UPPSALA
UNIVERSITET

Roger Kameugne[1,2]
Laure Pauline Fotso[3]
Joseph Scott[4]
Youcheu Ngo-Kateu[3]

[1] Dept of Mathematics
University of Maroua
[2] Dept of Mathematics,
University of Yaoundé
[3] Dept of Computer Science,
University of Yaoundé
[4] ASTRA Group,
Computing Science Div,
University of Uppsala

Quadratic algorithm for cumulative edge-finding

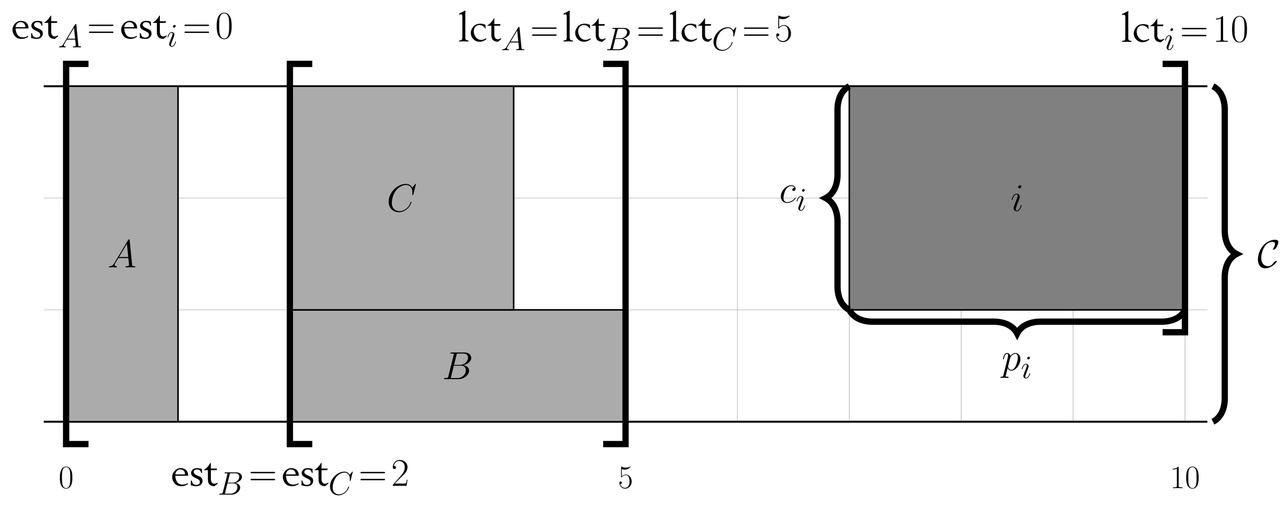
Introduction

Edge-finding is a commonly used filtering algorithm for resource constrained scheduling problems. For cumulative scheduling problems (i.e., resources with a capacity greater than 1), the state of the art was until recently the Θ-tree algorithm by Vilím[3], with a complexity of $\mathcal{O}(kn \log n)$, where n is the number of tasks and k the number of distinct resource requirements.

We present a new edge-finding algorithm of $\mathcal{O}(n^2)$ complexity, and demonstrate that in practice it outperforms earlier algorithms, while offering comparable performance to Vilím's more recent timetable edge-finding [4].

