Zuse Institute Berlin

Takustr. 7
14195 Berlin
Germany

ALEXANDER TESCH

A Nearly Exact Propagation Algorithm for Energetic Reasoning in $\mathcal{O}(n^2 \log n)$

ZIB Report 16-25 (May 2016)

Zuse Institute Berlin
Takustr. 7
D-14195 Berlin

Telefon: +49 30-84185-0
Telefax: +49 30-84185-125

e-mail: bibliothek@zib.de
URL: http://www.zib.de

ZIB-Report (Print) ISSN 1438-0064
ZIB-Report (Internet) ISSN 2192-7782

A Nearly Exact Propagation Algorithm for Energetic Reasoning in $\mathcal{O}(n^2 \log n)$

Alexander Tesch

Zuse Institute Berlin,
Takustrasse 7, 14195 Berlin, Germany
tesch@zib.de

Abstract. In constraint programming, energetic reasoning constitutes a powerful start time propagation rule for cumulative scheduling problems (CuSP). This article first presents an improved time interval checking algorithm that is derived from a polyhedral model. In a second step, we extend this algorithm to an energetic reasoning propagation algorithm with time complexity $\mathcal{O}(n^2 \log n)$ where n denotes the number of jobs. The idea is based on a new sweep line subroutine that efficiently evaluates energy overloads of each job on the relevant time intervals. In particular, our algorithm performs energetic reasoning propagations for every job. In addition, we show that on the vast number of relevant intervals our approach achieves the maximum possible propagations according to the energetic reasoning rule.

1 Introduction

The *cumulative scheduling problem* (CuSP) considers a set of jobs J where each job $j \in J$ is given a processing time $p_j \in \mathbb{Z}_{>0}$, a resource demand $d_j \in \mathbb{Z}_{>0}$ and a scheduling interval $[e_j, l_j] \subset \mathbb{R}$. For every job $j \in J$ we want to compute start times $s_j \in [e_j, l_j - p_j]$ such that at any point in time the resource consumption of all jobs does not exceed the maximum capacity $D \in \mathbb{Z}_{>0}$. Equivalently, the CuSP can be described as the feasibility problem

$$\begin{aligned} \text{find} \quad & s \in \mathbb{R}^n & & \\ \text{such that} \quad & \sum_{j \in J : s_j \le t < s_j + p_j} d_j \le D & \forall t \in \mathbb{R} & \quad (1) \\ & e_j \le s_j \le l_j - p_j & \forall j \in J. & \quad (2) \end{aligned}$$

Usually, CuSP feasibility tests are subroutines for more specific scheduling problems such as *makespan minimization* [8, 9]. A CuSP checks if a current subproblem may lead to feasible solution of the main problem. If this is not the case, the search tree can be pruned.

In order to solve the CuSP the literature proposes domain branching in combination with specific feasibility and propagation algorithms. The most common ones are *Time-Tabling* [6, 9, 15], *Edge-Finding* [7, 13], *Extended Edge-Finding* [14],

Time-Table Edge-Finding [8, 11, 12], *Energetic Reasoning* [5] and *Not-First/Not-Last Pruning* [10].

This paper focuses on energetic reasoning. Except for the last, energetic reasoning dominates all of the stated rules. In practice, however, the weaker but faster propagation rules are preferred over energetic reasoning due to its high complexity of $\mathcal{O}(n^3)$, see Baptiste et al. [5]. Therefore, it is natural to ask for faster implementations of energetic reasoning. In this context, some approaches aim to improve the energetic reasoning rule directly or its external conditions. Berthold, Heinz and Schulz [4] characterize time intervals where energetic reasoning cannot detect infeasibility. Such intervals can be neglected in the standard energetic reasoning algorithm which leads to a reasonable computation time improvement. Similarly, Derrien et al. [3] sharpen the original characterization of relevant time intervals of Baptiste et al. [5] and apply the standard energetic reasoning algorithm to the reduced set of intervals. Both approaches are *exact*, that is they compute the maximum energetic reasoning propagations for all jobs. Their complexity is $\mathcal{O}(n^3)$. Recently, Bonifas [2] presents a new $\mathcal{O}(n^2 \log n)$ algorithm that computes one beneficial job for each relevant interval and propagates energetic reasoning on it. Unlike previous approaches, his algorithm detects at least one possible energetic reasoning propagation. But in general, the total number of propagations is $\mathcal{O}(1)$. Thus, to ensure propagations for all jobs his algorithm generally needs to be executed $\mathcal{O}(n)$ times which gives a complexity of $\mathcal{O}(n^3 \log n)$. However, the method seems practically relevant, if one focuses on one job propagations [2]. The idea of Bonifas' algorithm is based on a geometric interpretation of energetic reasoning which leads to upper envelope computations of piecewise linear functions in the plane.

In this article, we proceed from a related geometric interpretation. In contrast to Bonifas [2], we state a different geometric problem which allows us to compute energetic reasoning propagations for all jobs instead of only one. Hence our approach yields $\mathcal{O}(n)$ more propagations in general. In addition, for each propagation that is found in [2] our algorithm finds an equal or stronger propagation. The complexity of our algorithm is $\mathcal{O}(n^2 \log n)$ compared to the complete $\mathcal{O}(n^3 \log n)$ algorithm in [2]. Hence, our algorithm supersedes Bonifas' algorithm. Compared to the exact $\mathcal{O}(n^3)$ approach of Derrien et al. [3], our algorithm does not provably compute the maximum energetic reasoning propagations. But we show that our approach yields maximum propagations on the huge majority of relevant intervals. We give a precise characterization of the corresponding interval set.

Our paper is organized as follows. In Section 2 we introduce the concepts of energetic reasoning. Section 3 gives an alternative characterization for the set of relevant time intervals from polyhedral theory. From this, we deduce an improved interval checking algorithm in Section 4. Furthermore, we extended this algorithm by energetic reasoning propagations in Section 5. Its basis forms a sweep line subroutine that is introduced in Section 6. In Section 7 we characterize the set of time intervals where our algorithm performs maximum propagations.

Section 8 compares our methods to the current state of the art. Finally, we conclude our results in Section 9.

2 Energetic Reasoning

In the following we introduce the basic concepts of energetic reasoning for CuSP, see Baptiste et al. [5]. Assume a CuSP instance as introduced in Section 1. For each job $j \in J$ define

$$\mu_j(t_1, t_2) = \min\{p_j, t_2 - t_1, \max\{0, e_j + p_j - t_1\}, \max\{0, t_2 - l_j + p_j\}\} \quad (3)$$

as the *minimum left-/right-shift duration* in the time interval $[t_1, t_2] \subset \mathbb{R}$. Moreover, define the *energy overload* in the time interval $[t_1, t_2] \subset \mathbb{R}$ as

$$\omega(t_1, t_2) = \sum_{j \in J} d_j \cdot \mu_j(t_1, t_2) - D \cdot (t_2 - t_1) \quad (4)$$

which is the slack between the *consumed energy* and the *available energy* in a time interval $[t_1, t_2]$. If in any time interval $[t_1, t_2] \subset \mathbb{R}$ the consumed energy exceeds the available energy, that is $\omega(t_1, t_2) > 0$, then the CuSP is infeasible. The time interval of maximum energy overload can be computed in $\mathcal{O}(n^2)$, see [5].

Besides checking infeasibility, energetic reasoning reduces the variable domain which consists of scheduling intervals $[e_j, l_j]$ for all jobs $j \in J$. Assume a job $j \in J$ is *left-shifted*, that is $s_j = e_j$, then

$$\mu_j^{left}(t_1, t_2) = \max\{0, \min\{t_2, e_j + p_j\} - \max\{t_1, e_j\}\} \quad (5)$$

defines the *left-shift duration* of job j in the interval $[t_1, t_2] \subset \mathbb{R}$. In addition,

$$\omega_j(t_1, t_2) = \omega(t_1, t_2) + d_j \cdot (\mu_j^{left}(t_1, t_2) - \mu_j(t_1, t_2)) \quad (6)$$

denotes the overload in the interval $[t_1, t_2] \subset \mathbb{R}$ that occurs if job $j \in J$ is left-shifted. The energetic reasoning propagation rule states: if there is an energy overload in the interval $[t_1, t_2] \subset \mathbb{R}$ due to left-shifting job $j \in J$, that is $\omega_j(t_1, t_2) > 0$, then e_j is an invalid earliest start time and thus can be delayed.

Theorem 1 (Baptiste et al. [5]). *Given a job $j \in J$ and a time interval $[t_1, t_2] \subset \mathbb{R}$ with $\omega_j(t_1, t_2) > 0$ then the earliest start time e_j can be updated to*

$$e_j = t_2 - \mu_j(t_1, t_2) + \left\lceil \frac{\omega(t_1, t_2)}{d_j} \right\rceil. \quad (7)$$

Note that the *right-shift* case is equivalent to the left-shift case by symmetry at time $t = 0$, see Derrien et al. [3]. Since there are $\mathcal{O}(n^2)$ relevant time intervals [5], the standard energetic reasoning algorithm checks Theorem 1 for all jobs on all relevant time intervals which yields an exact $\mathcal{O}(n^3)$ energetic reasoning propagation algorithm. The currently tightest characterization of the $\mathcal{O}(n^2)$ time intervals is given by Derrien et al. [3].

3 The Energetic Reasoning Polyhedron

Derrien et al. [3] characterize a set of relevant intervals that are sufficient for overload checking. However, they do not state an algorithm that reduces to their characterization. In this section, we introduce a polyhedral model from which we derive an alternative characterization for the same set of relevant time intervals. But our polyhedral model enables us to construct an improved overload checking algorithm that considers a subset of intervals than the algorithm stated in [3].

First, we model the problem of computing an interval $[t_1, t_2] \subset \mathbb{R}$ of maximum overload (4) by a simple linear program. Therefore, let $t_1, t_2 \in \mathbb{R}$ be continuous variables that represent the interval limits. In addition, the variables $\tilde{\mu}_j \geq 0$ model the piecewise linear expression $\mu_j(t_1, t_2)$, as given in (3), for all jobs $j \in J$. Then for any job subset $S \subseteq J$ with $S \neq \emptyset$ define the linear program

$$\max \sum_{j \in I} d_j \cdot \tilde{\mu}_j - D \cdot (t_2 - t_1)$$

$$\begin{aligned}
\tilde{\mu}_j &\leq p_j && \forall j \in S && (8)\\
\tilde{\mu}_j &\leq t_2 - t_1 && \forall j \in S && (9)\\
\tilde{\mu}_j &\leq e_j + p_j - t_1 && \forall j \in S && (10)\\
\tilde{\mu}_j &\leq t_2 - l_j + p_j && \forall j \in S && (11)\\
t_1 &\leq t_2 && && (12)\\
\tilde{\mu}_j &\geq 0 && \forall j \in S && (13)\\
t_1, t_2 &\in \mathbb{R} && &&
\end{aligned}$$

in $|S| + 2$ variables. With respect to (4), the objective function maximizes the energy overload in the variable interval $[t_1, t_2] \subset \mathbb{R}$, whose maximum energy is bounded by inequalities (8)-(11) for each variable $\tilde{\mu}_j$ with $j \in S$ according to (3). We define the associated polyhedron of inequalities (8) - (13) by

$$P_S = \{(t_1, t_2, \tilde{\mu}) \in \mathbb{R}^{|S|+2} \mid (t_1, t_2, \tilde{\mu}) \text{ satisfies } (8) - (13)\} \tag{14}$$

which we call the *energetic Reasoning polyhedron* for the job subset $S \subseteq J$.

