaining adjustments missed by the edge-finding algorithm. This algorithm uses the minimal slack notion, which has proved to be useful for edge-finding (Kameugne et al., 2011), to quickly locate the set Ω for which the detection and the adjustment rules hold. The combination of this algorithm with the quadratic edge-finding algorithm of Kameugne et al. (2011) yields a $\mathcal{O}(n^2)$ extended edge-finding algorithm. The new extended edge-finder reaches the same fix point as the conjunction of the standard and extended edge-finding rules (see Theorem 4). Through experimental results on RCPSP benchmarks from the Project Scheduling Problem Library and Baptiste and Le Pape sets, we verify the completeness of our algorithm, and we demonstrate the contribution extended edge-finding may make to the propagation of CUMULATIVE, when combined with an edge-finding filter.

In section 2, we define the extended edge-finding rule. In Section 3, the new extended edge-finding algorithm is presented; its correctness is proved in Section 4. Section 5 compares the combination of the algorithm proposed in Section 3 with the edge-finding algorithm of Kameugne et al. (2011), with the conjunction of the standard and extended edge-finding rules. Section 6 studies experimental results on RCPSP benchmarks from the Project Scheduling Problem Library (PSPLib, 2013), and the benchmarks of Baptiste and Le Pape (2000).

2 The (extended) edge-finding rules

2.1 The rules

Let $\Omega \subset T$ be a non-empty set of tasks of a CuSP and $i \in T \setminus \Omega$ be a task. If the scheduling of task i as early as possible (i.e., starting at r_i) would cause an overload in the interval $[r_\Omega, d_\Omega)$, then it is deduced that all the tasks in Ω end before the end of i. Following Vilím (2009a), we denote this relationship $\Omega \ll i$. Once an appropriate Ω and i have been located, the earliest start time of i can be adjusted using the following rule:

$$\Omega \ll i \implies r_i \geq r_\Theta + \left\lceil \frac{1}{c_i} \operatorname{rest}(\Theta, c_i) \right\rceil \tag{4}$$

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

$$\text{rest}(\Theta, c_i) = \begin{cases} e_\Theta - (C - c_i)\,(d_\Theta - r_\Theta) & \text{if } \Theta \neq \emptyset \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

The condition $\text{rest}(\Theta, c_i) > 0$ states that the total energy e_Ω that must be scheduled in the window $[r_\Omega, d_\Omega)$ is strictly larger than the energy that could be scheduled without making any start time of i in that window infeasible. The proof of these results can be found in Baptiste et al. (2001); Nuijten (1994).

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. *Let Ω be a set of tasks and let $i \notin \Omega$ be a task of a CuSP of capacity C.*

$$e_{\Omega\cup\{i\}} > C\left(d_\Omega - r_{\Omega\cup\{i\}}\right) \Rightarrow \Omega \ll i \ ; \tag{EF}$$

$$r_i + p_i \geq d_\Omega \quad \Rightarrow \quad \Omega \ll i \ ; \tag{EF1}$$

$$\left.\begin{array}{l} r_i \leq r_\Omega < r_i + p_i \\ e_\Omega + c_i(r_i + p_i - r_\Omega) > C\,(d_\Omega - r_\Omega) \end{array}\right\} \Rightarrow \Omega \ll i \ . \tag{EEF}$$

Proof. Proof For the proof (EF) and (EF1) see Kameugne et al. (2011, 2013b); Vilím (2009a); for the proof of (EEF) see Baptiste et al. (2001); Mercier and Van Hentenryck (2008); Nuijten (1994). □

Rules (EF) and (EF1) are known as the *edge-finding detection rules*, while (EEF) is known as the *extended edge-finding detection rule*. Combining (EF) and (EF1) (resp. (EEF)) with (4) gives us a formal definition of an edge-finding (resp. extended edge-finding) algorithm:

Definition 1 (Specification of a complete (extended) edge-finding algorithm)**.** *An (extended) edge-finding algorithm receives as input a CuSP, and produces as output a vector of updated lower bounds for the release times of tasks $\langle LB_1, \ldots, LB_n\rangle$, where:*

$$LB_i = \max\left(r_i, \max_{\substack{\Omega \subseteq T \\ i \notin \Omega \\ \beta(\Omega, i) \vee \alpha(\Omega, i)}} \ \max_{\substack{\Theta \subseteq \Omega \\ \text{rest}(\Theta, c_i) > 0}} r_\Theta + \left\lceil \frac{1}{c_i} \text{rest}(\Theta, c_i) \right\rceil\right) \tag{6}$$

with

$$\alpha(\Omega, i) \stackrel{def}{=} \left(C\left(d_\Omega - r_{\Omega\cup\{i\}}\right) < e_\Omega + e_i\right) \vee (r_i + p_i \geq d_\Omega) \tag{7}$$

and

$$\beta(\Omega, i) \stackrel{def}{=} (C\,(d_\Omega - r_\Omega) < e_\Omega + c_i(r_i + p_i - r_\Omega)) \wedge (r_i \leq r_\Omega < r_i + p_i) \tag{8}$$

and

$$\text{rest}(\Theta, c_i) = \begin{cases} e_\Theta - (C - c_i)\,(d_\Theta - r_\Theta) & \text{if } \Theta \neq \emptyset \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

2.2 Dominance properties

An (extended) edge-finder cannot efficiently consider all sets $\Theta \subseteq \Omega \subset T$ to update a task i. Some dominance properties are used to reduce the number of sets which must be considered. It is demonstrated by Mercier and Van Hentenryck (2008) that an (extended) edge-finder that considers only sets $\Omega \subseteq T$ and $\Theta \subseteq \Omega$ which are also task intervals can be complete.

Definition 2 (Task Intervals)**.** *(After Caseau and Laburthe (1994)) Let $L, U \in T$. The task intervals $\Omega_{L,U}$ is the set of tasks*

$$\Omega_{L,U} = \{j \in T \mid r_L \leq r_j \wedge d_j \leq d_U\} \quad . \tag{10}$$

If there exists a set of tasks $\Omega \subseteq T$ which cannot be scheduled in the window from r_Ω to d_Ω without exceeding the capacity, then the CuSP 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 (Vilím, 2009b; Wolf and Schrader, 2006).