Cumulative Resource Scheduling

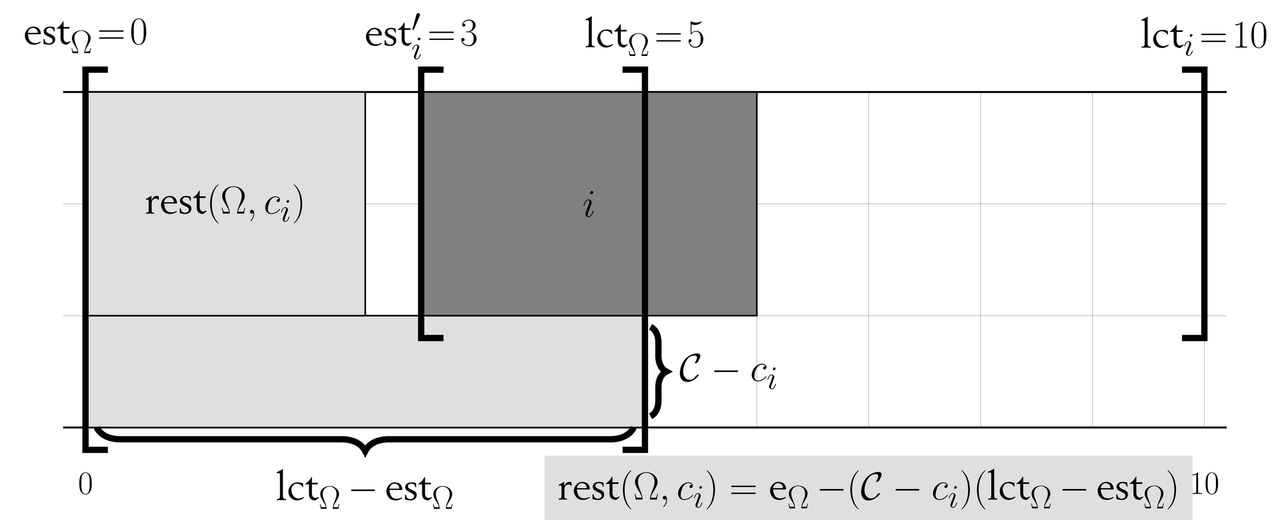
In a resource constrained scheduling problem, a set of tasks share a finite resource of capacity. Tasks have fixed resource requirements and durations, and a domain of start times. The problem is to assign each task a start time such that the capacity of the resource is never exceeded.



Cumulative scheduling is NP-Complete, but there exist several polynomial filtering algorithms for elastic relaxations of the constraint, with edge-finding being one of the most common.

Edge-Finding

Edge-finding attempts to deduce precedence relations between a set of tasks Ω and another task $i \notin \Omega$. If there is not enough capacity in the time between est_Ω and lct_Ω to schedule all the tasks in Ω as well as i, then i must end after the end (or begin before the beginning) of all tasks in Ω. This deduction justifies a tightening of the earliest start time (or latest completion time) of i.



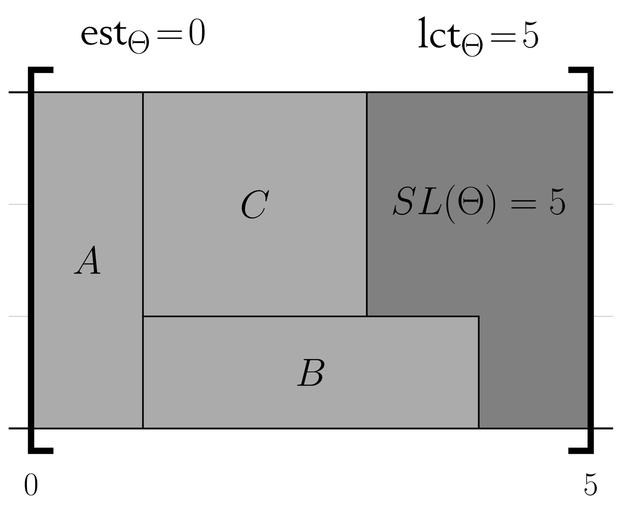
Not all sets of tasks need to be considered, only *task intervals*:

$$\Omega_L^U = \{t \in T \mid \mathrm{est}_t \geq \mathrm{est}_L, \mathrm{lct}_t \leq \mathrm{lct}_U\}$$

Determining that Ω precedes i is not enough, as the maximal update to est_i may come from a subset of Ω. Hence, the edge-finding rule, for each task $i \in T$[2]:

$$\mathrm{est}'_i = \max_{\substack{\Omega \subseteq T \\ i \notin \Omega \\ \mathcal{C}(\mathrm{lct}_\Omega - \mathrm{est}_{\Omega \cup \{i\}}) < e_{\Omega \cup \{i\}}}} \quad \max_{\substack{\Theta \subseteq \Omega \\ \mathrm{rest}(\Theta, c_i) > 0}} \mathrm{est}_\Theta + \left\lceil \frac{\mathrm{rest}(\Theta, c_i)}{c_i} \right\rceil \qquad \text{(EF)}$$

