# *Isospeed*: Improving (min,+) Convolution by Exploiting (min,+)/(max,+) Isomorphism

**Raffaele Zippo**
Dipartimento di Ingegneria dell'Informazione, University of Firenze, Italy
Dipartimento di Ingegneria dell'Informazione, University of Pisa, Italy
Distributed Computer Systems Lab (DISCO), TU Kaiserslautern, Germany

**Paul Nikolaus**
Distributed Computer Systems Lab (DISCO), TU Kaiserslautern, Germany

**Giovanni Stea**
Dipartimento di Ingegneria dell'Informazione, University of Pisa, Italy

## Abstract

(min,+) convolution is the key operation in (min,+) algebra, a theory often used to compute performance bounds in real-time systems. As already observed in many works, its algorithm can be computationally expensive, due to the fact that: i) its complexity is superquadratic with respect to the size of the operands; ii) operands must be extended *before* starting its computation, and iii) said extension is tied to the least common multiple of the operand periods.

In this paper, we leverage the isomorphism between (min,+) and (max,+) algebras to devise a new algorithm for (min,+) convolution, in which the need for operand extension is minimized. This algorithm is considerably faster than the ones known so far, and it allows us to reduce the computation times of (min,+) convolution by orders of magnitude.

**2012 ACM Subject Classification** Computer systems organization → Real-time systems; Networks → Network performance analysis; Mathematics of computing → Mathematical software performance

**Keywords and phrases** Deterministic Network Calculus, min-plus algebra, max-plus algebra, performance, algorithms

**Digital Object Identifier** 10.4230/LIPIcs.ECRTS.2023.12

**Supplementary Material** *Software (ECRTS 2023 Artifact Evaluation approved artifact)*: `https://doi.org/10.4230/DARTS.9.1.3`

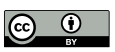
**Funding** This work was supported in part by the Italian Ministry of Education and Research (MIUR) in the framework of the FoReLab project (Departments of Excellence).

**Acknowledgements** This work is inspired by the results in [20] – we wish to thank Steffen Bondorf for pointing out this paper to us, as well as Raul-Paul Epure for suggestions with respect to some proofs.

## 1 Introduction

(min,+) and (max,+) algebras [2, 17] lie at the core of theories for the analysis of worst-case performance bounds.[1] More specifically, Deterministic Network Calculus (DNC) [10, 9, 8, 15, 4] – devised for network traffic – and Real-Time Calculus (RTC) [25] – devised for event-triggered systems – are both based on (min,+) and (max,+) algebra. Some recent papers analyzing real-time systems using DNC or RTC are [24, 19, 7, 3]. In both theories, flows of traffic (in DNC) or events (in RTC) are represented as cumulative functions of time,

[1] While commonly called *algebras* in DNC jargon, (min,+) and (max,+) are *semirings* [4, Ch. 2].









counting the traffic (or the events) arrived up to time $t$. The service guarantees offered by elements where resource contention may occur (e.g., packet schedulers at the output of a network) are also represented as functions of time, called *curves*.[2] I/O relationships at a node, represented as operations in (min,+) and (max,+) algebra, allow one to derive worst-case performance guarantees, e.g., bounds on the transit delay of a flow, or the backlog at a node. These theories are *compositional*, i.e., they allow one to model complex systems by first modeling their elements in isolation, and then composing their models, again using operations in (min,+) or (max,+) algebra. For instance, in DNC, the minimum service that a packet scheduler guarantees to a flow traversing it is represented by a *service curve* $\beta$. Accordingly, if a flow traverses two such schedulers, having service curves $\beta_1$ and $\beta_2$, the minimum service that the latter is guaranteed on an end-to-end basis can be obtained by computing the *(min,+) convolution* of $\beta_1$ and $\beta_2$. A similar property exists in RTC. In this paper, we concentrate on properties of (min,+) algebra. However, given the strong similarities between the two theories, we will often use $(\cdot,+)$ when the discussion applies to both (min,+) and (max,+) algebra.

The issue of *automated computation* of $(\cdot,+)$ algebra expressions is relevant: algebraic expressions which look simple on paper can in fact require lengthy computations. The research community has therefore developed several software packages to automate this task. In doing so, they have addressed the problem of finding efficient data structures to represent functions and curves, and efficient algorithms to implement basic $(\cdot,+)$ algebra operations. Works [6, 4] provided an "algorithmic toolbox" for DNC: they showed that piecewise-affine functions that are ultimately pseudo-periodic (UPP) represent suitable models for both traffic and service guarantees, and provided algorithms for most $(\cdot,+)$ algebra operations. The toolbox was first implemented in the COINC free library [5], which is not available anymore, and later by the commercial library RTaW-Pegase [21] and the open-source library Nancy [28]. A very similar model [4, p. 95], called *variability characterization curves* (VCCs), was implemented by the RTC toolbox [26]. Broadly speaking, in both UPP and VCC models functions and curves are represented as *sequences of segments*, and periodicity is leveraged to allow functions defined in $[0, +\infty)$ to be represented with a finite amount of information.

One of the most important operations in (min,+) algebra is (min,+) convolution. The latter has a complexity which is superquadratic with the size of its operands (i.e., the number of segments in their sequences), which makes it computationally expensive. What is worse, the algorithms that compute these operations require that the sequences of operands must be *extended* beforehand, which reflects on the overall complexity. Operand extension, in turn, depends heavily on numerical properties of the operands themselves, and is often related to the hyperperiod, i.e., the least common multiple (lcm) of the period lengths of the operands. The impact of the above issue grows exponentially when operations are chained together, which limits the scope of the studies that one can do in practice. Different techniques have been proposed in the literature to mitigate or avoid this issue, such as using containers and inclusion functions [16], avoiding the periodic parts altogether by bounding the study a priori [12, 13, 14], using a posteriori representation minimization to mitigate the impact on chained operations [29], devising more efficient algorithms for specific subclasses of operands and operations [29], namely subadditive functions and (min,+) convolution.

In this paper, we provide a novel technique to reduce the computation cost of (min,+) convolution, under general hypotheses on the operands, which we call *isospeed*. Our technique relies on exploiting the isomorphism between (min,+) and (max,+) algebra, thoroughly

[2] To be precise, in a curve an abscissa $\tau$ describes what may happen in any interval of length $\tau$.

described in [17]. It is shown therein that the result of a (min,+) convolution can be obtained by computing the *isomorphic* operation, i.e., the (max,+) convolution, if one applies a simple transformation to the operands beforehand, and to the result afterwards. This transformation is called *pseudoinversion*. This implies that there are always two ways to compute a (min,+) convolution: the *direct* one, e.g., using the algorithm described in [4], and the *inverse* one – i.e., the one based on pseudoinversion and isomorphism. This was first observed in [20]: the authors found that – in their specific use cases – the inverse algorithm for (min,+) convolution was considerably faster than the direct one. However, they did not provide an explanation as to why. Lacking the above, one cannot know whether this is a general result or just a stroke of luck. In this paper, we investigate the above in depth, providing novel results of both theoretical and practical significance. First, we offer a cogent explanation for the empirical observation in [20], which allows us to understand when and why one algorithm will be faster than the other *a priori*, and when instead no conclusion can be drawn. This is because the complexity of (min,+) convolution depends on numerical properties of the operands, and pseudoinversion modifies them, making the computational cost of the inverse algorithm different from the direct one's. Then, we build on the above observation to devise a new algorithm, that reduces the operand extension and the runtime of (min,+) convolution to a minimum. This makes it no worse (barring negligible overhead), and often much better, than the *fastest* between the direct algorithm and inverse one. Our algorithm improves the feasibility of performance studies that cannot benefit from the other methods mentioned above. For instance, it applies regardless of the shape of the operands, unlike the algorithms in [29], which requires operands to be subadditive, or the method described in [12, 13, 14], which requires service curves to be superadditive. Moreover, it optimizes single (min,+) convolutions, unlike the representation minimization in [29], which is only beneficial when *chaining* operations and does nothing to optimize individual ones. While the two methods can work in conjunction, individual (min,+) convolutions may take minutes or more using the standard algorithm. Due to space limitations, we can only focus on (min,+) convolution in this paper. However, our results can be generalized to (max,+) convolution as well, which can be improved in the same way, with the same performance benefits. This is because the isomorphism, as the name implies, works both ways.

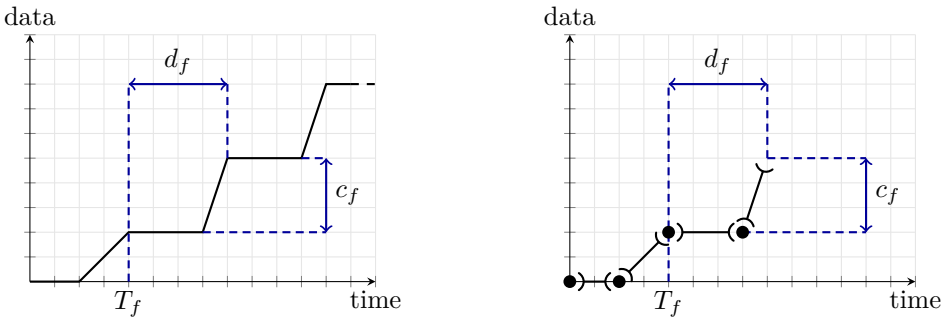
The rest of this paper is organized as follows: In Section 2, we introduce the mathematical background and state of the art. Then, in Section 3, we present our isospeed algorithm, which is evaluated in Section 4. Section 5 concludes the paper and highlights future work.

## 2 Background and Notation

In this section, we provide the mathematical and algorithmic background required for this paper. We define the types of functions that we use and the operations that we aim to improve. We use $a \wedge b = \min(a, b)$ and $a \vee b = \max(a, b)$. We also use $\mathbb{N}_0$ to denote the set of non-negative integers $\{0, 1, 2, 3, \dots\}$, $\mathbb{N}$ for the set of strictly positive integers $\{1, 2, 3, \dots\}$, and $\mathbb{Q}_+$ the set of non-negative rationals (including 0).

### 2.1 The UPP Model

To implement DNC and RTC computations in software, one needs to provide finite representations of functions and well-formed algorithms for $(\cdot,+)$ operations. In this paper, we use the model discussed in [29, 27] and implemented in Nancy [28], which we also use to implement our optimization. According to the widely accepted approach described in [6, 4],




**(a)** $f$. **(b)** $R_f$.

**Figure 1** An ultimately pseudo-periodic piecewise-affine function $f$ and its representation $R_f$.

a sufficiently generic class of functions useful for $(\cdot,+)$ computations is the set $\mathcal{U}$ of (i) ultimately pseudo-periodic $\mathbb{Q}_+ \to \mathbb{Q} \cup \{+\infty, -\infty\}$ (ii) piecewise affine functions. We define both properties (i) and (ii) separately:

▶ **Definition 1** (Ultimately Pseudo-Periodic Function [6, p. 8])**.** *Let $f$ be a function $\mathbb{Q}_+ \to \mathbb{Q} \cup \{+\infty, -\infty\}$. Then, $f$ is ultimately pseudo-periodic (UPP) if exists $T_f \in \mathbb{Q}_+, d_f \in \mathbb{Q}_+ \setminus \{0\}, c_f \in \mathbb{Q} \cup \{+\infty, -\infty\}$ such that*

$$f(t + k \cdot d_f) = f(t) + k \cdot c_f, \qquad \forall t \geq T_f, \forall k \in \mathbb{N}. \tag{1}$$

*We call $T_f$ the* pseudo-periodic start*, or length of the initial transient, $d_f$ the* period length*, and $c_f$ the* period height*. We also say that $f$ is UPP* from $T_f$*.*

▶ **Definition 2** (Piecewise Affine Function [6, p. 9])**.** *We say that a function $f$ is* piecewise affine *(PA) if there exists an increasing sequence $(a_i), i \in \mathbb{N}_0$ which tends to $+\infty$, such that $a_0 = 0$ and $\forall i \in \mathbb{N}_0$, it either holds that $f(t) = b_i + \rho_i t$ for some $b_i, \rho_i \in \mathbb{Q}$, or $f(t) = +\infty$, or $f(t) = -\infty$ for all $t \in ]a_i, a_{i+1}[$.*

We remark that functions in $\mathcal{U}$ are not necessarily non-decreasing, and can assume infinite values. Both these properties are useful for algebraic manipulations. Among the functions in $\mathcal{U}$, we distinguish *Ultimately Constant* (UC) ones. A function is UC if there exists a $T \in \mathbb{Q}_+$ such that $f(t) = f(T)\, \forall t \geq T$. Similarly, an *Ultimately Infinite* (UI) function is one such that $f(t) = +\infty$, or $f(t) = -\infty$ for all $t \geq T$. Typical cases of UI curves in DNC are the service curves of *delay elements*. Throughout this paper, we will exclude UC and UI functions, because some properties do not hold otherwise. This limitation is of negligible impact, since (min,+) convolution is computationally trivial when an operand is UC/UI.