Definition 3 (E-Feasibility)**.** *(Mercier and Van Hentenryck, 2008) A CuSP problem is E-feasible if :*

$$\forall \Omega \subseteq T, \quad \Omega \neq \emptyset, \quad C\,(d_\Omega - r_\Omega) \geq e_\Omega \quad . \tag{11}$$

In the rest of the paper, we only consider E-feasible CuSPs.

In a naive approach, every task can be compared with every task intervals in $\mathcal{O}(n^3)$ time; however, it is possible to further limit the number of intervals as follows:

Proposition 2 (Mercier and Van Hentenryck (2008))**.** *Let i be a task and Ω, Θ be two tasks set of an E-feasible CuSP with $\Theta \subseteq \Omega$. If the extended edge-finding rule* (EEF) *applied to task i with pair (Ω, Θ) allows to update the earliest start time of i then*

(i) there exists four tasks L, U, l, u such that $r_i \leq r_L \leq r_l < d_u \leq d_U < d_i$ and $r_L < r_i + p_i$;

(ii) the extended edge-finding rule (EEF) *applied to task i with the pair $(\Omega_{L,U}, \Omega_{l,u})$ allows at least the same update of the earliest start time of task i.*

In the example shown in Figure 1, the rule (EEF) correctly detects $\Omega \ll I$ for $\Omega = \{A, B, C\}$. Using the set $\Theta = \{A, B, C\}$ in formula (4), the release date of I is updated from 0 to 4.

3 A new extended edge-finding algorithm

In this section, we present a quadratic extended edge-finding algorithm which performs the additional adjustment missed by edge-finding. The combination of our new algorithm with the quadratic edge-finder of Kameugne et al. (2011) reaches the same fix point as the conjunction of the standard and extended edge-finding algorithms (see section 5). For a given task i, the algorithm identifies the set Ω of tasks of minimum slack that satisfy both $r_i < r_\Omega$ and $d_i > d_\Omega$. It then determines whether the relation $\Omega \ll i$ holds, and if so, updates the release date of task i (see. Section 4 for proof of correctness).

3.1 At the fix point of the edge-finding

The extended edge-finding algorithm proposed in the next section supposes that we are already at the fix point of the edge-finding rule. Under this hypothesis, we will prove that, if the extended edge-finding rule detects the relation $\Omega \ll i$ for a tasks set Ω and a task i, then the set Ω can help to compute the potential update value of r_i i.e.,

$$rest(\Omega, c_i) > 0$$

and

$$r_\Omega + \left\lceil \frac{1}{c_i} rest(\Omega, c_i) \right\rceil > r_i.$$

This property is formally proved in the following theorem.

Theorem 1. *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*

$$rest(\Omega, c_i) > 0$$

and

$$r_\Omega + \left\lceil \frac{1}{c_i} rest(\Omega, c_i) \right\rceil > r_i.$$

Proof. Proof Let $\Omega \subset T$ be a set of tasks of an instance of CuSP of capacity C. We suppose that the fix point of the edge-finding rule is reached i.e., for all $\Omega' \subset T$, for all task $i \in T \setminus \Omega'$, we have

$$C\left(d_{\Omega'} - r_{\Omega' \cup \{i\}}\right) \geq e_{\Omega'} + e_i \quad \text{and} \quad r_i + p_i < d_{\Omega'}.$$

If not, then for all $\forall \Theta' \subseteq \Omega'$ with $rest(\Theta', c_i) > 0$, we have

$$r_{\Theta'} + \left\lceil \frac{1}{c_i} rest(\Theta', c_i) \right\rceil \leq r_i.$$

Let $\Theta \subseteq \Omega$ be a set of tasks and $i \in T \setminus \Omega$ be a task such that the pair (Ω, Θ) justifies, by extended edge-finding rule, an update of the release date of task i; i.e,

$$C\left(d_\Omega - r_\Omega\right) < e_\Omega + c_i(r_i + p_i - r_\Omega) \tag{12}$$

and

$$r_i \leq r_\Omega < r_i + p_i \tag{13}$$

and

$$r_\Theta + \left\lceil \frac{1}{c_i} rest(\Theta, c_i) \right\rceil > r_i \tag{14}$$

with

$$rest(\Theta, c_i) > 0. \tag{15}$$

We have to prove that

$$rest(\Omega, c_i) > 0 \quad \text{and} \quad r_\Omega + \left\lceil \frac{1}{c_i} rest(\Omega, c_i) \right\rceil > r_i$$

hold. First of all, let us prove that $r_i < r_\Omega$ and $d_\Omega > r_i + p_i$.

- If $r_i = r_\Omega$ holds, then formula (12) is equivalent to

  $$C\,(d_\Omega - r_\Omega) < e_\Omega + e_i.$$

  Therefore, with (14) and (15), and the fact that $\Theta \subseteq \Omega$, we can conclude that the edge-finding rule (EF) holds. Therefore the release date of r_i is updated, and the fix point of the edge-finding is not yet reached. This contradicts our hypothesis; hence, $r_i < r_\Omega$, since $r_i \leq r_\Omega$ by formula (13).

- If $d_\Omega \leq r_i + p_i$ holds, then the edge-finding rule (EF1) holds. Using formulas (14) and (15), and the fact that $\Theta \subseteq \Omega$, the release date of r_i may be updated, so the fix point of edge-finding has not yet reached. Hence, $d_\Omega > r_i + p_i$.

The inequality (12) is algebraically equivalent to

$$rest(\Omega, c_i) > c_i(d_\Omega - r_i - p_i) \tag{16}$$

and from the inequality $d_\Omega > r_i + p_i$ we have $rest(\Omega, c_i) > 0$. From the inequality $r_i < r_\Omega$ it follows that

$$r_\Omega + \left\lceil \frac{1}{c_i} rest(\Omega, c_i) \right\rceil > r_i \ . \tag{17}$$

□

According to this theorem, once edge-finding is at fix point, the set of tasks used to detect the extended edge-finding condition can also serve to compute a potential update value. We use this approach to design a sound extended edge-finding algorithm. The algorithm may not find the best adjustments at the first run, but it must after a finite number of iterations. This "lazy" approach has recently been used with success to reduce the complexity of edge-finding filtering rule for cumulative resources in Kameugne et al. (2011).

