

¹ OMEinsumContractionOrders: A Julia package for tensor network contraction order optimization

³ Jin-Guo Liu  ^{1*}, Xuanzhao Gao^{2*}, and Richard Samuelson^{3*}

⁴ 1 Hong Kong University of Science and Technology (Guangzhou) 2 Center of Computational
⁵ Mathematics, Flatiron Institute 3 * These authors contributed equally.

DOI: [10.xxxxxx/draft](https://doi.org/10.xxxxxx/draft)

Software

- [Review](#) 
- [Repository](#) 
- [Archive](#) 

Editor: [Open Journals](#) 

Reviewers:

- [@openjournals](#)

Submitted: 01 January 1970

Published: unpublished

License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License ([CC BY 4.0](#))

⁶ Summary

⁷ 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 ⁸ tensor network contraction orders. OMECO is designed to search for near-optimal contraction ⁹ orders for exact tensor network contraction, and provides a comprehensive suite of optimization ¹⁰ algorithms for tensor network contraction orders, including greedy heuristics, simulated ¹¹ annealing, and tree width solvers. In this paper, we present the key features of OMECO, its ¹² integration with the Julia ecosystem, and performance benchmarks.

¹⁴ 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.

¹⁹ 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* ([Villegas 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](#)).

²⁶ The computational cost of tensor network contraction depends critically on the *contraction order*—the sequence in which pairwise tensor multiplications are performed. This order ²⁷ can be represented as a binary tree where leaves correspond to input tensors and internal ²⁸ nodes represent intermediate results. The optimization objective balances multiple complexity ²⁹ measures through the cost function:

$$\mathcal{L} = w_t \cdot tc + w_s \cdot \max(0, sc - sc_{target}) + w_{rw} \cdot rwc,$$

³¹ where w_t , w_s , and w_{rw} represent weights for time complexity (tc), space complexity (sc), and ³² read-write complexity (rwc), respectively. In practice, memory access costs typically dominate ³³ computational costs, motivating $w_{rw} > w_t$. The space complexity penalty activates only when ³⁴ $sc > sc_{target}$, allowing unconstrained optimization when memory fits within available device ³⁵ capacity.

³⁶ Finding the optimal contraction order—even when minimizing only time complexity—is NP-³⁷ complete ([Markov & Shi, 2008](#)). Algorithms for finding near-optimal contraction orders have ³⁸ been developed and achieve impressive scalability ([Gray & Kourtis, 2021](#); [Roa-Villegas et](#)

³⁹ al., 2024), handling tensor networks with over 10^4 tensors. However, an efficient and reliable
⁴⁰ implementation of these methods in Julia is still missing.

⁴¹ OMECO addresses this gap by offering a unified and extensible interface to a comprehensive
⁴² suite of optimization algorithms for tensor network contraction orders, including greedy heuris-
⁴³ tics, simulated annealing, and tree-width-based solvers. OMECO has been integrated into the
⁴⁴ OMEinsum package and powers several downstream applications: Yao (Luo et al., 2020) for quan-
⁴⁵ tum circuit simulation, GenericTensorNetworks (J.-G. Liu et al., 2023) and TensorBranching
⁴⁶ for combinatorial optimization, TensorInference (Roa-Villegas & Liu, 2023) for probabilistic
⁴⁷ inference, and TensorQEC for quantum error correction. These applications are reflected in the
⁴⁸ ecosystem built around OMECO, as illustrated in Figure 1. This infrastructure is expected to
⁴⁹ benefit other applications requiring tree or path decomposition, such as polynomial optimization
⁵⁰ (Magron & Wang, 2021).

