Zuse Institute Berlin

Takustr. 7
14195 Berlin
Germany

ALEXANDER TESCH

Exact Energetic Reasoning in $\mathcal{O}(n^2 \log^2 n)$

ZIB Report 16-46 (September 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

Exact Energetic Reasoning in $\mathcal{O}(n^2 \log^2 n)$

Alexander Tesch

Zuse Institute Berlin (ZIB)
Takustr. 7, 14195 Berlin, Germany
tesch@zib.de

Abstract. In this paper, we address the *Energetic Reasoning* propagation rule for the *Cumulative Scheduling Problem* (CuSP). An energetic reasoning propagation algorithm is called *exact*, if it computes the maximum possible energetic reasoning propagation for all the jobs. The currently best known exact energetic reasoning algorithm has complexity $\mathcal{O}(n^3)$, see [1]. In this paper, we present a new exact energetic reasoning propagation algorithm with improved complexity of $\mathcal{O}(n^2 \log^2 n)$.

1 Introduction

We consider the *Cumulative Scheduling Problem* (CuSP), in which we are given a set J of n jobs where each job $j \in J$ has a processing time $p_j > 0$ and a resource demand $d_j > 0$. Moreover, each job $j \in J$ is given a scheduling interval $[e_j, l_j] \subset \mathbb{R}$ in which it has to be processed. For given resource capacity D, the objective is to find feasible starting times $s_j \in [e_j, l_j - p_j]$ such that at any time point the total resource consumption does not exceed the total capacity. More formally, the CuSP can be formulated as the feasibility problem

$$\text{find} \qquad s \in \mathbb{R}^n \tag{1}$$

$$\text{such that} \quad \sum_{s_j \le t < s_j + p_j} d_j \le D \qquad \forall t \in \mathbb{R} \tag{2}$$

$$e_j \le s_j \le l_j - p_j \qquad \forall j \in J. \tag{3}$$

Normally, the CuSP is used as subroutine in more complex scheduling algorithms. In constraint programming, the CuSP is modeled as CUMULATIVE constraint. Usually, the CuSP is solved by specific branching and propagation methods. The most common propagations algorithms for the CuSP are *Time-Tabling* [10], *Edge-Finding* [7, 14], *Extended Edge-Finding* [8, 9], *Time-Table Edge-Finding* [12, 15], *Energetic Reasoning* [1, 2, 6] and *Not First-/Not Last* [11].

In this paper, we focus on energetic reasoning which, in general, constitutes a very strong propagation rule for the CuSP. Except for Not First-/Not Last, all stated propagation algorithms are relaxations of energetic reasoning. Due to the high running time of $\mathcal{O}(n^3)$ energetic reasoning is avoided in practice, since faster but weaker propagation algorithms generate better results [12, 15]. However, since its first practical examination in Baptiste et al. [1], it remained open if

there is a faster implementation of energetic reasoning than the straightforward $\mathcal{O}(n^3)$ algorithm.

Many approaches have been made to improve the practical complexity of energetic reasoning. In Berthold et al. [2], they introduce an approximation scheme in order to detect time intervals which do not contribute to any propagation of energetic reasoning. Such intervals can be eliminated in the search. Similarly, Derrien et al. [5] strengthen the original characterization of relevant time intervals of Baptiste et al. [1] which shows a decent practical impact. However, the general complexity still remains at $\mathcal{O}(n^3)$.

Recently, Bonifas [3] introduced an algorithm that computes energetic reasoning propagations for at least one job in $\mathcal{O}(n^2 \log n)$. His idea is based on a new geometric interpretation of energetic reasoning. In order to compute propagations for all jobs, his algorithm can be extended to an $\mathcal{O}(n^3 \log n)$ algorithm.

More recently, the algorithm was improved in Tesch [13] where an $\mathcal{O}(n^2 \log n)$ algorithm is proposed to compute energetic reasoning propagations for all jobs. The idea is similar to Bonifas' algorithm, but a more comprehensive subproblem is studied. From this context, a specific sweep line algorithm is developed which solves the subproblem efficiently. It is also shown that the algorithm computes exact propagations on the majority of relevant intervals.

In general, however, no algorithm for energetic reasoning was known that computes exact propagations for all jobs in time less than $\mathcal{O}(n^3)$.

Contribution. In this article, we proceed from a related geometric and algorithmic concept as in [13]. The main difference is the identification of a key property in the problem that can be handled well algorithmically by using a slightly extended data structure. This leads to an exact energetic reasoning algorithm with improved running time of $\mathcal{O}(n^2 \log^2 n)$.

Outline. Our paper is organized as follows. In Section 2, we introduce the basic principles of energetic reasoning. In Section 3, we prepare the main concepts to translate energetic reasoning into a geometric environment. Section 4, introduces a sweep line algorithm and its related data structres that solves the associated geometric problem. Finally, we conclude our results in Section 5.

2 Exact Energetic Reasoning

Given a job $j \in J$ and a time interval $[t_1, t_2] \subset \mathbb{R}$ define

$$\mu_j(t_1, t_2) = \min\{p_j, t_2 - t_1, \max\{0, e_j + p_j - t_1\}, \max\{0, t_2 - l_j + p_j\}\} \quad (4)$$

as the *minimum left-/right-shift duration* which is the minimum duration that job j is processed in the interval $[t_1, t_2]$. Furthermore, define the *energy overload* in a time interval $[t_1, t_2] \subset \mathbb{R}$ by

$$\omega(t_1, t_2) = \sum_{j \in J} d_j \cdot \mu_j(t_1, t_2) - D \cdot (t_2 - t_1) \quad (5)$$

which is the slack between the consumed and the available energy in the interval $[t_1, t_2]$. Apparently, if there is a time interval $[t_1, t_2] \subset \mathbb{R}$ with $\omega(t_1, t_2) > 0$ then the CuSP is infeasible. This feasibility condition is known as *overload checking* and can be tested in $\mathcal{O}(n^2)$, see Baptiste et al. [1].

Energetic reasoning extends this rule by start- and end time propagations. Given a job $j \in J$ define the *left-* and *right-shift duration* in a time interval $[t_1, t_2] \subset \mathbb{R}$ by

$$\mu_j^{left}(t_1, t_2) = \min\{e_j + p_j, t_2\} - \max\{e_j, t_1\} \tag{6}$$
$$\mu_j^{right}(t_1, t_2) = \min\{l_j, t_2\} - \max\{l_j - p_j, t_1\} \tag{7}$$

which is the duration of job j in the interval $[t_1, t_2]$, if job j is started as early as possible (left-shift) or as late as possible (right-shift). Then define the *left-* and *right-shift overload* of a job $j \in J$ in a time interval $[t_1, t_2] \subset \mathbb{R}$ as

$$\omega_j^{left}(t_1, t_2) = \omega(t_1, t_2) + d_j \cdot (\mu_j^{left}(t_1, t_2) - \mu_j(t_1, t_2)) \tag{8}$$
$$\omega_j^{right}(t_1, t_2) = \omega(t_1, t_2) + d_j \cdot (\mu_j^{right}(t_1, t_2) - \mu_j(t_1, t_2)) \tag{9}$$

which is the energy overload in the interval $[t_1, t_2]$, if job j is left- or right-shifted.

Theorem 1 (Baptiste et al. [1]). *If there is a job $j \in J$ and a time interval $[t_1, t_2] \subset \mathbb{R}$ such that $\omega_j^{left}(t_1, t_2) > 0$ holds then the earliest start time e_j of job j can be updated by*

$$e_j = t_2 - \mu_j(t_1, t_2) + \left\lceil \frac{\omega(t_1, t_2)}{d_j} \right\rceil. \tag{10}$$

Analogously, if $\omega_j^{right}(t_1, t_2) > 0$ holds then the latest completion time l_j of job j can be updated by

$$l_j = t_1 + \mu_j(t_1, t_2) - \left\lceil \frac{\omega(t_1, t_2)}{d_j} \right\rceil. \tag{11}$$

Note that the left- and right-shift case of energetic reasoning are equivalent by symmetry at time $t = 0$. The idea of energetic reasoning works as follows. Assume a job $j \in J$ is left-shifted. If the additional amount of energy that is caused by left-shifting job j exceeds the available energy in some time interval $[t_1, t_2] \subset \mathbb{R}$ then the earliest start time e_j is invalid. Hence, the earliest start time e_j can be increased by an amount such that left-shifting job j causes no overload anymore in the interval $[t_1, t_2]$. That is exactly expressed by the update function (10). The right-shift case (11) works equally.