3.2 The algorithm

Our new algorithm works essentially as follows. For each task i, the algorithm identifies the task intervals Ω of minimum slack which satisfy $r_i < r_\Omega$ and $d_\Omega \leq d_i$. If the extended edge-finding conditions hold for the set of tasks Ω and the task i, then Ω is used by the algorithm to update r_i. The means of locating the task intervals with the minimum *slack* is different from the one used in Kameugne et al. (2011).

Definition 4 (Kameugne et al. (2011))**.** *Let Ω be a task set of an E-feasible CuSP. The* slack *of the task set Ω, denoted SL_Ω, is given by:*

$$SL_\Omega = C(d_\Omega - r_\Omega) - e_\Omega.$$

Definition 5. *Let i and L be two tasks of an E-feasible CuSP with $r_i < r_L$. $\delta(L,i)$, where $d_{\delta(L,i)} \leq d_i$, defines the largest task interval with the* minimum slack*: for all $U \in T$ such that $d_U \leq d_i$*

$$C(d_{\delta(L,i)} - r_L) - e_{\Omega_{L,\delta(L,i)}} \leq C(d_U - r_L) - e_{\Omega_{L,U}} \ . \tag{18}$$

Given two tasks i and L, we are looking for the upper bound $\delta(L,i)$ of the largest task intervals of minimum slack; this differs from Kameugne et al. (2011), where the lower bound was used instead.

Algorithm 1 performs the extended edge-finding rule for all tasks $i \in T$ with the time complexity $\mathcal{O}(n^2)$. It works as follows:

- The outer loop (line 3) iterates through the tasks $L \in T$ forming the possible lower bounds of the task intervals.
- The inner loop (line 5) selects the tasks $i \in T$ that comprise the possible upper bounds for the task intervals, in non-decreasing order of deadlines. If $r_L \leq r_i$, then the energy and the slack of $\Omega_{L,i}$ are calculated. The slack is then compared to the slack of $\Omega_{L,\delta(L,i)}$; if the new slack is higher, $\delta(L,i)$ becomes i (line 10). If $r_L > r_i$, then the extended edge-finding condition is tested at line 12. If this condition holds, then the potential update value of the release date of i is calculated (line 15), based on the current $\delta(L,i)$ and the release date of task i is updated (line 16).
- At the next iteration of the outer loop, $\delta(L,i)$ is re-initialized.

4 Proof of correctness

Before showing that Algorithm 1 is correct, let us prove some properties of its inner loops.

Algorithm 1: Extended edge-finding algorithm in $\mathcal{O}(n^2)$ time and $\mathcal{O}(n)$ space

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

1 **for** $i \in T$ **do**
2 $LB'_i := r_i$;
3 **for** $L \in T$ **do**
4 $Energy := 0, maxEnergy := 0, d_\delta := r_L$;
5 **for** $i \in T$ *by non-decreasing deadlines* **do**
6 **if** $(r_L \leq r_i)$ **then**
7 $Energy := Energy + e_i$;
8 $minSlack :=$ **if** $(maxEnergy \neq 0 \wedge d_\delta > r_L)$ **then** $C(d_\delta - r_L) - maxEnergy$ **else** $+\infty$;
9 **if** $(C(d_i - r_L) - Energy \leq minSlack)$ **then**
10 $maxEnergy := Energy, d_\delta := d_i$;
11 **else**
12 **if** $(maxEnergy + c_i(r_i + p_i - r_L) > C(d_\delta - r_L))$ **then**
13 **if** $(r_i + p_i < d_\delta \wedge r_i + p_i > r_L \wedge d_\delta > r_L$ **then**
14 $rest := maxEnergy - (C - c_i)(d_\delta - r_L)$;
15 $upd := r_L + \lceil rest/c_i \rceil$;
16 $LB'_i := \max(LB'_i, upd)$;
17 **for** $i \in T$ **do**
18 $r_i := LB'_i$;

Proposition 3. *For each task i and for each task L with $r_i < r_L$, Algorithm 1 calculates a potential update to r_i based on the task intervals of minimum slack, such that*

$$\left.\begin{array}{c} r_i + p_i < d_{\delta(L,i)} \\ r_i + p_i > r_L \\ d_{\delta(L,i)} > r_L \\ e_{\Omega_{L,\delta(L,i)}} + c_i(r_i + p_i - r_L) > C(d_{\delta(L,i)} - r_L) \end{array}\right\} \implies upd = r_L + \left\lceil \text{rest}(\Omega_{L,\delta(L,i)}, c_i) \cdot \frac{1}{c_i} \right\rceil \quad . \tag{19}$$

Proof. Proof Let $i \in T$ be any task. Each choice of $L \in T$ in the outer loop (line 3) starts with the values $d_{\delta(L,i)} = r_L$ and $maxEnergy = 0$ (line 4). The inner loop at line 5 iterates through all tasks $i' \in T$ (T sorted in non-decreasing order of deadlines). For any task $i' \in T$ such that $d_{i'} \leq d_i$, if $r_L \leq r_{i'}$, then $i' \in \Omega_{L,i'}$, so $e_{i'}$ is added to $Energy$ (line 7). Hence $Energy = e_{\Omega_{L,i'}}$ at each iteration. The test on line 9 ensures that $d_{\delta(L,i)}$ and $maxEnergy = e_{\Omega_{L,\delta(L,i)}}$ are updated to reflect $\delta(L,i)$ for the current task interval $\Omega_{L,i'}$. Therefore, at the i^{th} iteration of the inner loop, if $r_i < r_L$ then line 14 computes $rest_{L,i} =$

$\text{rest}(\Omega_{L,\delta(L,i)}, c_i)$, and the potential update value :

$$\left.\begin{array}{r} r_i + p_i < d_{\delta(L,i)} \\ r_i + p_i > r_L \\ d_{\delta(L,i)} > r_L \\ maxEnergy + c_i(r_i + p_i - r_L) > C(d_{\delta(L,i)} - r_L) \end{array}\right\} \implies upd = r_L + \left\lceil \text{rest}_{L,i} \cdot \frac{1}{c_i} \right\rceil \quad . \tag{20}$$

The conditions of line 12 and line 13 imply that $rest_{L,i} > 0$ which justifies the computation of the potential update value at line 15. Therefore formula (19) holds and the proposition is correct since $maxEnergy = e_{\Omega_{L,\delta(L,i)}}$. □

We now provide a proof that at the fix point of the edge-finding rule, the extended edge-finding condition (EEF) can be checked using the minimum slack of task intervals, as provided in Definition 5.