Lemma 1. *There exists a job subset $S \subseteq J$ such that the maximum overload $\omega^* = \max_{t_1 < t_2} \omega(t_1, t_2)$ equals the optimal objective value of the linear program*

$$\max \sum_{j \in J} d_j \cdot \tilde{\mu}_j - D \cdot (t_2 - t_1),\ (t_1, t_2, \tilde{\mu}) \in P_S.$$

Proof. Let $(t_1^*, t_2^*) \in \mathbb{R}^2$ be the time interval of maximum overload and let $S = \{j \in J \mid \mu_j(t_1^*, t_2^*) > 0\}$. The optimal values $\tilde{\mu}_j^*$ are attained at the minimum right hand side of inequalities (8)-(11), that is $\tilde{\mu}_j^* = \mu_j(t_1^*, t_2^*)$ for all $j \in J$. □

In the following we characterize the vertices of P_S as they identify intervals of maximum overload. Without loss of generality, we restrict ourselves to vertices $(t_1, t_2, \tilde{\mu}) \in P_S$ with $\tilde{\mu}_j > 0$ for all $j \in S$. Otherwise, if $\tilde{\mu}_j = 0$ for any $j \in S$

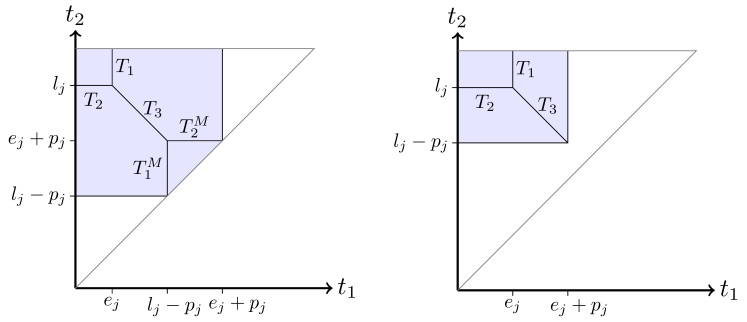

Fig. 1. Two possible shapes of the projected polyhedron P_j (here with lower and upper bounds for t_1 and t_2) with mandatory part (left) or without mandatory part (right).

then job j does not contribute to the objective function, so we can equivalently consider $P_{S'}$ with $S' = S \setminus \{j\}$. In the following we abbreviate notation and write $P_j = P_{\{j\}}$ and $P_{i,j} = P_{\{i,j\}}$.

Lemma 2. *Let $S \subseteq J$ be a job subset with $S \neq \emptyset$ and $P_S \neq \emptyset$. In addition, let $(t_1, t_2, \tilde{\mu}) \in P_S$ be a vertex of P_S with $\tilde{\mu}_j > 0$ for all $j \in S$. Then either one of the following holds:*

(i) There is a job $j \in S$ such that $(t_1, t_2, \tilde{\mu}_j)$ is a vertex of P_j.
(ii) There are two distinct jobs $i, j \in S$ such that $(t_1, t_2, \tilde{\mu}_i, \tilde{\mu}_j)$ is the intersection of one edge of P_i and one edge of P_j and thus a vertex of $P_{i,j}$.

In order to determine the time interval of maximum energy overload, Lemma 2 allows us to restrict to vertices of P_j or $P_{i,j}$ for dedicated jobs $i, j \in S$ with $i \neq j$. Since the vertices of case (i) and (ii) correspond to the intersection of edges of P_j, or P_i and P_j respectively, Lemma 2 motivates to project the edges of the polyhedron P_j with $j \in J$ to the (t_1, t_2)-plane. The next lemma gives a similar geometric interpretation as presented in Artigues et al. [1].

Lemma 3. *Given a job $j \in J$ and assume the projection of the polyhedron P_j to the (t_1, t_2)-plane. The projected line segments of the edges of P_j that contain a vertex $(t_1, t_2, \tilde{\mu}_j)$ of P_j with $\tilde{\mu}_j > 0$ are given by*

$$
\begin{aligned}
T_1(j) &= \{(e_j, t_2) \in \mathbb{R}^2 \mid t_2 \geq l_j\} \\
T_2(j) &= \{(t_1, l_j) \in \mathbb{R}^2 \mid t_1 \leq e_j\} \\
T_3(j) &= \{(t_1, t_2) \in \mathbb{R}^2 \mid t_1 + t_2 = e_j + l_j,\ e_j \leq t_1 \leq \min\{e_j + p_j, l_j - p_j\}\} \\
T_1^M(j) &= \{(l_j - p_j, t_2) \in \mathbb{R}^2 \mid l_j - p_j \leq t_2 \leq e_j + p_j\} \\
T_2^M(j) &= \{(t_1, e_j + p_j) \in \mathbb{R}^2 \mid l_j - p_j \leq t_1 \leq e_j + p_j\}.
\end{aligned}
$$

We say job $j \in J$ has a *mandatory part*, if it holds $l_j - p_j < e_j + p_j$. Let $J^M = \{j \in J \mid l_j - p_j < e_j + p_j\}$ denote the set of jobs with mandatory part. Consider Figure 1. From Lemma 3 we deduce that for any job $j \in J$ the

polyhedron P_j can have two combinatorial types: if j has a mandatory part or if j has no mandatory part. If job j has a mandatory part then inequality (9) cuts P_j which yields the additional line segments $T_1^M(j)$ and $T_2^M(j)$. If job j has no mandatory part inequality (9) is dominated by inequalities (8), (10) and (11) which implies $T_1^M(j) = T_2^M(j) = \emptyset$. Combining Lemmas 2 and 3, define for any two jobs $i, j \in J$ with $i \neq j$ the line segment intersection points $\mathcal{T}_j = \{(e_j, l_j)\}, \mathcal{T}_j^M = \{(l_j - p_j, e_j + p_j)\}, \mathcal{T}_{ij} = \big(T_1(i) \cup T_1^M(i)\big) \cap \big(T_2(j) \cup T_2^M(j)\big)$ and $\mathcal{T}'_{ij} = \big(T_1(i) \cup T_1^M(i) \cup T_2(i) \cup T_2^M(i)\big) \cap T_3(j)$.

Thus, the set of *relevant time intervals* results as the union

$$\mathcal{T} = \bigcup_{j \in J} \mathcal{T}_j \cup \bigcup_{j \in J^M} \mathcal{T}_j^M \cup \bigcup_{i,j \in J : i \neq j} \mathcal{T}_{ij} \cup \bigcup_{i,j \in J : i \neq j} \mathcal{T}'_{ij} \tag{15}$$

which forms a set of points in the plane.

Theorem 2. *If $(t_1, t_2) \in \mathbb{R}^2$ is a time interval of maximum energy overload then it holds $(t_1, t_2) \in \mathcal{T}$.*

Note that the interval set $\mathcal{T}$ is equivalent to the characterization of Derrien et al. [3]. However, in the next section we derive an improved overload checking algorithm from this characterization of projected line segments.

4 Dynamic Overload Checking Algorithm

In this section we introduce an improved overload checking algorithm that considers a subset of intervals than the checker presented in [3], see Algorithm 1 in the appendix. We modify the $\mathcal{O}(n^2)$ overload checking algorithm of Baptiste et al. [5]. The basic algorithm iterates over all t_1 values of vertical line segments $T_1(i) \cup T_1^M(i)$ with $i \in J$ and over all t_2 values of horizontal and diagonal line segments $T_2(j) \cup T_2^M(j) \cup T_3(j)$ with $j \in J$. At each pair (t_1, t_2) we check for an energy overload $\omega(t_1, t_2) > 0$ which implies infeasibility (line 9).

While the overload checker of [3] iterates all possible t_2 values, our algorithm uses a dynamic list to store only those t_2 values that intersect with the current vertical t_1 line segment. In this context, the following fact is crucial: for any current t_1 value, there is at most one intersecting segment of either $T_2(j)$, $T_2^M(j)$ or $T_3(j)$ for every job $j \in J$, see Figure 1. For non-decreasing t_1 values they appear in the sequence of either $T_2(j) \to T_3(j) \to T_2^M(j)$ or $T_2(j) \to T_3(j)$ depending on whether it holds $j \in J^M$ or $j \in J \setminus J^M$. Whenever we traverse the intersection point $(e_j, l_j) \in \mathbb{R}^2$ of the line segments $T_2(j)$ and $T_3(j)$ we delete the current $T_2(j)$ line segment and add the $T_3(j)$ line segment to the list (lines 13-16). Analogously, if $j \in J^M$ and we traverse the intersection point $(l_j - p_j, e_j + p_j)$ of the line segments $T_3(j)$ and $T_2^M(j)$ we delete the $T_3(j)$ line segment and add the $T_2^M(j)$ line segment to the list (lines 17-19). Moreover, if we detect a t_2 value that corresponds to a line segment $T_2^M(j)$ or $T_3(j)$ and $t_1 \geq e_j + p_j$ we delete the t_2 segment from the list because it is not defined according to Lemma 3 (lines 10-12). By construction of the line segments $T_1(j)$ and $T_1^M(j)$ it is ensured that

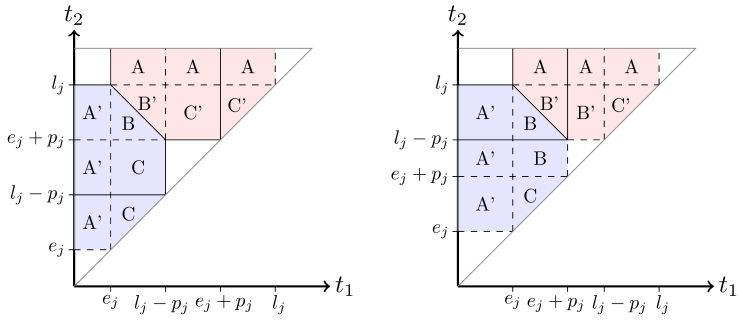

Fig. 2. Figure 1 continued: The slack regions between the left-shift polyhedron P'_j and P_j (lower left region) and between the right-shift polyhedron P''_j and P_j (upper right region). For each job $j \in J$, energetic reasoning takes effect only on such intervals.

all intersection points are traversed by the algorithm. In particular, insertions and deletions are always performed at the current list element. Therefore, all modifications to the original algorithm can be implemented in $\mathcal{O}(1)$. The number of iterated intervals remains $\mathcal{O}(n^2)$ in general, hence the complexity of our overload checking algorithm is also $\mathcal{O}(n^2)$.

Note that the intersection relations $\big(T_2(i) \cup T_2^M(i)\big) \cap T_3(i)$ of $\mathcal{T}'_{ij}$ in (15) are not included in our algorithms since they are symmetric to $\big(T_1(i) \cup T_1^M(i)\big) \cap T_3(i)$ at time $t = 0$. We execute our algorithm also for its symmetric version to include all relevant intervals.

5 Energetic Reasoning Propagation

In this section, we extend the overload checking algorithm of Section 4 by start and end time propagations based on energetic reasoning, as in (7).

Extension. In principle, finding a left-shift energetic reasoning propagation for a job $j \in J$ corresponds to finding an energy overload restricting to polyhedra P'_j and P_i for all $i \in J \setminus \{j\}$, where P'_j emerges from P_j by setting $l_j = e_j + p_j$. Analogously for right-shift propagations, where P''_j emerges from P_j by setting $e_j = l_j - p_j$, see Figure 2. In order to include all relevant intervals that are implied by the polyhedra P'_j and P''_j we need to take the extended line segments

$$T_1(j) = \{(e_j, t_2) \in \mathbb{R}^2 \mid t_2 \geq t_1\}, \quad T_1^M(j) = \{(l_j - p_j, t_2) \in \mathbb{R}^2 \mid l_j - p_j \leq t_2\}$$
$$T_2(j) = \{(t_1, l_j) \in \mathbb{R}^2 \mid t_1 \leq l_j\}, \quad T_2^M(j) = \{(t_1, e_j + p_j) \in \mathbb{R}^2 \mid t_1 \leq e_j + p_j\}$$

of Lemma 3 for all jobs $j \in J$. Now these line segments corresponds to vertical and horizontal lines that cross the entire interval plane $t_1 \leq t_2$. Therefore, any dynamic update of our overload checking algorithm of Section 4 becomes obsolete for these line segments. Hence, only the diagonal line segments of $T_3(j)$ for each $j \in J$ are dynamically updated. This gives an equivalent but simpler

characterization of relevant intervals for energetic reasoning, as given in [3]. On the basis of this characterization, we also get a simple $\mathcal{O}(n^3)$ energetic reasoning propagation algorithm which is equivalent to [3], see Algorithm 2 in the appendix. In the following we consider the set $\mathcal{T}$ as defined in (15) but using the extended line segments.