For functions in $\mathcal{U}$, it is enough to store a representation of the initial transient part and of one period, which is a finite amount of information. This is exemplified in Figure 1. Accordingly, we call a *representation* $R_f$ of a function $f$ the tuple $(S, T, d, c)$, where $T, d, c$ are the values described above, and $S$ is a sequence of points and open segments describing $f$ in $[0, T + d[$. We use both points and open segments in order to easily model discontinuities. We will use the umbrella term *elements* to encompass both when convenient. We denote with $n(S)$ the cardinality of a sequence $S$, i.e., the number of its elements. Note that, given $R_f$, one can compute $f(t)$ for all $t \geq 0$, and also $S_f^I$, i.e., a sequence describing $f$ in the finite interval $I$ for any $I \subset \mathbb{Q}_0^+$. Furthermore, being finite, $R_f$ can be used as a data structure to represent $f$ in code. As it will be useful in the following, we define *Cut* to be an (obvious) algorithm that, given $R_f$ and an interval $I$, computes $S_f^I$. With a little abuse of notation,

we will use $(\cdot,+)$ operators directly on finite sequences such as $S_f^I$. For instance, given the (min,+) convolution (formally defined later), we will write $S_f^{I_f} \otimes S_g^{I_g}$ to express that we are computing the (min,+) convolution $f \otimes g$, limited to the values of $f$ in interval $I_f$ and those of $g$ in interval $I_g$. It will be useful in the following to consider a function $f \in \mathcal{U}$ in a *restricted support* $D$.[3] This is done as follows:

▶ **Definition 3** (Min and Max Restrictions). *Let $f \in \mathcal{U}$ and $D \subseteq \mathbb{Q}_+$. Then, its* min restriction *over a support $D$ is defined as*

$$f|_D^{\wedge} \coloneqq \begin{cases} f(t), & \text{if } t \in D, \\ +\infty, & \text{otherwise.} \end{cases}$$

*Moreover, its* max restriction *over a support $D$ is defined as*

$$f|_D^{\vee} \coloneqq \begin{cases} f(t), & \text{if } t \in D, \\ -\infty, & \text{otherwise.} \end{cases}$$

In this work, we will often consider $D$ to be an interval $I$ of the form $[0, a[$ or $[a, +\infty[$. In many cases, it will be useful to restrict a function to its *transient part*, i.e, to interval $I = [0, T_f[$, or to its *periodic part*, i.e., $I = [T_f, +\infty[$, using shorthands $f_t^{\wedge}$ and $f_t^{\vee}$, as well as $f_p^{\wedge}$ and $f_p^{\vee}$, respectively. Accordingly, one can decompose $f$ as $f = f_t^{\wedge} \wedge f_p^{\wedge}$ or $f = f_t^{\vee} \vee f_p^{\vee}$.

A $(\cdot,+)$ operator can be defined computationally as an algorithm that takes UPP representations of its input functions and yields a UPP representation of the result. Considering a generic binary operator[4] $[\cdot] * [\cdot]$, in order to compute $f * g$ we need an algorithm that computes $R_{f*g}$ from $R_f$ and $R_g$, i.e., $R_f, R_g \to R_{f*g}$. We call this *by-curve* algorithm. Such an algorithm consists of the following steps:

1. compute valid parameters $T_{f*g}, d_{f*g}$ and $c_{f*g}$ for the result.
2. compute the intervals $I_f$ and $I_g$, for the sequences $S_f^{I_f} = \mathrm{Cut}(R_f, I_f)$ and, likewise, $S_g^{I_g}$;
3. compute $S_f^{I_f}, S_g^{I_g} \to S_{f*g}^{I_{f*g}}$ where $I_{f*g} = [0, T_{f*g} + d_{f*g}[$, i.e., use an algorithm that computes the resulting sequence from the sequences of the operands. We call this *by-sequence* algorithm for operator $[\cdot] * [\cdot]$;
4. return $R_{f*g} = (S_{f*g}, T_{f*g}, d_{f*g}, c_{f*g})$.

The *by-curve* algorithm for operator $[\cdot] * [\cdot]$ allows us to compute the result with any operands. Works [6, 4] provide such computational descriptions for most DNC operators, such as $(\cdot,+)$ convolution and deconvolution, while [27] provides the same for pseudoinverses (formally defined in the next section).

▶ Remark 4. Parameters $T_{f*g}, d_{f*g}$ and $c_{f*g}$, as well as intervals $I_f$ and $I_g$, are *sufficient* to compute a representation $R_{f*g}$. There may be, in general, more than one way to compute them for an algorithm, resulting in different performance.

Intuitively, dealing with shorter sequences leads to faster *by-sequence* algorithms. Optimized parameters and intervals can be found either by making restrictive assumptions on the shape of the operands, or – as we do in this paper – by exploiting algebraic properties.

[3] Inspired by [6, p. 7], we use *support* $D$ to assign a subset outside of which the function is constantly $-\infty$ or $+\infty$. Note that this does not necessarily mean, in general, that $f$ is finite on $D$. We mostly use it to define a set within which the properties of $f$ are observed.

[4] The same process applies also, with minor adjustments, to unitary operators.

## 2.2 Upper and Lower Pseudoinverses

It was first shown in [17] that (min,+) and (max,+) algebra can be regarded as specular images of each other. In fact, results in one algebra can be mapped to the other via *pseudoinversion* of operands and results. This *isomorphism* will be exploited throughout this paper. Hereafter, we discuss the essential definitions and properties of pseudoinverses, taken from [27].[5]

▶ **Definition 5** (Lower and Upper Pseudoinverse). *Let $f \in \mathcal{U}$ be non-decreasing. Then its* lower pseudoinverse $f_{\downarrow}^{-1}$ *and its* upper pseudoinverse $f_{\uparrow}^{-1}$ *are defined as*

$$f_{\downarrow}^{-1}(y) \coloneqq \inf\{t \geq 0 \mid f(t) \geq y\},$$
$$f_{\uparrow}^{-1}(y) \coloneqq \sup\{t \geq 0 \mid f(t) \leq y\}.$$

The lower pseudoinverse is always left-continuous and the upper pseudoinverse is right-continuous. Moreover, upper and lower pseudoinverses can be combined to yield something close to an involutive property. Consider a non-decreasing function $f$. Then, if $f$ is left-continuous, $f(t) = \left(f_{\uparrow}^{-1}\right)_{\downarrow}^{-1}(t)$. If $f$ is right-continuous, $f(t) = \left(f_{\downarrow}^{-1}\right)_{\uparrow}^{-1}(t)$. UPP properties and algorithms for the pseudoinverses are summarized here.

▶ **Theorem 6** ([27], Theorem 9, Theorem 10). *Let $f$ be a non-decreasing UPP function that is neither UC nor UI. Then, $f_{\downarrow}^{-1}$ and $f_{\uparrow}^{-1}$ are function of $\mathcal{U}$ with*

$$T_{f_{\downarrow}^{-1}} = f\left(T_f + d_f\right), \tag{2}$$
$$T_{f_{\uparrow}^{-1}} = f\left(T_f\right) \tag{3}$$
$$d_{f_{\downarrow}^{-1}} = d_{f_{\uparrow}^{-1}} = c_f, \tag{4}$$
$$c_{f_{\downarrow}^{-1}} = c_{f_{\uparrow}^{-1}} = d_f. \tag{5}$$

The exact algorithm for upper/lower pseudoinverses is reported in [27]. In both cases, it can be computed in linear time with the operand's sequence size, i.e., it is $\mathcal{O}\left(n(S)\right)$. This makes pseudoinversion considerably less complex than (min,+) convolution, which – as we will discuss below – is superquadratic. Both operations yield a result whose sequence has a cardinality *similar* to its operand's. The two cardinalities are not exactly *equal* because constant segments in the operand map to discontinuities in the pseudoinverse and vice versa. Segments count as elements in the cardinality, whereas discontinuities do not. This will be recalled later on, when we discuss performance.

## 2.3 (min,+) and (max,+) Convolution

Convolution is one of the most common operations in $(\cdot,+)$ algebra. We introduce here both (min,+) and (max,+) convolution.

▶ **Definition 7** (Convolution in (min,+) / (max,+) Algebra). *Let $f, g$ be non-decreasing. Their (min,+) convolution is defined for all $t \geq 0$ as*

$$f \otimes g(t) \coloneqq \inf_{0 \leq s \leq t} \{f(s) + g(t-s)\}.$$

[5] These definitions differ from the ones in [17, p. 60]. In fact, [17] considers functions defined in $\mathbb{R}$, hence having no boundaries, whereas functions in $\mathcal{U}$ are defined in $\mathbb{Q}_+$, hence 0 and $f(0)$ constitute a boundary.

*Their (max,+) convolution is defined for all $t \geq 0$ as*

$$f \overline{\otimes} g(t) \coloneqq \sup_{0 \leq s \leq t} \{f(s) + g(t-s)\}.$$

A fundamental result, which uses the above properties, is the isomorphism between (min,+) and (max,+) convolution, which enables us to replace one with the other, via pseudoinversion of operands and results.

▶ **Theorem 8** (Isomorphism of Convolution For Left-Continuous Functions $\in \mathcal{U}$). *Let $f, g \in \mathcal{U}$ be left-continuous and non-decreasing. Then,*

$$(f \otimes g)_{\uparrow}^{-1} = \left(f_{\uparrow}^{-1}\right) \overline{\otimes} \left(g_{\uparrow}^{-1}\right). \tag{6}$$

A proof can be derived by following along the lines of [20, Theorem 1], [17, Theorem 10.3b], but adapting to the fact that we consider functions $\in \mathcal{U}$. As a consequence, since the (min,+) convolution of left-continuous functions is itself left-continuous, we obtain:

$$f \otimes g = \left((f \otimes g)_{\uparrow}^{-1}\right)_{\downarrow}^{-1} = \left(f_{\uparrow}^{-1} \overline{\otimes} g_{\uparrow}^{-1}\right)_{\downarrow}^{-1}. \tag{7}$$

The algorithms for $(\cdot, +)$ convolution of UPP curves require one to specialize the generic steps described in Section 2.1. Due to space limitations, we discuss in depth (min,+) convolution only. (max,+) convolution can be presented along the same lines.

It was proved in [6] that (min,+) convolution $f \otimes g$ can be computed if one decomposes its operands into their transient and periodic parts, according to Definition 3. More specifically, the procedure is as follows:

1. Decompose the operands as $f = f_t^{\wedge} \wedge f_p^{\wedge}$ and $g = g_t^{\wedge} \wedge g_p^{\wedge}$.
2. Compute partial convolutions involving at least one transient part: $h_{tt} \coloneqq f_t^{\wedge} \otimes g_t^{\wedge}$, $h_{tp} \coloneqq f_t^{\wedge} \otimes g_p^{\wedge}$, $h_{pt} \coloneqq f_p^{\wedge} \otimes g_t^{\wedge}$. These can be computed using the algorithms described in [6]. These computations are not particularly complex, since at least one of the operands is defined in a finite interval.
3. Compute the partial convolution of the *periodic* parts, $h_{pp} \coloneqq f_p^{\wedge} \otimes g_p^{\wedge}$. This is the computationally complex part, as we detail below.
4. Compute $f \otimes g = h_{tt} \wedge h_{tp} \wedge h_{pt} \wedge h_{pp}$.

In [6, Proposition 4.5], UPP properties are derived for all these parts ($h_{tt}$, $h_{tp}$, $h_{pt}$, $h_{pp}$) and their minimum. We summarize this result here as Proposition 9.

▶ **Proposition 9** ([6], Proposition 4.5). *Let $f, g \in \mathcal{U}$. Then their (min,+) convolution $f \otimes g$ is again $\in \mathcal{U}$. Moreover,*

$$d_{f \otimes g} = \operatorname{lcm}(d_f, d_g), \tag{8}$$

$$c_{f \otimes g} = d_{f \otimes g} \cdot \min\left(\frac{c_f}{d_f}, \frac{c_g}{d_g}\right) = \operatorname{lcm}(d_f, d_g) \cdot \min\left(\frac{c_f}{d_f}, \frac{c_g}{d_g}\right) \tag{9}$$

*are a period length and height for $f \otimes g$.*

From the proof of [6, Proposition 4.5] we can extract the following result for, in particular, the convolution of periodic parts.