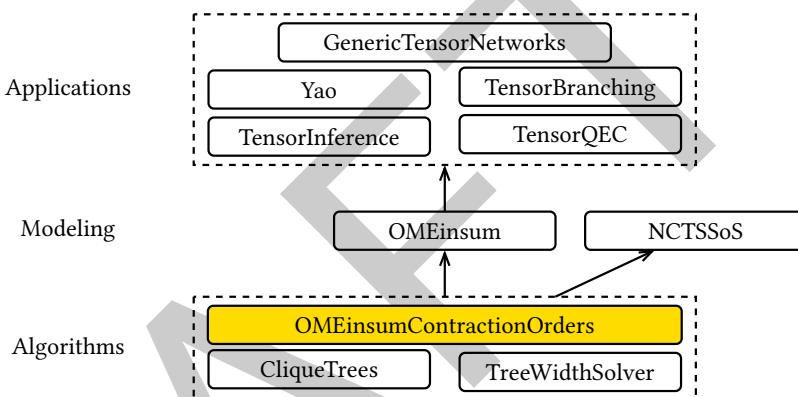


Figure 1: The ecosystem built around OMEinsumContractionOrders and its dependencies. OMECO serves as a core component of the tensor network contractor OMEinsum, which powers applications including Yao (quantum simulation), TensorQEC (quantum error correction), TensorInference (probabilistic inference),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

51 Features and benchmarks

⁵² The major feature of OMECO is contraction order optimization. OMECO provides several
⁵³ algorithms with complementary performance characteristics that can be simply called by the
⁵⁴ 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
SABipartite	Simulated annealing bipartition method, pure Julia implementation

Optimizer	Description
ExactTreewidth	Exact algorithm with exponential runtime (Bouchitté & Todinca, 2001), based on TreeWidthSolver
Treewidth	Clique tree elimination methods from CliqueTrees package (Samuelson & Fairbanks, 2025)

55 The algorithms HyperND, Treewidth, and ExactTreewidth operate on the tensor network's line
 56 graph and utilize the [CliqueTrees](#) and [TreeWidthSolver](#) packages, as illustrated in [Figure 1](#).
 57 Additionally, the PathSA optimizer implements path decomposition by constraining contraction
 58 orders to path graphs, serving as a variant of TreesA.
 59 These methods balance optimization time against solution quality. [Figure 2](#) displays benchmark
 60 results for the Sycamore quantum supremacy circuit([Arute et al., 2019](#); [Pan & Zhang, 2021](#)),
 61 highlighting the Pareto front where contraction order quality is balanced with optimization
 62 runtime. Real-world examples demonstrating applications to quantum circuit simulation,
 63 combinatorial optimization, and probabilistic inference are available in the [OMEinsumContractionOrdersBenchmark](#) repository. Optimizer performance is highly problem-dependent, with no
 64 single algorithm dominating across all metrics and graph topologies.

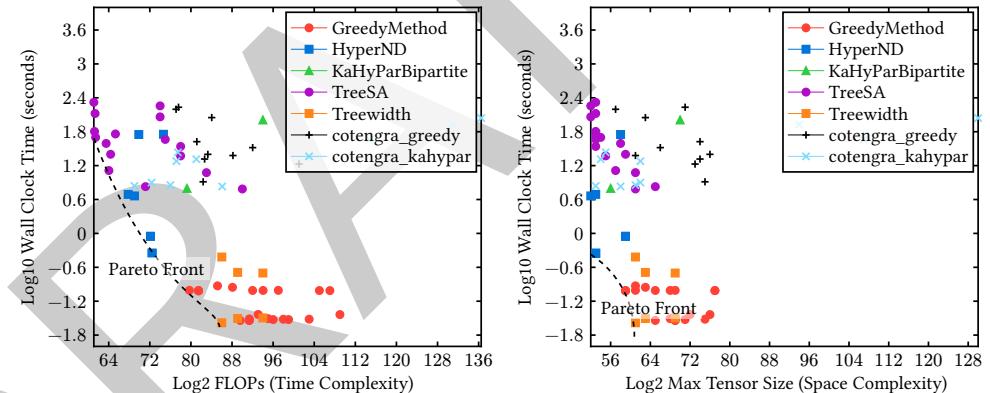


Figure 2: Time complexity (left) and space complexity (right) benchmark results for contraction order optimization on the Sycamore quantum circuit tensor network (Intel Xeon Gold 6226R CPU @ 2.90GHz, single-threaded). The x -axis shows contraction cost, y -axis shows optimization time. Each point represents 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](#).