Problem Decomposition. We call an energetic reasoning propagation algorithm *exact*, if it computes

$$\max_{(t_1,t_2)\in\mathcal{T}:\ \omega_j(t_1,t_2)>0} \ t_2 - \mu_j(t_1,t_2) + \left\lceil \frac{\omega(t_1,t_2)}{d_j} \right\rceil \tag{16}$$

for all jobs $j \in J$, that is the maximum earliest start time update for all jobs $j \in J$ according to (7). In particular, problem (16) implies two nested subproblems for each job $j \in J$. First, we have to find intervals $(t_1', t_2') \in \mathcal{T}$ of positive energy overload $\omega_j(t_1', t_2') > 0$. Second, among such intervals $(t_1', t_2') \in \mathcal{T}$ with $\omega_j(t_1', t_2') > 0$ we have to determine an interval $(t_1, t_2) \in \mathcal{T}$ that yields the maximum update with respect to (16). Our idea is to decompose (16) and to compute the maxima of the two incorporated functions

$$\max_{(t_1,t_2)\in\mathcal{T}} \ \omega_j(t_1,t_2) \tag{17}$$

$$\max_{(t_1,t_2)\in\mathcal{T}} \ t_2 - \mu_j(t_1,t_2) + \left\lceil \frac{\omega(t_1,t_2)}{d_j} \right\rceil \tag{18}$$

for all jobs $j \in J$. If for any job $j \in J$ the maxima of functions (17) or (18) are attained at an interval $(t_1, t_2) \in \mathcal{T}$ and it holds $\omega_j(t_1, t_2) > 0$ we update the earliest start time according to (7).

The approach of Bonifas [2] reversely computes $\max_{j\in J} \ \omega_j(t_1, t_2)$ for each relevant time interval $(t_1, t_2) \in \mathcal{T}$. If the time interval $(t_1, t_2) \in \mathcal{T}$ attains its maximum at job $j \in J$ he updates the earliest start time e_j according to (7). Since one job can dominate on many intervals, his algorithm generally takes $\mathcal{O}(n^3 \log n)$ to propagate all jobs. In the following we construct an $\mathcal{O}(n^2 \log n)$ algorithm that propagates all jobs according to (17) and (18). Due to the additional computation of (18) each propagation is equivalent or stronger than in [2]. Hence, our approach dominates Bonifas' algorithm with respect to complexity and propagation strength.

In particular, the computation of (17) yields at least one possible energetic reasoning propagation for every job. But in general, our approach may not detect the maximum propagations in one step. Thus, we apply our algorithm until a fixpoint is reached. To our knowledge it is unknown if the number of fixpoint iterations for energetic reasoning is polynomially bounded. At least, it is not strongly polynomial, see Mercier and van Hentenryck [14]. In practice, however, it rarely takes more than two iterations to reach the fixpoint and mostly the fixpoints of our approach and exact energetic reasoning are equal.

Geometry. The overload checking algorithm of Section 4 first loops over all t_1 values and then over all t_2 values with $(t_1, t_2) \in \mathcal{T}$ while checking for potential

energy overloads $\omega(t_1, t_2) > 0$. In the following we consider a fixed iteration of the main t_1-loop for a fixed value $\bar{t}_1$.

According to fixed $\bar{t}_1$, we reformulate functions (17) and (18) equivalently as

$$\max_{(\bar{t}_1,t_2)\in\mathcal{T}} \ \omega(\bar{t}_1, t_2) + d_j \cdot (\mu_j^{left}(\bar{t}_1, t_2) - \mu_j(\bar{t}_1, t_2)) \tag{19}$$

$$\max_{(\bar{t}_1,t_2)\in\mathcal{T}} \ \omega(\bar{t}_1, t_2) + d_j \cdot (t_2 - \mu_j(\bar{t}_1, t_2)) \tag{20}$$

for all jobs $j \in J$ by using definition (6) and the fact that we only need the intervals $(\bar{t}_1, t_2) \in \mathcal{T}$ where the maximum is attained. Now, the values $\omega(\bar{t}_1, t_2)$ with $(\bar{t}_1, t_2) \in \mathcal{T}$ form the set of points $\mathcal{P} = \{(t_2, \omega(\bar{t}_1, t_2) \in \mathbb{R}^2 \mid (\bar{t}_1, t_2) \in \mathcal{T}\}$ which are collected during the overload checking algorithm. The remaining two piecewise linear functions $d_j \cdot (\mu_j^{left}(\bar{t}_1, t_2) - \mu_j(\bar{t}_1, t_2))$ and $d_j \cdot (t_2 - \mu_j(\bar{t}_1, t_2))$ will be decomposed into a set of line segments $\mathcal{L}_j$ in $\mathbb{R}^2$, see Lemmas 4 and 5.

Geometrically in $\mathbb{R}^2$, problems (19) and (20) compute for each job $j \in J$ and each line segment $l \in \mathcal{L}_j$ the point $(t_2, \omega(\bar{t}_1, t_2)) \in \mathcal{P}$ in the domain of line l such that the sum of their function values at t_2 is maximal. Since by Lemmas 4 and 5 it holds $|\mathcal{L}_j| \in \mathcal{O}(1)$, we select for each job $j \in J$ the maximum value among all line segments $l \in \mathcal{L}_j$. In Section 6 we introduce a new sweep line algorithm that solves this problem in $\mathcal{O}((|\mathcal{L}| + |\mathcal{P}|) \cdot \log(|\mathcal{L}| + |\mathcal{P}|))$, where $\mathcal{L} = \bigcup_{j\in J} \mathcal{L}_j$. From $|\mathcal{L}| \in \mathcal{O}(n)$ and $|\mathcal{P}| \in \mathcal{O}(n)$ it follows that the subproblems (19) and (20) can be solved in $\mathcal{O}(n \log n)$, which gives a total running time of $\mathcal{O}(n^2 \log n)$.

Line Segment Decomposition. Lemmas 4 and 5 show how the functions $d_j \cdot (\mu_j^{left}(\bar{t}_1, t_2) - \mu_j(\bar{t}_1, t_2))$ and $d_j \cdot (t_2 - \mu_j(\bar{t}_1, t_2))$ can be decomposed into line segments $\mathcal{L}_j$ for all jobs $j \in J$. This applies only to the left-shift case. For the right-shift case, Lemmas 6 and 7 provide line segment decompositions for the analogous functions $d_j \cdot (\mu_j^{right}(\bar{t}_1, t_2) - \mu_j(\bar{t}_1, t_2))$ and $-d_j \cdot (\bar{t}_1 + \mu_j(\bar{t}_1, t_2))$. The simultaneous consideration of left- and right-shift propagations enables us to compute all interesting propagations in one step, compare Section 7 and Algorithm 4. We restrict both cases to their specific interval region where energetic reasoning propagations may occur, see Figure 2. For this, define $\theta_1 = \max\{e_j, \bar{t}_1\}$, $\theta_2 = \min\{e_j + p_j, l_j - p_j\}$, $\theta_3 = \max\{e_j + p_j, l_j - p_j\}$ and $\theta_4 = \min\{l_j, l_j + e_j - \bar{t}_1\}$.

Lemma 4. *For any job $j \in J$ and $\bar{t}_1 \leq \theta_2$ the piecewise linear function $f_j(t_2) = d_j \cdot (\mu_j^{left}(\bar{t}_1, t_2) - \mu_j(\bar{t}_1, t_2))$ on the interval $[\theta_1, \theta_4]$ decomposes into*

$$\begin{aligned} f_j^1(t_2) &= d_j \cdot (t_2 - \theta_1) \ , \quad t_2 \in [\theta_1, \theta_2] \\ f_j^2(t_2) &= d_j \cdot (\theta_2 - \theta_1) \ , \quad t_2 \in [\theta_2, \theta_3] \\ f_j^3(t_2) &= -d_j \cdot (t_2 - \theta_4) \ , \quad t_2 \in [\theta_3, \theta_4]. \end{aligned}$$

Lemma 5. *For any job $j \in J$ and $\bar{t}_1 \leq \theta_2$ the piecewise linear function $f_j(t_2) = d_j \cdot (t_2 - \mu_j(\bar{t}_1, t_2))$ on the interval $[\theta_1, \theta_4]$ decomposes into the linear functions*

$$\begin{aligned} f_j^1(t_2) &= d_j \cdot t_2 \ , \quad t_2 \in [\theta_1, l_j - p_j] \\ f_j^2(t_2) &= d_j \cdot (l_j - p_j) \ , \quad t_2 \in [l_j - p_j, \theta_4]. \end{aligned}$$

Lemma 6. *Let* $\theta' = \max\{\theta_3, \theta_4, \bar{t}_1\}$. *For any job* $j \in J$ *and* $\bar{t}_1 \in [e_j, l_j]$ *the piecewise linear function* $f_j(t_2) = d_j \cdot (\mu_j^{right}(\bar{t}_1, t_2) - \mu_j(\bar{t}_1, t_2))$ *on the interval* $[\theta', \infty)$ *decomposes into the linear function segments*

$$f_j^1(t_2) = d_j \cdot (t_2 - \theta') \;, \quad t_2 \in [\theta', l_j]$$
$$f_j^2(t_2) = d_j \cdot (l_j - \theta') \;, \quad t_2 \in [l_j, \infty).$$

Lemma 7. *Let* $\theta' = \max\{\theta_3, \theta_4, \bar{t}_1\}$. *For any job* $j \in J$ *and* $\bar{t}_1 \in [e_j, l_j]$ *the piecewise linear function* $f_j(t_2) = -d_j \cdot (\bar{t}_1 + \mu_j(\bar{t}_1, t_2))$ *is constant on* $[\theta', \infty)$.

6 Sweep Line Algorithm

In this section we introduce a sweep line new algorithm that solves the geometric problems (19) and (20) that occur during energetic reasoning. Compared to Section 5, we restate the problem more generally.