Theorem 2. *At the fix point of the edge-finding rule, for any task $i \in T$ and set of tasks $\Omega \subseteq T \setminus \{i\}$,*

$$r_i < r_\Omega < r_i + p_i \ \wedge \ e_\Omega + c_i(r_i + p_i - r_\Omega) > C(d_\Omega - r_\Omega) \tag{21}$$

if and only if

$$r_i < r_L < r_i + p_i \ \wedge \ e_{\Omega_{L,\delta(L,i)}} + c_i(r_i + p_i - r_L) > C(d_{\delta(L,i)} - r_L) \tag{22}$$

for some task $L \in T$ such that $r_i < r_L$, and $\delta(L,i)$ as specified in Definition 5.

Proof. Proof Let $i \in T$ be any task. We start by demonstrating that (21) implies (22). Assume there exists a subset $\Omega \subseteq T \setminus \{i\}$ such that

$$r_i < r_\Omega < r_i + p_i \wedge e_\Omega + c_i(r_i + p_i - r_\Omega) > C(d_\Omega - r_\Omega) \tag{23}$$

By (EEF), we have $\Omega \lessdot i$. By Proposition 2, there exists a task interval $\Omega_{L,U} \lessdot i$, such that $r_i \leq r_L < r_i + p_i$. By Definition 5 we have

$$C(d_{\delta(L,i)} - r_L) - e_{\Omega_{L,\delta(L,i)}} \leq C(d_U - r_L) - e_{\Omega_{L,U}}. \tag{24}$$

Adding $-c_i(r_i + p_i - r_L)$ to both sides of (24) and using the fact that

$$C(d_U - r_L) - e_{\Omega_{L,U}} - c_i(r_i + p_i - r_L) < 0 \tag{25}$$

it follows that

$$C(d_{\delta(L,i)} - r_L) < e_{\Omega_{L,\delta(L,i)}} + c_i(r_i + p_i - r_L). \tag{26}$$

Now we show that (22) implies (21). Let $L \in T$ be a task such that $r_i < r_L < r_i + p_i$, and $\delta(L,i) \in T$ be tasks that satisfy (22). It is obvious that (21) is satisfied for $\Omega = \Omega_{L,\delta(L,i)}$. □

Proposition 3 has shown that $\delta(L, i)$ and the minimum slack are correctly computed by the loop at line 5. Combined with Theorem 1, this justifies the use of the characteristics of the task intervals of minimum slack on line 12 and 13 to check (EEF). Thus, at the fix point of the edge-finding rule, for every task i, Algorithm 1 correctly detects the sets $\Omega \subseteq T \setminus \{i\}$ for which the rule (EEF) demonstrates $\Omega \Ll i$.

A complete extended edge-finder would always choose the set Θ for each task i that yielded the strongest update to the bound of i. In the following theorem, we demonstrate that our algorithm has the slightly weaker property of soundness; that is, the algorithm updates the bounds correctly, but might not always make the strongest adjustment to a bound at the first iteration.

Theorem 3. *At the fix point of the edge-finding rule, for every task $i \in T$, and given the strongest lower bound LB_i as specified in Definition 1, Algorithm 1 computes some lower bound LB'_i, such that $r_i < LB'_i \leq LB_i$ if $r_i < LB_i$, and $LB'_i = r_i$ if $r_i = LB_i$. It time complexity is $\mathcal{O}(n^2)$ in time.*

Proof. Proof Let $i \in T$ be any task.

LB'_i is initialized to r_i. We have $LB'_i \geq r_i$ since the value LB'_i is only updated by $\max(upd, LB'_i)$ (line 16) after each detection (line 12 and 13). If the equality $LB_i = r_i$ holds, then no detection is found by the extended edge-finding rule, therefore by Algorithm 1, and thus $LB'_i = r_i$ holds from the loop at line 17. In the rest of the proof, we assume that $r_i < LB_i$. The extended edge-finding condition is detected by the pair (Ω, i) where Ω is a set of tasks and the release date of task i is updated. According to Theorem 2, there exist tasks L and $\delta(L, i)$ such that the relation $\Omega_{L,\delta(L,i)} \Ll i$ is detected by Algorithm 1. According to Theorem 1, the set $\Omega_{L,\delta(L,i)}$ can also serve to update r_i, exactly as we have done in our algorithm. As demonstrated by Proposition 3, $\delta(L, i)$ and the minimum slack are correctly computed by Algorithm 1. Therefore, after the detection condition is fulfilled at lines 12 and 13, the release date of task i is updated to $LB'_i = upd > r_i$. Hence, Algorithm 1 correctly detects and adjusts the release date of task i. Thus, Algorithm 1 is sound.