Moreover, there exists an explicit description for the set of *relevant time intervals* $\mathcal{T} \subset \mathbb{R}^2$ for energetic reasoning, see for example Baptiste et al. [1], Derrien et al. [5] and Tesch [13]. Notably, there are $\mathcal{O}(n^2)$ many of such relevant time intervals.

Definition 1. *A propagation algorithm for energetic reasoning is called exact, if it computes for all jobs $j \in J$ both*

$$\max_{(t_1,t_2)\in\mathcal{T}:\ \omega_j^{left}(t_1,t_2)>0} \quad t_2 - \mu_j(t_1,t_2) + \left\lceil \frac{\omega(t_1,t_2)}{d_j} \right\rceil \tag{12}$$

$$\min_{(t_1,t_2)\in\mathcal{T}:\ \omega_j^{right}(t_1,t_2)>0} \quad t_1 + \mu_j(t_1,t_2) - \left\lceil \frac{\omega(t_1,t_2)}{d_j} \right\rceil \tag{13}$$

that is the maximum start- and end time propagations according to the energetic reasoning rule.

Thus, we aim to construct an algorithm that solves the problems (12) and (13) efficiently.

3 Slack Polyhedra

In this section, we convert problems (12) and (13) to geometric problems in $\mathbb{R}^2$.

First, we extract the slack functions from definitions (8) and (9) and define

$$g_j^{left}(t_1,t_2) = d_j \cdot (\mu_j^{left}(t_1,t_2) - \mu_j(t_1,t_2)) \tag{14}$$

$$g_j^{right}(t_1,t_2) = d_j \cdot (\mu_j^{right}(t_1,t_2) - \mu_j(t_1,t_2)) \tag{15}$$

for every job $j \in J$. In particular, it holds $g_j^{left}(t_1,t_2) \geq 0$ and $g_j^{right}(t_1,t_2) \geq 0$ for all $(t_1,t_2) \in \mathbb{R}^2$. Moreover, due to the update functions (10) and (11) define

$$f_j^{left}(t_1,t_2) = d_j \cdot (t_2 - \mu_j(t_1,t_2)) \tag{16}$$

$$f_j^{right}(t_1,t_2) = -d_j \cdot (t_1 + \mu_j(t_1,t_2)) \tag{17}$$

for every job $j \in J$. The exactness conditions (12) and (13) can then be written equivalently for all jobs $j \in J$ as

$$\max_{(t_1,t_2)\in\mathcal{T}:\omega(t_1,t_2)+g_j^{left}(t_1,t_2)>0} \quad \omega(t_1,t_2) + f_j^{left}(t_1,t_2) \tag{18}$$

$$\max_{(t_1,t_2)\in\mathcal{T}:\omega(t_1,t_2)+g_j^{right}(t_1,t_2)>0} \quad \omega(t_1,t_2) + f_j^{right}(t_1,t_2) \tag{19}$$

because the time intervals where the maximum is attained in (12) and (13) are the same. As shown in [13], the functions $g_j^{left}(t_1,t_2)$ and $g_j^{right}(t_1,t_2)$ can be intepreted as three-dimensional slack polyhedra. In the next step, we restrict problems (18) and (19) to two dimensions.

3.1 Two Dimensions

Our main algorithm iterates over all relevant t_1 values in an outer loop. In the following we consider one fixed iteration with fixed value $\bar{t}_1 \in \mathbb{R}$. Then

problems (18) and (19) translate to

$$\max_{(\bar{t}_1,t_2)\in\mathcal{T}:\omega(\bar{t}_1,t_2)+g_j^{left}(\bar{t}_1,t_2)>0} \omega(\bar{t}_1,t_2)+f_j^{left}(\bar{t}_1,t_2) \tag{20}$$

$$\max_{(\bar{t}_1,t_2)\in\mathcal{T}:\omega(\bar{t}_1,t_2)+g_j^{right}(\bar{t}_1,t_2)>0} \omega(\bar{t}_1,t_2)+f_j^{right}(\bar{t}_1,t_2) \tag{21}$$

for all jobs $j \in J$ which are two-dimensional problems in the (t_2, ω)-plane. Note that for fixed value $\bar{t}_1 \in \mathbb{R}$, the algorithm of Baptiste et al. [1] computes all relevant values $\omega(\bar{t}_1, t_2)$ with $(\bar{t}_1, t_2) \in \mathcal{T}$ in $\mathcal{O}(n)$. Thus, in the following we assume these values are given. The geometric version of problem (20) computes the point $(t_2, \omega(\bar{t}_1, t_2)) \in \mathbb{R}^2$ with $(\bar{t}_1, t_2) \in \mathcal{T}$ that lies above the function $-g_j^{left}(\bar{t}_1, t_2)$ and which has maximal distance to the function $-f_j^{left}(\bar{t}_1, t_2)$. The right-shift case (21) states equally.

The functions $g_j^{left}(\bar{t}_1, t_2)$ and $f_j^{left}(\bar{t}_1, t_2)$ are piecewise linear, respectively $g_j^{right}(\bar{t}_1, t_2)$ and $f_j^{right}(\bar{t}_1, t_2)$. In the next section, we decompose them into pairs of linear function segments over the same domain.

3.2 Decomposition into Line Segment Pairs

For any job $j \in J$ and any value $\bar{t}_1 \in \mathbb{R}$ define the parameters

$$\begin{aligned} \theta_1 &= \max\{e_j, \bar{t}_1\}, & \theta_2 &= \min\{e_j + p_j, l_j - p_j\}, \\ \theta_3 &= \max\{e_j + p_j, l_j - p_j\}, & \theta_4 &= \min\{l_j, e_j + l_j - \bar{t}_1\}, \\ \theta_5 &= \max\{e_j + l_j - \bar{t}_1, l_j, \bar{t}_1\}. \end{aligned}$$

We may assume that $\omega(\bar{t}_1, t_2) \leq 0$ for all $(\bar{t}_1, t_2) \in \mathcal{T}$, otherwise the CuSP is already infeasible. Therefore, the exactness conditions in (20) and (21) allow us to restrict to interval regions where it holds $g_j^{left}(\bar{t}_1, t_2) > 0$ and $g_j^{right}(\bar{t}_1, t_2) > 0$. In this connection, define the interval regions

$$\mathcal{T}_j^{left} = \{(\bar{t}_1, t_2) \in \mathbb{R}^2 \mid (\bar{t}_1, t_2) \in (-\infty, \theta_2] \times [\theta_1, \theta_4]\} \tag{22}$$

$$\mathcal{T}_j^{right} = \{(\bar{t}_1, t_2) \in \mathbb{R}^2 \mid (\bar{t}_1, t_2) \in [e_j, l_j] \times [\theta_5, \infty)\} \tag{23}$$

which are mutually disjoint. That implies that a time interval $[t_1, t_2] \subset \mathbb{R}$ is either suited for a left-shift propagation, right-shift propagation or none of both.