Let $\mathcal{P}$ be a set of pairwise distinct points in $\mathbb{R}^2$ where each point $q \in \mathcal{P}$ has coordinates $(x_q, y_q) \in \mathbb{R}^2$. Additionally, let $\mathcal{L}$ be a set of line segments in $\mathbb{R}^2$ where each line segment $l \in \mathcal{L}$ is given a slope $a_l \in \mathbb{R}$, an intercept $b_l \in \mathbb{R}$ and an interval $[\underline{x}_l, \overline{x}_l] \subset \mathbb{R}$. Thus, each line segment $l \in \mathcal{L}$ corresponds to the set of points $(x, y) \in \mathbb{R}^2$ that satisfy $y = a_l \cdot x + b_l$ with $x \in [\underline{x}_l, \overline{x}_l]$. For each line segment $l \in \mathcal{L}$ we want to compute a point $q \in \mathcal{P}$ with $x_q \in [\underline{x}_l, \overline{x}_l]$ that has maximum y-distance to the line segment l. More formally, we compute

$$\max_{q \in \mathcal{P}: x_q \in [\underline{x}_l, \overline{x}_l]} a_l \cdot x_q + y_q + b_l \tag{21}$$

for all line segments $l \in \mathcal{L}$. Since b_l is constant in this term we reduce to

$$\max_{q \in \mathcal{P}: x_q \in [\underline{x}_l, \overline{x}_l]} a_l \cdot x_q + y_q \tag{22}$$

for all line segments $l \in \mathcal{L}$. If there is no $q \in \mathcal{P}$ with $x_q \in [\underline{x}_l, \overline{x}_l]$ we assume the function has value $-\infty$. Our approach is to dualize problem (22) and translate it from the (x, y)-plane to the (a, y)-plane with respect to the slopes a_l of the line segments $l \in \mathcal{L}$, see Figure 3. In this dual setting, each point $q \in \mathcal{P}$ corresponds to a line with slope x_q and intercept y_q that contains all points $(a, y) \in \mathbb{R}^2$ with $y = x_q \cdot a + y_q$. Moreover, each line segment $l \in \mathcal{L}$ translates to a point $(a_l, 0) \in \mathbb{R}^2$. The equivalent dual problem is to compute for each point $(a_l, 0)$ with $l \in \mathcal{L}$ the line $q \in \mathcal{P}$ with $x_q \in [\underline{x}_l, \overline{x}_l]$ of maximum value $y = x_q \cdot a_l + y_q$. The difficulty of the dual problem is to consider for each point $(a_l, 0) \in \mathbb{R}^2$ with $l \in \mathcal{L}$ a subset of lines $q \in \mathcal{P}$ with slopes x_q in the range $[\underline{x}_l, \overline{x}_l]$. Therefore, an upper envelope computation of all dual lines $q \in \mathcal{P}$ is not sufficient.

In the following we will stick to the *dual setting*, that means we consider $\mathcal{P}$ as a set of lines and $\mathcal{L}$ as a set of points in $\mathbb{R}^2$. A detailed pseudo-code of the following algorithms can be found in Algorithms 5-8 in the appendix.

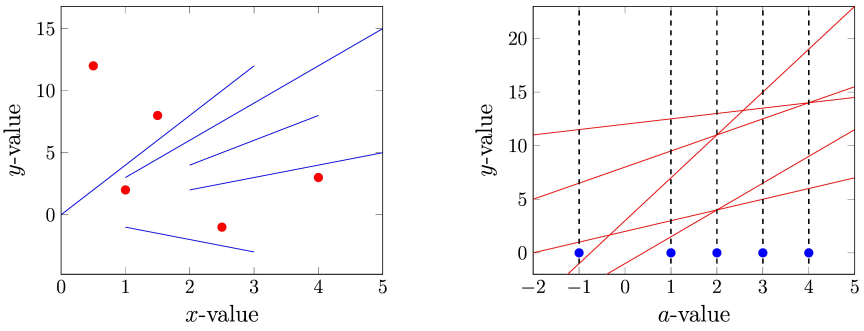

Fig. 3. Primal problem (left): given a set of points $(x_q, y_q) \in \mathcal{P}$ and a set of line segments $l \in \mathcal{L}$ with slope a_l. Dual problem (right): converts to a set of lines $q \in \mathcal{P}$ with slope x_q, intercept y_q and a set of evaluation points $(a_l, 0)$ for all $l \in \mathcal{L}$ where the sweep line (dashed) is evaluated. The interval tree B stores the state of the sweep line.

Sweeping (Algorithm 5). The sweep line algorithm sweeps over all points $(a_l, 0) \in \mathbb{R}^2$ with $l \in \mathcal{L}$ in non-decreasing order of a_l. All a_l values are stored as *evaluation events* in a *min-heap* H. At each point $(a_l, 0)$ we evaluate function (22), which can be done efficiently for any interval $[\underline{x}_l, \overline{x}_l]$ by using a *binary interval tree* B which stores the current state of the sweep line. While sweeping over all a_l values with $l \in \mathcal{L}$ the state of the sweep line changes, so the interval tree B must be updated dynamically. Therefore, the heap H additionally stores *resolve events* which constitute events where the tree structure must be updated. New resolve events are added dynamically during the sweep.

The main sweep line algorithm successively extracts the minimum element from the heap H. If it is an evaluation event, we call the subroutine *evaluate* and if it is a resolve event we call the subroutine *resolve*. The main concepts of the sweep line algorithm are explained in the following.

Interval Tree (Algorithm 6). Let V_B denote the set of nodes of the interval tree B. Each tree node $v \in V_B$ stores four data members: an *interval* $[\underline{x}_v, \overline{x}_v] \subset \mathbb{R}$, a *dominating line* $\pi_v \in \mathcal{P}$, a *resolve point* $\alpha_v \in \mathbb{R}$ and a *minimum resolve point* $\beta_v \in \mathbb{R}$. The data members α_v, β_v and π_v change dynamically while sweeping over all a_l values with $l \in \mathcal{L}$. For an initial sweep value $a_0 \in \mathbb{R}$ the tree is build up recursively from bottom to top. The leaves of B, from left to right, correspond to lines $q \in \mathcal{P}$ sorted by x_q first and by y_q second in non-decreasing order. The data members of a *leaf node* $v \in V_B$ that is associated with one line $q \in \mathcal{P}$ is initialized by $[\underline{x}_v, \overline{x}_v] = [x_q, x_q]$, $\pi_v = q$, $\alpha_v = \infty$ and $\beta_v = \infty$.

Conversely, the data members of a *non-leaf node* $v \in V_B$ with child nodes $v.left \in V_B$ and $v.right \in V_B$ are defined recursively as follows. The interval $[\underline{x}_v, \overline{x}_v] = [\underline{x}_{v.left}, \overline{x}_{v.right}]$ spans the intervals of the child nodes of v. Its dominating line π_v is equal to the line $q \in \{\pi_{v.left}, \pi_{v.right}\}$ that has the higher value of $a_0 \cdot x_q + y_q$ (lines 15-16). Furthermore, for the dominating lines $q = \pi_{v.left}$ and $q' = \pi_{v.right}$ the resolve point $\alpha_v = \frac{y_q - y_{q'}}{x_{q'} - x_q}$ equals the intersection point of

the lines q and q', if $x_q \neq x_{q'}$. Otherwise, if $x_q = x_{q'}$ set $\alpha_v = \infty$ (lines 18-24). Finally, let $\beta_v = \min\{\alpha_{v.left}, \alpha_{v.right}, \beta_{v.left}, \beta_{v.right}\}$ be the minimum value of a resolve point of any node in the subtree rooted at v (line 14). If, during the construction, it holds $\alpha_v < \beta_v$ for some $v \in V_B$ we add a resolve event with value α_v to the heap H (lines 22-23), see also *resolve*.

Evaluate (Algorithm 7). This subroutine evaluates (22) for a sweep value a_l and an interval $[\underline{x}_l, \overline{x}_l]$ with $l \in \mathcal{L}$. For this, we descend the interval tree B recursively from the root along nodes $v \in V_B$ with $[\underline{x}_v, \overline{x}_v] \cap [\underline{x}_l, \overline{x}_l] \neq \emptyset$. For nodes $v \in V_B$ with $[\underline{x}_v \overline{x}_v] \subseteq [\underline{x}_l, \overline{x}_l]$ function (22) can be evaluated in $\mathcal{O}(1)$ because the dominating line $\pi_v \in \mathcal{P}$ (line 11) yields the maximum value of (22) in the interval $[\underline{x}_v, \overline{x}_v]$ by construction. Therefore, the recursion descends the tree B only along the interval limits $\underline{x}_l$ and $\overline{x}_l$. Hence, one evaluation takes $\mathcal{O}(\log |\mathcal{P}|)$.

Resolve (Algorithm 8). This subroutine resolves a node $v \in V_B$ at its sweep value α_v. Recall that α_v denotes the intersection value of the dominating lines $\pi_{v.left}$, $\pi_{v.right} \in \mathcal{P}$. Since resolving means that $\pi_{v.right}$ replaces $\pi_{v.left}$ as dominating line with respect to (22) for all sweep values $a_l > \alpha_v$ we set $\pi_v = \pi_{v.right}$. Additionally, we set $\alpha_v = \infty$ since the lines $\pi_{v.left}$ and $\pi_{v.right}$ have no future intersection because $\pi_{v.right}$ has the higher slope by construction (lines 1-3).

By its recursive definition, changing the dominating line $\pi_v \in \mathcal{P}$ may change the values of α_u, β_u and π_u of all nodes u on the path from v to the root of B. Thus, we propagate those values along this path (lines 4-18). In particular, α_u is updated only if it was not already resolved, that is $\alpha_u < \infty$. If, after the propagation, it holds $\alpha_u < \beta_u$ for any node u on the path from v to the root of B then the resolve event at α_u is added to the heap H (line 14). This is because we only add the resolve event at the intersection point α_v to the heap H, if α_v does not change due to recursion. Otherwise, if $\beta_v \leq \alpha_v$ then resolving a child node of v may still affect the recursive value of α_v. In this case, we add at most one resolve event for each tree node which is crucial for the running time of our algorithm. Since only data members along the path from a node $v \in V_B$ to the root of B are changed, one resolve takes $\mathcal{O}(\log |\mathcal{P}|)$.

Theorem 3. *The sweep line algorithm runs in* $\mathcal{O}((|\mathcal{L}| + |\mathcal{P}|) \cdot \log(|\mathcal{L}| + |\mathcal{P}|))$.

Proof. The interval tree is initialized in $\mathcal{O}(|\mathcal{P}|)$. The event heap H contains at most $\mathcal{O}(|\mathcal{P}|)$ resolve events and $\mathcal{O}(|\mathcal{L}|)$ evaluation events, so extracting all elements gives $\mathcal{O}((|\mathcal{L}|+|\mathcal{P}|)\cdot\log(|\mathcal{L}|+|\mathcal{P}|))$. The main loop extracts $\mathcal{O}(|\mathcal{L}|+|\mathcal{P}|)$ events from the heap H and each event is either evaluated or resolved, where each has complexity $\mathcal{O}(\log |\mathcal{P}|)$. Hence, the statement follows. □

7 Exact Intervals

Finally, we characterize interval subsets $\mathcal{T}_j \subseteq \mathcal{T}$ for all jobs $j \in J$ where our sweep line propagation algorithm of Section 5 is exact. That means, if the interval $(t_1, t_2) \in \mathcal{T}$ yields the maximum earliest start time update (7) for job $j \in J$

and it holds $(t_1, t_2) \in \mathcal{T}_j$ then the sweep line propagator finds this propagation. Therefore, recall the functions

$$\max_{(\bar{t}_1, t_2) \in \mathcal{T}} \ \omega(\bar{t}_1, t_2) + d_j \cdot (\mu_j^{left}(\bar{t}_1, t_2) - \mu_j(\bar{t}_1, t_2)) \tag{23}$$

$$\max_{(\bar{t}_1, t_2) \in \mathcal{T}} \ \omega(\bar{t}_1, t_2) + d_j \cdot (t_2 - \mu_j(\bar{t}_1, t_2)) \tag{24}$$

as given in (19) and (20) for fixed value $\bar{t}_1 \in \mathbb{R}$.