It is obvious that the complexity of Algorithm 1 is $\mathcal{O}(n^2)$ in time and $\mathcal{O}(n)$ in space. Indeed, the array T can be sorted in $\mathcal{O}(n \log n)$ time complexity and the outer loop (line 3) contain an inner loop of complexity $\mathcal{O}(n)$ in time. Therefore, the complexity of the algorithm is $\mathcal{O}(n \log n) + \mathcal{O}(n^2) = \mathcal{O}(n^2)$ in time. □

5 Comparison with the conjunction of the standard and extended edge-finding rules

According to Theorem 3, at the fix point of the edge-finding rule, Algorithm 1 will always make some update to r_i if an update is justified by the extended edge-finding rule, although possibly not always the strongest update. As there are a finite number of updating sets, the combination of Algorithm 1 and

the quadratic edge-finding of Kameugne et al. (2011) must reach the same fix point as conjunction of the standard and the extended edge-finding rule. This "lazy" approach has recently been used to speedup the filtering mechanism (Kameugne et al., 2011; Vilím, 2011) and to reduce the complexity of not-first/not-last filtering for cumulative resources (Kameugne and Fotso, 2013a; Schutt and Wolf, 2010). Theorem 4 proves that when the combination of Algorithm 1 with the quadratic edge-finding algorithm of Kameugne et al. (2011) reaches the fix point, then the conjunction of the standard and extended edge-finding rules cannot propagate anything.

Theorem 4. *When the combination of the edge-finding algorithm of Kameugne et al. (2011) and the quadratic extended edge-finding algorithm of Algorithm 1 reaches the fix point, then both standard and extended edge-finding rule cannot propagate anything.*

Proof. Proof by contradiction we will assume that the combination of the edge-finding algorithm of (Kameugne et al., 2011, Algorithm 1) and the quadratic extended edge-finding of Algorithm 1 reaches a fix point, however standard and extended edge-finding rule can be propagated; i.e., there exist an i, Ω, and $\Theta \subseteq \Omega$ such that one of (EF) or (EF1) or (EEF) holds, and (4) improves r_i using Θ. In this case, we prove that either the edge-finding algorithm of Kameugne et al. (2011) or the quadratic extended edge-finding algorithm of Algorithm 1 can updated the time bounds of task i. We distinguish two cases:

1. (EF) **or** (EF1) **holds and the time bound of a task is updated.** The soundness of the edge-finding algorithm of Kameugne et al. (2011) implies that the combination of this algorithm and Algorithm 1 updates the time bounds of some task, thus contradicting our hypothesis.
2. (EEF) **holds and the time bound of a task is updated.** We distinguish two subcases.
   (a) **The fix point of** (EF) **and** (EF1) **is reached.** The soundness of Algorithm 1 at the fix point of (EF) and (EF1) implies that the combination of this algorithm with the edge-finding algorithm of Kameugne et al. (2011) improves the time bounds of a task, thus contradicting our hypothesis.
   (b) **The fix point of** (EF) **or** (EF1) **is not reached.** We return to the first item and the contradiction is detected.

□

6 Experimental results

We evaluate the usefulness of the extended edge-finding rule in two areas. The first is efficiency, as demonstrated by both running time and search tree size

	static			dynamic		
	time	**nodes**	**solve**	**time**	**nodes**	**solve**
CalcEEF	7	769	1001	15	825	1042
QuadEF	369	864	1044	453	948	1075
QuadEEF	77	922	1002	0	948	1075
TTEF	597	917	1046	611	980	1074

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 static or dynamic branching schemes.

when solving standard benchmark problems. The second is pruning strength, evaluated on a large number of randomly generated domains.

Edge-finding is one of several algorithms typically used in conjunction for the filtering of the CUMULATIVE constraint. For our tests, we implemented the following four versions of CUMULATIVE using the Gecode 3.7.3 (2012) constraint solver:

1. **QuadEF**: A sequence of three filters: a timetabling algorithm (Baptiste et al., 2001), overload checking (Vilím, 2009b), and the $\mathcal{O}(n^2)$ edge-finding algorithm from Kameugne et al. (2011).

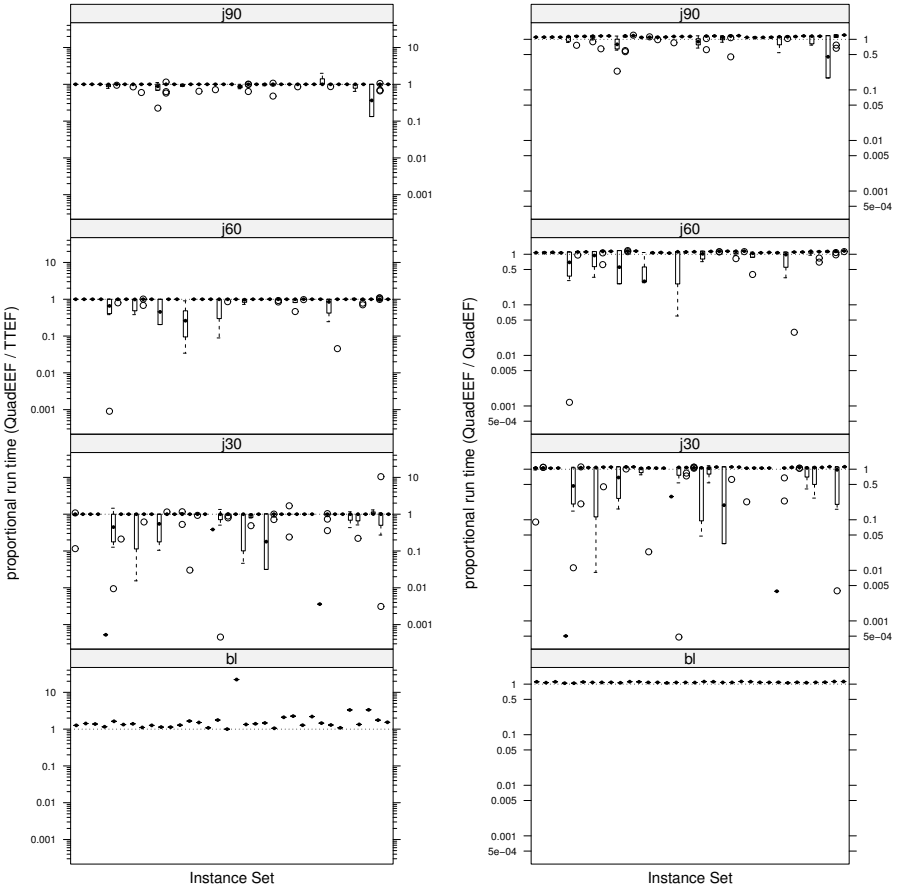
2. **CalcEEF**: The same sequence of filters, but replacing the edge-finding algorithm with the $\mathcal{O}(kn^2)$ combined edge-finding and extended edge-finding algorithm of Mercier and Van Hentenryck (2008).