▶ **Corollary 10** ((min,+) convolution of periodic parts.). *Let $f$ and $g \in \mathcal{U}$. Then, $h_{pp} = f_p^\wedge \otimes g_p^\wedge$ is again a function of $\mathcal{U}$ with*

$$d_{h_{pp}} = d_{f\otimes g} = \operatorname{lcm}(d_f, d_g)\,, \tag{10}$$

$$c_{h_{pp}} = c_{f\otimes g} = d_{h_{pp}} \cdot \min\left(\frac{c_f}{d_f}, \frac{c_g}{d_g}\right), \tag{11}$$

$$T_{h_{pp}} = T_f + T_g + \operatorname{lcm}(d_f, d_g)\,. \tag{12}$$

And that, in order to compute $f_p^\wedge \otimes g_p^\wedge$, it is sufficient to use

$$\begin{aligned} I_{f_p^\wedge} &= \left[T_f, T_f + 2\cdot d_{h_{pp}}\right[, \\ I_{g_p^\wedge} &= \left[T_g, T_g + 2\cdot d_{h_{pp}}\right[, \\ I_{h_{pp}} &= \left[T_f + T_g, T_f + T_g + 2\cdot d_{h_{pp}}\right[. \end{aligned} \tag{13}$$

Corollary 10 shows that the period of the convolution of the periodic parts depends on the lcm of the periods of the operands. Algorithm 1 reports the pseudocode for the (min,+) convolution algorithm, based on the above decomposition [6]. When computing the convolution of the periodic parts, operands have to be *extended*, i.e., computed in intervals $I_{f_p^\wedge}$, $I_{g_p^\wedge}$. The complexity of the by-sequence algorithm for the convolution (line 6) is [6, p. 43]

$$\mathcal{O}\left(n\left(S_{f_p^\wedge}^{I_{f_p^\wedge}}\right)\cdot n\left(S_{g_p^\wedge}^{I_{g_p^\wedge}}\right)\cdot \log\left(n\left(S_{f_p^\wedge}^{I_{f_p^\wedge}}\right)\cdot n\left(S_{g_p^\wedge}^{I_{g_p^\wedge}}\right)\right)\right). \tag{14}$$

However, domains $I_{f_p^\wedge}$, $I_{g_p^\wedge}$ depend on $\operatorname{lcm}(d_f, d_g)$. On one hand, this corroborates the observation that computing $h_{pp}$ is the most complex task. On the other hand, the number of operations required may vary considerably depending on *numerical properties* of the operands. In fact, $\operatorname{lcm}(d_f, d_g)$ ranges from $\max(d_f, d_g)$ to the product of the numerators of $d_f$ and $d_g$[6]. Therefore, the runtime of the convolution of the periodic parts may vary a lot.

We briefly discuss (max,+) convolution, to highlight that it can be computed via the same decomposition, at similar big-O complexity, as the (min,+) convolution [6]. The main difference is that, since a *supremum* is used in place of an *infimum*, the decomposition is based on the maximum, rather than the minimum, of the parts. The procedure is as follows:

1. Decompose the operands as $f = f_t^\vee \vee f_p^\vee$ and $g = g_t^\vee \vee g_p^\vee$.
2. Compute partial convolutions involving at least one transient part: $\overline{h}_{tt} \coloneqq f_t^\vee \mathbin{\overline{\otimes}} g_t^\vee$, $\overline{h}_{tp} \coloneqq f_t^\vee \mathbin{\overline{\otimes}} g_p^\vee$, $\overline{h}_{pt} \coloneqq f_p^\vee \mathbin{\overline{\otimes}} g_t^\vee$. These can be computed by adapting the algorithms for the (min,+) convolution described in [6]. Again, these computations are not particularly complex, since at least one of the operands is defined in a finite interval.
3. Compute the partial convolution of the *periodic* parts, $\overline{h}_{pp} \coloneqq f_p^\vee \mathbin{\overline{\otimes}} g_p^\vee$.
4. Compute $f \mathbin{\overline{\otimes}} g = \overline{h}_{tt} \vee \overline{h}_{tp} \vee \overline{h}_{pt} \vee \overline{h}_{pp}$.

When computing the convolution of the periodic parts, operands have to be computed in intervals $I_{f_p^\vee}$, $I_{g_p^\vee}$, which do depend on $d_{\overline{h}_{pp}} = \operatorname{lcm}(d_f, d_g)$. The worst-case complexity of the by-sequence algorithm for the (max,+) convolution can be derived following the same steps as for the (min,+) convolution in [6, p. 43], and is the same as in (14), provided that one substitutes $S_{x_p^\vee}^{I_{x_p^\vee}}$ for $S_{x_p^\wedge}^{I_{x_p^\wedge}}$ for both operands $x$. Like with (min,+) convolution, domains $I_{f_p^\vee}$, $I_{g_p^\vee}$ depend on $\operatorname{lcm}(d_f, d_g)$. Therefore, the same observations already discussed apply here as well: computing $\overline{h}_{pp}$ is the most complex task, and the number of operations it requires may vary considerably depending on *numerical properties* of the operands.

[6] We recall that the lcm of two fractions is the lcm of their numerators divided by the greatest common divisor of their denominators.

**Algorithm 1** Pseudocode for (min,+) convolution.

**Input** Functions $f$ and $g$.
**Return** Their (min,+) convolution $f \otimes g$.

1: Decompose the operands as $f = \min\left(f_t^\wedge, f_p^\wedge\right)$ and $g = \min\left(g_t^\wedge, g_p^\wedge\right)$
2: Compute $h_{tt} \coloneqq f_t^\wedge \otimes g_t^\wedge$, $h_{tp} \coloneqq f_t^\wedge \otimes g_p^\wedge$, $h_{pt} \coloneqq f_p^\wedge \otimes g_t^\wedge$ as described in [6]
3: Compute $h_{pp} \coloneqq f_p^\wedge \otimes g_p^\wedge$ as follows:
4: Let $d_{h_{pp}} = \text{lcm}(d_f, d_g)$; $c_{h_{pp}} = d_{h_{pp}} \cdot \min\left(\frac{c_f}{d_f}, \frac{c_g}{d_g}\right)$; $T_{h_{pp}} = T_f + T_g + d_{h_{pp}}$.
5: Let
$$I_{f_p^\wedge} = \left[T_f, T_f + 2 \cdot d_{h_{pp}}\right[;$$
$$I_{g_p^\wedge} = \left[T_g, T_g + 2 \cdot d_{h_{pp}}\right[;$$
$$I_{h_{pp}} = \left[T_f + T_g, T_f + T_g + 2 \cdot d_{h_{pp}}\right[.$$
6: Compute $S_{h_{pp}}^{I_{h_{pp}}} = S_{f_p^\wedge}^{I_{f_p^\wedge}} \otimes S_{g_p^\wedge}^{I_{g_p^\wedge}}$
7: $R_{h_{pp}} = \left(S_{h_{pp}}^{I_{h_{pp}}}, T_{h_{pp}}, d_{h_{pp}}, c_{h_{pp}}\right)$
8: $f \otimes g = \min\left(h_{tt}, h_{tp}, h_{pt}, h_{pp}\right)$

# 3 Improving the Runtime of (min,+) Convolution

This section reports our contributions. First, we observe that there are always two algorithms to compute the (min,+) convolution, and discuss *why* one can be faster than the other. Our observations motivate our improved (min,+) convolution algorithm, which outperforms both the above.

## 3.1 Alternative Algorithms for (min,+) Convolution

The algorithms for (min,+) and (max,+) convolution are very similar, and they have the same complexity *on the same operands*. However, a (min,+) convolution $f \otimes g$ can be computed via a (max,+) convolution of *pseudoinverse operands* $f_\uparrow^{-1}$, $g_\uparrow^{-1}$, as per (7) [20].

This means that one can *always* compute a (min,+) convolution in two different ways:

1. the *direct* method, using Algorithm 1;
2. the *inverse* method, using pseudoinversion of the operands, (max,+) convolution, and pseudoinversion of the result, via (7).

Now, both methods have the same big-O complexity. In fact, pseudoinversion is linear, and – as explained in the previous section – both (min,+) and (max,+) convolutions are superquadratic. Despite this, it was observed in [20] that the inverse algorithm was significantly *faster* than the direct one, which is counterintuitive, since pseudoinversion adds overhead.[7]

We present here a sound explanation of the above phenomenon, which is missing in [20], and serves as a basis for our improved method. In both the (min,+) and (max,+) convolutions, the dominant factor for the complexity is the length of the *extended* sequences (as per (14)), which depends on the *number of periods that each operand must be extended*. For the

[7] In [20], this result was presented within the context of VCC curves using RTC Toolbox [26] and event-based service curves, which differs from the one referenced here, which is instead based on the UPP model implemented in Nancy [28]. However, this distinction does not affect the following discussion.

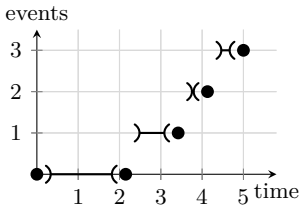



**Figure 2** Example of event-based service curve used in [20], having abscissas in $\mathbb{R}_+$ and ordinates in $\mathbb{N}_0$.

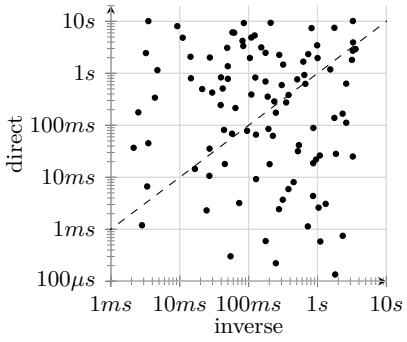



**Figure 3** runtimes of the *direct* vs. *inverse* algorithms for (min,+) convolution.

*direct* algorithm, such extension occurs on hyperperiod $\operatorname{lcm}(d_f, d_g)$. Hence, we can write $\operatorname{lcm}(d_f, d_g) = k_{d_f} \cdot d_f = k_{d_g} \cdot d_g$. We call $k_{d_f}$ and $k_{d_g}$ *extension multipliers*. Computing the cut of $f$ in $\left[T_f, T_f + 2 \cdot d_{h_{pp}}\right[$ (13) entails extending $f$ by $2 \cdot k_{d_f}$ periods – and $g$ by $2 \cdot k_{d_g}$.

On the other hand, the *inverse* algorithm uses pseudoinversion of the operands, which *swaps their period lengths* $d_f, d_g$ *with their period heights* $c_f, c_g$. Consider in fact computing $f \otimes g$ via (7). We obtain for the inner function $f_{\uparrow}^{-1} \mathbin{\overline{\otimes}} g_{\uparrow}^{-1}$, using Theorem 6,

$$\begin{aligned} d_{f_{\uparrow}^{-1} \mathbin{\overline{\otimes}} g_{\uparrow}^{-1}} &= \operatorname{lcm}\left(d_{f_{\uparrow}^{-1}}, d_{g_{\uparrow}^{-1}}\right) = \operatorname{lcm}(c_f, c_g)\,, \\ c_{f_{\uparrow}^{-1} \mathbin{\overline{\otimes}} g_{\uparrow}^{-1}} &= \max\left(\frac{c_{f_{\uparrow}^{-1}}}{d_{f_{\uparrow}^{-1}}}, \frac{c_{g_{\uparrow}^{-1}}}{d_{g_{\uparrow}^{-1}}}\right) \cdot d_{f_{\uparrow}^{-1} \mathbin{\overline{\otimes}} g_{\uparrow}^{-1}} = \max\left(\frac{d_f}{c_f}, \frac{d_g}{c_g}\right) \cdot \operatorname{lcm}(c_f, c_g)\,. \end{aligned} \tag{15}$$

Thus, in the *inverse* algorithm, the hyperperiod is $\operatorname{lcm}(c_f, c_g) = k_{c_f} \cdot c_f = k_{c_g} \cdot c_g$, and the extension multipliers are instead $k_{c_f}$ and $k_{c_g}$.