Lemma 8. *If the slopes of the functions (23) and (24) coincide on an interval $[\underline{t}_2, \bar{t}_2] \subset \mathbb{R}$ then both functions on the interval $[\underline{t}_2, \bar{t}_2]$ attain their maximum at the same point $(\bar{t}_1, t_2) \in \mathcal{T}$ with $t_2 \in [\underline{t}_2, \bar{t}_2]$, if the maximum exists.*

Lemma 8 implies that the two nested subproblems (23) and (24) of the exactness condition (16) attain their maximum at the same point $t_2 \in [\underline{t}_2, \bar{t}_2]$, if their slopes are equal. Thus, only one function of (23) and (24) must be considered. Since the sweep line algorithm of Section 5 computes such maxima, we obtain exact energetic reasoning propagations on the interval $[\underline{t}_2, \bar{t}_2]$. This suggests to characterize intervals where the difference $d_j \cdot (t_2 - \mu_j^{left}(\bar{t}_1, t_2))$ of functions (23) and (24) is constant. Analogously for the right-shift case, we study intervals where the function $\mu_j^{right}(\bar{t}_1, t_2) - 2 \cdot \mu_j(\bar{t}_1, t_2)$ is constant.

Recall Figure 2 and consider the following subdivision of interval areas

$$\begin{aligned}
\mathcal{T}_j^A &= \{(t_1, t_2) \in \mathbb{R}^2 \mid t_1 \in [e_j, l_j],\ t_2 \in [l_j, \infty)\} \\
\mathcal{T}_j^B &= \{(t_1, t_2) \in \mathbb{R}^2 \mid t_1 \in [e_j, \theta_2],\ t_2 \in [e_j + p_j, \theta_4]\} \\
\mathcal{T}_j^{B'} &= \{(t_1, t_2) \in \mathbb{R}^2 \mid t_1 \in [e_j, l_j - p_j],\ t_2 \in [\max\{\theta_3, \theta_4\}, l_j]\} \\
\mathcal{T}_j^C &= \{(t_1, t_2) \in \mathbb{R}^2 \mid t_1 \in [e_j, \theta_2],\ t_2 \in [t_1, e_j + p_j]\}
\end{aligned}$$

where $\theta_1 = \max\{e_j, \bar{t}_1\}$, $\theta_2 = \min\{e_j + p_j, l_j - p_j\}$, $\theta_3 = \max\{e_j + p_j, l_j - p_j\}$ and $\theta_4 = \min\{l_j, l_j + e_j - \bar{t}_1\}$.

Lemma 9. *For fixed value $\bar{t}_1 \in [e_j, l_j]$ the function $\mu_j^{right}(\bar{t}_1, t_2) - 2 \cdot \mu_j(\bar{t}_1, t_2)$ has slope zero for all $t_2 \in [l_j, \infty)$.*

Lemma 10. *For fixed value $\bar{t}_1 \in [e_j, \theta_2]$ the function $t_2 - \mu_j^{left}(\bar{t}_1, t_2)$ has slope one the interval $t_2 \in [e_j + p_j, \theta_4]$.*

Lemma 11. *For fixed value $\bar{t}_1 \in [e_j, \theta_2]$ the function $\mu_j^{right}(\bar{t}_1, t_2) - 2 \cdot \mu_j(\bar{t}_1, t_2)$ has slope one in the interval $[\max\{\theta_3, \theta_4\}, l_j]$.*

Lemma 12. *For fixed value $\bar{t}_1 \in [e_j, \theta_2]$ the function $t_2 - \mu_j^{left}(\bar{t}_1, t_2)$ has slope zero in the interval $[t_1, \theta_2]$.*

From Lemmas 9 and 12 we deduce that our approach is exact on the interval sets $\mathcal{T}_j^A$ and $\mathcal{T}_j^C$. Thus, we execute our algorithm also for the symmetric version of the problem to imply exactness on their symmetric copies $\mathcal{T}_j^{A'}$ and $\mathcal{T}_j^{C'}$, see

Figure 2. In contrast, Lemmas 10 and 11 show that our approach is not exact on the symmetric interval sets $\mathcal{T}_j^B$ and $\mathcal{T}_j^{B'}$. But in practice, both sets form only a minor part of the whole interval region where energetic reasoning is propagated. Fr our computational results, we implemented two versions. One version omits the line segments that belong to the non-exact interval regions and the other includes the slopes of both functions (23) and (24) to enhance our chance to find the exact propagation, see Algorithm 4.

8 Computational Results

Our algorithms are implemented in C++ using Linux GCC compiler version 4.8.4 on a 3.20 GHz Intel Xeon CPU and 16GB RAM. The test set is taken from RCPSP instances of the PSPLIB [16] and contains 480 instances of 30 jobs, 480 instances of 60 jobs and 600 instances of 120 jobs.

We implemented the basic overload checker of Derrien et al. [3] (CD) and our improved dynamic overload checker (CC) of Section 4. As energetic reasoning propagators, we implemented the original $\mathcal{O}(n^3)$ propagator of Baptiste et al. [5] (ERB) and a simpler but equivalent version of the $\mathcal{O}(n^3)$ propagator of Derrien et al. [3] (ERD), see Algorithm 2. Moreover, we implemented two versions of our $\mathcal{O}(n^2 \log n)$ sweep line propagator. The full version (SWF) adds all line segments of the exact and non-exact interval regions, compare Algorithm 4 and Section 7. For the non-exact regions, only lines with the slopes of the corresponding overload and update function are added. The relaxed version (SWR) considers only line segments of the exact interval regions. In addition to all algorithms, we apply a fast time-tabling and precedence propagator [9].

We compute lower bounds by destructive improvement [8]. In order to show the real performance of the algorithms we first apply the static *SetTimes* [17] branching rule. To show the practical performance we also apply a dynamic branching rule, similar to Schutt et al. [9], which stores a score value for every job that is increased by one, if fixing this job to its earliest start time leads to a direct failure, otherwise the score is decreased by one. We select the job with the highest score value and break ties with the earliest start and latest completion time. The time limit for each lower bound is 3600 seconds.

Tables 1 and 2 compare the results of the static and the dynamic branching rule. The column *opt* shows the number of optimally solved instances, ΔLB is the sum of the computed lower bounds normed to the weakest algorithm and *nodes/s* is the average number of nodes per second of the optimally solved instances. Our results show that the precise polyhedral interpretation of the dynamic overload checker (CC) leads to a speedup of factor two compared to the checker of Derrien et al. [3]. Due to this gain, the checker performs also very well as standalone algorithm combined with dynamic branching. We have to note that the PSPLIB instances are rather cumulative than disjunctive, that is pure checkers perform very reasonable compared to pure energetic approaches.

Our sweep line propagators also show a very positive performance. In terms of computation time, full and relaxed sweep line propagation highly dominate

Table 1. Results for the checkers and propagators using static branching.

	J30			J60			J120		
	opt	Δ LB	nodes/s	opt	Δ LB	nodes/s	opt	Δ LB	nodes/s
CD	375	90	815.16	329	0	253.84	159	0	78.86
CC	382	150	1577.41	330	31	590.97	159	25	198.42
ERB	376	0	36.49	343	31	6.38	167	100	1.27
ERD	386	87	137.14	344	62	21.05	168	128	4.20
SWF	386	52	288.04	343	48	84.01	169	97	29.09
SWR	391	97	456.73	342	80	139.65	169	132	50.85

Table 2. Results for the checkers and propagators using dynamic branching.

	J30			J60			J120		
	opt	Δ LB	nodes/s	opt	Δ LB	nodes/s	opt	Δ LB	nodes/s
CD	458	48	1065.78	386	168	266.79	211	93	90.21
CC	461	78	1747.14	389	210	596.91	217	196	205.78
ERB	446	0	40.93	369	0	6.58	184	0	1.38
ERD	454	44	137.25	380	71	22.01	197	61	4.70
SWF	455	27	299.33	377	83	80.26	209	136	30.29
SWR	460	52	483.42	387	151	134.84	214	205	53.05

the previous energetic reasoning propagators by a factor up to twelve on large instances. In terms of propagation power, however, full sweep line propagation (SWF) is slightly inferior to the propagator of Derrien et al. [3] using static branching. Using dynamic branching, full sweep line propagation dominates again. In particular, relaxing the non-exact line segments (SWR) results in a vast computational speedup where much more propagations are performed in the same time. The relationship between almost the same propagation power as exact energetic reasoning and much faster computation time lets the relaxed sweep line algorithm (SWR) outperform all other energetic reasoning algorithms.

On the 120 job test set, the relaxed sweep line algorithm (SWR) further improved four best known lower bounds: 103 (34_2), 128 (47_6), 104 (59_5), 88 (60_3) and our dynamic checker (CC) improved six additional best known lower bounds: 116 (12_6), 218 (36_3), 119 (47_3), 128 (47_10), 124 (53_10), 126 (58_9). The improvement is +1 for each instance, we compared with [18].

9 Conclusion

In this paper we propose an improved overload checking algorithm and a new energetic reasoning propagation algorithm for solving the CuSP. Our approaches are derived from a novel polyhedral interpretation of energetic reasoning. On that basis, we develop practically efficient algorithms that improve the current state of the art. Further research may focus on practical refinements of the presented algorithms and the development of more sophisticated combinatorial methods from a related polyhedral background.

References

1. Artigues, C., & Lopez, P.: Energetic reasoning for energy-constrained scheduling with a continuous resource. Journal of Scheduling, 18(3), 225-241 (2015)
2. Bonifas, N. (2016): A $O(n^2log(n))$ propagation for the Energy Reasoning, Conference Paper, Roadef 2016 (2016)
3. Derrien, A., & Petit, T.: A new characterization of relevant intervals for energetic reasoning. In Principles and Practice of Constraint Programming (pp. 289-297). Springer International Publishing (2014)
4. Berthold, T., Heinz, S., & Schulz, J.: An approximative criterion for the potential of energetic reasoning. In Theory and Practice of Algorithms in (Computer) systems (pp. 229-239). Springer Berlin Heidelberg (2011)
5. Baptiste, P., Le Pape, C., & Nuijten, W.: Satisfiability tests and time bound adjustments for cumulative scheduling problems. Annals of Operations research, 92, 305-333 (1999)
6. Baptiste, P., Le Pape, C., & Nuijten, W. (2012). Constraint-based scheduling: Applying Constraint Programming to Scheduling Problems (Vol. 39). Springer Science & Business Media.
7. Vilim, P.: Edge Finding Filtering Algorithm for Discrete Cumulative Resources in $\mathcal{O}(kn \log n)$. In Principles and Practice of Constraint Programming - CP 2009 (pp. 802-816). Springer Berlin Heidelberg (2009)
8. Vilim, P.: Timetable edge finding filtering algorithm for discrete cumulative resources. In Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (pp. 230-245). Springer Berlin Heidelberg (2011)
9. Schutt, A., Feydy, T., Stuckey, P. J., & Wallace, M. G.: Explaining the cumulative propagator. Constraints, 16(3), 250-282 (2011)
10. Schutt, A., & Wolf, A.: A New $\mathcal{O}(n^2 \log n)$ Not-First/Not-Last Pruning Algorithm for Cumulative Resource Constraints. In Principles and Practice of Constraint Programming - CP 2010 (pp. 445-459). Springer Berlin Heidelberg (2010)
11. Schutt, A., Feydy, T., & Stuckey, P. J.: Explaining time-table-edge-finding propagation for the cumulative resource constraint. In Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (pp. 234-250). Springer Berlin Heidelberg (2013)
12. Ouellet, P., & Quimper, C. G.: Time-table extended-edge-finding for the cumulative constraint. In Principles and Practice of Constraint Programming (pp. 562-577). Springer Berlin Heidelberg (2013)
13. Kameugne, R., Fotso, L. P., Scott, J., & Ngo-Kateu, Y.: A quadratic edge-finding filtering algorithm for cumulative resource constraints. Constraints, 19(3), 243-269 (2014)
14. Mercier, L., & Van Hentenryck, P. (2008). Edge finding for cumulative scheduling. INFORMS Journal on Computing, 20(1), 143-153.
15. Letort, A., Beldiceanu, N., & Carlsson, M.: A scalable sweep algorithm for the cumulative constraint. In Principles and Practice of Constraint Programming (pp. 439-454). Springer Berlin Heidelberg (2012)
16. Kolisch, R., & Sprecher, A. (1997). PSPLIB-a project scheduling problem library: OR software-ORSEP operations research software exchange program. European journal of operational research, 96(1), 205-216.
17. Godard, D., Laborie, P., & Nuijten, W. (2005, June). Randomized Large Neighborhood Search for Cumulative Scheduling. In ICAPS (Vol. 5, pp. 81-89).
18. HP Peter Stuckey: `http://people.eng.unimelb.edu.au/pstuckey/rcpsp/`

A Proofs

A.1 Proof of Lemma 2

We first show the following helping lemma.