3. **QuadEEF**: An extension of **QuadEF** to perform extended edge-finding as well. At each node of the search tree, if the fix point of the edge-finding algorithm is reached, then Algorithm 1 is executed. This process is repeated until the global fix point is reached.

4. **TTEF**: Timetable edge-finding (Vilím, 2011).

6.1 Efficiency

For benchmarks, we used resource-constrained project scheduling problems (RCPSP). An RCPSP consists of: a set of resources of finite capacities; a set of tasks of given processing time;, 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 equals to the horizon.



(a) QuadEEF vs. TTEF (b) QuadEEF vs. QuadEF

Figure 2: Comparison of runtimes when using static branching, sorted by instance.

Tests were performed on the RCPSP single-mode J30, J60 and J90 test sets of the well-established benchmark library PSPLib (PSPLib, 2013) as well as on the library of Baptiste and Le Pape (2000) (BL). The data sets J30, J60 and J90 consists 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, we have modeled each problem 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 (Aggoun and Beldiceanu, 1993).

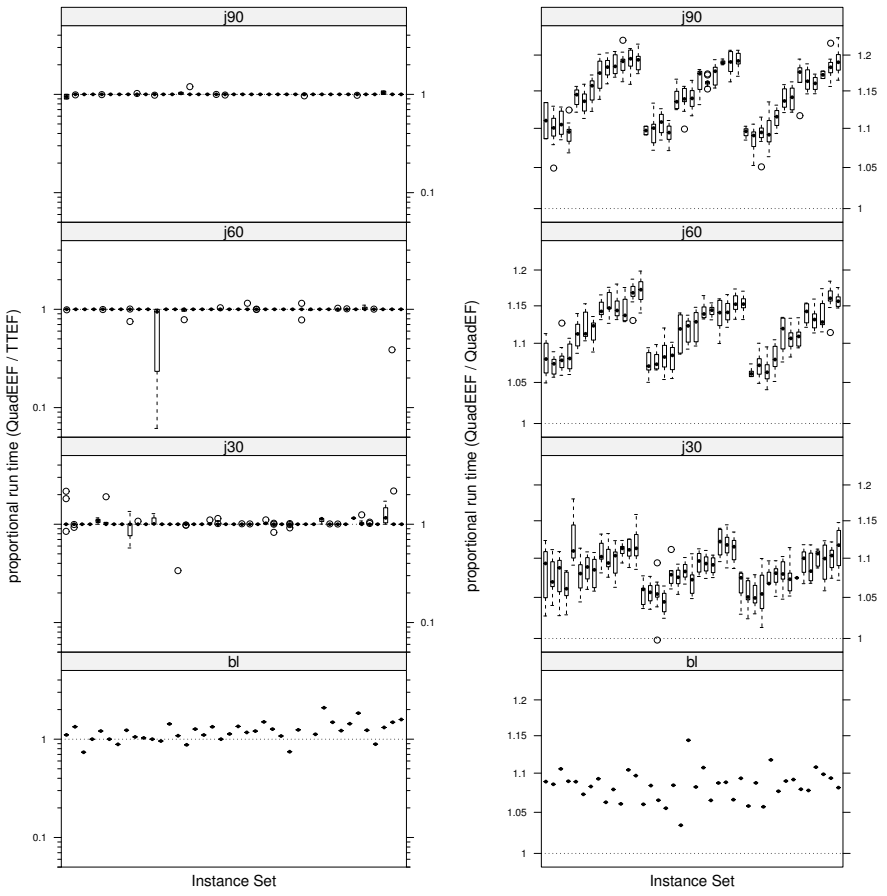
Dynamic branching schemes are the most used branching strategy in CP, as they typically result in smaller search trees. However, when comparing filtering algorithms of differing pruning strengths, dynamic branching can be misleading: in some cases the domain resulting from weaker pruning may result in a choice point which results in 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, we consider two branching models:

a. 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 Gecode 3.7.3 (2012).

b. For static branching, we selected the first unassigned variable, and the smallest value in the domain.

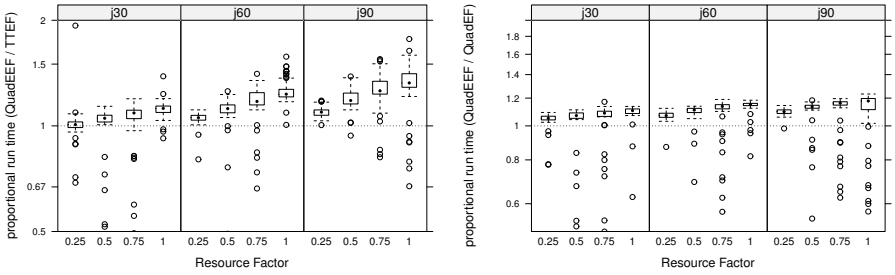
All tests were performed on a 3.07 GHz Intel Core i7 processor running OpenSUSE Linux. The code was implemented in C++, using the Gecode constraint programming toolkit, version 3.7.3, and compiled with GCC 4.5.1. For each benchmark instance, we used branch and bound search to minimize the makespan, stopping 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.