Both algorithms have the same complexity, but the operands they work on may have considerably different size (i.e., the cardinalities of their extended sequences), hence their runtime can be vastly different. For instance, if $k_{d_f} > k_{c_f}$ and $k_{d_g} > k_{c_g}$, the inverse algorithm will be faster. This is likely the case in the experiments of work [20], which uses event-based service curves, exemplified in Figure 2. However, depending on the parameters of the operands, two more cases can be given, i.e.:

- $k_{d_f} < k_{c_f}$ and $k_{d_g} < k_{c_g}$, in which case the direct algorithm will generally be faster;
- $k_{d_f} < k_{c_f}$ and $k_{d_g} > k_{c_g}$ (or vice versa), in which case the comparison is inconclusive.

Figure 3 compares the direct and the inverse approach. We generated 100 pairs of operands randomly, and reported the runtimes of each (min,+) convolution as coordinates of points on the cartesian plane, with the direct algorithm on the ordinates and the inverse one on the abscissas. The horizontal (or vertical) distance between a point and the bisector indicates the difference (in orders of magnitude) between choosing one algorithm or the other. The above figure clearly shows that the comparison may swing either way, *and* that there is a lot to be gained in choosing wisely.

Hereafter, we follow up on the above observations. We show that the (min,+)/(max,+) isomorphism holds in more general settings than those described in Section 2, and that this, in turn, can be leveraged to define an improved algorithm for (min,+) convolution.

## 3.2 Exploiting Isomorphism to Speed up (min,+) Convolution

As discussed above, the most expensive part of (min,+) convolution is computing $h_{pp} = f_p^\wedge \otimes g_p^\wedge$, and this is due to the problem of operand extension. We have observed in the previous section that pseudoinversion may considerably alter the way operands are extended. Our intuition is that we can *limit the extension of each operand individually* to the minimum of what the direct and inverse algorithms would do, i.e., we can always choose *the smallest extension multiplier* operand by operand, independently. This will allow $h_{pp}$ to be computed using smaller sequences, in considerably less time. We obtain this result via incremental steps. First, we show that isomorphism allows us to find *another set of parameters* for $h_{pp}$.

▶ **Theorem 11.** *Let $f, g \in \mathcal{U}$ be left-continuous and non-decreasing functions. Then, $h_{pp} = f_p^\wedge \otimes g_p^\wedge$ is again a function of $\mathcal{U}$ with*

$$d_{h_{pp}} = \max\left(\frac{d_f}{c_f}, \frac{d_g}{c_g}\right) \cdot \operatorname{lcm}(c_f, c_g) = \max\left(k_{c_g} \cdot d_g, k_{c_f} \cdot d_f\right), \tag{16}$$

$$c_{h_{pp}} = \operatorname{lcm}(c_f, c_g), \tag{17}$$

$$T_{h_{pp}} = \sup\left\{t \geq T_f + T_g \mid f_p^\wedge \otimes g_p^\wedge(t) \leq f(T_f) + g(T_g) + \operatorname{lcm}(c_f, c_g)\right\}. \tag{18}$$

The proof is reported in Appendix A.2. Now, since *both* Corollary 10 and Theorem 11 compute valid parameters for $h_{pp}$, we can always use the *minimum* of each:

$$d_{h_{pp}} = \min\left(\operatorname{lcm}(d_f, d_g), \max\left(k_{c_g} \cdot d_g, k_{c_f} \cdot d_f\right)\right), \tag{19}$$

$$c_{h_{pp}} = d_{h_{pp}} \cdot \min\left(\frac{c_f}{d_f}, \frac{c_g}{d_g}\right), \tag{20}$$

$$\begin{aligned} T_{h_{pp}} = &\min(T_f + T_g + \operatorname{lcm}(d_f, d_g), \\ &\sup\left\{t \geq T_f + T_g \mid f_p^\wedge \otimes g_p^\wedge(t) \leq f(T_f) + g(T_g) + \operatorname{lcm}(c_f, c_g)\right\}). \end{aligned} \tag{21}$$

Then, we show that we can find alternative *cuts* of $f_p^\wedge$ and $g_p^\wedge$ required to compute $h_{pp}$.

▶ **Corollary 12.** *Given $f$ and $g \in \mathcal{U}$ which are left-continuous and non-decreasing in $[T_f, +\infty[$ and $[T_g, +\infty[$, respectively, and are neither UC nor UI. Then, to compute $f_p^\wedge \otimes g_p^\wedge$ via $(\max, +)$ isomorphism of restricted functions, it is sufficient to use sequences $S_{f_p^\wedge}^{I'_{f_p^\wedge}}$ and $S_{g_p^\wedge}^{I'_{g_p^\wedge}}$, with*

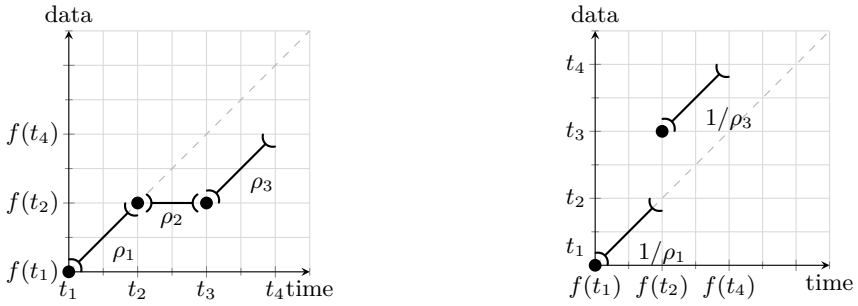
$$I'_{f_p^\wedge} = \left[T_f, T'_f + 2 \cdot k_{c_f} \cdot d_f\right], I'_{g_p^\wedge} = \left[T_g, T'_g + 2 \cdot k_{c_g} \cdot d_g\right]. \tag{22}$$

*where we used $T'_f = \sup\{t \geq T_f \mid f(t) = f(T_f)\}$ and $T'_g = \sup\{t \geq T_g \mid g(t) = g(T_g)\}$.*[8]

The proof is reported in Appendix A.2. Corollary 12 states that, instead of computing $f_p^\wedge \otimes g_p^\wedge$ using domains $I_{f_p^\wedge}$ and $I_{g_p^\wedge}$ defined in (13), we can use *both* $I'_{f_p^\wedge}$ and $I'_{g_p^\wedge}$, whose size depends on $k_{c_f}, k_{c_g}$ instead of $k_{d_f}, k_{d_g}$. Finally, we show that we can *mix and match* the above intervals, to minimize the extension of *each* operand, independently.

▶ **Theorem 13** (Mix and Match ((min,+) Convolution))**.** *Let $f$ and $g \in \mathcal{U}$ which are neither UC nor UI, and are left-continuous and non-decreasing in $[T_f, +\infty[$ and $[T_g, +\infty[$, respectively. Let $I_{f_p^\wedge}, I_{g_p^\wedge}$ be the intervals to compute $f_p^\wedge \otimes g_p^\wedge$ according to (13), and let $I'_{f_p^\wedge}, I'_{g_p^\wedge}$ be the intervals to compute the same through Corollary 12. Then $I_{f_p^\wedge} \cap I'_{f_p^\wedge}, I_{g_p^\wedge} \cap I'_{g_p^\wedge}$ are valid intervals to compute $f_p^\wedge \otimes g_p^\wedge$.*

[8] The suprema are attainable since the functions are left-continuous over the respective intervals.




**(a)** $S_f$. **(b)** $S_{f^{-1}_{\uparrow}}$.

**Figure 4** Example of upper pseudoinverse of a sequence $S_f$.

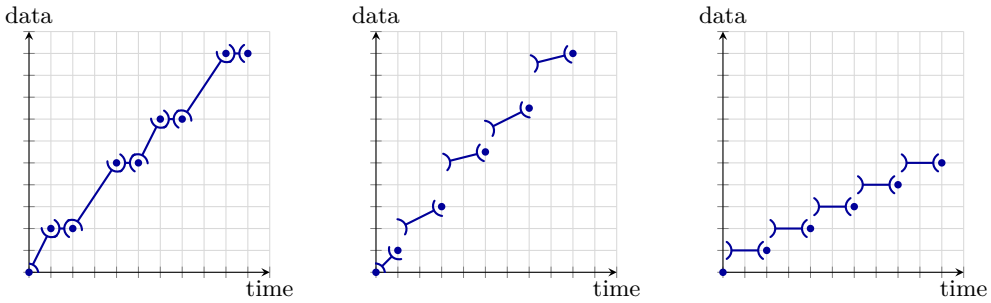
The proof is reported in Appendix A.2. This theorem allows us to compute $h_{pp}$ using extended sequences that have the minimum cardinality between those that the *direct* and the *inverse* methods would compute. We exemplify this through a simple example.

Consider $f$ and $g$ with $d_f = 2$, $c_f = 13$, $d_g = 11$ and $c_g = 3$. Then, using the *direct* method we will compute $2 \cdot k_{d_f} = 22$ period extensions of $f$ and $2 \cdot k_{d_g} = 4$ period extensions of $g$. On the other hand, using the *inverse* method, we will compute the same result with $2 \cdot k_{c_f} = 6$ period extensions of $f$ and $2 \cdot k_{c_g} = 26$ period extensions of $g$. Which of the two will be faster depends both on the number of elements contained in each period of $f$ and $g$, but also on the topological properties of these elements, which are difficult to understand *ex ante* [29]. Using the above theorem, though, we can just take the best option for each independently: $2 \cdot k_{c_f} = 6$ period extensions of $f$ and $2 \cdot k_{d_g} = 4$ period extensions of $g$ – which is clearly better than both the previous options.

Based on the above, we can define a new algorithm, called *isospeed*, which outperforms both the *direct* and the *inverse* ones. It consists in modifying in Algorithm 1 including the new, optimized values for the parameters, i.e., (19), (20) and (21) at line 4, and Theorem 13 at line 5. Moreover, we also optimize line 6, i.e., the *by-sequence* convolution, still leveraging isomorphism. We start from the algorithm in [6], summarized below.

Given two sequences $S_a$ and $S_b$, consider their elements $e^a_1, \ldots, e^a_n$ and $e^b_1, \ldots, e^b_m$, where $n = n(S_a)$, $m = n(S_b)$. For each pair $e^a_i, e^b_j$, we can then compute the *elementary* (min,+) convolution $e^a_i \otimes e^b_j$, i.e., $n(S_a) \cdot n(S_b)$ elementary convolutions. Then, $S_a \otimes S_b$ is computed as the lower envelope of these elementary convolutions. However, it is easy to see that not all pairs $e^a_i, e^b_j$ will contribute to the end result. Indeed, the convolution result is relevant only for a given interval (e.g., in Algorithm 1 $I_{h_{pp}} = [T_f + T_g, T_f + T_g + 2 \cdot \mathrm{lcm}(d_f, d_g)[$), thus any elementary convolution whose abscissas fall outside such interval can be safely skipped. We call this *horizontal filtering*. Similarly, when applying the optimizations described in the previous section, we can ignore elementary convolutions whose *ordinates* fall outside $[f(T_f) + g(T_g), f(T_f) + g(T_g) + 2 \cdot \mathrm{lcm}(c_f, c_g)]$ (*vertical filtering*). Horizontal and vertical filtering further reduce the computation time.

Moreover, we recall that – under pseudoinversion – constant segments become discontinuities and vice versa, leading to different cardinalities for the sequences of an operand. This is exemplified by Figure 4, where $S_f$ has six elements, while $S_{f^{-1}_{\uparrow}}$ has four. While constant segments do contribute to the complexity of computing the convolution, discontinuities do not, being only a difference in value between two elements. Thus, also within the by-sequence convolution there may be a runtime difference between the direct and inverse approach.




**(a)** "Horizontal" curve. **(b)** "Vertical" curve. **(c)** "Balanced" curve.

**Figure 5** Shapes of curves used in our experiments.

Accordingly, we optimize the by-sequence convolution via the following heuristic: we count the total number of constant segments and discontinuities of the two operands, call them $C$ and $D$, respectively. Then, if $D > C$, we perform the by-sequence convolution of the operands as they are. If, instead $D < C$, we pseudo-invert the sequences first, perform a (max,+) by-sequence convolution, and then pseudo-invert the result again. This heuristic is cheap, being $\mathcal{O}(n(S_f) + n(S_g))$. However, it may not have 100% accuracy. In fact, work [29] discusses that the topological properties of the elements, which are difficult to understand *ex ante*, may also influence the runtime of by-sequence convolution.

Finally, we discuss the algorithmic cost of the isospeed algorithm. Applying the mix-and-match theorem requires computing the extension multipliers of both operands and comparing constants, which is $\mathcal{O}(1)$. However, testing the hypotheses that each operand must be left-continuous and non-decreasing is – strictly speaking – $\mathcal{O}(n(S))$, where $S$ is the base sequence, not the extended one. Note that the same cost has to be paid in the *inverse* algorithm as well. However, we observe that such a cost can easily be amortized by testing these properties *once* per operand and caching the result. Moreover, computing $C$ and $D$ for the heuristic also has a linear cost.