Minimum slack intervals



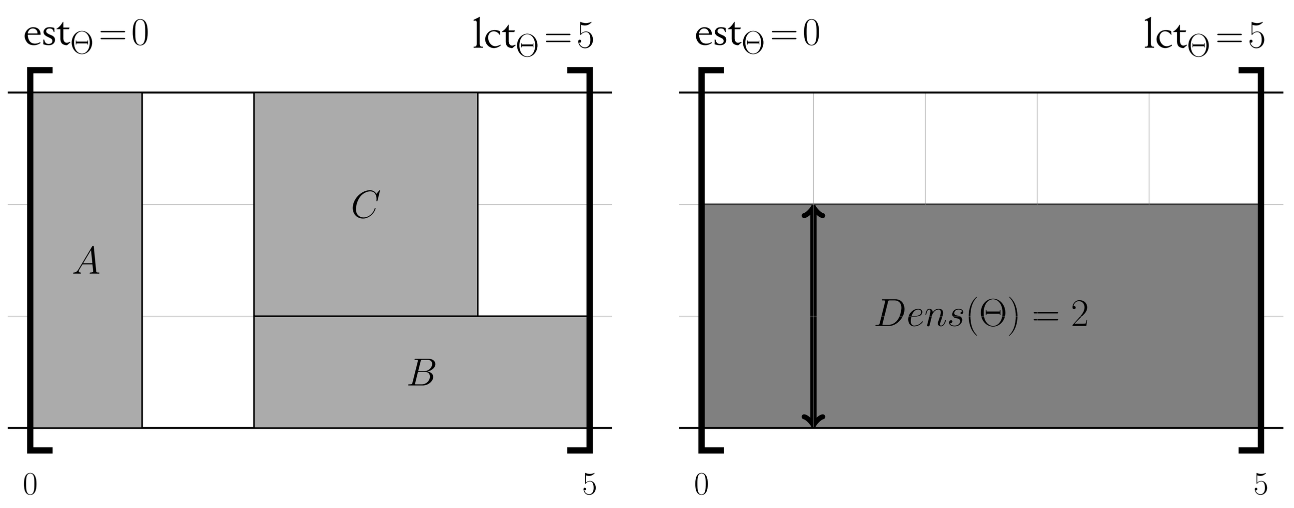
To find the intervals that lead to the best update, edge-finders such as [2, 1, 3] consider interval *slack*: the amount of capacity unused by the task interval.

Intuitively: the less slack an interval has, the more likely it is to lead to a good bound update.

The minimum slack interval can be used to check the condition on the outer maximization of (EF). We also consider the interval with the least slack of any interval Θ_ℓ^u such that $\mathrm{est}_\ell \leq \mathrm{est}_i$; as we prove in the paper, if the Θ from the inner maximization of (EF) has $\mathrm{est}_\Theta \leq \mathrm{est}_i$, then Θ will be this interval of minimum slack.

Maximum density intervals

To correctly locate Θ when $\mathrm{est}_\Theta > \mathrm{est}_i$, we introduce the notion of *maximum density*, where density measures the average resource usage by a task interval.



We prove that when $\mathrm{est}_\Theta > \mathrm{est}_i$, the interval Θ_ℓ^u with $\mathrm{est}_\ell > \mathrm{est}_i$ with the maximum density is either (a) the interval Θ that satisfies the inner maximization of (EF), or (b) an interval where $\mathrm{est}_\ell > \mathrm{est}_\Theta$, which must also lead to a (possibly weaker) update of est_i. In the latter case, we further demonstrate that the algorithm moves closer to finding the true maximal update on each propagation, requiring in the worst case propagations on the order of $\mathcal{O}(n)$; experimental results suggest that, in practice, our algorithm rarely requires more propagations than other edge-finders.