Table 1 summarizes the results for all PSPLib and BL instances. **TTEF** perfromed the strongest of the four algorithms, followed by **QuadEF**; **QuadEEF** was generally somewhat slower, while the older **CalcEEF** unsurprisingly performed the worst. As can be observed form Figure 2, using static branching the running time of the three algorithms were nearly the same in the majority of instances, with the additional work of the extended edge-finding filtering requiring only slightly more time. In contrast, the 77 instances in which **QuadEEF** had the lowest running time, it frequently beat the other two filtering algorithms by a more substantial margin. Unfortunately, use of a dynamic



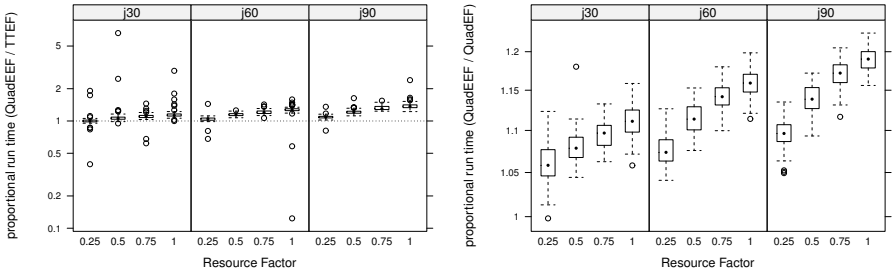
(a) QuadEEF vs. TTEF (b) QuadEEF vs. QuadEF

Figure 3: Comparison of runtimes when using dynamic branching, sorted by instance.



(a) QuadEEF vs. TTEF (b) QuadEEF vs. QuadEF

Figure 4: Comparison of runtimes on PSP instances using static branching, with instances grouped by resource factor. Several outliers are omitted form the bottom of the graphs for purposes of scaling.



(a) QuadEEF vs. TTEF (b) QuadEEF vs. QuadEF

Figure 5: Comparison of runtimes on PSP instances using dynamic branching, with instances grouped by resource factor.

branching scheme appears to hide any advantage of **QuadEEF**; in the instances tested, it was consistently slower than **QuadEF** alone (although never by a large margin).

It is not clear how to categorize these instances where **QuadEEF** outperforms other filtering algorithms. The instances in the PSPLib data set may be categorized according to several criteria; for example, Figures 4 and 5 show the benchmarks grouped by *resource factor* as defined in Kolisch and Sprecher (1996), a measure of the proportion of the available resource required by the tasks. The BL instances are omitted, as the resource factor of those instances is not included with the benchmark. The plot is focused on the bulk of the instances, in which **QuadEEF** ran slightly slower than its competitors (most instances where **QuadEEF** was faster occur as outliers, omitted from the graph for space). We observe that for these instances, the proportional increase in running time against both **QuadEF** and **TTEF** is directly proportional to the resource factor of the test instances (this accounts for the periodicity observed

# tasks	**QuadEF**		TTEF	
	weaker	**stronger**	**weaker**	**stronger**
20	6	0	912	43826
30	8	0	869	43430
40	15	0	1907	89240
50	31	0	3450	126631
60	35	0	5424	152114

Table 2: Comparison of pruning strength on randomly generated instances. Columns indicate number of instances (out of 1 million generated) where propagation by **QuadEF** (resp. **TTEF**) led to a weaker or stronger domain than propagation by **QuadEEF**.

in Figure 3(b) as well — it is present in all the runtime plots, but the magnitude of the difference caused by resource factor is too small in comparison to running time differences stemming from different tree sizes in the other plots). Clearly, **QuadEEF** suffers more from the added complexity found in problems with a higher density of tasks; however, resource factor does not appear to correlate with the incidence of outliers in which **QuadEEF** is faster. No correlation is observed with the other known parameters of the PSPLib instances, either.

To more clearly demonstrate the effect of the additional filtering of the extended edge-finding rule, we consider the size of the resulting search tree. In the majority of instances, no difference in search tree size (as indicated by node count) is observed among any of the algorithms. Figure 6 shows only the instances where using **QuadEEF** resulted in a different number of nodes then either **QuadEF** or **TTEF**. Interestingly, despite the fact that these later two algorithms use substantially different filtering rules, the difference in node counts compared to the extended edge-finding rule was quite similar, with **TTEF** resulting in fewer nodes than standard edge-finding in most of these cases. The implication is that timetable edge-finding performs some, but not all, of the additional filtering justified by the extended edge-finding rule. In these cases, application of extended edge-finding appears to result in additional filtering. The faster running times of **TTEF** when compared to **QuadEEF** appear to result from a sacrifice of filtering strength in order to gain efficiency.

6.2 Pruning Strength

To more directly evaluate the relative pruning strength of the various algorithms, we generated a large number of random cumulative instances. For each instance, between 20 and 60 tasks were generated with random lengths and resource requirements. Each set of tasks was tested for e-feasibility on a single cumulative resource of capacity 10, using the standard overload-checking algo-

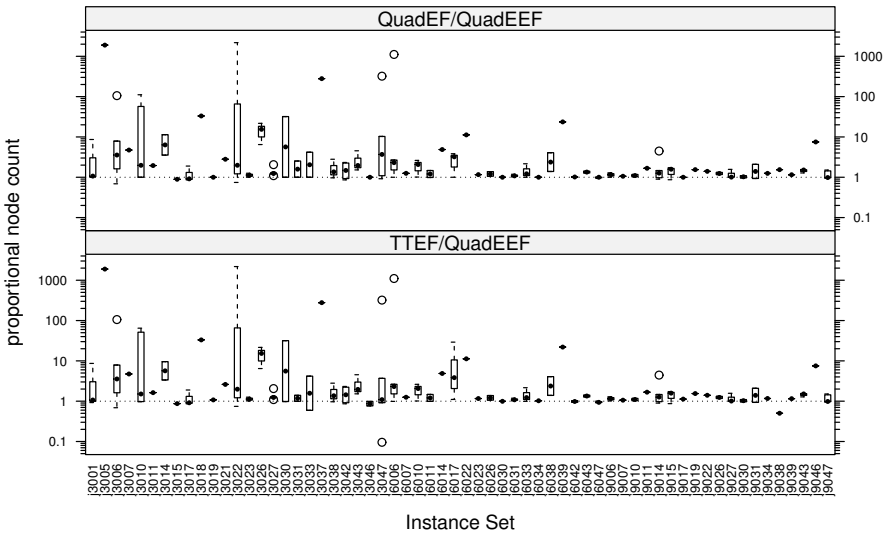


Figure 6: Proportional difference in node count of **QuadEF** and **TTEF** compared to **QuadEEF**, with static branching. Only instances with some reported difference in search tree sizes are shown, and BL results are omitted.

