The Journal of Open Source Software

OMEinsumContractionOrders: A Julia package for tensor network contraction order optimization

Jin-Guo Liu[1*], **Xuanzhao Gao**[2*], and **Richard Samuelson**[3*]

1 Hong Kong University of Science and Technology (Guangzhou) **2** Center of Computational Mathematics, Flatiron Institute **3** ***** These authors contributed equally.

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Open Journals

Reviewers:

- @openjournals

Submitted: 01 January 1970
Published: unpublished

Summary

`OMEinsumContractionOrders` (One More Einsum Contraction Orders, or OMECO) is a Julia package (Bezanson et al., 2012) that implements state-of-the-art algorithms for optimizing tensor network contraction orders. OMECO is designed to search for near-optimal contraction orders for exact tensor network contraction, and provides a comprehensive suite of optimization algorithms for tensor network contraction orders, including greedy heuristics, simulated annealing, and tree width solvers. In this paper, we present the key features of OMECO, its integration with the Julia ecosystem, and performance benchmarks.

Statement of need

A *tensor network* is a mathematical framework that represents multilinear algebra operations as graphical structures, where tensors are nodes and shared indices are edges. This diagrammatic approach transforms complex high-dimensional contractions into visual networks that expose underlying computational structure.

The framework has remarkable universality across diverse domains: *einsum* notation (Harris et al., 2020) in numerical computing, *factor graphs* (Bishop & Nasrabadi, 2006) in probabilistic inference, *sum-product networks* in machine learning, and *junction trees* (Villescas et al., 2023) in graphical models. Tensor networks have enabled breakthroughs in quantum circuit simulation (Markov & Shi, 2008), quantum error correction (Piveteau et al., 2024), neural network compression (Qing et al., 2024), strongly correlated quantum materials (Haegeman et al., 2016), and combinatorial optimization problems (J.-G. Liu et al., 2023).

The computational cost of tensor network contraction depends critically on the *contraction order*—the sequence in which pairwise tensor multiplications are performed. This order can be represented as a binary tree where leaves correspond to input tensors and internal nodes represent intermediate results. The optimization objective balances multiple complexity measures through the cost function:

$$\mathcal{L} = w_{\mathsf{t}} \cdot \mathsf{tc} + w_{\mathsf{s}} \cdot \max(0, \mathsf{sc} - \mathsf{sc}_{\text{target}}) + w_{\mathsf{rw}} \cdot \mathsf{rwc},$$

where w_{t}, w_{s}, and w_{rw} represent weights for time complexity (tc), space complexity (sc), and read-write complexity (rwc), respectively. In practice, memory access costs typically dominate computational costs, motivating $w_{\mathsf{rw}} > w_{\mathsf{t}}$. The space complexity penalty activates only when $\mathsf{sc} > \mathsf{sc}_{\text{target}}$, allowing unconstrained optimization when memory fits within available device capacity.

Finding the optimal contraction order—even when minimizing only time complexity—is NP-complete (Markov & Shi, 2008). Algorithms for finding near-optimal contraction orders have been developed and achieve impressive scalability (Gray & Kourtis, 2021; Roa-Villescas et

The Journal of Open Source Software

al., 2024), handling tensor networks with over 10^4 tensors. However, an efficient and reliable implementation of these methods in Julia is still missing.

OMECO addresses this gap by offering a unified and extensible interface to a comprehensive suite of optimization algorithms for tensor network contraction orders, including greedy heuristics, simulated annealing, and tree-width-based solvers. OMECO has been integrated into the `OMEinsum` package and powers several downstream applications: `Yao` (Luo et al., 2020) for quantum circuit simulation, `GenericTensorNetworks` (J.-G. Liu et al., 2023) and `TensorBranching` for combinatorial optimization, `TensorInference` (Roa-Villescas & Liu, 2023) for probabilistic inference, and `TensorQEC` for quantum error correction. This infrastructure is expected to benefit other applications requiring tree or path decomposition, such as polynomial optimization (Magron & Wang, 2021). These applications are reflected in the ecosystem built around OMECO, as illustrated in Figure 1. This infrastructure is expected to benefit other applications requiring tree or path decomposition, such as polynomial optimization (Magron & Wang, 2021).