Results

The algorithm was implemented in Gecode 3.6.0, and compared with several previous edge-finding filters. The j30, j60 and j90 problem sets of the Project Scheduling Problem Library (PSPLib) were used as benchmarks. Runtimes are shown for the time required to find the best solution on a 3.07 ghz Intel Core i7 processor, with a time limit of 300 seconds.

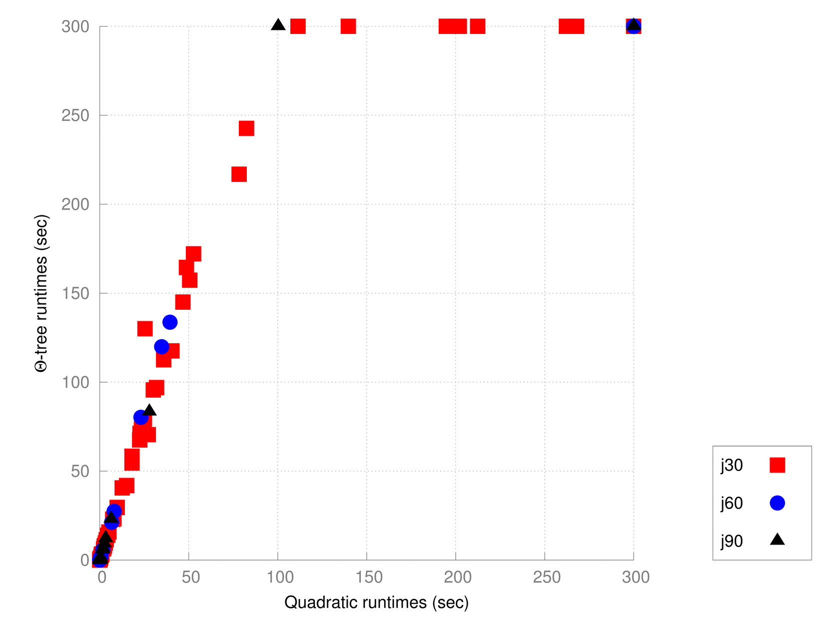


Figure 1: Θ-tree edge-finding[3] has a complexity of $\mathcal{O}(kn \log n)$, where k is the number of distinct capacity requirements among the tasks, which is not strictly dominated by the $\mathcal{O}(n^2)$ complexity of our algorithm, especially for small k. Nevertheless, in terms of runtime our algorithm consistently outperformed the Gecode Θ-tree edge-finder by a factor of three.

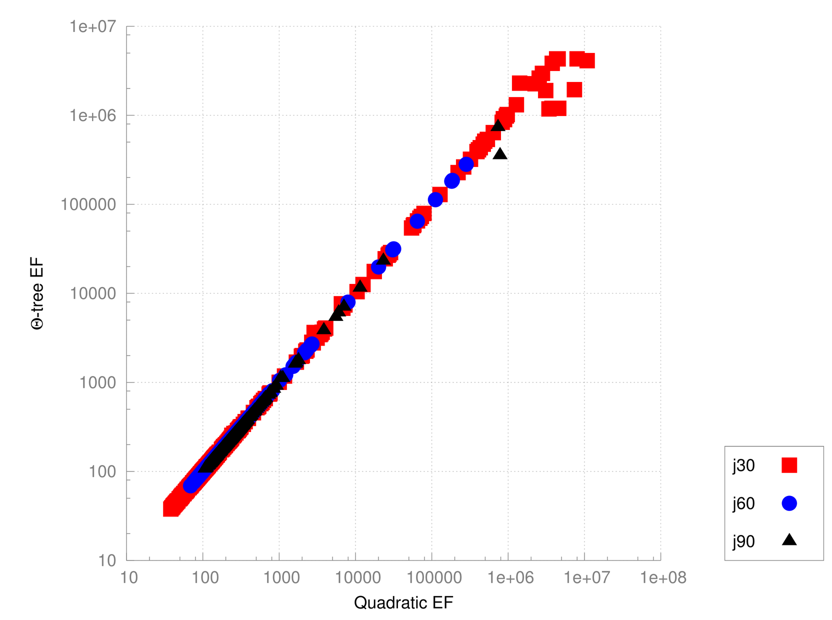


Figure 2: Despite the theoretical possibility that our algorithm would require additional propagations to reach the same fixpoint as earlier algorithms (specifically for the Θ-tree algorithm[?] here), in practice the number of propagations varied by less than one percent in almost all cases.