Lemma 13. *Let $S \subseteq J$ be a job subset with $S \neq \emptyset$ and $P_S \neq \emptyset$. In addition, let $(t_1, t_2, \tilde{\mu}) \in P_S$ be a vertex of P_S with $\tilde{\mu}_j > 0$ for all $j \in S$. Then either one of the following holds:*

(i) $(t_1, t_2, \tilde{\mu})$ satisfies three inequalities of (8)-(11) with equality and all of them correspond to one job $j \in S$
(ii) $(t_1, t_2, \tilde{\mu})$ satisfies four inequalities of (8)-(11) with equality where two correspond to one job $i \in S$ and two correspond to one job $j \in S$ with $i \neq j$.

Proof. We first show that P_S has full dimension. Let $m = |S|$ and let $\delta_j \in \mathbb{R}^m$ be the j-th unit vector. Furthermore, let $0_m, 1_m \in \{0,1\}^m$ be the vectors that contain m zeros or one respectively. Consider the $m+2$ vectors $(-T, T, p_j \cdot \delta_j)_{j \in S}$, $(0, T, 0_m)$ and $(T, T, 0_m)$ where T denotes a large constant. We verify that these vectors are linearly independent and satisfy inequalities (8)-(13). Consequently, P_S contains $m+2$ linearly independent vectors, so it has full dimension $m+2$.

It follows that the vertex $(t_1, t_2, \tilde{\mu}) \in P_S$ satisfies $m+2$ inequalities of (8)-(13) with equality. If it satisfies inequality (12) or (13) with equality it holds $\tilde{\mu}_j = 0$ for some $j \in S$ which contradicts the assumption. Hence, we can restrict to inequalities (8)-(11) which yields the reduced constraint matrix $A \in \{0,1\}^{4 \cdot m \times m+2}$ of the form

$$A = \begin{pmatrix} 0_m & 0_m & I_m \\ 1_m & -1_m & I_m \\ 1_m & 0_m & I_m \\ 0_m & -1_m & I_m \end{pmatrix} \begin{matrix} (8) \\ (9) \\ (10) \\ (11) \end{matrix}$$

where the first two columns of A correspond to variables t_1, t_2 and the last m columns correspond to variables $\tilde{\mu}_j$ with $j \in S$. Here, $I_m \in \{0,1\}^{m \times m}$ equals the $m \times m$ identity matrix.

Thus, the vertex $(t_1, t_2, \tilde{\mu}) \in P_S$ corresponds to a selection of $m+2$ linearly independent rows of A whose associated submatrix we denote by A_B. Since every column of A_B must contain at least one non-zero entry and each row of A has exactly one non-zero coefficient for some variable $\tilde{\mu}_j$ it follows that A_B contains m rows with non-zero entries for each variable $\tilde{\mu}_j$ with $j \in S$. The remaining two rows of A_B either have a non-zero entry for one job $j \in S$ or for two distinct jobs $i, j \in S$. This is equivalent to cases (i) and (ii) which proves the lemma. □

Lemma 2. *Let $S \subseteq J$ be a job subset with $S \neq \emptyset$ and $P_S \neq \emptyset$. In addition, let $(t_1, t_2, \tilde{\mu}) \in P_S$ be a vertex of P_S with $\tilde{\mu}_j > 0$ for all $j \in S$. Then either one of the following holds:*

(i) There is a job $j \in S$ such that $(t_1, t_2, \tilde{\mu}_j)$ is a vertex of P_j.
(ii) There are two distinct jobs $i, j \in S$ such that $(t_1, t_2, \tilde{\mu}_i, \tilde{\mu}_j)$ is the intersection of one edge of P_i and one edge of P_j and thus a vertex of $P_{i,j}$.

Proof. It either holds case (i) or (ii) of Lemma 13 because the assumptions are equal. From the proof of Lemma 13, the polyhedra P_i and $P_{i,j}$ have dimensions three and four respectively. Case (i) of Lemma 13 implies that the projected vertex $(t_1, t_2, \tilde{\mu}_j)$ is a vertex of P_j.

An edge of P_j satisfies two inequalities of (8)-(11) with equality that correspond to job j. Therefore, case (ii) of Lemma 13 yields that the projected vertex $(t_1, t_2, \tilde{\mu}_i, \tilde{\mu}_j)$ is the intersection of one edge of P_i and one edge of P_j and hence a vertex of $P_{i,j}$. □

A.2 Proof of Lemma 3

Lemma 3. *Given a job $j \in J$ and assume the projection of the polyhedron P_j to the (t_1, t_2)-plane. The projected line segments of the edges of P_j that contain a vertex $(t_1, t_2, \tilde{\mu}_j)$ of P_j with $\tilde{\mu}_j > 0$ are given by*

$$
\begin{aligned}
T_1(j) &= \{(e_j, t_2) \in \mathbb{R}^2 \mid t_2 \geq l_j\} \\
T_2(j) &= \{(t_1, l_j) \in \mathbb{R}^2 \mid t_1 \leq e_j\} \\
T_3(j) &= \{(t_1, t_2) \in \mathbb{R}^2 \mid t_1 + t_2 = e_j + l_j,\ e_j \leq t_1 \leq \min\{e_j + p_j, l_j - p_j\}\} \\
T_1^M(j) &= \{(l_j - p_j, t_2) \in \mathbb{R}^2 \mid l_j - p_j \leq t_2 \leq e_j + p_j\} \\
T_2^M(j) &= \{(t_1, e_j + p_j) \in \mathbb{R}^2 \mid l_j - p_j \leq t_1 \leq e_j + p_j\}.
\end{aligned}
$$

Proof. By the proof of Lemma 13, it suffices to restrict to inequalities (8)-(11). An edge of P_j satisfies two inequalities of (8)-(11) with equality. Thus, there are six possible cases:

(i) If inequalities (8) and (10) hold with equality then it holds $\tilde{\mu}_j = p_j = e_j + p_j - t_1$ which implies $t_1 = e_j$. By inequalities (11) and (9) it follows $t_2 \geq l_j$ and $t_2 \geq e_j + p_j$.

(ii) If inequalities (8) and (11) hold with equality then it holds $\tilde{\mu}_j = p_j = t_2 - l_j + p_j$ which implies $t_2 = l_j$. By inequalities (10) and (9) it follows $t_1 \leq e_j$ and $t_1 \leq l_j - p_j$.

(iii) If inequalities (10) and (11) hold with equality then it holds $\tilde{\mu}_j = e_j + p_j - t_1 = t_2 - l_j + p_j$ which implies $t_1 + t_2 = e_j + l_j$. By inequalities (8) and (9) it follows $t_1 \geq e_j$ and $t_1 \leq l_j - p_j$. In addition, inequality (13) yields $t_1 \leq e_j + p_j$.

(iv) If inequalities (9) and (11) hold with equality then it holds $\tilde{\mu}_j = t_2 - t_1 = t_2 - l_j + p_j$ which implies $t_1 = l_j - p_j$. By inequalities (10) and (8) it follows $t_2 \leq e_j + p_j$ and $t_2 \leq l_j$. In addition, inequality (12) yields $t_2 \geq l_j - p_j$.

(v) If inequalities (9) and (10) hold with equality then it holds $\tilde{\mu}_j = t_2 - t_1 = e_j + p_j - t_1$ which implies $t_2 = e_j + p_j$. By inequalities (11) and (8) it follows $t_1 \geq l_j - p_j$ and $t_1 \geq e_j$. In addition, inequality (12) yields $t_1 \leq e_j + p_j$.

(vi) If inequalities (8) and (9) hold with equality then it holds $\tilde{\mu}_j = p_j = t_2 - t_1$. By inequalities (10) and (11) it follows $t_1 \leq e_j$ and $t_2 \geq l_j$. Adding both yields $l_j - e_j \leq t_2 - t_1 = p_j \leq l_j - e_j$ which implies $p_j = l_j - e_j$. Thus, it holds $t_1 = e_j$ and $t_2 = l_j$. Therefore, all inequalities of (8)-(11) are satisfied with equality. This case is already included in cases (i)-(v).

Since $e_j \leq l_j - p_j$ and $e_j + p_j \leq l_j$ always holds the cases (i)-(v), in order of appearance, correspond to the line segments $T_1(j), T_2(j), T_3(j), T_1^M(j), T_2^M(j)$ which proves the lemma. □

A.3 Proof of Theorem 2

Theorem 2. *If $(t_1, t_2) \in \mathbb{R}^2$ is a time interval of maximum energy overload then it holds $(t_1, t_2) \in \mathcal{T}$.*

Proof. By Lemma 1 there exists a job subset $S \subseteq J$ with $S \neq \emptyset$ such that $(t_1, t_2, \tilde{\mu}) \in \mathbb{R}^{|S|+2}$ is a vertex of P_S with $\tilde{\mu}_j > 0$ for all $j \in S$. Therefore, Lemma 2 holds. We distinguish between cases (i) and (ii) of Lemma 2.

If case (i) holds then there is a job $j \in S$ such that $(t_1, t_2, \tilde{\mu}_j)$ is a vertex of P_j. Since P_j is a three-dimensional polyhedron the vertex $(t_1, t_2, \tilde{\mu}_j)$ of P_j has at least three incident edges. By Lemma 3, the only intersection points of at least three projected edges of P_j are $(t_1, t_2) = (e_j, l_j)$ and $(t_1, t_2) = (l_j - p_j, e_j + p_j)$, if $j \in J^M$. This case is equivalent to $(t_1, t_2) \in \mathcal{T}_j$ and $(t_1, t_2) \in \mathcal{T}_j^M$, if $j \in J^M$.