Lemma 1. *Given a job $j \in J$ and an interval $(t_1, t_2) \in \mathbb{R}^2$ then $g_j^{left}(t_1, t_2) > 0$ implies $(t_1, t_2) \in \mathcal{T}_j^{left}$ and $g_j^{right}(t_1, t_2) > 0$ implies $(t_1, t_2) \in \mathcal{T}_j^{right}$.*

Proof. If $(t_1, t_2) \notin \mathcal{T}_j^{left}$ holds, it follows $\mu_j^{left}(t_1, t_2) = \mu_j(t_1, t_2)$ which implies $g_j^{left}(t_1, t_2) = 0$. Similarly, if $(t_1, t_2) \notin \mathcal{T}_j^{right}$ holds, it follows $\mu_j^{right}(t_1, t_2) = \mu_j(t_1, t_2)$ which implies $g_j^{right}(t_1, t_2) = 0$. This means, if the functions $g_j^{left}(t_1, t_2)$ and $g_j^{right}(t_1, t_2)$ are positive then only in the interval regions $\mathcal{T}_j^{left}$ and $\mathcal{T}_j^{right}$. □

By Lemma 1, we will restrict to the interval regions $\mathcal{T}_j^{left}$ and $\mathcal{T}_j^{right}$.

Lemma 2 (Left-Shift Decomposition). *For all jobs $j \in J$ and any fixed value $\bar{t}_1 \leq \theta_2$ the piecewise linear functions $g_j^{left}(\bar{t}_1, t_2)$ and $f_j^{left}(\bar{t}_1, t_2)$ on the interval $[\theta_1, \theta_4]$ decompose into the linear function segments pairs:*

(i) $t_2 \in [\theta_1, \theta_2]$

$$\begin{aligned} g_{j,1}^{left}(\bar{t}_1, t_2) &= d_j \cdot (t_2 - \theta_1) \\ f_{j,1}^{left}(\bar{t}_1, t_2) &= d_j \cdot t_2 \end{aligned} \tag{24}$$

(ii) $t_2 \in [\theta_2, \theta_3]$

$$\begin{aligned} g_{j,2}^{left}(\bar{t}_1, t_2) &= d_j \cdot (\theta_2 - \theta_1) \\ f_{j,2}^{left}(\bar{t}_1, t_2) &= \begin{cases} d_j \cdot t_2 & e_j + p_j \leq l_j - p_j \\ d_j \cdot (l_j - p_j) & e_j + p_j > l_j - p_j \end{cases} \end{aligned} \tag{25}$$

(iii) $t_2 \in [\theta_3, \theta_4]$

$$\begin{aligned} g_{j,3}^{left}(\bar{t}_1, t_2) &= d_j \cdot (t_2 - \theta_4) \\ f_{j,3}^{left}(\bar{t}_1, t_2) &= d_j \cdot (l_j - p_j). \end{aligned} \tag{26}$$

Lemma 3 (Right-Shift Decomposition). *For all jobs $j \in J$ and any fixed value $\bar{t}_1 \in [e_j, l_j]$ the piecewise linear functions $g_j^{right}(\bar{t}_1, t_2)$ and $f_j^{right}(\bar{t}_1, t_2)$ on the interval $[\theta_5, \infty)$ decompose into the linear function segments pairs:*

(i) $t_2 \in [\theta : 5, l_j]$

$$\begin{aligned} g_{j,1}^{right}(\bar{t}_1, t_2) &= d_j \cdot (t_2 - \theta_5) \\ f_{j,1}^{right}(\bar{t}_1, t_2) &= -d_j \cdot (\bar{t}_1 + \max\{0, e_j + p_j - \bar{t}_1\}) \end{aligned} \tag{27}$$

(ii) $t_2 \in [l_j, \infty)$

$$\begin{aligned} g_{j,2}^{right}(\bar{t}_1, t_2) &= d_j \cdot (l_j - \theta_5) \\ f_{j,2}^{right}(\bar{t}_1, t_2) &= -d_j \cdot (\bar{t}_1 + \max\{0, e_j + p_j - \bar{t}_1\}). \end{aligned} \tag{28}$$

The proofs of Lemmas 2 and 3 can be found in [13]. From Lemmas 2 and 3 we also derive that the linear function segments of $g_j^{left}(\bar{t}_1, t_2)$, $g_j^{right}(\bar{t}_1, t_2)$, $f_j^{left}(\bar{t}_1, t_2)$ and $f_j^{right}(\bar{t}_1, t_2)$ have slopes in $\{-d_j, 0, d_j\}$.

Lemma 4. *Let either $g_j = g_j^{left}$ and $f_j = f_j^{left}$ or $g_j = g_j^{right}$ and $f_j = f_j^{right}$. For all jobs $j \in J$ and any fixed value $\bar{t}_1 \in \mathbb{R}$ it holds one of the following*

(i) $g_j(\bar{t}_1, t_2)$ and $f_j(\bar{t}_1, t_2)$ have the same slope
(ii) $g_j(\bar{t}_1, t_2)$ or $f_j(\bar{t}_1, t_2)$ has slope zero

for any value $t_2 \geq \bar{t}_1$.

Proof. By Lemmas 2 and 3, the functions $g_j(\bar{t}_1, t_2)$ and $f_j(\bar{t}_1, t_2)$ can be decomposed into pairs of linear function segments. Comparing their slopes with respect to variable t_2 yields the statement. □

Lemma 4 is the *key lemma* of this paper, because the restrictions allow us to perform the computations more efficiently.

3.3 Points and Line Segments

We encode a line segment $l \subset \mathbb{R}^2$ in the plane by a 4-tuple $(a_l, b_l, \underline{x}_l, \overline{x}_l) \in \mathbb{R}^4$ such that $l = \{(x, y) \in \mathbb{R}^2 \mid y = a_l \cdot x + b_l \wedge x \in [\underline{x}_l, \overline{x}_l]\}$. In this respect, we write $l \simeq (a_l, b_l, \underline{x}_l, \overline{x}_l)$.

Consider a fixed value $\bar{t}_1 \in \mathbb{R}$ and job $j \in J$. If it holds $\bar{t}_1 \in (-\infty, \theta_2]$ define

$$\begin{aligned} \mathcal{L}_{j,1}^{left} &= \{(k,l) \subset \mathbb{R}^2 \times \mathbb{R}^2 \mid k \simeq (d_j, -e_j \cdot p_j, \theta_1, \theta_2) \wedge l \simeq (d_j, 0, \theta_1, \theta_2)\} \\ \mathcal{L}_{j,3}^{left} &= \{(k,l) \subset \mathbb{R}^2 \times \mathbb{R}^2 \mid k \simeq (-d_j, \theta_3 \cdot d_j, \theta_3, \theta_4) \wedge l \simeq (0, 0, \theta_3, \theta_4)\} \end{aligned}$$

and, if $e_j + p_j \leq l_j - p_j$ holds, define

$$\mathcal{L}_{j,2}^{left} = \{(k,l) \subset \mathbb{R}^2 \times \mathbb{R}^2 \mid k \simeq (0, p_j \cdot d_j, \theta_2, \theta_3) \wedge l \simeq (d_j, 0, \theta_2, \theta_3)\}$$

and otherwise, if $e_j + p_j > l_j - p_j$ holds, define

$$\mathcal{L}_{j,2}^{left} = \{(k,l) \subset \mathbb{R}^2 \times \mathbb{R}^2 \mid k \simeq (0, d_j \cdot \theta_2, \theta_2, \theta_3) \wedge l \simeq (0, 0, \theta_2, \theta_3)\}$$

as the line segment pairs of the left-shift decomposition.

Additionally, if it holds $\bar{t}_1 \in [e_j, l_j]$, define

$$\begin{aligned} \mathcal{L}_{j,1}^{right} &= \{(k,l) \subset \mathbb{R}^2 \times \mathbb{R}^2 \mid k \simeq (0, p_j \cdot d_j, l_j, \infty) \wedge l \simeq (0, 0, l_j, \infty)\} \\ \mathcal{L}_{j,2}^{right} &= \{(k,l) \subset \mathbb{R}^2 \times \mathbb{R}^2 \mid k \simeq (d_j, -\theta_5 \cdot d_j, \theta_5, l_j) \wedge l \simeq (0, 0, \theta_5, l_j)\} \end{aligned}$$

as the line segment pairs of the right-shift decomposition.