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

72 OMECO has been integrated into the `OMEinsum` package and powers several downstream
 73 applications: Yao ([Luo et al., 2020](#)) for quantum circuit simulation, `GenericTensorNetworks`
 74 ([J.-G. Liu et al., 2023](#)) and `TensorBranching` for combinatorial optimization, `TensorInference`
 75 ([Roa-Villescas & Liu, 2023](#)) for probabilistic inference, and `TensorQEC` for quantum error

⁷⁸ correction. This infrastructure is expected to benefit other applications requiring tree or path
⁷⁹ decomposition, such as polynomial optimization ([Magron & Wang, 2021](#)).

⁸⁰ Another key feature of OMECO is index slicing, a technique that trades time complexity for
⁸¹ reduced space complexity by explicitly looping over a subset of tensor indices. OMECO provides
⁸² the `slice_code` interface for this purpose, currently supporting the TreeSASlicer algorithm,
⁸³ which implements dynamic slicing based on the TreeSA optimizer. [Figure 3](#) demonstrates this
⁸⁴ capability using the Sycamore quantum circuit, where slicing reduces the space complexity
⁸⁵ from 2^{52} to 2^{31} .

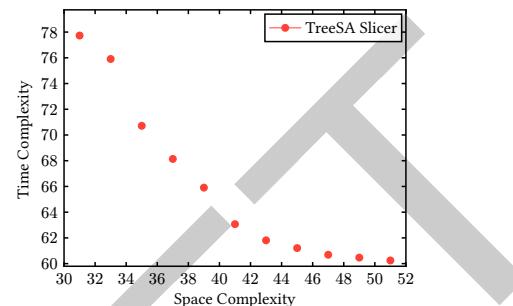


Figure 3: Trade-off between time complexity and target space complexity using TreeSASlicer on the Sycamore quantum circuit. The original network has a space complexity of 2^{52} .

⁸⁶ The figure shows that moderate slicing increases time complexity only slightly, while aggressive
⁸⁷ slicing can induce significant overhead.

References

- ⁸⁹ Arute, F., Arya, K., Babbush, R., Bacon, D., Bardin, J. C., Barends, R., Biswas, R., Boixo, S.,
⁹⁰ Brandao, F. G., Buell, D. A., & others. (2019). Quantum supremacy using a programmable
⁹¹ superconducting processor. *Nature*, *574*(7779), 505–510.
- ⁹² Bezanson, J., Karpinski, S., Shah, V. B., & Edelman, A. (2012). Julia: A fast dynamic language
⁹³ for technical computing. *arXiv:1209.5145 [Cs]*. <https://doi.org/10.48550/arXiv.1209.5145>
- ⁹⁴ Bishop, C. M., & Nasrabadi, N. M. (2006). *Pattern recognition and machine learning* (Vol.
⁹⁵ 4). Springer.
- ⁹⁶ Bouchitté, V., & Todinca, I. (2001). Treewidth and minimum fill-in: Grouping the minimal
⁹⁷ separators. *SIAM Journal on Computing*, *31*(1), 212–232.
- ⁹⁸ Gray, J., & Kourtis, S. (2021). Hyper-optimized tensor network contraction. *Quantum*, *5*, 410.
⁹⁹ <https://doi.org/10.22331/q-2021-03-15-410>
- ¹⁰⁰ Haegeman, J., Lubich, C., Oseledets, I., Vandereycken, B., & Verstraete, F. (2016). Unifying
¹⁰¹ time evolution and optimization with matrix product states. *Physical Review B*. <https://doi.org/10.1103/PhysRevB.94.165116>
- ¹⁰³ Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau,
¹⁰⁴ D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van
¹⁰⁵ Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., ...
¹⁰⁶ Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362.
¹⁰⁷ <https://doi.org/10.1038/s41586-020-2649-2>
- ¹⁰⁸ Kalachev, G., Panteleev, P., & Yung, M.-H. (2021). *Recursive multi-tensor contraction for
¹⁰⁹ XEB verification of quantum circuits*. <https://arxiv.org/abs/2108.05665>