## 4 Performance Evaluation

The isospeed algorithm has been implemented by extending Nancy [28], an open-source library implementing the algorithms from [6, 29, 27].We compared it against two baselines, i.e., the *direct* algorithm, [6], recalled in Algorithm 1, and the *inverse* one [20]. To highlight the impact of the by-sequence convolution and its heuristic, we run the experiments using three different shapes of operands, shown in Figure 5. *Horizontal* curves have constant segments but no discontinuities, whereas *vertical* curves have discontinuities but no constant segments. *Balanced* curves are similar to the type of curves studied in [20], and have an equal number of constant segments and discontinuities.

We run the experiments on a cloud Virtual Machine (Intel Xeon Processors (CascadeLake) cores @2.2 GHz, 32 GB of DRAM, Ubuntu 22.04), using randomly generated parameters for the shapes discussed above. We run all algorithms in serial mode (rather than *parallel*, which is the default in Nancy). To make the comparison more challenging, *horizontal* filtering is included in the baseline algorithms as well, since it does not depend on the results of this paper, whereas vertical filtering – which is a consequence of isomorphism – is used only in the isospeed algorithm. Moreover, we include the cost of testing operand properties in the *isospeed* and *inverse* algorithms (there is nothing to test for the *direct* one). We measured the time to compute the convolution using the three methods.

Our results are shown in Figures 6–8. The results of Figures 6a, 7a, and 8a and Figures 6b, 7b, and 8b clearly show that the *isospeed* algorithm outperforms blindly choosing either the *direct* or the *inverse* approach, reducing runtimes often by several orders of magnitude. Moreover, Figures 6c, 7c, and 8c show that *isospeed* performs at least as well as the best between *direct* and *inverse* in most cases, and sometimes even better, being up to one order of magnitude faster. The improvements occur whenever both baseline algorithms extend one operand more than necessary, whereas the isospeed does not, due of its mix-and-match approach. At the risk of stating the obvious, we remark that you do not know which of the two baseline algorithms is the best *beforehand*.

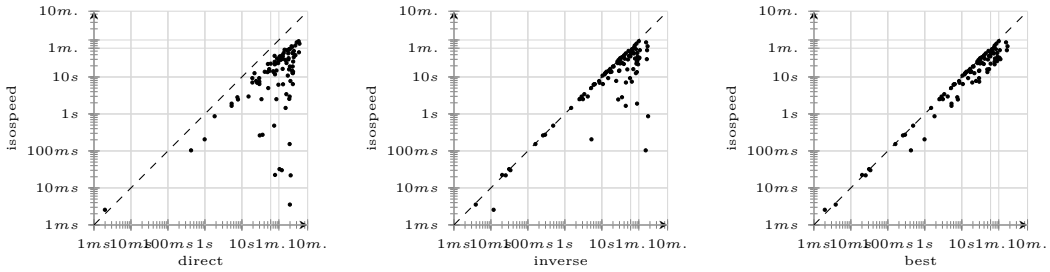
There are indeed some cases when isospeed adds a modicum of overhead – see the few points above the bisector in the figures. This is due to two different reasons: the points in the bottom-left corner of, e.g., Figures 7a and 7c are experiments where runtimes are in the order of milliseconds, and the overhead of testing hypotheses is significant against such a short timespan. We do not see this as a relevant shortcoming – there is little to optimize if the baseline is already that fast. The points above the bisector in the top-right region of Figures 8b and 8c are instead experiments when our heuristic fails to select the most efficient way to perform the by-sequence convolution. This only occurs with *balanced* curves, and for a reason: with *horizontal* and *vertical* curves, the choice is quite clear-cut – it is either $D \gg C$ or $C \ll D$, respectively, and the heuristic always selects the best approach. *Balanced* curves, instead, are designed to thwart our heuristic – they have, in fact, $D \approx C$.

All the above experiments highlighted speedups of up to one order of magnitude against the (clairvoyant) best baseline. Such speedups depend on numerical properties of the operands, and random generation of operand parameters seldom hits on the most interesting cases. As a last set of experiments, we generate horizontal operands in such a way that $k_{d_f} > k_{c_f}$ and $k_{d_g} < k_{c_g}$ (or vice versa), as in the example reported at the end of Section 3.2, so that each baseline algorithm will always extend one operand more than necessary. The results of these experiments are reported in Figure 9, and they show more frequent and significant speedups against the best baseline (e.g., compared with Figure 6) – reaching two orders of magnitude.

As a last remark, we observe that the absolute magnitude of the runtimes involved in these experiments is a few tens of seconds at most. This was done on purpose to keep experiments manageable, and it certainly does not imply that (min,+) convolutions are always that fast. One can always devise cases where runtimes are in the order of hours or more – all it takes is period lengths and heights that are products of large primes. Moreover, it is well known that chaining convolutions leads to exponentially increasing runtimes, much like chaining lcms does, a phenomenon called *state explosion* [12, 13, 29]. In these cases, *isospeed* may act as an enabler of otherwise unfeasible performance studies.

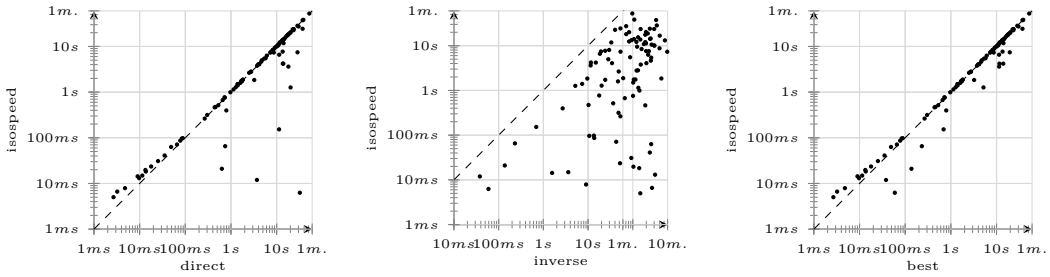
## 5 Conclusions

In this paper, we have investigated what is perhaps the most common operation in (min,+) algebra, i.e., (min,+) convolution. Starting from the observation that – due to (min,+)/(max,+) isomorphism – there are always two ways to compute a (min,+) convolution, the *direct* and *inverse* algorithm, we provide a technically sound explanation of an observation first appeared in [20], i.e., that one algorithm can be considerably faster than the other. The reason lies in the way the two algorithm extend operands, which is the key factor in the complexity of said algorithms. Based on the above observation, we prove algebraic properties that allow one to minimize operand extension, for each operand independently. This allows



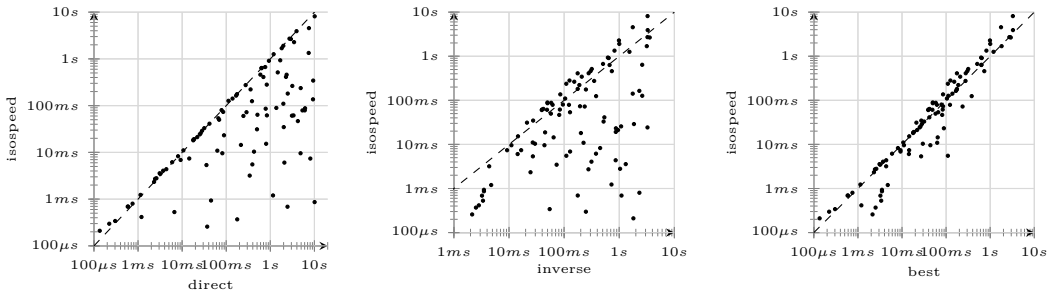

**(a)** *isospeed* vs. *direct*. **(b)** *isospeed* vs. *inverse*. **(c)** *isospeed* vs. *best*.

**Figure 6** Performance comparison of the three algorithms. Operands are *horizontal* curves.




**(a)** *isospeed* vs. *direct*. **(b)** *isospeed* vs. *inverse*. **(c)** *isospeed* vs. *best*.

**Figure 7** Performance comparison of the three algorithms. Operands are *vertical* curves.

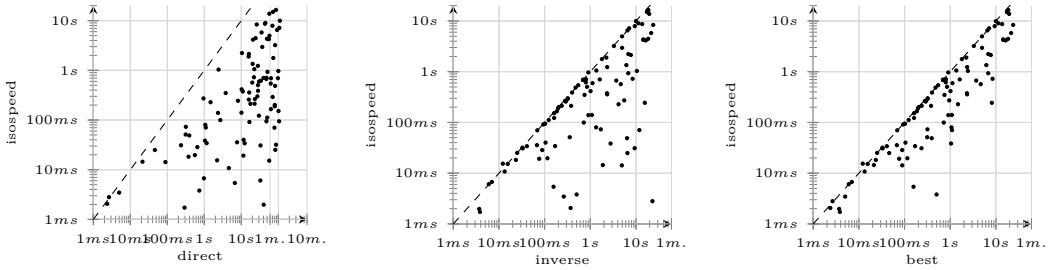



**(a)** *isospeed* vs. *direct*. **(b)** *isospeed* vs. *inverse*. **(c)** *isospeed* vs. *best*.

**Figure 8** Performance comparison of the three algorithms. Operands are *balanced* curves.

us to devise a novel algorithm, called *isospeed*, that outperforms both the *direct* and *inverse* algorithms, reducing runtimes often by several orders of magnitude. More interestingly, our isospeed algorithm also beats a clairvoyant heuristic that guesses the best of the above two baselines: except in few cases when it adds a modicum of overhead, isospeed is at least as fast as that, and can be one or two orders of magnitude faster.

Abating the cost of (min,+) convolution by orders of magnitude is not just a performance improvement: it may also enable performance studies that were previously considered to be beyond the realm of doable, e.g., because of state explosion. Some examples of this problem are reported in [29]. Our findings allow us to reduce this problem as much as possible, in the most general settings: unlike the techniques described in [12, 13, 14, 29], which only apply to specific classes of operands, isospeed only requires operands to be left-continuous and non-decreasing.




**(a)** *isospeed* vs. *direct*. **(b)** *isospeed* vs. *inverse*. **(c)** *isospeed* vs. *best*.

**Figure 9** Performance comparison of the three algorithms. Operands are *horizontal* curves, and their parameters are set so that comparing extension multipliers is inconclusive.

Due to space limitations, we were unable to discuss (max,+) convolution in this paper. However, all the results in this paper apply – via minor, straightforward changes – to that as well, because isomorphism works both ways. More to the point, the performance improvements for (max,+) convolution are exactly the same, since – as we discussed briefly in Section 2 – the algorithm for the (max,+) convolution is not different from that of (min,+) convolution. This increases the significance of our findings.

As a future work, we plan to devise more precise heuristics, that allow one to identify the most efficient by-sequence convolution more effectively. Moreover, we believe that the same process highlighted in this paper could be used to find alternative, improved algorithms for other operations, e.g., the $(\cdot, +)$ *deconvolution*.

Finally, many works in real-time literature deal with *supply functions* and *demand bound functions*, which are similar to the curves discussed here [23, 11, 18, 22, 1]. A further avenue of research is then to explore the possibility to express these results using Real-Time Calculus, hence making the computational improvements discussed in this paper available to speed up the resolution of these problems.

## References


1 Luis Almeida and Paulo Pedreiras. Scheduling within temporal partitions: response-time analysis and server design. In *Proceedings of the 4th ACM international conference on Embedded software*, pages 95–103, 2004.

2 François Baccelli, Guy Cohen, Geert Jan Olsder, and Jean-Pierre Quadrat. *Synchronization and linearity: an algebra for discrete event systems*. John Wiley & Sons Ltd, 1992.

3 Mahmoud Bazzal, Lukas Krawczyk, and Carsten Wolff. RTCAnalysis: Practical Modular Performance Analysis of Automotive Systems with RTC. In Audrius Lopata, Daina Gudonienė, and Rita Butkienė, editors, *Information and Software Technologies*, pages 209–223, Cham, 2021. Springer International Publishing.

4 Anne Bouillard, Marc Boyer, and Euriell Le Corronc. *Deterministic Network Calculus: From Theory to Practical Implementation*. Wiley, Hoboken, NJ, 2018.

5 Anne Bouillard, Bertrand Cottenceau, Bruno Gaujal, Laurent Hardouin, Sébastien Lagrange, Mehdi Lhommeau, and Éric Thierry. COINC library: a toolbox for the Network Calculus. In *VALUETOOLS'09*, 2009.