In particular, the line segments $\mathcal{L}_{j,1}^{left}, \mathcal{L}_{j,2}^{left}, \mathcal{L}_{j,3}^{left}$ correspond to the decompositions (24)-(26) and the sets $\mathcal{L}_{j,1}^{right}, \mathcal{L}_{j,2}^{right}$ correspond to the decompositions (27) and (28). Finally, for fixed value $\bar{t}_1 \in \mathbb{R}$ let

$$\mathcal{L} = \bigcup_{j \in J} \left(\mathcal{L}_{j,1}^{left} \cup \mathcal{L}_{j,2}^{left} \cup \mathcal{L}_{j,3}^{left} \cup \mathcal{L}_{j,1}^{right} \cup \mathcal{L}_{j,2}^{right} \right) \tag{29}$$

be the set of all such decomposed line segment pairs. Note that for simplification we have set $b_l = 0$ for any line segment pair $(k,l) \in \mathcal{L}$ because the latter geometric problem is invariant for any value of b_l. Furthermore, define

$$\mathcal{P} = \{(t_2, \omega(\bar{t}_1, t_2)) \in \mathbb{R}^2 \mid (\bar{t}_1, t_2) \in \mathcal{T}\} \tag{30}$$

as the point set in the plane which consists of all relevant overload values. The data sets $\mathcal{P}$ and $\mathcal{L}$ serve as input for the geometric problem of the next section.

4 Sweep Line Algorithm

In this section, we translate the subproblems (20) and (21) into a more general geometric context.

There we are given a set points $(x_q, y_q) \in \mathcal{P}$ in the plane and a set of line segment pairs $(k,l) \in \mathcal{L}$. A line segment $l \subset \mathbb{R}^2$ is represented by a 4-tuple

$(a_l, b_l, \underline{x}_l, \overline{x}_l)$ with slope a_l, intercept b_l and a domain $[\underline{x}_l, \overline{x}_l] \subset \mathbb{R}$. The line segment contains all points $(x, y) \in \mathbb{R}^2$ with $x \in [\underline{x}_l, \overline{x}_l]$ which satisfy the equation $y = a_l \cdot x + b_l$. For our problem, we assume that $[\underline{x}_k, \overline{x}_k] = [\underline{x}_l, \overline{x}_l]$ holds for all line segment pairs $(k, l) \in \mathcal{L}$ and $x_q \neq x_{q'}$ for all dual lines $q, q' \in \mathcal{P}$ with $q \neq q'$.

The geometric problem is to compute for all line segment pairs $(k, l) \in \mathcal{L}$

$$\max_{q \in \mathcal{P} : x_q \in [\underline{x}_k, \overline{x}_k] \wedge a_k \cdot x_q + y_q + b_k > 0} \quad a_l \cdot x_q + y_q + b_l$$

what is invariant for b_l, so we equivalently consider the problem

$$\max_{q \in \mathcal{P} : x_q \in [\underline{x}_k, \overline{x}_k] \wedge a_k \cdot x_q + y_q + b_k > 0} \quad a_l \cdot x_q + y_q. \tag{31}$$

That means we compute for all line segment pairs $(k, l) \in \mathcal{L}$ the point $q \in \mathcal{P}$ that lies above the mirrored line segment k such that the y-distance to the mirrored line segment l is maximal. Let us consider the dual version of the problem.

Dualization. Instead of examining the problem in the (x, y)-plane we convert the problem to the (a, y)-plane where the line segment slopes are continuous. In this case, every point $(x_q, y_q) \in \mathcal{P}$ converts to a dual line with slope x_q and intercept y_q which consists of all points $(a, y) \in \mathbb{R}^2$ that satisfy the equation $y = x_q \cdot a + y_q$. Moreover, every pair of line segments $(k, l) \in \mathcal{L}$ converts to two points $(a_k, b_k) \in \mathbb{R}^2$ and $(a_l, 0) \in \mathbb{R}^2$ since $b_l = 0$ holds in problem (31).

The dual problem is to find for every pair of dual points $(k, l) \in \mathcal{L}$ the dual line $q \in \mathcal{P}$ with maximum y-distance to the dual point $(a_l, 0)$ such the dual line q lies above the dual point (a_k, b_k) and has a slope $x_q \in [\underline{x}_l, \overline{x}_l]$.

In the following we present an algorithm that solves the dual problem. It is based on a *sweep line algorithm* [4] that sweeps over the dual points (a_k, b_k) and $(a_l, 0)$ with $(k, l) \in \mathcal{L}$ and efficiently retrieves the optimal dual line $(x_q, y_q) \in \mathcal{P}$ that intersects with the current sweep line. For this, we construct a modified version of a *range tree* [4] that efficiently performs queries of the form

$$\max_{q \in \mathcal{P} : (x_q, y_q) \in [\underline{x}, \overline{x}] \times [\underline{y}, \overline{y}]} \quad a \cdot x_q + y_q \tag{32}$$

for some current sweep value $a \in \mathbb{R}$. Additionally, we use an *event heap* to perform the sweeping and to update the domination order of the dual lines on the current sweep line.

Exploiting the Problem Structure. In the following we show that our main problem (31) can be solved by queries of the form (32). By Lemma 4, for any line segment pair $(k, l) \in \mathcal{L}$ at least one of the two statements hold: (i) $\tilde{a} = a_k = a_l$, (ii) $a_k = 0$ or $a_l = 0$. This leaves three possible cases.

For case (i), problem (31) turns into

$$\max_{q \in \mathcal{P} : x_q \in [\underline{x}_k, \overline{x}_k] \wedge \tilde{a} \cdot x_q + y_q + b_k > 0} \quad \tilde{a} \cdot x_q + y_q$$

which is equivalent to the problem

$$\max_{q \in \mathcal{P}: x_q \in [\underline{x}_k, \overline{x}_k]} \quad \tilde{a} \cdot x_q + y_q \tag{33}$$

since maximizing $\tilde{a} \cdot x_q + y_q$, in turn, maximizes $\tilde{a} \cdot x_q + y_q + b_k$ because b_k is constant. In the end, we only need to check if the condition $\tilde{a} \cdot x_q + y_q + b_k > 0$ holds for the dual line $q \in \mathcal{P}$ where the maximum is attained. We can perform this evaluation efficiently, since problem (33) is of the form (32).

For case (ii), problem (31) converts into one of the two problems:

$$\max_{q \in \mathcal{P}: x_q \in [\underline{x}_k, \overline{x}_k] \wedge y_q > -b_k} \quad a_l \cdot x_q + y_q \tag{34}$$

$$\max_{q \in \mathcal{P}: x_q \in [\underline{x}_k, \overline{x}_k] \wedge a_k \cdot x_q + y_q > -b_k} \quad y_q \tag{35}$$

where problem (34) is of the form (32) and thus can be evaluated efficiently. Conversely, problem (35) can be seen as the dual problem to (34). Instead of maximizing the objective function under some constraint we enforce the constraint to have maximal value under the condition that the objective function is bounded by some fixed value. In our range tree, such queries can also be performed efficiently. There we search for the highest value $y_q \in [\underline{y}, \overline{y}]$ while at the same time we require that the dual line $q \in \mathcal{P}$ with maximum value $a_k \cdot x_q + y_q$ satisfies $a_k \cdot x_q + y_q > -b_k$. The evaluation algorithms are described in more detail in Section 4.2.

4.1 Data Structures

In the following we explain the two data structures of our algorithm: a *two-dimensional range tree* and an *event heap*.

Range Tree. We construct a two-dimensional range tree that stores all points $\mathcal{P}$ such that we can efficiently find the point $(x_q, y_q) \in \mathcal{P}$ that lies in an arbitrary rectangle $[\underline{x}, \overline{x}] \times [\underline{y}, \overline{y}]$ and maximizes the linear function $a \cdot x_q + y_q$ for some current sweep value $a \in \mathbb{R}$. For an overview of range trees, see De Berg et al. [4].