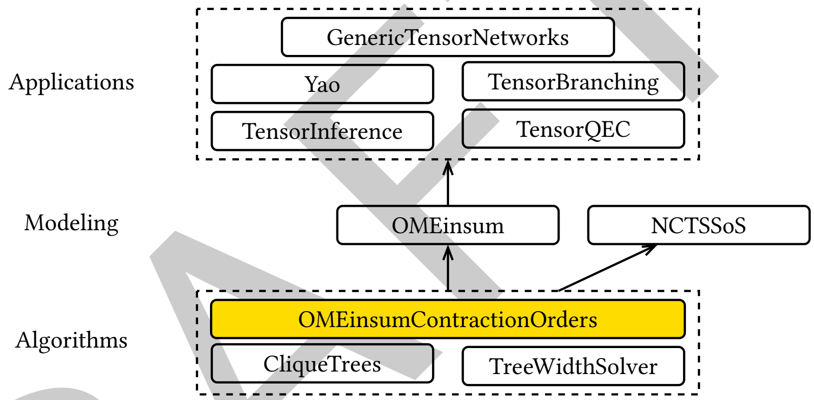

Figure 1: The ecosystem built around `OMEinsumContractionOrders` and its dependencies. OMECO serves as a core component of the tensor network contractor `OMEinsum`, which powers applications including `Yao` (quantum simulation), `TensorQEC` (quantum error correction), `TensorInference` (probabilistic inference), `GenericTensorNetworks and TensorBranching` (combinatorial optimization).

Features and benchmarks

The major feature of OMECO is contraction order optimization. OMECO provides several algorithms with complementary performance characteristics that can be simply called by the `optimize_code` function:

Optimizer	Description
`GreedyMethod`	Fast greedy heuristic with modest solution quality
`TreeSA`	Reliable simulated annealing optimizer (Kalachev et al., 2021) with high-quality solutions
`PathSA`	Simulated annealing optimizer for path decomposition
`HyperND`	Nested dissection algorithm for hypergraphs, requires `KaHyPar` or `Metis`
`KaHyParBipartite`	Graph bipartition method for large tensor networks (Gray & Kourtis, 2021), requires `KaHyPar`

Optimizer	Description
`SABipartite`	Simulated annealing bipartition method, pure Julia implementation
`ExactTreewidth`	Exact algorithm with exponential runtime (Bouchitté & Todinca, 2001), based on `TreeWidthSolver`
`Treewidth`	Clique tree elimination methods from `CliqueTrees` package (Samuelson & Fairbanks, 2025)

The algorithms `HyperND`, `Treewidth`, and `ExactTreewidth` operate on the tensor network's line graph and utilize the `CliqueTrees` and `TreeWidthSolver` packages, as illustrated in Figure 1. Additionally, the `PathSA` optimizer implements path decomposition by constraining contraction orders to path graphs, serving as a variant of `TreeSA`.

These methods balance optimization time against solution quality. Figure 2 displays benchmark results for the tensor network of the Sycamore quantum circuit(Arute et al., 2019; Pan & Zhang, 2021) that widely used as a benchmark for quantum supremacy, which is believed to have an optimal space complexity of 52. The Pareto front highlights the optimal trade-off between optimization time and solution quality.