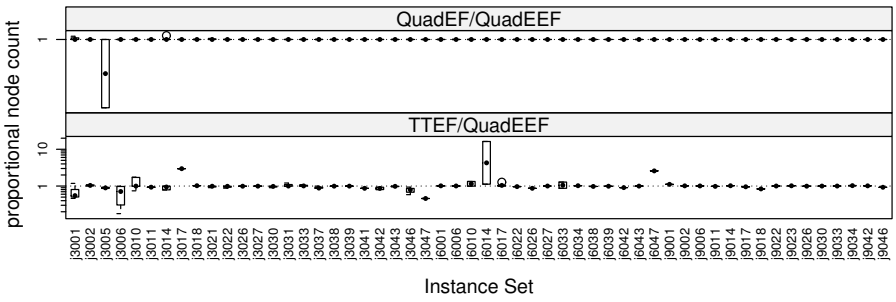


Figure 7: Using dynamic branching, almost no differences in search tree size were observed. Once again, BL results are omitted.

rithm. We generated a total of 1 million e-feasible instances using this naive approach, and then ran **QuadEF**, **TTEF**, and **QuadEEF** on each instance. The resulting domains were evaluated by comparing the amount of reduction in the domains of the start times.

Results, summarized in Table 2, demonstrate that **TTEF** results in stronger pruning much more frequently, not quite one half of all generated instances. In a much smaller number of cases, the conjunction of edge-finding and extended edge-finding resulted in stronger pruning. Most interestingly, extended edge-finding resulted in additional pruning over standard edge-finding in just under 100 instances, a quite small proportion. This suggests that the condition detected by the extended edge-finding rule is, in fact, quite rare. Nevertheless, the increased performance of **QuadEEF** over **QuadEF** on the benchmark instances suggests that in more structured test cases this condition is more likely to arise.

7 Conclusion

We have presented a sound quadratic extended edge-finding filtering algorithm, complement of the edge-finding algorithm for cumulative scheduling. The combination of this algorithm with the edge-finder of Kameugne et al. (2011) reaches the same fix point as the conjunction of the standard and extended edge-finding rules, although it may take more propagations to do so. Experimental results on a standard benchmark suite demonstrate that, while slightly slower in most instances, our algorithm is substantially faster on a few instances, and can lead to a significant reduction in search tree size.

Future work will focus on improve the complexity of this algorithm from $\mathcal{O}(n^2)$ to $\mathcal{O}(n \log n)$. Another idea is to apply the same approach to the not-first/not-last rule to implementing a similar sound quadratic algorithm for this rule.

References

Aggoun, A., & Beldiceanu, N. (1993). Extending CHIP in order to solve complex scheduling and placement problems. *Mathematical and Computer Modelling*, 17(7), 57–73.

Baptiste, P., & Le Pape, C. (2000). Constraint propagation and decomposition techniques for highly disjunctive and highly cumulative project scheduling problems. *Constraints*, 5(1), 119–139.

Baptiste, P., Le Pape, C., & Nuijten, W.P.M. (2001). *Constraint-based scheduling: applying constraint programming to scheduling problems*. Berlin: Springer.

Caseau, Y., & Laburthe, F. (1994). Improved CLP scheduling with task intervals. In P. Van Hentenryck (Ed.), *ICLP 1994—Logic Programming* (pp. 369–383). MIT Press.

Gecode. (2012) http://www.gecode.org. Accessed 30 August, 2012.

Kameugne, R., Fotso, L.P., Scott, J. & Ngo-Kateu, Y. (2011) A quadratic edge-finding filtering algorithm for cumulative resource constraints, In: Lee, J. (ed.) *CP 2011*, LNCS vol. 6876, pp. 478–492. Springer, Heidelberg.

Kameugne, R., Fotso, L.P., Scott, J. & Ngo-Kateu, Y. (2013b) A quadratic edge-finding filtering algorithm for cumulative resource constraints, Extended version of the CP paper, Submitted for revision in Constraints.

Kameugne, R. and Fotso, L.P. (2013a) A Cumulative Not-First/Not-Last Filtering Algorithm in $\mathcal{O}(n^2 \log(n))$, Indian J. Pure Appl. Math., **44**(1): pp 95-115. Springer-Verlag (2013). DOI 10.1007/s13226-013-0005-z.

Kolisch, R., & Sprecher, A. (1996). PSPLIB - a project scheduling library. *European Journal of Operational Research*, 96,205–216.

Mercier, L., & Van Hentenryck, P. (2008). Edge finding for cumulative scheduling. *INFORMS Journal on Computing*, 20(1),143–153.

Nuijten, W.P.M. (1994). *Time and resource constrained scheduling*. PhD thesis, Technische Universiteit Eindhoven.

PSPLib. (2013)—project scheduling problem library. http://129.187.106.231/psplib/

Schutt, A., & Wolf, A. (2010). 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, LNCS*, (vol. 6308, pp. 445–459). Berlin: Springer.

Vilím, P. (2009a). 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, LNCS*, (vol. 5732, pp. 802–816). Berlin: Springer.

Vilím, P. (2009b). Max energy filtering algorithm for discrete cumulative resources. In W.J. van Hoeve, & J.N. Hooker (Eds.), *CPAIOR 2009—Integration of AI and OR Techniques in Constraint Programming, LNCS*, (vol. 5547, pp. 294–308). Berlin: Springer.

Vilím, P. (2011). Timetable edge finding filtering algorithm for discrete cumulative resources. In T. Achterberg, & J.C. Beck (Eds.), *CPAIOR 2011—Integration of AI and OR Techniques in Constraint Programming, LNCS*, (vol. 6697, pp. 230–245). Berlin: Springer.

Wolf, A., Schrader, G. (2006). $O(n \log n)$ overload checking for the cumulative constraint and its application. In M. Umeda, A. Wolf, O. Bartenstein, U. Geske, D. Seipel, O. Takata (Eds.), *INAP 2005—Applications of Declarative Programming for Knowledge Management, LNCS*, (vol. 4369, pp. 88–101). Berlin: Springer.