The first level of the range tree consists of a binary tree B with node set $V(B)$ which is constructed bottom-up. Every node $v \in V(B)$ stores a certain interval $[\underline{x}_v, \overline{x}_v] \subset \mathbb{R}$. The leaf nodes of B, from left to right, correspond to all points $(x_q, y_q) \in \mathcal{P}$ sorted by the x_q coordinate in non-decreasing order. For the leaf nodes v of B we initialize $[\underline{x}_v, \overline{x}_v] = [x_q, x_q]$. Every inner node $v \in V(B)$ stores the interval $[\underline{x}_{v.left}, \overline{x}_{v.right}] \subset \mathbb{R}$ where $v.left \in V(B)$ and $v.right \in V(B)$ are the left- and the right child node of node v respectively.

In the second level, every node $v \in V(B)$ additionally stores a binary tree B_v with node set $V(B_v)$ which is constructed bottom-up. The leaves of B_v correspond to the dual line set $\mathcal{P}_v = \{q \in \mathcal{P} \mid x_q \in [\underline{x}_v, \overline{x}_v]\}$ first sorted by their y_q coordinates in non-decreasing order and second by their x_q coordinates in non-decreasing order. Every inner node $w \in V(B_v)$ stores the interval $[\underline{y}_{w.left}, \overline{y}_{w.right}] \subset \mathbb{R}$ and, additionally, a *dominating dual line* $\pi_w \in \mathcal{P}_v$, a

resolve value $\alpha_w \in \mathbb{R}$ and a *minimum resolve value* $\beta_w \in \mathbb{R}$. For every leaf node $w \in V(B_v)$ that corresponds to a dual line $(x_q, y_q) \in \mathcal{P}_v$ we initialize $\pi_w = (x_q, y_q)$ and $\alpha_w = \beta_w = \infty$. For every inner node $w \in V(B_v)$ the values are defined recursively as follows:

$$\pi_w = \begin{cases} \pi_{w.left} & a \cdot x_q + y_q \geq a \cdot x_{q'} + y_{q'} \\ \pi_{w.right} & \text{else} \end{cases} \tag{36}$$

$$\alpha_w = \frac{y_{q'} - y_q}{x_q - x_{q'}} \tag{37}$$

$$\beta_w = \min\{\alpha_{w.left}, \alpha_{w.right}, \beta_{w.left}, \beta_{w.right}\} \tag{38}$$

where $q = \pi_{w.left}$ and $q' = \pi_{w.right}$. The dual line $\pi_w \in \mathcal{P}_v$ represents the dual line $(x_q, y_q) \in \mathcal{P}_v$ with maximum value $a \cdot x_q + y_q$ according to the current sweep value $a \in \mathbb{R}$. The resolve value $\alpha_w \in \mathbb{R}$ corresponds to the a-value where the two dominating dual lines $\pi_{w.left}$ and $\pi_{w.right}$ intersect. That means, if the sweep line reaches the value α_w the domination order of the two dual lines changes, so the tree B_v must be *resolved*, see Section 4.2. In this case, we eventually add a *resolve event* to the *event heap*, see the next paragraph. But resolve events are added to the event heap, only if the values of the dual lines $\pi_{w.left}$ and $\pi_{w.right}$ are not overwritten due to any resolve event earlier than α_w. Otherwise, the resolve event becomes obsolete and resolving is not necessary anymore. For this reason, we additionally store the minimum resolve value $\beta_w \in \mathbb{R}$ which marks the earliest sweep value that may affect the recursive definition of $\pi_{w.left}$ or $\pi_{w.right}$. Hence, we add a resolve event at any node $w \in V(B_v)$ to the event heap only if it holds $\alpha_w < \beta_w$.

The node components π_w, α_w, β_w are equivalent to those used in [13]. Here they apply for every node in every second-level tree B_v with $v \in V(B)$.

Event Heap. The *event heap* stores two types of events: *evaluation events* and *resolve events* which are sorted by a-value, respectively α_w-value, in non-decreasing order. At evaluation events, the sweep line is *evaluated* and at resolve events a second-level tree node $w \in V(B_v)$ is *resolved*, see Section 4.2. All evaluation events are added to the heap before the sweeping, while resolve events can be added dynamically during the sweeping.

4.2 Algorithm

In the following we describe the main phases of our sweep line algorithm.

Sweeping. The sweeping is performed by successively extracting the minimal element from the event heap. If the extracted event is an evaluation event we call the subroutine *evaluate* and otherwise, if it is a resolve event, we call the subroutine *resolve*. In the following the single subroutines are explained more explicitly.

Evaluate. At an evaluation event, we compute for a line segment pair $(k, l) \in \mathcal{L}$ the point $q^* \in \mathcal{P}$ for which

$$\max_{q \in \mathcal{P}: x_q \in [\underline{x}_k, \overline{x}_k] \wedge a_k \cdot x_q + y_q + b_k > 0} \quad a_l \cdot x_q + y_q$$

attains the maximum. For this, we iterate the first-level range tree B recursively. If we traverse a tree node $v \in V(B)$ with $[\underline{x}_v, \overline{x}_v] \subseteq [\underline{x}_k, \overline{x}_k]$ we call a evaluation subroutine on the second-level tree B_v that is either: easy-, standard- or dual evaluate, see below. If we reach a tree node $v \in V(B)$ with $[\underline{x}_v, \overline{x}_v] \cap [\underline{x}_k, \overline{x}_k] \neq \emptyset$ we perform recursion on the child nodes *v.left* and *v.right*, otherwise no recursion is performed. In total, $\mathcal{O}(\log |\mathcal{P}|)$ first-level tree nodes are traversed.

(i) Easy Evaluate. This case is called for $a_l = a_k$ to compute problem (33):

$$\max_{q \in \mathcal{P}_v} \quad a_l \cdot x_q + y_q$$

which can be done in $\mathcal{O}(1)$ since $\pi_w \in \mathcal{P}_v$, where w is the root node of B_v, yields the desired point. We only have to ensure that $a_l \cdot x_q + y_q > -b_k$ holds.

(ii) Standard Evaluate. This case is called for $a_l \neq 0$ to compute problem (34):

$$\max_{q \in \mathcal{P}_v: y_q > -b_k} \quad a_l \cdot x_q + y_q$$

which, again, can be performed by descending the second-level tree B_v along tree nodes $w \in V(B_v)$ with $[\underline{y}_w, \overline{y}_w] \cap (-b_k, \infty) \neq \emptyset$. If it holds $[\underline{y}_w, \overline{y}_w] \subset (-b_k, \infty)$, by construction the dual line $\pi_w \in \mathcal{P}_v$ already represents the dual line for which the objective function attains the maximum value. Hence, the globally optimal dual line can be computed in $\mathcal{O}(\log |\mathcal{P}_v|) = \mathcal{O}(\log |\mathcal{P}|)$.

(iii) Dual Evaluate. In contrast to (i) and (ii), this case is called for the sweep value $a_k \neq 0$ instead of a_l to compute problem (35):

$$\max_{q \in \mathcal{P}_v: a_k \cdot x_q + y_q > -b_k} \quad y_q$$

which is done by traversing B_v from the root to the rightmost leaf node $w \in V(B_v)$ with $\tilde{q} = \pi_w$ that satisfies $a_k \cdot x_{\tilde{q}} + y_{\tilde{q}} > -b_k$. For any node $w \in V(B_v)$ on this path, the latter condition is maintained by testing whether it holds $a_k \cdot x_q + y_q > -b_k$ or $a_k \cdot x_{q'} + y_{q'} > -b_k$ for $q = \pi_{w.left}$ and $q' = \pi_{w.right}$. If the latter condition is true we enter the right subtree, since its leaves represent dual lines with higher y_q value. Otherwise, we enter the left subtree. Hence, one dual evaluation takes $\mathcal{O}(\log |\mathcal{P}_v|) = \mathcal{O}(|\mathcal{P}|)$.

Resolve. This subroutine is called at a sweep value $\alpha_w \in \mathbb{R}$ with $w \in V(B_v)$ and $v \in V(B)$ to update the dual line ordering on the current sweep line.