Otherwise, if case (ii) holds then there are two distinct jobs $i, j \in S$ such that (t_1, t_2) is an intersection point of the projected edges of P_i and P_j respectively. Since $T_1(i), T_1^M(i)$ are vertical, $T_2(i), T_2^M(i)$ horizontal and $T_3(i)$ diagonal line segments the possible intersection relations are vertical-horizontal, vertical-diagonal and horizontal-diagonal. The relation vertical-horizontal corresponds to $(t_1, t_2) \in \mathcal{T}_{ij}$ and the relations vertical-diagonal and horizontal-diagonal to $(t_1, t_2) \in \mathcal{T}'_{ij}$. If the line segments of jobs i and j intersect in more than one point, we can always find an intersection point of the previous characterizations along the intersecting line. It follows that $(t_1, t_2) \in \mathcal{T}$ which shows the theorem. □

A.4 Proof of Lemmas 4-7

Lemma 4. *For any job $j \in J$ and $\bar{t}_1 \leq \theta_2$ the piecewise linear function $f_j(t_2) = d_j \cdot (\mu_j^{left}(\bar{t}_1, t_2) - \mu_j(\bar{t}_1, t_2))$ on the interval $[\theta_1, \theta_4]$ decomposes into the linear function segments*

$$\begin{aligned} f_j^1(t_2) &= d_j \cdot (t_2 - \theta_1)\,, \quad && t_2 \in [\theta_1, \theta_2] \\ f_j^2(t_2) &= d_j \cdot (\theta_2 - \theta_1)\,, \quad && t_2 \in [\theta_2, \theta_3] \\ f_j^3(t_2) &= -d_j \cdot (t_2 - \theta_4)\,, \quad && t_2 \in [\theta_3, \theta_4]. \end{aligned}$$

Proof. Since $\bar{t}_1 \leq \theta_2$, the function $\mu_j^{left}(\bar{t}_1, t_2)$ has slope one in the interval $t_2 \in [\theta_1, e_j + p_j]$ and zero otherwise. The function $\mu_j(\bar{t}_1, t_2)$ has slope one in the interval $t_2 \in [l_j - p_j, \theta_4]$ and zero otherwise. Hence $\mu_j^{left}(\bar{t}_1, t_2) - \mu_j(\bar{t}_1, t_2)$ has slope one the interval $[\theta_1, \theta_2]$, constant slope in $[\theta_2, \theta_3]$ and slope minus one in $[\theta_3, \theta_4]$. Scaling by d_j shows the statement. □

Lemma 5. *For any job $j \in J$ and $\bar{t}_1 \leq \theta_2$ the piecewise linear function $f_j(t_2) = d_j \cdot (t_2 - \mu_j(\bar{t}_1, t_2))$ on the interval $[\theta_1, \theta_4]$ decomposes into the linear function segments*

$$f_j^1(t_2) = d_j \cdot t_2 \ , \quad t_2 \in [\theta_1, l_j - p_j]$$
$$f_j^2(t_2) = d_j \cdot (l_j - p_j) \ , \quad t_2 \in [l_j - p_j, \theta_4].$$

Proof. Since $\bar{t}_1 \leq \theta_2$, the function $\mu_j(\bar{t}_1, t_2)$ has slope zero in $[\theta_1, l_j - p_j]$ and slope one in the interval $[l_j - p_j, \theta_4]$. Hence, the function $t_2 - \mu_j(\bar{t}_1, t_2)$ has slope one in the interval $[\theta_1, l_j - p_j]$ and slope zero in the interval $[l_j - p_j, \theta_4]$. Scaling by d_j shows the statement. □

Lemma 6. *Let $\theta' = \max\{\theta_3, \theta_4, \bar{t}_1\}$. For any job $j \in J$ and $\bar{t}_1 \in [e_j, l_j]$ the piecewise linear function $f_j(t_2) = d_j \cdot (\mu_j^{right}(\bar{t}_1, t_2) - \mu_j(\bar{t}_1, t_2))$ on the interval $[\theta', \infty)$ decomposes into the linear function segments*

$$f_j^1(t_2) = d_j \cdot (t_2 - \theta') \ , \quad t_2 \in [\theta', l_j]$$
$$f_j^2(t_2) = d_j \cdot (l_j - \theta') \ , \quad t_2 \in [l_j, \infty).$$

Proof. Since $\bar{t}_1 \in t_1 \in [e_j, l_j]$, the function $\mu_j^{right}(\bar{t}_1, t_2)$ has slope one in the interval $[\theta', l_j]$ and slope zero in the interval $[l_j, \infty)$. The function $\mu_j(\bar{t}_1, t_2)$ is constant for all $t_2 \in [\theta', \infty)$. Hence, the function $\mu_j^{right}(\bar{t}_1, t_2) - \mu_j(\bar{t}_1, t_2)$ has slope one in the interval $[\theta', l_j]$ and slope zero in the interval $[l_j, \infty)$. Scaling by d_j shows the statement. □

Lemma 7. *Let $\theta' = \max\{\theta_3, \theta_4, \bar{t}_1\}$. For any job $j \in J$ and $\bar{t}_1 \in [e_j, l_j]$ the piecewise linear function $f_j(t_2) = -d_j \cdot (\bar{t}_1 + \mu_j(\bar{t}_1, t_2))$ is constant on $[\theta', \infty)$.*

Proof. By construction, it holds $\mu_j(\bar{t}_1, t_2) = \mu_j(\bar{t}_1, \theta')$ for all $t_2 \in [\theta', \infty)$ which is constant. Consequently, $f_j(t_2)$ is constant for all $t_2 \in [\theta', \infty)$. □

A.5 Proof of Lemma 8

Lemma 8. *If the slopes of the functions (23) and (24) coincide on an interval $[\underline{t}_2, \bar{t}_2] \subset \mathbb{R}$ then both functions on the interval $[\underline{t}_2, \bar{t}_2]$ attain their maximum at the same point $(\bar{t}_1, t_2) \in \mathcal{T}$ with $t_2 \in [\underline{t}_2, \bar{t}_2]$, if the maximum exists.*

Proof. Since the slope of (23) equals the slope of (24) the quotient of the functions $\omega(\bar{t}_1, t_2) + d_j \cdot (\mu_j^{left}(\bar{t}_1, t_2) - \mu_j(\bar{t}_1, t_2))$ and $\omega(\bar{t}_1, t_2) + d_j \cdot (t_2 - \mu_j(\bar{t}_1, t_2))$ is constant for all $t_2 \in [\underline{t}_2, \bar{t}_2]$. Hence, if there exists an interval $(\bar{t}_1, t_2) \in \mathcal{T}$ with $t_2 \in [\underline{t}_2, \bar{t}_2]$ that maximizes (23) it also maximizes (24) and conversely. □

A.6 Proof of Lemmas 9-12

Lemma 9. *For fixed value $\bar{t}_1 \in [e_j, l_j]$ the function $\mu_j^{right}(\bar{t}_1, t_2) - 2 \cdot \mu_j(\bar{t}_1, t_2)$ has slope zero for all $t_2 \in [l_j, \infty)$.*

Proof. Both functions $\mu_j^{right}(\bar{t}_1, t_2)$ and $\mu_j(\bar{t}_1, t_2)$ have slope zero in the interval $t_2 \in [l_j, \infty)$, so the stated function has slope zero in this interval. □

Lemma 10. *For fixed value* $\bar{t}_1 \in [e_j, \theta_2]$ *the function* $t_2 - \mu_j^{left}(\bar{t}_1, t_2)$ *has slope one the interval* $t_2 \in [e_j + p_j, \theta_4]$.

Proof. The function $\mu_j^{left}(\bar{t}_1, t_2)$ has slope zero in the interval $t_2 \in [e_j + p_j, \theta_4]$, so $t_2 - \mu_j^{left}(\bar{t}_1, t_2)$ has slope one in the interval $t_2 \in [e_j + p_j, \theta_4]$. □

Lemma 11. *For fixed value* $\bar{t}_1 \in [e_j, \theta_2]$ *the function* $\mu_j^{right}(\bar{t}_1, t_2) - 2 \cdot \mu_j(\bar{t}_1, t_2)$ *has slope one in the interval* $[\max\{\theta_3, \theta_4\}, l_j]$.

Proof. The function $\mu_j^{right}(\bar{t}_1, t_2)$ has slope one and the function $\mu_j(\bar{t}_1, t_2)$ has slope zero in the interval $[\max\{\theta_3, \theta_4\}, l_j]$, so the stated function has slope one. □

Lemma 12. *For fixed value* $\bar{t}_1 \in [e_j, \theta_2]$ *the function* $t_2 - \mu_j^{left}(\bar{t}_1, t_2)$ *has slope zero in the interval* $[t_1, \theta_2]$.

Proof. The function $\mu_j^{left}(\bar{t}_1, t_2)$ has slope one in the interval $[t_1, \theta_2]$, therefore $t_2 - \mu_j^{left}(\bar{t}_1, t_2)$ has slope zero. □

B Algorithms

Notes for the algorithms:

- $(j, t_2, \tau_2) \in O_3(t_1) \iff (j, t_2 + t_1, \tau_2) \in O_3$
- the computation of the energy overloads $\omega(t_1, t_2)$ is analogous to the checker of Baptiste et al. [5] and involves dynamic slope updates

Algorithm 1: Dynamic $\mathcal{O}(n^2)$ interval checker

Input: CuSP instance
Output: false, if there exists a time interval positive overload

$O_1 \leftarrow \{(i, e_i, T_1) \mid i \in J\} \cup \{(i, l_i - p_i, T_1^M) \mid i \in J^M\}$
$O_2 \leftarrow \{(i, l_i, T_2) \mid i \in J\}$
$O_3 \leftarrow \emptyset$
$t_1^{old} \leftarrow \infty$
for $(i, t_1, \tau_1) \in O_1$ *non-decreasing* t_1 **do**
  if $t_1 = t_1^{old}$ **then** *continue*
  $t_1^{old} \leftarrow t_1$
  for $curr = (j, t_2, \tau_2) \in O_2 \cup O_3(t_1)$ *non-decreasing* $t_2 > t_1$ **do**
    if $\omega(t_1, t_2) > 0$ **then return** $false$ `// see Baptiste et al.`
    if $t_1 \geq e_j + p_j$ **then** `// update list elements of` O_2 `and` O_3
      if $curr \in O_2$ **then** $O_2.delete(curr)$
      else $O_3.delete(curr)$
    else if $\tau_2 = T_2 \wedge t_1 = e_j$ **then**
      $O_2.delete(curr)$
      if $e_j + p_j = l_i$ **then** $O_2.insert(j, t_2, T_2^M)$
      else $O_3.insert(i, e_i + l_i, T_3)$
    else if $\tau_2 = T_3 \wedge t_1 = l_j - p_j$ **then**
      $O_3.delete(curr)$
      $O_2.insert(j, t_2, T_2^M)$
return *true*

Algorithm 2: Reduced $\mathcal{O}(n^3)$ propagator

Input: CuSP instance
Output: false, if there exists a time interval with positive overload, otherwise propagate earliest start and latest completion times

$O_1 \leftarrow \{(j, e_j, T_1) \mid j \in J\} \cup \{(j, l_j - p_j, T_1^M) \mid j \in J\}$
$O_2 \leftarrow \{(j, l_j, T_2) \mid j \in J\} \cup \{(j, e_j + p_j, T_2^M) \mid j \in J\}$
$O_3 \leftarrow \{(j, e_j + l_j, T_3) \mid j \in J\}$
$t_1^{old} \leftarrow \infty$

for $(i, t_1, \tau_1) \in O_1$ *non-decreasing* t_1 **do**
  if $t_1 = t_1^{old}$ **then** *continue*
  $t_1^{old} = t_1$
  for $curr = (j, t_2, \tau_2) \in O_2 \cup O_3(t_1)$ *non-decreasing* $t_2 > t_1$ **do**
    if $\omega(t_1, t_2) > 0$ **then return** *false* `// see Baptiste et al.`
    `// left- and right-shift propagations`
    for $j \in J$ **do**
      if $\omega(t_1, t_2) + d_j \cdot (\mu_j^{left}(t_1, t_2) - \mu_j(t_1, t_2)) > 0$ **then**
        $e_j \leftarrow t_2 - \mu_j(t_1, t_2) + \left\lceil \frac{\omega(t_1, t_2)}{d_j} \right\rceil$
      if $\omega(t_1, t_2) + d_j \cdot (\mu_j^{right}(t_1, t_2) - \mu_j(t_1, t_2)) > 0$ **then**
        $l_j \leftarrow t_1 + \mu_j(t_1, t_2) - \left\lceil \frac{\omega(t_1, t_2)}{d_j} \right\rceil$

return *true*

Algorithm 3: $\mathcal{O}(n^2 \log n)$ Sweep Line Propagator

Input: CuSP instance
Output: false, if there exists a time interval with positive overload, otherwise propagate earliest start and latest completion times