- 110 Liu, J. W. (1985). Modification of the minimum-degree algorithm by multiple elimination.
111 *ACM Transactions on Mathematical Software (TOMS)*, 11(2), 141–153.
- 112 Liu, J.-G., Gao, X., Cain, M., Lukin, M. D., & Wang, S.-T. (2023). Computing solution
113 space properties of combinatorial optimization problems via generic tensor networks. *SIAM*
114 *Journal on Scientific Computing*, 45(3), A1239–A1270.
- 115 Luo, X.-Z., Liu, J.-G., Zhang, P., & Wang, L. (2020). Yao. Jl: Extensible, efficient framework
116 for quantum algorithm design. *Quantum*, 4, 341.
- 117 Magron, V., & Wang, J. (2021). TSSOS: A julia library to exploit sparsity for large-scale
118 polynomial optimization. *arXiv:2103.00915*.
- 119 Markov, I. L., & Shi, Y. (2008). Simulating Quantum Computation by Contracting Tensor
120 Networks. *SIAM Journal on Computing*, 38(3), 963–981. <https://doi.org/10.1137/050644756>
- 122 Ng, E. G., & Peyton, B. W. (2014). Fast implementation of the minimum local fill ordering
123 heuristic. *CSC14: The Sixth SIAM Workshop on Combinatorial Scientific Computing*, 4.
- 124 Pan, F., & Zhang, P. (2021). *Simulating the sycamore quantum supremacy circuits*. <https://arxiv.org/abs/2103.03074>
- 126 Piveteau, C., Chubb, C. T., & Renes, J. M. (2024). Tensor-Network Decoding Beyond 2D.
127 *PRX Quantum*, 5(4), 040303. <https://doi.org/10.1103/PRXQuantum.5.040303>
- 128 Qing, Y., Li, K., Zhou, P.-F., & Ran, S.-J. (2024). *Compressing neural network by tensor*
129 *network with exponentially fewer variational parameters* (No. arXiv:2305.06058). arXiv.
130 <https://doi.org/10.48550/arXiv.2305.06058>
- 131 Roa-Villegas, M., Gao, X., Stuijk, S., Corporaal, H., & Liu, J.-G. (2024). Probabilistic
132 Inference in the Era of Tensor Networks and Differential Programming. *Physical Review*
133 *Research*, 6(3), 033261. <https://doi.org/10.1103/PhysRevResearch.6.033261>
- 134 Roa-Villegas, M., & Liu, J.-G. (2023). TensorInference: A julia package for tensor-based
135 probabilistic inference. *Journal of Open Source Software*, 8(90), 5700.
- 136 Rothberg, E., & Eisenstat, S. C. (1998). Node selection strategies for bottom-up sparse matrix
137 ordering. *SIAM Journal on Matrix Analysis and Applications*, 19(3), 682–695.
- 138 Samuelson, R., & Fairbanks, J. (2025). *CliqueTrees.jl: A julia library for computing tree*
139 *decompositions and chordal completions of graphs*. <https://github.com/AlgebraicJulia/CliqueTrees.jl>
- 141 Villegas, M. R., Liu, J.-G., Wijnings, P. W. A., Stuijk, S., & Corporaal, H. (2023). Scaling
142 Probabilistic Inference Through Message Contraction Optimization. *2023 Congress in*
143 *Computer Science, Computer Engineering, & Applied Computing (CSCE)*, 123–130. <https://doi.org/10.1109/CSCE60160.2023.00025>