If it holds $\pi_w = \pi_{w.left}$ we update $\pi_w = \pi_{w.right}$ or vice versa. Furthermore, we set $\alpha_w = \infty$ and $\beta_w = \min\{\alpha_{w.right}, \beta_{w.right}\}$ or vice versa. Changing the

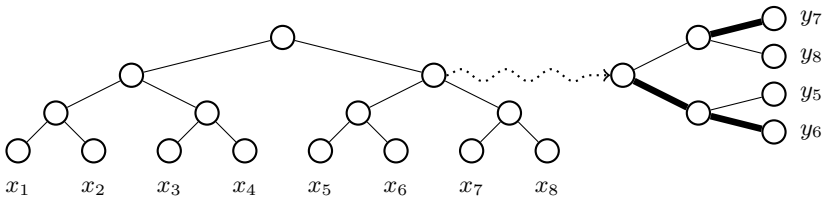

Fig. 1. Two-dimensional range tree: The first-level tree B (left) sorts all points $q \in \mathcal{P}$ (leaves) according to their x_q-coordinate. Every inner node $v \in V(B)$ stores a second-level tree B_v (right) that sorts all nodes $q \in \mathcal{P}_v$ according to their y_q-coordinate. Every second-level tree B_v additionally stores the dual line ordering of P_v on the current sweep line. This is indicated by the thick lines in B_v which represent the paths to the currently dominating dual line $\pi_w \in \mathcal{P}_v$ in the interval $[\underline{y}_w, \overline{y}_w]$.

values of π_w, α_w, β_w may change the values $\pi_{w'}, \alpha_{w'}, \beta_{w'}$ of each parent node w' of w due to the recursive definition in (36)-(38). Hence, we propagate the changes from w to the root node of B_v. Therefore, one resolve has complexity $\mathcal{O}(\log|\mathcal{P}_v|) = \mathcal{O}(\log|\mathcal{P}|)$.

Lemma 5. *For every first-level tree node $v \in V(B)$ every second-level tree node $w \in V(B_v)$ is resolved at most once.*

Proof. Assume there exists a first-level tree node $v \in V(B)$ such that a second-level tree node $w \in V(B_v)$ is resolved twice.

Then there are three dual lines $q_1, q_2, q_3 \in \mathcal{P}_v$ that successively, in the order of non-decreasing sweep value, represent the dominating dual line $\pi_w \in \mathcal{P}_v$. That is, it holds $\pi_w = q_1$, after the first resolve it holds $\pi_w = q_2$ and after the second resolve it holds $\pi_w = q_3$. It follows that $x_1 < x_2 < x_3$ since dual lines with higher slope will dominate with non-decreasing sweep value. Moreover, the dual line q_2 belongs to a different subtree of w than the dual lines q_1 and q_3, because resolve events are added only for intersecting dual lines of distinct subtrees. Without loss of generality, let the dual lines q_1 and q_3 correspond to leaves of the right subtree of w and the dual line q_2 to a leaf of the left subtree of w. By the ordering of the subtree, this implies $y_2 \leq y_1$ and $y_2 \leq y_3$. Let $\alpha_w \in \mathbb{R}$ be the sweep value of the first resolve event between the dual lines q_1 and q_2 and let $\alpha'_w \in \mathbb{R}$ be the sweep value of the second resolve event between the dual lines q_2 and q_3. It holds $\alpha_w < \alpha'_w$ because $\alpha_w = \alpha'_w$ violates the adding condition $\alpha_w < \beta_w$. It follows $\alpha_w \cdot x_1 + y_1 = \alpha_w \cdot x_2 + y_2$ and $\alpha'_w \cdot x_2 + y_2 = \alpha'_w \cdot x_3 + y_3$ which is equivalent to

$$0 \leq \frac{y_1 - y_2}{x_2 - x_1} = \alpha < \alpha' = \frac{y_2 - y_3}{x_3 - x_2} \leq 0$$

that forms a contradiction. The case of switching the subtrees of the dual lines q_1, q_3 and q_2 is analogous. □

An explicit description of all algorithms can be found in the appendix.

4.3 Complexity

In this section, we investigate the total complexity of the sweep line algorithm, which consists of four parts: *tree construction*, *adding/deleting elements to/from the event heap*, *evaluating all evaluation events* and *resolving all resolve events*.

Lemma 6. *The construction of the range tree takes* $\mathcal{O}(|\mathcal{P}| \cdot \log |\mathcal{P}|)$.

Proof. For every inner tree node $v \in V(B)$ of height h a second-level tree B_v with $\mathcal{O}\left(\frac{|\mathcal{P}|}{2^h}\right)$ nodes is constructed. The data members of every node $w \in V(B_v)$ are generated in $\mathcal{O}(1)$. Hence, the total construction takes at most

$$\sum_{h=0}^{\lceil \log |\mathcal{P}| \rceil} 2^h \cdot \mathcal{O}\left(\frac{|\mathcal{P}|}{2^h}\right) = \mathcal{O}(|\mathcal{P}| \cdot \log |\mathcal{P}|)$$

which shows that lemma. □

Lemma 7. *Let* $m = |\mathcal{L}| + |\mathcal{P}| \cdot \log |\mathcal{P}|$. *The complexity of adding and deleting elements from the event heap is* $\mathcal{O}(m \cdot \log m)$.

Proof. By Lemma 5, every second-level tree node is resolved at most once. Hence, there are $\mathcal{O}(|\mathcal{P}| \cdot \log |\mathcal{P}|)$ resolve events. Additionally, there are $|\mathcal{L}|$ evaluation events which gives a total complexity of $\mathcal{O}(m \cdot \log m)$ for adding and deleting elements from the event heap. □

Lemma 8. *The complexity of processing all evaluation events is* $O(|\mathcal{L}| \cdot \log^2 |\mathcal{P}|)$.

Proof. Every line segment pair $(k, l) \in \mathcal{L}$ is evaluated exactly once. In one evaluation call $\mathcal{O}(\log |\mathcal{P}|)$ first-level tree nodes $v \in V(B)$ are traversed, while for each such tree node $\mathcal{O}(\log |V(B_v)|)$ second-level tree nodes are traversed. This gives

$$\sum_{h=0}^{\lceil \log |\mathcal{P}| \rceil} \mathcal{O}\left(\log\left(\frac{|\mathcal{P}|}{2^h}\right)\right) = \sum_{h=0}^{\lceil \log |\mathcal{P}| \rceil} \mathcal{O}(\log |\mathcal{P}|) - \mathcal{O}(h) = \mathcal{O}(\log^2 |\mathcal{P}|)$$

which yields a total complexity of $\mathcal{O}(|\mathcal{L}| \cdot \log^2 |\mathcal{P}|)$. □

Lemma 9. *The complexity of processing all resolve events is* $O(|\mathcal{P}| \cdot \log^2 |\mathcal{P}|)$.