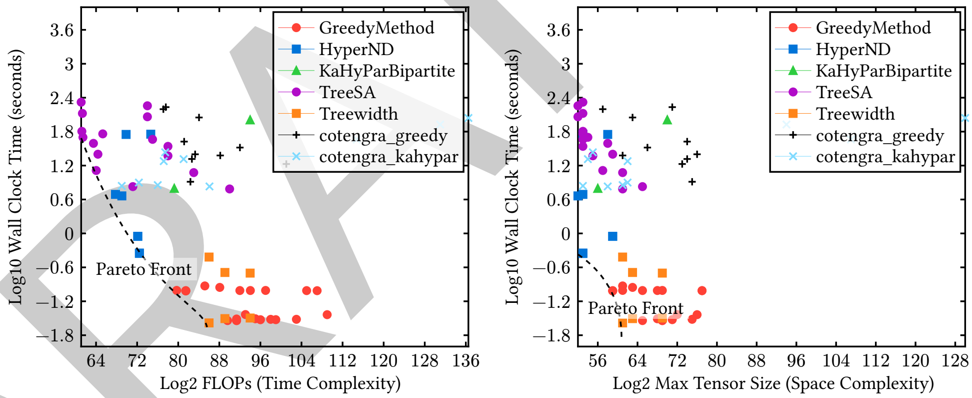

Figure 2: Time complexity (left) and space complexity (right) benchmark results for contraction order optimization on the Sycamore quantum circuit tensor network (Intel Xeon Gold 6226R CPU @ 2.90GHz, single-threaded). The x-axis shows contraction cost, y-axis shows optimization time. Each point represents a different optimizer configuration tested with varying parameters. `TreeSA` and `HyperND` achieve the lowest contraction costs, while `GreedyMethod` offers the fastest optimization time. The parameter setup for each optimizer is detailed in our benchmark repository `OMEinsumContractionOrdersBenchmark`.

Optimizers prefixed with `cotengra_` are from the Python package cotengra (Gray & Kourtis, 2021); all others are OMECO implementations. For both optimization objectives (minimizing time and space complexity), OMECO optimizers dominate the Pareto front. Given sufficient optimization time, `TreeSA` consistently achieves the lowest time and space complexity. `GreedyMethod` and `Treewidth` (backed by minimum fill (MF) (Ng & Peyton, 2014), multiple minimum degree (MMD) (J. W. Liu, 1985), and approximate minimum fill (AMF) (Rothberg & Eisenstat, 1998)) provides the fastest optimization but yields suboptimal contraction orders, while `HyperND` offers a favorable balance between optimization time and solution quality.

More real-world examples demonstrating applications to quantum circuit simulation, combinatorial optimization, and probabilistic inference are available in the OMEinsumContractionOrdersBenchmark repository. We find that optimizer performance is highly problem-dependent, with no single algorithm dominating across all metrics and graph topologies.

Another key feature of OMECO is index slicing, a technique that trades time complexity for reduced space complexity by explicitly looping over a subset of tensor indices. OMECO provides the `slice_code` interface for this purpose, currently supporting the `TreeSASlicer` algorithm, which implements dynamic slicing based on the `TreeSA` optimizer. Figure 3 demonstrates this capability using the Sycamore quantum circuit, where slicing reduces the space complexity from 2^{52} to 2^{31}.

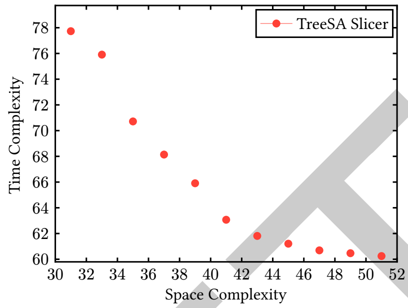

Figure 3: Trade-off between time complexity and target space complexity using `TreeSASlicer` on the Sycamore quantum circuit. The original network has a space complexity of 2^{52}.

The numerical experiments show that moderate slicing increases time complexity only slightly, while aggressive slicing can induce significant overhead. There seems to be a critical point at around 42 where the time complexity starts to increase significantly.

References

Arute, F., Arya, K., Babbush, R., Bacon, D., Bardin, J. C., Barends, R., Biswas, R., Boixo, S., Brandao, F. G., Buell, D. A., & others. (2019). Quantum supremacy using a programmable superconducting processor. *Nature*, *574*(7779), 505–510.

Bezanson, J., Karpinski, S., Shah, V. B., & Edelman, A. (2012). Julia: A fast dynamic language for technical computing. *arXiv:1209.5145 [Cs]*. https://doi.org/10.48550/arXiv.1209.5145

Bishop, C. M., & Nasrabadi, N. M. (2006). *Pattern recognition and machine learning* (Vol. 4). Springer.