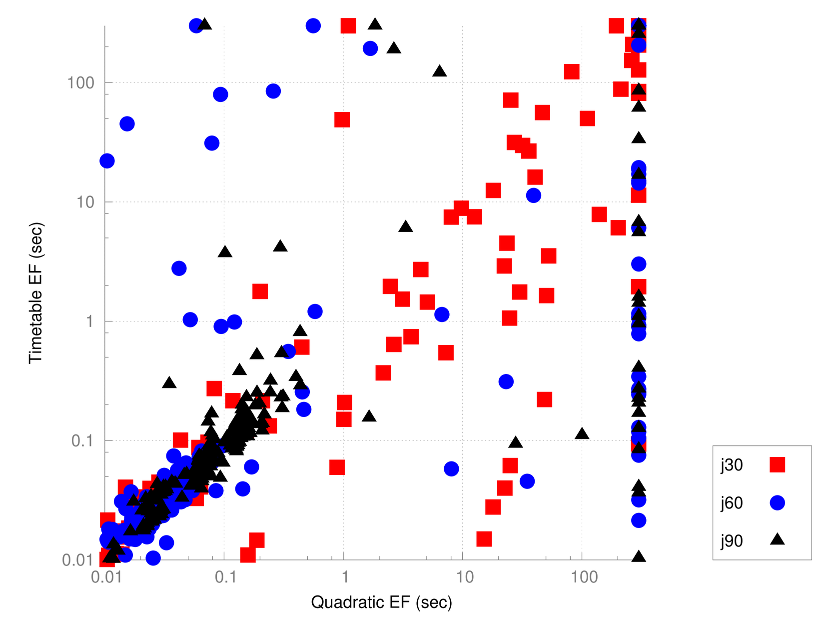


Figure 3: A comparison with Vilím's $\mathcal{O}(n^2)$ timetable edge-finding[4] was less clear-cut. Each algorithm performed better on several instances, with timetable edge-finding performing better on difficult j30 instances, but worse on many j60 and j90 instances. Interestingly, the Vilím algorithm, which is strictly speaking stronger than traditional edge-finding, was able to solve many of the harder instances that all other algorithms we implemented timed out on.

Conclusion

- **We present a $\mathcal{O}(n^2)$ algorithm for cumulative edge-finding which does not suffer from the incompleteness of the original quadratic algorithm [1].**
- **Our algorithm outperforms the state of the art Θ-tree algorithm [3], while offering a simpler implementation with no custom data structures.**

[1] Mercier L., Van Hentenryck P. Edge finding for cumulative scheduling. *INFORMS*, 20:pp. 143–153 (2008).

[2] Nuijten W.P.M. *Time and resource constrained scheduling : a constraint satisfaction approach.* Ph.D. thesis, Technische Universiteit Eindhoven (1994).

[3] Vilím P. Edge finding filtering algorithm for discrete cumulative resources in O(*kn* log *n*). In: Gent (ed.), *CP2009*, pp. 802–816. Berlin: Springer (2009).

[4] Vilím P. Timetable edge finding filtering algorithm for discrete cumulative resources. In: Achterberg, Beck (eds.), *CPAIOR2011*, pp. 230–245. Berlin: Springer (2011).

Acknowledgements: The authors would like to thank Christian Schulte, Pascal van Hentenryck and Petr Vilím for their suggestions; also the Swedish Research Council, as the third author is supported by grant 2009-4384.