Proof. By Lemma 5, every inner tree node $w \in V(B_v)$ of height h of some tree node $v \in V(B)$ is resolved at most once and each resolve takes h steps. Hence, resolving one second-level tree B_v with m leaf nodes takes

$$\sum_{h=0}^{\lceil \log m \rceil} h \cdot 2^h$$

which we substitute by $r = 2$ and $M = \lceil \log m \rceil$. It follows

$$\begin{aligned}\sum_{h=0}^{M} h \cdot r^h = r \cdot \frac{\Delta}{\Delta r}(\sum_{h=0}^{M} r^h) &= r \cdot \frac{\Delta}{\Delta r}\left(\frac{1 - r^{M+1}}{1 - r}\right) \\ &= \frac{(M+1) \cdot r^{M+1} \cdot (r-1) + r^{M+2} - r}{(1-r)^2} \\ &= \mathcal{O}(M \cdot r^M) = \mathcal{O}(m \cdot \log m)\end{aligned}$$

that means resolving one subtree of size m takes $\mathcal{O}(m \cdot \log m)$. Hence, the total complexity of resolving every second-level tree node is

$$\begin{aligned}\sum_{h=0}^{\lceil \log |\mathcal{P}| \rceil} 2^h \cdot \mathcal{O}\left(\frac{|\mathcal{P}|}{2^h} \cdot \log\left(\frac{|\mathcal{P}|}{2^h}\right)\right) &= |\mathcal{P}| \cdot \sum_{h=0}^{\lceil \log |\mathcal{P}| \rceil} \mathcal{O}\left(\log\left(\frac{|\mathcal{P}|}{2^h}\right)\right) \\ &= \mathcal{O}(|\mathcal{P}| \cdot \log^2 |\mathcal{P}|)\end{aligned}$$

which completes the proof. □

Theorem 2. *Let $m = |\mathcal{L}| + |\mathcal{P}| \cdot \log |\mathcal{P}|$. The sweep line algorithm has a total complexity $\mathcal{O}(m \cdot \log m)$.*

Proof. The computation time is dominated by adding and deleting elements from the event heap. By Lemma 7, this takes $\mathcal{O}(m \cdot \log m)$. □

Theorem 3. *The maximum energetic reasoning propagations for all jobs can be computed in $\mathcal{O}(n^2 \cdot \log^2 n)$.*

Proof. In our case, we have $|\mathcal{L}| = \mathcal{O}(n)$ and $|\mathcal{P}| = \mathcal{O}(n)$. By Theorem 2, every sweep line iteration takes

$$\mathcal{O}(n \cdot \log n \cdot \log(n \cdot \log n)) = \mathcal{O}(n \cdot \log^2 n).$$

The sweep line algorithm is called for $\mathcal{O}(n)$ values of $\bar{t}_1$, so the total complexity is $\mathcal{O}(n^2 \cdot \log^2 n)$. □

Note that, compared to [13], the sweep line algorithm does not need to be called twice, since all exact propagations are detected in one call of the algorithm.

5 Conclusion

In this paper, we introduced a new exact energetic reasoning propagation algorithm with complexity $\mathcal{O}(n^2 \cdot \log^2 n)$ which improves the currently best known complexity of $\mathcal{O}(n^3)$ for exact energetic reasoning. Our algorithm is based on a geometric interpretation of the underlying slack polyhedra. A special property of the problem allows us to solve the geometric problem efficiently by using a specific two-dimensional range tree. Besides the theoretical improvement it remains to study the practical performance of the algorithm.

References

1. 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)
2. 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)
3. Bonifas, N. (2016): A $O(n^2log(n))$ propagation for the Energy Reasoning, Conference Paper, Roadef 2016 (2016)
4. De Berg, M., Van Kreveld, M., Overmars, M., & Schwarzkopf, O. C. (2000). Computational geometry. In Computational geometry (pp. 1-17). Springer Berlin Heidelberg.
5. 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)
6. Erschler, J., & Lopez, P. (1990, June). Energy-based approach for task scheduling under time and resources constraints. In 2nd international workshop on project management and scheduling (pp. 115-121).
7. 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)
8. Mercier, L., & Van Hentenryck, P. (2008). Edge finding for cumulative scheduling. INFORMS Journal on Computing, 20(1), 143-153.
9. 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)
10. Schutt, A., Feydy, T., Stuckey, P. J., & Wallace, M. G.: Explaining the cumulative propagator. Constraints, 16(3), 250-282 (2011)
11. 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)
12. 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)
13. Tesch, A. (2016, September). A Nearly Exact Propagation Algorithm for Energetic Reasoning in $\mathcal{O}(n^2 \log n)$. In International Conference on Principles and Practice of Constraint Programming (pp. 493-519). Springer International Publishing.
14. 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)
15. 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)

A Algorithms

Notes on the algorithms:

- define $O_3(t_1) = \{(j, t_2 - t_1) \mid (j, t_2) \in O_3\}$
- the computation of the overload values $\omega(t_1, t_2)$ is equal to Baptiste et al. and involves dynamic slope updates.
- to keep the stated algorithms simple no dynamic line updates, as introduced in [13], are added to the algorithms

Algorithm 1: $\mathcal{O}(n^2 \log^2 n)$ Sweep Line Propagator

Input: CuSP instance
Output: false, if there exists a time interval with positive overload, otherwise propagate exact energetic reasoning

$O_1 \leftarrow \{(j, e_j) \mid j \in J\} \cup \{(j, l_j - p_j) \mid j \in J\}$
$O_2 \leftarrow \{(j, l_j) \mid j \in J\} \cup \{(j, e_j + p_j) \mid j \in J\}$
$O_3 \leftarrow \{(j, e_j + l_j) \mid j \in J\}$
$t_1^{old} \leftarrow \infty$
$e'_j \leftarrow e_j \quad \forall j \in J$
$l'_j \leftarrow l_j \quad \forall j \in J$
for $(i, t_1) \in O_1$ *non-decreasing* t_1 **do**
    if $t_1 = t_1^{old}$ **then** *continue*
    $t_1^{old} = t_1$
    $t_2^{old} \leftarrow \infty$
    for $curr = (j, t_2) \in O_2 \cup O_3(t_1)$ *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`
        $t_2^{old} \leftarrow t_2$
    $\mathcal{L} \leftarrow initializeLineSegmentPairs(t_1)$ `// 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 2: Initialize Line Segment Pairs

Input: jobs J, fixed interval value $\bar{t}_1$
Output: set of line segment pairs $\mathcal{L}$

$\mathcal{L} \leftarrow \emptyset$
for $j \in J$ **do**
    if $\bar{t}_1 \leq \min\{e_j + p_j, l_j - p_j\}$ **then**
        $\mathcal{L} \leftarrow \mathcal{L}_{j,1}^{left} \cup \mathcal{L}_{j,2}^{left} \cup \mathcal{L}_{j,3}^{left}$ `// see Section 3.2`
    if $\bar{t}_1 \in [e_j, l_j]$ **then**
        $\mathcal{L} \leftarrow \mathcal{L}_{j,1}^{right} \cup \mathcal{L}_{j,2}^{right}$ `// see Section 3.2`
return $\mathcal{L}$

Algorithm 3: Sweep Line Propagation

Input: points $\mathcal{P}$, line segment pairs $\mathcal{L}$, earliest start times e'_j, latest completion times l'_j, fixed interval value $\bar{t}_1$

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

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

for $(k,l) \in \mathcal{L}$ **do** // fill heap with evaluation events

  if $a_l = a_k$ **then**

    $H.insert(k, l, easy)$

  else if $a_k = 0$ **then**

    $H.insert(k, l, standard)$

  else if $a_l = 0$ **then**

    $H.insert(k, l, dual)$

$a_0 \leftarrow H.getMin.Value$ // start slope

$B \leftarrow initializeRangeTree(\mathcal{P}, a_0, H)$ // build range tree

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

  $event \leftarrow H.extractMin()$

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

    $(k,l) \leftarrow event.lines$

    $j \leftarrow l.job$

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

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

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

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

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

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

    if $l.shift = left \wedge \omega_j^{left}(t_1,t_2) > 0$ **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 $l.shift = right \wedge \omega_j^{right}(t_1,t_2) > 0$ **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**

    $v \leftarrow event.firstLevelTreeNode$

    $w \leftarrow event.secondLevelTreeNode$

    $B.resolve(B_v, w, H)$ // resolve

return e', l'

Algorithm 4: Initialize Range Tree

Input: set of points $\mathcal{P}$, initial slope a_0, event heap H
Output: range 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 in non-decreasing order
$q \leftarrow \mathcal{P}.first$

`// leaf nodes`
for $v \in V^{leaf}(B)$ *from left to right* $\wedge\ q \neq \mathcal{P}.end$ **do**
    $[\underline{x}_v, \overline{x}_v] \leftarrow [x_q, x_q]$
    $B_v \leftarrow \{v\}$
    $q \leftarrow \mathcal{P}.next$