6 Anne Bouillard and Éric Thierry. An algorithmic toolbox for network calculus. *Discrete Event Dynamic Systems*, 18(1):3–49, 2008.

7 Marc Boyer, Amaury Graillat, Benoît Dupont de Dinechin, and Jörn Migge. Bounding the delays of the MPPA network-on-chip with network calculus: Models and benchmarks. *Performance Evaluation*, 143:102124, 2020. doi:10.1016/j.peva.2020.102124.

8 Cheng-Shang Chang. *Performance guarantees in communication networks*. Springer-Verlang, New York, USA, 2000.

**9** Rene L Cruz. A calculus for network delay, part II: Network analysis. *IEEE Transactions on information theory*, 37(1):132–141, 1991.

**10** Rene L Cruz et al. A calculus for network delay, part I: Network elements in isolation. *IEEE Transactions on information theory*, 37(1):114–131, 1991.

**11** Xiang Feng and Aloysius K Mok. A model of hierarchical real-time virtual resources. In *23rd IEEE Real-Time Systems Symposium, 2002. RTSS 2002.*, pages 26–35. IEEE, 2002.

**12** Nan Guan and Wang Yi. Finitary real-time calculus: Efficient performance analysis of distributed embedded systems. In *2013 IEEE 34th Real-Time Systems Symposium*, pages 330–339, 2013.

**13** Kai Lampka, Steffen Bondorf, and Jens Schmitt. Achieving efficiency without sacrificing model accuracy: Network calculus on compact domains. In *2016 IEEE 24th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 313–318. IEEE, 2016.

**14** Kai Lampka, Steffen Bondorf, Jens B. Schmitt, Nan Guan, and Wang Yi. Generalized finitary Real-Time calculus. In *Proc. of the 36th IEEE International Conference on Computer Communications (INFOCOM 2017)*, 2017.

**15** Jean-Yves Le Boudec and Patrick Thiran. *Network calculus: a theory of deterministic queuing systems for the internet*. Springer Science & Business Media, Berlin, Germany, 2001.

**16** Euriell Le Corronc, Bertrand Cottenceau, and Laurent Hardouin. Container of (min,+)-linear systems. *Discrete Event Dynamic Systems*, 24(1):15–52, 2014.

**17** Jörg Liebeherr. Duality of the Max-Plus and Min-Plus Network Calculus. *Foundations and Trends in Networking*, 11(3-4):139–282, 2017. doi:10.1561/1300000059.

**18** Giuseppe Lipari and Enrico Bini. Resource partitioning among real-time applications. In *15th Euromicro Conference on Real-Time Systems, 2003. Proceedings.*, pages 151–158. IEEE, 2003.

**19** David A. Nascimento, Steffen Bondorf, and Divanilson R. Campelo. Modeling and Analysis of Time-Aware Shaper on Half-Duplex Ethernet PLCA Multidrop. *IEEE Transactions on Communications*, 71(4):2216–2229, 2023. doi:10.1109/TCOMM.2023.3246080.

**20** Victor Pollex, Henrik Lipskoch, Frank Slomka, and Steffen Kollmann. Runtime Improved Computation of Path Latencies with the Real-Time Calculus. In *Proceedings of the 1st International Workshop on Worst-Case Traversal Time*, pages 58–65, 2011.

**21** RealTime-at-Work. RTaW-Pegase (min,+) library. https://www.realtimeatwork.com/rtaw-pegase-libraries/. Accessed: 2022-04-05.

**22** Insik Shin and Insup Lee. Periodic resource model for compositional real-time guarantees. In *RTSS 2003. 24th IEEE Real-Time Systems Symposium, 2003*, pages 2–13. IEEE, 2003.

**23** Martin Stigge, Pontus Ekberg, Nan Guan, and Wang Yi. The digraph real-time task model. In *2011 17th IEEE real-time and embedded technology and applications symposium*, pages 71–80. IEEE, 2011.

**24** Deepak Vedha Raj Sudhakar, Karsten Albers, and Frank Slomka. Generalized and scalable offset-based response time analysis of fixed priority systems. *Journal of Systems Architecture*, 112:101856, 2021. doi:10.1016/j.sysarc.2020.101856.

**25** L. Thiele, S. Chakraborty, and M. Naedele. Real-time calculus for scheduling hard real-time systems. In *2000 IEEE International Symposium on Circuits and Systems (ISCAS)*, volume 4, pages 101–104 vol.4, 2000. doi:10.1109/ISCAS.2000.858698.

**26** Ernesto Wandeler and Lothar Thiele. Real-Time Calculus (RTC) Toolbox. http://www.mpa.ethz.ch/Rtctoolbox. URL: http://www.mpa.ethz.ch/Rtctoolbox.

**27** Raffaele Zippo, Paul Nikolaus, and Giovanni Stea. Extending the Network Calculus Algorithmic Toolbox for Ultimately Pseudo-Periodic Functions: Pseudo-Inverse and Composition, 2022. doi:10.48550/arXiv.2205.12139.

**28** Raffaele Zippo and Giovanni Stea. Nancy: An efficient parallel Network Calculus library. *SoftwareX*, 19:101178, 2022. doi:10.1016/j.softx.2022.101178.

**29** Raffaele Zippo and Giovanni Stea. Computationally efficient worst-case analysis of flow-controlled networks with Network Calculus. *IEEE Transactions on Information Theory*, 2023. doi:10.1109/TIT.2023.3244276.

# A Appendix

This appendix reports the proofs of the main results of this paper, i.e., Theorem 11, Corollary 12 and Theorem 13. In order to make these proofs rigorous, we need a few preliminary technical clarifications. These preliminaries state that the (min,+)/(max,+) isomorphism stated in [20, 17] holds for the convolution of periodic parts as well. This is fairly intuitive, but requires to be rigorously stated nonetheless. Said preliminaries re-state existing results from [6, 4, 17, 27], tweaking the existing proofs to accommodate restricted functions in $\mathcal{U}$. The preliminary results are reported in the first part of this appendix, and the proof of our main results follow in the second part.

## A.1 Preliminary results

In the main results of this work, we exploit properties analogue to the isomorphisms proved in [17], such as Theorem 8, which need to be applied to functions restricted over a support, e.g., $f_p^{\wedge}$. Now, pseudoinverses – and, consequently, isomorphism – are defined for *non-decreasing* functions, and restricted functions are not. Thus, in this section we generalize pseudoinversion and isomorphism to include restricted functions.

▶ **Definition 14** (Lower and Upper Pseudoinverse over an Interval)**.** *Let $f \in \mathcal{U}$ be non-decreasing over $I$, where $I = [a, +\infty[ \subset \mathbb{Q}$. Then, its* lower pseudoinverse over (the interval) *$I$ is defined as*

$$f^{-1}_{\downarrow,I}(y) \coloneqq \begin{cases} \inf\{t \in I \mid f(t) \geq y\}, & \text{if } y \geq f(a), \\ +\infty, & \text{otherwise,} \end{cases} \tag{23}$$

*and its* upper pseudoinverse over (the interval) *$I$ is defined as*

$$f^{-1}_{\uparrow,I}(y) \coloneqq \begin{cases} \sup\{t \in I \mid f(t) \leq y\}, & \text{if } y \geq f(a), \\ -\infty, & \text{otherwise.} \end{cases} \tag{24}$$

Note that it does not hold in general that $\sup\{t \in I \mid f(t) < y\} = \inf\{t \in I \mid f(t) \geq y\}$, for the lower pseudoinverse, as well as $\sup\{t \in I \mid f(t) \leq y\} = \inf\{t \in I \mid f(t) > y\}$, for the upper pseudoinverse. However, if $y > f(a)$ (for $I = [a, +\infty[$), the two equations hold. We also note that, since these pseudoinverses consider only values of $f(t)$ for $t \in I$, it follows that $f^{-1}_{\downarrow,I} = \left(f|_I^{\wedge}\right)^{-1}_{\downarrow,I} = \left(f|_I^{\vee}\right)^{-1}_{\downarrow,I}$, and similarly for $f^{-1}_{\uparrow,I}$. Finally, given $f(I) \coloneqq [f(a), +\infty[$, the lower pseudoinverse over $I$ is left-continuous over $f(I)$, and the upper pseudoinverse over $I$ is right-continuous over $f(I)$.

▶ **Theorem 15.** *Let $I$ be an interval of the form $[a, +\infty[$. Let $f \in \mathcal{U}$ be neither UC nor UI, and is non-decreasing over $I$. Then, its lower pseudoinverse over $I$, $f^{-1}_{\downarrow,I}$, is again a function of $\mathcal{U}$ with*

$$T_{f^{-1}_{\downarrow,I}} = \begin{cases} f\left(T_f + d_f\right), & \text{if } a \leq T_f, \\ f\left(a + d_f\right), & \text{if } a > T_f, \end{cases} \tag{25}$$

$$d_{f^{-1}_{\downarrow,I}} = c_f, \tag{26}$$

$$c_{f^{-1}_{\downarrow,I}} = d_f, \tag{27}$$

*and its upper pseudoinverse over $I$, $f^{-1}_{\uparrow,I}$, is again a function of $\mathcal{U}$ with*

$$T_{f^{-1}_{\uparrow,I}} = \begin{cases} f(T_f), & \text{if } a \leq T_f, \\ f(a), & \text{if } a > T_f, \end{cases} \tag{28}$$

$$d_{f^{-1}_{\uparrow,I}} = c_f, \tag{29}$$

$$c_{f^{-1}_{\uparrow,I}} = d_f. \tag{30}$$

The proofs are easily derived following the steps of those in Theorem 9 and Theorem 10 in [27]. The only difference is that, since we are considering values of $f(t)$ only for $t \in [a, +\infty[$, we can only consider $f(t)$ to be UPP from $\max(T_f, a)$. Note that, in the rest of this paper, we will always use $a \leq T_f$, thus only the first branch of (25) and (28) apply. As briefly mentioned in [27], one can improve the result of (25) using additional assumptions on the shape of $f$. As this will be useful in this work, we derive these results explicitly.

▶ **Lemma 16.** *Let $f \in \mathcal{U}$ be neither UC nor UI, right-continuous, and non-decreasing over the interval $I = [a, +\infty[$, where $a \leq T_f$. Let*

$$T^*_f \coloneqq f^{-1}_{\downarrow,I}(f(T_f)) = \inf\{t \geq a \mid f(t) = f(T_f)\}, \tag{31}$$

$$T^{**}_f \coloneqq f^{-1}_{\downarrow,I}(f(T^*_1 + d_f)) = \inf\{t \geq a \mid f(t) = f(T^*_1 + d_f)\}. \tag{32}$$

*Then, if $f$ is UPP from $T^*_f$ and if $T^{**}_f = T^*_f + d_f$, the pseudo-periodic start $T_{f^{-1}_{\uparrow}}$ in (25) can be improved into*

$$T_{f^{-1}_{\downarrow,I}} = f(T_f). \tag{33}$$

The properties addressed by Lemma 16 have to do with constant segments at the start and the end of the pseudo-period which, as discussed in [27], can lead to $T_{f^{-1}_{\downarrow,I}} > f(T_f)$. The conditions of Lemma 16 ensure that this does not happen, by checking that either a) there are no constant segments before $T_f$ and $T_f + d$, or b) these constant segments are of equal length, such that $f^{-1}_{\downarrow,I}$ is UPP from $f(T_f)$. The proof consists, in fact, in verifying that due to right-continuity of $f$ and the definitions of $T^*_f$, $T^{**}_f$, it follows that a) or b) are verified and thus $T_{f^{-1}_{\downarrow,I}} = f(T_f)$.

▶ **Lemma 17.** *Let $f \in \mathcal{U}$ be neither UC nor UI, left-continuous, and non-decreasing over the interval $I = [a, +\infty[$, where $a \leq T_f$. Moreover, let $f^{-1}_{\uparrow,I}$ be its upper pseudoinverse over $I$. Then, $f^{-1}_{\uparrow,I}$ satisfies the conditions of Lemma 16, thus its lower pseudoinverse $\left(f^{-1}_{\uparrow,I}\right)^{-1}_{\downarrow,[f(a),+\infty[}$ is UPP from $f^{-1}_{\uparrow,I}(T_{f^{-1}_{\uparrow,I}}) = T_f$.*

The proof is easily derived by observing that, in order for $f^{-1}_{\uparrow,I}$ to violate the conditions of Lemma 16, $f$ would need to have left-discontinuities, which is a contradiction.

