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

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 abate 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/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 [2] – 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.

Related Article Raffaele Zippo, Paul Nikolaus, and Giovanni Stea, "Isospeed: Improving (min,+) Convolution by Exploiting (min,+)/(max,+) Isomorphism", in 35th Euromicro Conference on Real-Time Systems (ECRTS 2023), LIPIcs, Vol. 262, pp. 12:1–12:24, 2023.
`https://doi.org/10.4230/LIPIcs.ECRTS.2023.12`

Related Conference 35th Euromicro Conference on Real-Time Systems (ECRTS 2023), July 11–14, 2023, Vienna, Austria

1 Artifact

This document introduces the *artifact* related to [3], i.e., the code implementation of the algorithm described therein and the benchmarks used to show their improvement. This artifact is provided, with an MIT license, as a GitHub repository at `https://github.com/rzippo/ecrts-2023-artifact`, and is composed of

- the source code of the Nancy library [4], extended with the (min,+) isospeed algorithm discussed in the paper
- a benchmark program that runs the experiments whose results are discussed in the paper
- a parser program that makes TikZ figures from the results produced by the benchmarks

The algorithmic implementation provided here is the same used to generate the results discussed in the paper. This will slightly differ from what we will soon publish on `https://github.com/rzippo/nancy` – we found some useful tweaks in the past two months.

2 Experiments and claims

We present in [3] a new method for (min,+) convolution called *isospeed*, which leverages the isomorphism between (min,+) and (max,+) algebras to reduce the computation times of said operation, under very general hypotheses on the operands. Our expriments compare our isospeed algorithm against two baselines:

- the direct algorithm, i.e., the standard algorithm for (min,+) convolution found in [1];
- the inverse algorithm, i.e., the algorithm that uses pseudoinversion of the operands, a (max,+) convolution, and pseudoinversion of the results. This algorithm is described in [2].

These experiments measure the runtime of (min,+) convolution of randomly generated operands, using the three algorithms (isospeed, direct, and inverse). They show that isospeed is (almost always) as fast as, and often much faster than, the best between the direct algorithm and the inverse one. There are few cases when isospeed is not as fast as the best baselines. These are due to different causes:

- a particular (min,+) convolution is computationally inexpensive (i.e., in the order of milliseconds), and unoptimizable (i.e., either the direct or the inverse algorithms cannot be optimized further). In this case, there is little to do, and checking the isospeed requirements only adds a modicum of overhead;
- the heuristic that selects the fastest between the direct and inverse by-sequence convolution (by counting horizontal segments and discontinuities) fails to identify the correct choice. This happens mostly when operands are staircase curves, hence have an equal number of horizontal segments and discontinuities.

We ran our 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.

Note that, since we measure runtime, the raw results are *expected* to vary, as they depend on the hardware setup. The comparison between different methods, instead, should match the observations in [3].

3 To reproduce the results

3.1 Requirements

Compiling and running this code requires .NET 6.0 (`https://dotnet.microsoft.com/en-us/download`), as well as network connectivity to download the other dependencies during compilation.

Table 1 The .tikz files produced, and their corresponding figure in [3].

TikZ file	Figure
IsoConvolutionHorizontalStaircaseBenchmarks-minp-iso.tikz	Figure 6a
IsoConvolutionHorizontalStaircaseBenchmarks-maxp-iso.tikz	Figure 6b
IsoConvolutionHorizontalStaircaseBenchmarks-best-iso.tikz	Figure 6c
IsoConvolutionVerticalStaircaseBenchmarks-minp-iso.tikz	Figure 7a
IsoConvolutionVerticalStaircaseBenchmarks-maxp-iso.tikz	Figure 7b
IsoConvolutionVerticalStaircaseBenchmarks-best-iso.tikz	Figure 7c
IsoConvolutionBalancedStaircaseBenchmarks-minp-iso.tikz	Figure 8a
IsoConvolutionBalancedStaircaseBenchmarks-maxp-iso.tikz	Figure 8b
IsoConvolutionBalancedStaircaseBenchmarks-best-iso.tikz	Figure 8c
IsoConvolutionHorizontalKTradeoffStaircaseBenchmarks-minp-iso.tikz	Figure 9a
IsoConvolutionHorizontalKTradeoffStaircaseBenchmarks-maxp-iso.tikz	Figure 9b
IsoConvolutionHorizontalKTradeoffStaircaseBenchmarks-best-iso.tikz	Figure 9c

3.2 Run the experiment

```
dotnet build -c Release
dotnet run -c Release --project ./isospeed-convolution-benchmarks/
    isospeed-convolution-benchmarks.csproj -- --filter "*"
```

3.3 Gather the results

The benchmark results can be then found in `BenchmarkDotNet.Artifacts/results`, in .md, .html, and .csv formats.

The `Method` column will report whether direct, inverse or isospeed was used. The `Pair` column will contain (as C# code) the parameters of the operands. The `Mean` column will contain the runtime of a single (min,+) convolution.

Grouping these results by `Pair` and comparing the runtime for each method, one should observe similar runtime improvements as shown in the paper.

3.4 Automated plots

To produce the plots in the paper, we used the C# program `isospeed-bench-to-tikz`, that parses the .csv files and generates TikZ plots with our desired layout.

```
dotnet run -c Release --project ./isospeed-bench-to-tikz/isospeed-bench-
    to-tikz.csproj
```

The program will leave the .tikz files produced in `BenchmarkDotNet.Artifacts/results`. Table 1 shows the tikz files produced and the corresponding figures in [3].

4 Expected runtime and tweaks

The parameters of this benchmark suite are found in the `Globals` static class, in `Program.cs` starting at line 142. This suite runs for about 24 hours on our hardware configuration.

For a smaller functional test, one may reduce the number of test run (`TEST_COUNT`) or (heuristically) reduce the size of the computation (`RNG_MAX`, `LARGE_EXTENSION_LCM_THRESHOLD`).

5 Documentation

The documentation of the Nancy library [4] can be found at `https://nancy.unipi.it`, which includes tutorials and example of use. Note that the above will be about the public version of the library – which we will soon update with the algorithms discussed here.

6 License

The artifact is available under license MIT.

7 MD5 sum of the artifact

97e476a410907eaa75027f18af97f806

8 Size of the artifact

592 KB

References

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

2 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.

3 Raffaele Zippo, Paul Nikolaus, and Giovanni Stea. Isospeed: Improving (min,+) Convolution by Exploiting (min,+)/(max,+) Isomorphism. In *35th Euromicro Conference on Real-Time Systems (ECRTS 2023)*, volume 262 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 12:1–12:24. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. `doi:10.4230/LIPIcs.ECRTS.2023.12`.

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