Bouchitté, V., & Todinca, I. (2001). Treewidth and minimum fill-in: Grouping the minimal separators. *SIAM Journal on Computing*, *31*(1), 212–232.

Gray, J., & Kourtis, S. (2021). Hyper-optimized tensor network contraction. *Quantum*, *5*, 410. https://doi.org/10.22331/q-2021-03-15-410

Haegeman, J., Lubich, C., Oseledets, I., Vandereycken, B., & Verstraete, F. (2016). Unifying time evolution and optimization with matrix product states. *Physical Review B*. https://doi.org/10.1103/PhysRevB.94.165116

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., … Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

Kalachev, G., Panteleev, P., & Yung, M.-H. (2021). *Recursive multi-tensor contraction for XEB verification of quantum circuits*. https://arxiv.org/abs/2108.05665

Liu, J. W. (1985). Modification of the minimum-degree algorithm by multiple elimination. *ACM Transactions on Mathematical Software (TOMS)*, *11*(2), 141–153.

Liu, J.-G., Gao, X., Cain, M., Lukin, M. D., & Wang, S.-T. (2023). Computing solution space properties of combinatorial optimization problems via generic tensor networks. *SIAM Journal on Scientific Computing*, *45*(3), A1239–A1270.

Luo, X.-Z., Liu, J.-G., Zhang, P., & Wang, L. (2020). Yao. Jl: Extensible, efficient framework for quantum algorithm design. *Quantum*, *4*, 341.

Magron, V., & Wang, J. (2021). TSSOS: A julia library to exploit sparsity for large-scale polynomial optimization. *arXiv:2103.00915*.

Markov, I. L., & Shi, Y. (2008). Simulating Quantum Computation by Contracting Tensor Networks. *SIAM Journal on Computing*, *38*(3), 963–981. https://doi.org/10.1137/050644756

Ng, E. G., & Peyton, B. W. (2014). Fast implementation of the minimum local fill ordering heuristic. *CSC14: The Sixth SIAM Workshop on Combinatorial Scientific Computing*, 4.

Pan, F., & Zhang, P. (2021). *Simulating the sycamore quantum supremacy circuits*. https://arxiv.org/abs/2103.03074

Piveteau, C., Chubb, C. T., & Renes, J. M. (2024). Tensor-Network Decoding Beyond 2D. *PRX Quantum*, *5*(4), 040303. https://doi.org/10.1103/PRXQuantum.5.040303

Qing, Y., Li, K., Zhou, P.-F., & Ran, S.-J. (2024). *Compressing neural network by tensor network with exponentially fewer variational parameters* (No. arXiv:2305.06058). arXiv. https://doi.org/10.48550/arXiv.2305.06058

Roa-Villescas, M., Gao, X., Stuijk, S., Corporaal, H., & Liu, J.-G. (2024). Probabilistic Inference in the Era of Tensor Networks and Differential Programming. *Physical Review Research*, *6*(3), 033261. https://doi.org/10.1103/PhysRevResearch.6.033261

Roa-Villescas, M., & Liu, J.-G. (2023). TensorInference: A julia package for tensor-based probabilistic inference. *Journal of Open Source Software*, *8*(90), 5700.

Rothberg, E., & Eisenstat, S. C. (1998). Node selection strategies for bottom-up sparse matrix ordering. *SIAM Journal on Matrix Analysis and Applications*, *19*(3), 682–695.

Samuelson, R., & Fairbanks, J. (2025). *CliqueTrees.jl: A julia library for computing tree decompositions and chordal completions of graphs*. https://github.com/AlgebraicJulia/CliqueTrees.jl

Villescas, M. R., Liu, J.-G., Wijnings, P. W. A., Stuijk, S., & Corporaal, H. (2023). Scaling Probabilistic Inference Through Message Contraction Optimization. *2023 Congress in Computer Science, Computer Engineering, & Applied Computing (CSCE)*, 123–130. https://doi.org/10.1109/CSCE60160.2023.00025