▶ **Lemma 18** (Sufficient cut for Upper Pseudoinverse over Interval)**.** *Let $f \in \mathcal{U}$ be neither UC nor UI, and is left-continuous and non-decreasing over $I = [a, +\infty[$. Then, in order to compute $f^{-1}_{\uparrow,I}(x)$ and $x \in [x_1, x_2] \subset [f(a), +\infty[$ with $x_1 < x_2$, it is sufficient to use $f(t)$ with $t \in [t_1, t_2]$, where $t_1 \coloneqq f^{-1}_{\uparrow,I}(x_1)$ and $t_2 \coloneqq f^{-1}_{\uparrow,I}(x_2)$.*

The proof is easily derived by observing that, for any $x \in [x_1, x_2]$, $\sup\{t \geq a \mid f(t) \leq x\} = \sup\{t_1 \leq t \leq t_2 \mid f(t) \leq x\}$.

▶ **Lemma 19** (Sufficient cut for Lower Pseudoinverse over Interval). *Let $f \in \mathcal{U}$ be neither UC nor UI, and is right-continuous and non-decreasing over $I = [a, +\infty[$. Then, in order to compute $f^{-1}_{\downarrow,I}(x)$ with $x \in [x_1, x_2] \subset [f(a), +\infty[$ and $x_1 < x_2$, it is sufficient to use $f(t)$ with $t \in [t_1, t_2]$, where $t_1 \coloneqq f^{-1}_{\downarrow,I}(x_1)$ and $t_2 \coloneqq f^{-1}_{\downarrow,I}(x_2)$.*

The proof is similar to the one for Lemma 18.

▶ **Lemma 20.** *Let $f \in \mathcal{U}$ be non-decreasing and $I = [a, +\infty[ \subset \mathbb{Q}_+$. Let $x \in I$. If $f(x) \leq y$, then $f^{-1}_{\uparrow,I}(y) \geq x$.*

The proof follows along the lines of [17, pp. 62], since it follows from $x \in I$ that we are in the supremum case of (24). Next, we generalize Proposition 3.10 in [4, pp. 48].

▶ **Proposition 21.** *Let $f, g \in \mathcal{U}$ be left-continuous and non-decreasing. Then, for any $t \in [T_f + T_g, +\infty[$ it exists $s^* \in [T_f, t - T_g]$ such that*

$$(f_p^{\wedge} \otimes g_p^{\wedge})(t) = \inf_{T_f \leq s \leq t - T_g} \left\{ f_p^{\wedge}(s) + g_p^{\wedge}(t-s) \right\} = f(s^*) + g(t - s^*).$$

*In other words, the infimum is attainable.*

The proof follows along the lines of [4, pp. 48]. The only difference is related to the use of Bolzano-Weierstrass theorem for real sequences, obtaining $s^* \in \mathbb{R}_+$. However, it is easy to show that, since functions of $\mathcal{U}$ are piecewise affine $\mathbb{Q}_+ \to \mathbb{Q} \cup \{+\infty, -\infty\}$, given any convergent sequence $f(s_n)$, $s_n \in \mathbb{Q}_+$ and $s^*$ attains the limit of this sequence, then $s^* \in \mathbb{Q}_+$.

▶ **Proposition 22.** *Let $f$ and $g$ be non-decreasing and right-continuous functions of $\mathcal{U}$. Then, for any $t \in [T_f + T_g, +\infty[$ it exists $s^* \in [T_f, t - T_g]$ such that*

$$(f_p^{\vee} \mathbin{\overline{\otimes}} g_p^{\vee})(t) = \sup_{T_f \leq s \leq t - T_g} \left\{ f_p^{\vee}(s) + g_p^{\vee}(t-s) \right\} = f(s^*) + g(t - s^*)$$

*In other words, the supremum is attainable.*

The proof follows along the same lines as Proposition 21. Next, we provide a generalization of Theorem 8 for functions restricted to their periodic part.

▶ **Theorem 23.** *Let $f, g \in \mathcal{U}$ be left-continuous and non-decreasing. Then*

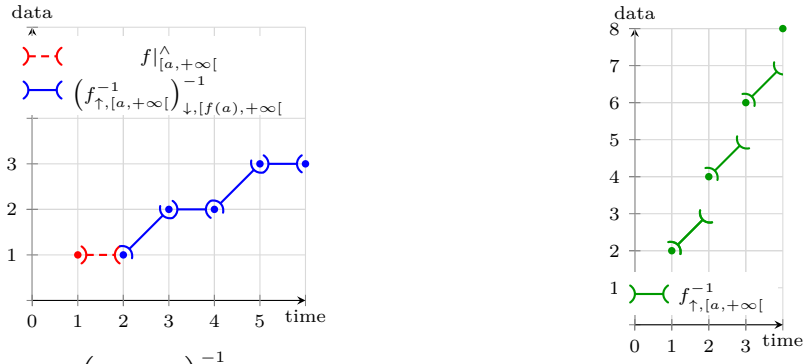
$$\left(f_p^{\wedge} \otimes g_p^{\wedge}\right)^{-1}_{\uparrow,[T_f+T_g,+\infty[} = \left(f^{-1}_{\uparrow,p} \mathbin{\overline{\otimes}} g^{-1}_{\uparrow,p}\right). \tag{34}$$

The proof follows along the same lines of [17, Theorem 10.3, p. 69]. The only difference is that, for $x < f(T_f) + g(T_g)$, we derive that both sides are $-\infty$. Next, we also generalize the involutive property of pseudoinverses.

▶ **Proposition 24.** *Let $f \in \mathcal{U}$ be left-continuous and non-decreasing on the interval $I = [a, +\infty[$. Let $a' \coloneqq \sup\{t \geq a \mid f(t) = f(a)\} \geq a$. Then*

$$\left(f^{-1}_{\uparrow,[a,+\infty[}\right)^{-1}_{\downarrow,[f(a),+\infty[} = f|^{\wedge}_{[a',+\infty[}.$$

The proof follows along the same lines of [17, Lemma 10.1 (c), pp. 64-6]. The only difference is that, for $t < a'$, we derive that for both sides are $+\infty$. Note that Proposition 24 implies that performing the lower pseudoinverse (over an interval) of an upper pseudoinverse (over an interval) does not reconstitute $f$ over that same interval, but only a subset of it: in fact, we obtain $f|^{\wedge}_{[a',+\infty[}$ instead of $f|^{\wedge}_{[a,+\infty[}$, where $a' \geq a$. This information loss




**(a)** $f|^{\wedge}_{[a,+\infty[}$ vs. $\left(f^{-1}_{\uparrow,[a,+\infty[}\right)^{-1}_{\downarrow,[f(a),+\infty[}$. **(b)** $f^{-1}_{\uparrow,[a,+\infty[}$.

**Figure 10** Example of loss of information when computing pseudoinverses over an interval.

is exemplified in Figure 10, where we show that the values of $f$ between 1 and 2 are lost. However, if the value $a$ is known, it is easy to reconstitute $f|^{\wedge}_{[a,+\infty[}$ by observing that the missing values of $f$ in $[a, a'[$ are all $f(a')$, which is part of the result. Again referencing Figure 10, we can see how the constant segment in red is the missing piece that can be reconstituted by knowing $a = 1$. We formalize this process through the *reconstruction operator*, $[f]_a$. Given a function $f$ that is either $+\infty$ or $-\infty$ in $[0, a'[$, and finite in $[a', +\infty[$, then

$$[f]_a(t) \coloneqq \begin{cases} f(t), & \text{if } t \in [0, a[, \\ f(a'), & \text{if } t \in [a, a'[, \\ f(t), & \text{if } t \in [a', +\infty[. \end{cases} \tag{35}$$

Using the reconstruction operator, we can state a stronger version of Proposition 24.

▶ **Proposition 25.** *Let $f \in \mathcal{U}$ be left-continuous and non-decreasing on the interval $I = [a, +\infty[$. Let $a' \coloneqq \sup\{t \geq a \mid f(t) = f(a)\} \geq a$. Then*

$$\left[\left(f^{-1}_{\uparrow,[a,+\infty[}\right)^{-1}_{\downarrow,[f(a),+\infty[}\right]_a = f|^{\wedge}_{[a,+\infty[}. \tag{36}$$

The proof is easily derived by applying (35). Combining these results, we can derive an alternative expression for $f^{\wedge}_p \otimes g^{\wedge}_p$, which is the analogous to (7) for restricted functions.

$$f^{\wedge}_p \otimes g^{\wedge}_p = \left[\left(f^{-1}_{\uparrow,p} \mathbin{\overline{\otimes}} g^{-1}_{\uparrow,p}\right)^{-1}_{\downarrow,[f(T_f)+g(T_g),+\infty[}\right]_{T_f+T_g}. \tag{37}$$

## A.2 Proofs of the results in Section 3.2

▶ **Theorem 11.** *Let $f, g \in \mathcal{U}$ be left-continuous and non-decreasing functions. Then, $h_{pp} = f^{\wedge}_p \otimes g^{\wedge}_p$ is again a function of $\mathcal{U}$ with*

$$d_{h_{pp}} = \max\left(\frac{d_f}{c_f}, \frac{d_g}{c_g}\right) \cdot \operatorname{lcm}(c_f, c_g) = \max\left(k_{c_g} \cdot d_g, k_{c_f} \cdot d_f\right), \tag{16}$$

$$c_{h_{pp}} = \operatorname{lcm}(c_f, c_g), \tag{17}$$

$$T_{h_{pp}} = \sup\left\{t \geq T_f + T_g \mid f^{\wedge}_p \otimes g^{\wedge}_p(t) \leq f(T_f) + g(T_g) + \operatorname{lcm}(c_f, c_g)\right\}. \tag{18}$$

**Proof.** Using Proposition 25, we have that

$$f_p^\wedge \otimes g_p^\wedge \overset{(36)}{=} \left[\left(\left(f_p^\wedge \otimes g_p^\wedge\right)^{-1}_{\uparrow,[T_f+T_g,+\infty[}\right)^{-1}_{\downarrow,[f(T_f)+g(T_g),+\infty[}\right]_{(T_f+T_g)}$$
$$\overset{(34)}{=} \left[\left(f_{\uparrow,p}^{-1} \mathbin{\overline{\otimes}} g_{\uparrow,p}^{-1}\right)^{-1}_{\downarrow,[f(T_f)+g(T_g),+\infty[}\right]_{(T_f+T_g)},$$

where we used Theorem 23 in the second line. For the inner part (the (max,+) convolution), we obtain for all $x \geq f(T_f) + g(T_g) + c_{h_{pp}} = f(T_f) + g(T_g) + \operatorname{lcm}(c_f, c_g)$ that

$$\begin{aligned}
&\left(f_{\uparrow,p}^{-1} \mathbin{\overline{\otimes}} g_{\uparrow,p}^{-1}\right)(x + c_{h_{pp}}) \\
&= \sup_{f(T_f)\leq u\leq x+c_{h_{pp}}-g(T_g)} \left\{f_{\uparrow,p}^{-1}(u) + g_{\uparrow,p}^{-1}(x + c_{h_{pp}} - u)\right\} \\
&\overset{(x-g(T_g)\geq f(T_f)+c_{h_{pp}})}{=} \sup_{f(T_f)\leq u\leq x-g(T_g)} \left\{f_{\uparrow,p}^{-1}(u) + g_{\uparrow,p}^{-1}(x + c_{h_{pp}} - u)\right\} \\
&\quad \vee \sup_{f(T_f)+c_{h_{pp}}\leq u\leq x+c_{h_{pp}}-g(T_g)} \left\{f_{\uparrow,p}^{-1}(u) + g_{\uparrow,p}^{-1}(x + c_{h_{pp}} - u)\right\},
\end{aligned}$$