$O_1 \leftarrow \{(j, e_j, T_1) \mid i \in J\} \cup \{(j, l_j - p_j, T_1^M) \mid j \in J\}$
$O_2 \leftarrow \{(j, l_j, T_2) \mid i \in J\} \cup \{(j, e_j + p_j, T_2^M) \mid j \in J\}$
$O_3 \leftarrow \{(j, e_j + l_j, T_3) \mid i \in J\}$
$t_1^{old} \leftarrow \infty$
$e'_j \leftarrow e_j \quad \forall j \in J$
$l'_j \leftarrow l_j \quad \forall j \in J$
for $(i, t_1, \tau_1) \in O_1$ *non-decreasing* t_1 **do**
    if $t_1 = t_1^{old}$ **then** *continue*
    $t_1^{old} = t_1$
    $t_2^{old} \leftarrow \infty$
    for $curr = (j, t_2, \tau_2) \in O_2 \cup O_3(t_1)$ *non-decreasing* $t_2 > t_1$ **do**
        if $\omega(t_1, t_2) > 0$ **then return** $false$ `// see Baptiste et al.`
        if $t_2 \neq t_2^{old}$ **then**
            $\mathcal{P} \leftarrow \mathcal{P} \cup \{(t_2, \omega(t_1, t_2))\}$ `// add points`
        $t_2^{old} \leftarrow t_2$
    $\mathcal{L} \leftarrow initializeLineSegments(t_1)$ `// add line segments`
    if $\mathcal{P} \neq \emptyset \wedge \mathcal{L} \neq \emptyset$ **then**
        $sweepLinePropagation(\mathcal{P}, \mathcal{L}, e', l', t_1)$ `// propagate`
for $j \in J$ **do**
    if $e'_j > e_j$ **then** $e_j = e'_j$
    if $l'_j < l_j$ **then** $l_j = l'_j$
    if $e_j + p_j > l_j$ **then return** $false$
return *true*

Algorithm 4: Initialize Line Segments

Input: jobs J, lower interval limit t_1
Output: set of line segments $\mathcal{L}$
$\mathcal{L} \leftarrow \emptyset$

for $j \in J$ **do**
  $\theta_1 = \max\{e_j, t_1\}$
  $\theta_2 = \min\{e_j + p_j, l_j - p_j\}$
  $\theta_3 = \max\{e_j + p_j, l_j - p_j\}$
  $\theta_4 = \min\{l_j, e_j + l_j - t_1\}$
  // add lines
  // format: (job, $\underline{x}_l, \overline{x}_l$, slope, left-/right-shift)
  if $t_1 \in [e_j, \theta_2)$ **then**
    // exact
    $\mathcal{L} \leftarrow \mathcal{L} \cup line(j, l_j, \infty, 0, right)$ // region A
    $\mathcal{L} \leftarrow \mathcal{L} \cup line(j, \theta_1, \theta_2, d_j, left)$ // region C
    // this line segment is always added
    $\mathcal{L} \leftarrow \mathcal{L} \cup line(j, \theta_2, \theta_3, 0, left)$ // region B or C
    // non-exact
    // add slopes of overload and update function
    if $e_j + p_j < l_j - p_j$ **then**
      $\mathcal{L} \leftarrow \mathcal{L} \cup line(j, \theta_2, \theta_3, d_j, left)$ // region B
    $\mathcal{L} \leftarrow \mathcal{L} \cup line(j, \theta_3, \theta_4, 0, left)$ // region B
    $\mathcal{L} \leftarrow \mathcal{L} \cup line(j, \theta_3, \theta_4, -d_j, left)$ // region B
    $\mathcal{L} \leftarrow \mathcal{L} \cup line(j, \theta_4, l_j, d_j, right)$ // region B'
    $\mathcal{L} \leftarrow \mathcal{L} \cup line(j, \theta_4, l_j, 0, right)$ // region B'
  else if $t_1 \in [\theta_2, \theta_3)$ **then**
    // exact
    $\mathcal{L} \leftarrow \mathcal{L} \cup line(j, l_j, \infty, 0, right)$ // region A
    // non-exact
    // add slopes of overload and update function
    if $e_j + p_j < l_j - p_j$ **then**
      $\mathcal{L} \leftarrow \mathcal{L} \cup line(j, \theta_3, l_j, d_j, right)$ // region B'
      $\mathcal{L} \leftarrow \mathcal{L} \cup line(j, \theta_3, l_j, 0, right)$ // region B'
  else if $t_1 \in [\theta_3, l_j)$ **then**
    // exact
    $\mathcal{L} \leftarrow \mathcal{L} \cup line(j, l_j, \infty, 0, right)$ // region A
return $\mathcal{L}$

Algorithm 5: Sweep Line Propagation

Input: points $\mathcal{P}$, line segments $\mathcal{L}$, earliest start times e'_j, latest completion times l'_j, lower interval limit t_1

Output: updated start times e'_j and completion times l'_j for all $j \in J$

$a_0 \leftarrow \min\{a_l \mid l \in \mathcal{L}\}$ `// start slope`

$H \leftarrow \emptyset$ `// empty event heap`

for $l \in \mathcal{L}$ **do** `// fill heap with evaluation events`

  $H.insert(a_l, l, evaluation)$

$B \leftarrow initializeIntervalTree(\mathcal{P}, a_0, H)$ `// build interval tree`

while $\neg H.empty$ **do** `// sweep over all events`

  $event \leftarrow H.extractMin$

  if $event.type = evaluation$ **then**

    $l \leftarrow event.getLine$

    $j \leftarrow l.job$

    $(t_2, \omega(t_1, t_2)) \leftarrow B.evaluate(l)$ `// evaluate`

    if $l.shift = left \wedge \omega(t_1, t_2) + d_j \cdot (\mu_j^{left}(t_1, t_2) - \mu_j(t_1, t_2)) > 0$ **then**

      $update \leftarrow t_2 - \mu_j(t_1, t_2) + \left\lceil \frac{\omega(t_1, t_2)}{d_j} \right\rceil$

      if $update > e'_j$ **then** $e'_j \leftarrow update$

    else if $l.shift = right \wedge \omega(t_1, t_2) + d_j \cdot (\mu_j^{right}(t_1, t_2) - \mu_j(t_1, t_2)) > 0$ **then**

      $update \leftarrow t_1 + \mu_j(t_1, t_2) - \left\lceil \frac{\omega(t_1, t_2)}{d_j} \right\rceil$

      if $update < l'_j$ **then** $l'_j \leftarrow update$

  else if $event.type = resolve$ **then**

    $B.resolve(event.getTreeNode, H)$ `// resolve`

return e', l'

Algorithm 6: Initialize Interval Tree

Input: set of points $\mathcal{P}$, initial slope a_0 event heap H
Output: interval tree B

$B \leftarrow$ binary interval tree with $2^{\lceil \log_2 \mathcal{P} \rceil + 1}$ empty nodes
$\mathcal{P} \leftarrow \mathcal{P}$ sorted by x_q first then by y_q for all $q \in \mathcal{P}$ in non-decreasing order
$q \leftarrow \mathcal{P}.first$

`// leaf nodes`

for $v \in V_B^{leaf}$ *from left to right* $\wedge\ q \neq \mathcal{P}.end$ **do**
    $[\underline{x}_v, \overline{x}_v] = [x_q, x_q]$
    $\pi_v \leftarrow v$
    $\alpha_v \leftarrow \infty$
    $\beta_v \leftarrow \infty$
    $q \leftarrow \mathcal{P}.next$

`// non-leaf nodes`

for $v \in V_B^{noleaf}$ *sorted from bottom to top and from left to right* **do**
    if $v.left = null$ **then** *continue*
    if $v.right \neq null$ **then**
        $[\underline{x}_v, \overline{x}_v] \leftarrow [\underline{x}_{v.left}, \overline{x}_{v.right}]$
        $\beta_v \leftarrow \min\{\alpha_{v.left}, \beta_{v.left}, \alpha_{v.right}, \beta_{v.right}\}$
        if $a_0 \cdot \pi_{v.left}.x + \pi_{v.left}.y > a_0 \cdot \pi_{v.right}.x + \pi_{v.right}.y$ **then**
            $\pi_v \leftarrow \pi_{v.left}$
        else $\pi_v \leftarrow \pi_{v.right}$
        if $\pi_{v.left}.x \neq \pi_{v.right}.x$ **then**
            $val \leftarrow \frac{\pi_{v.right}.y - \pi_{v.left}.y}{\pi_{v.left}.x - \pi_{v.right}.x}$
            if $val > a_0$ **then**
                $\alpha_v \leftarrow val$
                if $\alpha_v < \beta_v$ **then**
                    $H.insert(\alpha_v, v, resolve)$ `// add resolve event`
            else $\alpha_v \leftarrow \infty$
        else $\alpha_v \leftarrow \infty$
    else
        $[\underline{x}_v, \overline{x}_v] \leftarrow [\underline{x}_{v.left}, \overline{x}_{v.left}]$
        $\pi_v \leftarrow \pi_{v.left}$
        $\alpha_v \leftarrow \infty$
        $\beta_v \leftarrow \min\{\alpha_{v.left}, \beta_{v.left}\}$

return B

Algorithm 7: Evaluate

Input: interval tree B, line segment $l \in \mathcal{L}$
Output: point $q^* = (x_{q^*}, y_{q^*}) \in \mathcal{P}$ with
$q^* = \arg\max_{q \in \mathcal{P} : x_q \in [\underline{x}_l, \overline{x}_l]} a_l \cdot x_q + y_q$

$max \leftarrow -\infty$
$S \leftarrow \emptyset$ `// empty stack`
$S.push(B.root)$
while $\neq S.empty$ **do**
  $v \leftarrow S.pop$
  if $v = null$ **then** *continue*
  if $[\underline{x}_v, \overline{x}_v] \subseteq [\underline{x}_l, \overline{x}_l]$ **then**
    $val \leftarrow a_l \cdot \pi_v.x + \pi_v.y$
    if $val > max$ **then**
      $max \leftarrow val$
      $q^* \leftarrow \pi_v$
  else if $[\underline{x}_v, \overline{x}_v] \cap [\underline{x}_l, \overline{x}_l] \neq \emptyset$ **then**
    $S.push(v.left)$
    $S.push(v.right)$
return q^*

Algorithm 8: Resolve

Input: interval tree B, interval tree node $v \in V_B$, event heap H
Output: updates the interval tree and propagates changes to the root node, add new resolve events to heap

`// resolve current node`
$\pi_v \leftarrow \pi_{v.right}$
$\alpha_v \leftarrow \infty$
$\beta_v \leftarrow \min\{\alpha_{v.right}, \beta_{v.right}\}$
`// propagate changes to root node`
$prev \leftarrow v$
$v \leftarrow v.parent$
while $prev \neq T.root$ **do**
  if $v.right \neq null$ **then**
    if $\alpha_v < \infty$ **then**
      if $\pi_{v.right}.x \neq \pi_{v.left}.x$ **then**
        $\alpha_v \leftarrow \frac{\pi_{v.left}.y - \pi_{v.right}.y}{\pi_{v.right}.x - \pi_{v.left}.x}$
      else $\alpha_v \leftarrow \infty$
      $\beta_v \leftarrow \min\{\alpha_{v.left}, \beta_{v.left}, \alpha_{v.right}, \beta_{v.right}\}$
    else $\beta_v \leftarrow \min\{\alpha_{v.right}, \beta_{v.right}\}$
    if $\alpha_v < \beta_v$ **then**
      $H.insert(\alpha_v, v, resolve)$ `// add resolve event`
  else $\beta_v \leftarrow \min\{\alpha_{v.left}, \beta_{v.left}\}$
  $prev \leftarrow v$
  $v \leftarrow v.parent$