`// inner nodes`
for $v \in V^{noleaf}(B)$ *sorted bottom-up and left-right* **do**
    if $v.left = null$ **then** *continue*
    if $v.right \neq null$ **then**
        $[\underline{x}_v, \overline{x}_v] \leftarrow [\underline{x}_{v.left}, \overline{x}_{v.right}]$
        $\mathcal{P}_{v.left} \leftarrow B_{v.left}.leaves$
        $\mathcal{P}_{v.right} \leftarrow B_{v.right}.leaves$
        `// merge by` y_q`-value first, then by` x_q`-value`
        $\mathcal{P}_v \leftarrow merge(\mathcal{P}_{v.left}, \mathcal{P}_{v.right})$
        $B_v \leftarrow initializeSecondLevelTree(a_0, \mathcal{P}_v, H)$
    else
        $[\underline{x}_v, \overline{x}_v] \leftarrow [\underline{x}_{v.left}, \overline{x}_{v.left}]$
        $\mathcal{P}_v \leftarrow B_{v.left}.leaves$
        $B_v \leftarrow initializeSecondLevelTree(a_0, \mathcal{P}_v, H)$

return B

Algorithm 5: Initialize Second-Level Range Tree

Input: initial slope a_0, set of points $\mathcal{P}_v$ sorted by y-coordinate, event heap H
Output: second-level range tree B_v

$B_v \leftarrow$ binary interval tree with $2^{\lceil \log_2 \mathcal{P}_v \rceil + 1}$ empty nodes
$q \leftarrow \mathcal{P}_v.first$

`// leaf nodes`
for $w \in V^{leaf}(B_v)$ *from left to right* $\wedge\ q \neq \mathcal{P}_v.end$ **do**
    $[\underline{x}_w, \overline{x}_w] \leftarrow [x_q, x_q]$
    $\pi_w \leftarrow q$
    $\alpha_w \leftarrow \infty$
    $\beta_w \leftarrow \infty$
    $q \leftarrow \mathcal{P}_v.next$

`// inner nodes`
for $w \in V^{noleaf}(B_v)$ *sorted bottom-up and left-right* **do**
    if $w.left = null$ **then** *continue*
    if $w.right \neq null$ **then**
        $[\underline{x}_w, \overline{x}_w] \leftarrow [\underline{x}_{w.left}, \overline{x}_{w.right}]$
        $\beta_w \leftarrow \min\{\alpha_{w.left}, \alpha_{w.right}, \beta_{w.left}, \beta_{w.right}\}$
        $q \leftarrow \pi_{w.left}$
        $q' \leftarrow \pi_{w.right}$
        if $a_0 \cdot x_q + y_q > a_0 \cdot x_{q'} + y_{q'}$ **then**
            $\pi_w \leftarrow q$
        else
            $\pi_w \leftarrow q'$
        $val \leftarrow (y_{q'} - y_q)/(x_q - x_{q'})$
        if $val > a_0$ **then**
            $\alpha_w \leftarrow val$
            if $\alpha_w < \beta_w$ **then**
                $H.insert(\alpha_w, v, w, resolve)$ `// add resolve event`
        else $\alpha_w \leftarrow \infty$
    else
        $[\underline{x}_w, \overline{x}_w] \leftarrow [\underline{x}_{w.left}, \overline{x}_{w.left}]$
        $\pi_w \leftarrow \pi_{w.left}$
        $\alpha_w \leftarrow \infty$
        $\beta_w \leftarrow \min\{\alpha_{w.left}, \beta_{w.left}\}$
return B_v

Algorithm 6: Evaluate

Input: range tree B, line segment pair (k, l), evaluation type $eval$
Output: optimal point $q^* \in \mathcal{P}$ with respect to $eval$

```
1  max ← −∞
2  q* ← null
3  S ← ∅                                                    // empty stack
4  S.push(B.root)
5  while S ≠ ∅ do
6      v ← S.pop
7      if v = null then continue
8      if [x_v, x̄_v] ⊆ [x_l, x̄_l] then
9          if eval = easy then                              // easy evaluate
10             w ← B_v.root
11             q ← π_w
12         else if eval = standard then                     // standard evaluate
13             q ← standardEvaluate(B_v, a_l, b_k)
14         else if eval = dual then                         // dual evaluate
15             q ← dualEvaluate(B_v, a_l, b_k)
16         if q ≠ null then
17             if eval = easy ∨ eval = standard then
18                 val ← a_l · x_q + y_q
19             else if eval = dual then
20                 val ← y_q
21             if val > max then
22                 max ← val
23                 q* ← q
24     else if [x_v, x̄_v] ∩ [x_l, x̄_l] ≠ ∅ then
25         S.push(v.left)
26         S.push(v.right)
27 return q*
```

Algorithm 7: Standard Evaluate

Input: second-level range tree B_v, slope value a_l, intercept b_k
Output: point $q \in \mathcal{P}$ with $y_q > -b_k$ such that $a_l \cdot x_q + y_q$ is maximal

```
1  max ← −∞
2  q ← null
3  S ← ∅                                                    // empty stack
4  S.push(B_v.root)
5  while S ≠ ∅ do
6      w ← S.pop
7      if w = null then continue
8      if y_w > −b_k then
9          q' ← π_w
10         val ← a_l · x_q' + y_q'
11         if val > max then
12             max ← val
13             q ← q'
14     else if ȳ_w ≥ −b_k then
15         S.push(w.left)
16         S.push(w.right)
17 return q
```

Algorithm 8: Dual Evaluate

Input: second-level range tree B_v, slope value a_k and intercept b_k
Output: point $q \in \mathcal{P}$ with $a_k \cdot x_q + y_q > -b_k$ such that y_q is maximal

```
1  w ← B_v.root
2  if a_k · x_{π_w} + y_{π_w} ≤ −b_k then
3      return null
4  while w ∉ V^leaf(B_v) do
5      if a_k · x_{π_{w.right}} + y_{π_{w.right}} > −b_k then
6          w ← w.right
7      else if a_k · x_{π_{w.left}} + y_{π_{w.left}} > −b_k then
8          w ← w.left
9  q ← π_w
10 return q
```

Algorithm 9: Resolve

Input: second-level range tree B_v, range tree node $w \in V(B_v)$, event heap H
Output: updates π_w and all values $\pi_{w'}, \alpha_{w'}, \beta_{w'}$ from w to the root of B_v, eventually adds new resolve events to the event heap H

```
// resolve current node
```
$\alpha_w \leftarrow \infty$
if $\pi_w = \pi_{w.right}$ **then**
  $\pi_w = \pi_{w.left}$
  $\beta_w \leftarrow \min\{\alpha_{w.left}, \beta_{w.left}\}$
else
  $\pi_w \leftarrow \pi_{w.right}$
  $\beta_w \leftarrow \min\{\alpha_{w.right}, \beta_{w.right}\}$

```
// propagate changes to root node
```
$prev \leftarrow w$
$w \leftarrow w.parent$
while $prev \neq B_v.root$ **do**
  if $w.right \neq null$ **then**
    if $\alpha_w < \infty$ **then**
      $q \leftarrow \pi_{w.left}$
      $q' \leftarrow \pi_{w.right}$
      $\alpha_w \leftarrow (y_q - y_{q'})/(x_{q'} - x_q)$
      $\beta_w \leftarrow \min\{\alpha_{w.left}, \alpha_{w.right}, \beta_{w.left}, \beta_{w.right}\}$
    else if $\pi_w = \pi_{w.left}$ **then**
      $\beta_w \leftarrow \min\{\alpha_{w.left}, \beta_{w.left}\}$
    else if $\pi_w = \pi_{w.right}$ **then**
      $\beta_w \leftarrow \min\{\alpha_{w.right}, \beta_{w.right}\}$
    if $\alpha_w < \beta_w$ **then**
      $H.insert(\alpha_w, v, w, resolve)$ `// add resolve event`
  else $\beta_w \leftarrow \min\{\alpha_{w.left}, \beta_{w.left}\}$
  $prev \leftarrow w$
  $w \leftarrow w.parent$