We continue by substituting $v \coloneqq x + c_{h_{pp}} - u$:

$$\begin{aligned}
&\left(f_{\uparrow,p}^{-1} \mathbin{\overline{\otimes}} g_{\uparrow,p}^{-1}\right)(x + c_{h_{pp}}) \\
&\overset{(v:=x+c_{h_{pp}}-u)}{=} \sup_{f(T_f)\leq u\leq x-g(T_g)} \left\{f_{\uparrow,p}^{-1}(u) + g_{\uparrow,p}^{-1}(x + \underbrace{\operatorname{lcm}(c_f, c_g)}_{=k_{c_g}c_g} - u)\right\} \\
&\quad \vee \sup_{g(T_g)\leq v\leq x-f(T_f)} \left\{f_{\uparrow,p}^{-1}(x + \underbrace{\operatorname{lcm}(c_f, c_g)}_{=k_{c_f}c_f} - v) + g_{\uparrow,p}^{-1}(v)\right\} \\
&\overset{(x-u\geq g(T_g),x-v\geq f(T_f))}{=} \sup_{f(T_f)\leq u\leq x-g(T_g)} \left\{f_{\uparrow,p}^{-1}(u) + g_{\uparrow,p}^{-1}(x - u)\right\} + k_{c_g}d_g \\
&\quad \vee \sup_{g(T_g)\leq v\leq x-f(T_f)} \left\{f_{\uparrow,p}^{-1}(x - v) + g_{\uparrow,p}^{-1}(v)\right\} + k_{c_f}d_f \\
&= \left(f_{\uparrow,p}^{-1} \mathbin{\overline{\otimes}} g_{\uparrow,p}^{-1}\right)(x) + \max\left(k_{c_g} \cdot d_g, k_{c_f} \cdot d_f\right),
\end{aligned}$$

where we used Theorem 6 in the fourth line. It follows then that

$$T_{\overline{\otimes}_p^{-1}} = f(T_f) + g(T_g) + \operatorname{lcm}(c_f, c_g), \tag{38}$$
$$d_{\overline{\otimes}_p^{-1}} = c_{h_{pp}} = \operatorname{lcm}(c_f, c_g), \tag{39}$$
$$c_{\overline{\otimes}_p^{-1}} = \max\left(k_{c_g} \cdot d_g, k_{c_f} \cdot d_f\right). \tag{40}$$

Next, for the outer part we consider the lower pseudoinverse of the above result, over the interval $[f(T_f) + g(T_g), +\infty[$. From Lemmas 16 and 17, it follows that

$$\begin{aligned}
T_{h_{pp}} &\overset{(33)}{=} \left(f_p^\wedge \otimes g_p^\wedge\right)^{-1}_{\uparrow,[T_f+T_g,+\infty[}\left(T_{\overline{\otimes}_p^{-1}}\right) \\
&\overset{(24)}{=} \sup\left\{t \geq T_f + T_g \mid f_p^\wedge \otimes g_p^\wedge(t) \leq T_{\overline{\otimes}_p^{-1}}\right\}.
\end{aligned}$$

From Theorem 15 it follows also that

$$d_{h_{pp}} = c_{\overline{\otimes}_p^{-1}} = \max\left(k_{c_g} \cdot d_g, k_{c_f} \cdot d_f\right),$$
$$c_{h_{pp}} = d_{\overline{\otimes}_p^{-1}} = \operatorname{lcm}(c_f, c_g).$$

This finishes the proof. ◀

▶ **Corollary 12.** *Given $f$ and $g \in \mathcal{U}$ which are left-continuous and non-decreasing in $[T_f, +\infty[$ and $[T_g, +\infty[$, respectively, and are neither UC nor UI. Then, to compute $f_p^\wedge \otimes g_p^\wedge$ via $(\max, +)$ isomorphism of restricted functions, it is sufficient to use sequences $S_{f_p^\wedge}^{I'_{f_p^\wedge}}$ and $S_{g_p^\wedge}^{I'_{g_p^\wedge}}$, with*

$$I'_{f_p^\wedge} = \left[T_f, T'_f + 2 \cdot k_{c_f} \cdot d_f\right], I'_{g_p^\wedge} = \left[T_g, T'_g + 2 \cdot k_{c_g} \cdot d_g\right]. \tag{22}$$

*where we used $T'_f = \sup\{t \geq T_f \mid f(t) = f(T_f)\}$ and $T'_g = \sup\{t \geq T_g \mid g(t) = g(T_g)\}$.*[9]

**Proof.** The proof is based on using Proposition 25, as we did in the proof of Theorem 11. We thus compute $f_p^\wedge \otimes g_p^\wedge$ through $f_{\uparrow,p}^{-1} \overline{\otimes} g_{\uparrow,p}^{-1}$. In the proof of Theorem 11, we derived the UPP properties of the latter as (38), (39) and (40). Thus, we need to compute $S_{\overline{\otimes}_p^{-1}}^{I_{\overline{\otimes}_p^{-1}}} = S_q^{I_q} \overline{\otimes} S_r^{I_r}$, where $q \coloneqq f_{\uparrow,p}^{-1}$ and $r \coloneqq g_{\uparrow,p}^{-1}$ with

$$I_{\overline{\otimes}_p^{-1}} = [f(T_f) + g(T_g), f(T_f) + g(T_g) + 2 \cdot \operatorname{lcm}(c_f, c_g)[, \tag{41}$$
$$I_{f_{\uparrow,p}^{-1}} = [f(T_f), f(T_f) + 2 \cdot \operatorname{lcm}(c_f, c_g)] = \left[f(T_f), f(T_f) + 2 \cdot k_{c_f} \cdot c_f\right], \tag{42}$$
$$I_{g_{\uparrow,p}^{-1}} = [g(T_g), g(T_g) + 2 \cdot \operatorname{lcm}(c_f, c_g)] = \left[g(T_g), g(T_g) + 2 \cdot k_{c_g} \cdot c_g\right]. \tag{43}$$

Next, we derive which values of $f_p^\wedge$ and $g_p^\wedge$ are needed in order to compute the values of $f_{\uparrow,p}^{-1}$ in $I_{f_{\uparrow,p}^{-1}}$ and $g_{\uparrow,p}^{-1}$ in $I_{g_{\uparrow,p}^{-1}}$. We focus, without loss of generality, on $f_p^\wedge$, and obtain via Lemma 18 that

$$f_{\uparrow,p}^{-1}(f(T_f)) = \sup\{t \geq T_f \mid f(t) \leq f(T_f)\} = T'_f,$$

and using Theorem 6

$$f_{\uparrow,p}^{-1}(f(T_f) + 2 \cdot \operatorname{lcm}(c_f, c_g)) = f_{\uparrow,p}^{-1}(f(T_f)) + 2 \cdot k_{c_f} \cdot d_f = T'_f + 2 \cdot k_{c_f} \cdot d_f.$$

Hence, it is sufficient to use $\left[T'_f, T'_f + 2 \cdot k_{c_f} \cdot d_f\right]$ for $f_{\uparrow,p}^{-1}$, and $\left[T'_g, T'_g + 2 \cdot k_{c_g} \cdot d_g\right]$ for $g_{\uparrow,p}^{-1}$, to compute $f_{\uparrow,p}^{-1} \overline{\otimes} g_{\uparrow,p}^{-1}$. Then, to compute the lower pseudoinverse, we do not require any additional value from $f$ and $g$. We do so however for the last step, due to the loss of information implied by having $T'_f$ and $T'_g$ as left boundaries. Using Proposition 25, the reconstruction operator requires us to know that $f(t) = f(T_f)\ \forall T_f \leq t \leq T'_f$, we obtain

$$I'_{f_p^\wedge} = \left[T_f, T'_f + 2 \cdot k_{c_f} \cdot d_f\right], I'_{g_p^\wedge} = \left[T_g, T'_g + 2 \cdot k_{c_g} \cdot d_g\right]. \quad ◀$$

▶ **Theorem 13** (Mix and Match ((min,+) Convolution))**.** *Let $f$ and $g \in \mathcal{U}$ which are neither UC nor UI, and are left-continuous and non-decreasing in $[T_f, +\infty[$ and $[T_g, +\infty[$, respectively. Let $I_{f_p^\wedge}, I_{g_p^\wedge}$ be the intervals to compute $f_p^\wedge \otimes g_p^\wedge$ according to (13), and let $I'_{f_p^\wedge}, I'_{g_p^\wedge}$ be the intervals to compute the same through Corollary 12. Then $I_{f_p^\wedge} \cap I'_{f_p^\wedge}, I_{g_p^\wedge} \cap I'_{g_p^\wedge}$ are valid intervals to compute $f_p^\wedge \otimes g_p^\wedge$.*

[9] The suprema are attainable since the functions are left-continuous over the respective intervals.

**Proof.** We prove the result for $I_{f_p^\wedge} \supset I'_{f_p^\wedge}$, $I_{g_p^\wedge} = I'_{g_p^\wedge}$, and $T_{h_{pp}}$ as given by (18). The remaining cases follow by commutativity and / or along the same lines. Let $I_{f_p^\wedge} = [T_f, b]$, $I'_{f_p^\wedge} = [T_f, a]$ with $a < b$. Moreover, we define $d_{h_{pp}}$ according to (19). We show now that the values of $f$ in $]a, b]$ are not necessary for the computation. Therefore, assume that this is not the case, i.e., there exists some $t^* \in \left[T_f + T_g, T_{h_{pp}} + d_{h_{pp}}\right]$, $s^* \in ]a, b]$ such that

$$\inf_{0 \le s \le t^*,\, s \notin ]a,b]} \left\{ f_p^\wedge(s) + g_p^\wedge(t^* - s) \right\} > f_p^\wedge \otimes g_p^\wedge(t^*) = f_p^\wedge(s^*) + g_p^\wedge\left(t^* - s^*\right) =: z^*.$$

From $I'_{f_p^\wedge} = [T_f, a]$ and Lemma 18 it follows that $I_{f^{-1}_{\uparrow,p}} = [f(T_f), f(a)]$. Let us consider now $\left(f_p^\wedge \otimes g_p^\wedge\right)^{-1}_\uparrow(z^*) = f^{-1}_{\uparrow,p} \mathbin{\overline{\otimes}} g^{-1}_{\uparrow,p}(z^*)$. We distinguish two cases: either $z^* \in I_{\overline{\otimes}_p^{-1}}$, computed according to (41), or it is larger than the upper boundary of $I_{\overline{\otimes}_p^{-1}}$. In the first case, i.e., $z^* < f(T_f) + g(T_g) + 2 \cdot \mathrm{lcm}(c_f, c_g)$, we have

$$f^{-1}_{\uparrow,p} \mathbin{\overline{\otimes}} g^{-1}_{\uparrow,p}(z^*) = \sup_{0 \le v \le z^*} \left\{ f^{-1}_{\uparrow,p}(v) + g^{-1}_{\uparrow,p}(z^* - v) \right\} = f^{-1}_{\uparrow,p}(v^*) + g^{-1}_{\uparrow,p}\left(z^* - v^*\right),$$

where $v^* \in I'_{f^{-1}_{\uparrow,p}} = [f(T_f), f(a)]$ such that the supremum is attained (which exists being the upper pseudoinverses right-continuous and due to Proposition 22). Moreover, since $I'_{f^{-1}_{\uparrow,p}}$ and $I_{g^{-1}_{\uparrow,p}}$ are sufficient to compute $f^{-1}_{\uparrow,p} \mathbin{\overline{\otimes}} g^{-1}_{\uparrow,p}(z)$ for any $z \in I_{\overline{\otimes}_p^{-1}}$ (Corollary 12), it follows that $v^* \in I'_{f^{-1}_{\uparrow,p}}$ and $z^* - v^* \in I_{g^{-1}_{\uparrow,p}}$, hence we do not need any value of $f$ and $g$ outside of these intervals to perform the computation for $z^*$ in particular. But, since the interval $]a, b]$ was not used to compute $I'_{f^{-1}_{\uparrow,p}}$, this is a contradiction to $s^* \in ]a, b]$ being needed for $f_p^\wedge \otimes g_p^\wedge$.

In the second case ($z^* \ge f(T_f) + g(T_g) + 2 \cdot \mathrm{lcm}(c_f, c_g)$), $f^{-1}_{\uparrow,p} \mathbin{\overline{\otimes}} g^{-1}_{\uparrow,p}(z^*)$ can be computed by applying the UPP property meaning that

$$f^{-1}_{\uparrow,p} \mathbin{\overline{\otimes}} g^{-1}_{\uparrow,p}(z^*) = f^{-1}_{\uparrow,p} \mathbin{\overline{\otimes}} g^{-1}_{\uparrow,p}\left(z^* - k \cdot d_{\overline{\otimes}_p^{-1}}\right) + k \cdot c_{\overline{\otimes}_p^{-1}}$$

for $d_{\overline{\otimes}_p^{-1}}$ and $c_{\overline{\otimes}_p^{-1}}$ described in (39) and (40), respectively, and some $k \in \mathbb{N}$ such that $z^* - k \cdot d_{\overline{\otimes}_p^{-1}} \in I_{\overline{\otimes}_p^{-1}}$. We can follow for the latter the same reasoning as in the first case, thus having again a contradiction to the assumption that $s^* \in ]a, b]$ is needed for the computation of $f_p^\wedge \otimes g_p^\wedge$. ◀