

# <sup>1</sup> OMEinsumContractionOrders: A Julia package for tensor network contraction order optimization

<sup>3</sup> Jin-Guo Liu  <sup>1\*</sup>, Xuanzhao Gao<sup>2\*</sup>, and Richard Samuelson<sup>3\*</sup>

<sup>4</sup> 1 Hong Kong University of Science and Technology (Guangzhou) 2 Center of Computational  
<sup>5</sup> 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Published: unpublished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](#))

## <sup>6</sup> Summary

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

## <sup>14</sup> 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.

<sup>19</sup> 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 ([Liu et al., 2023](#)).

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

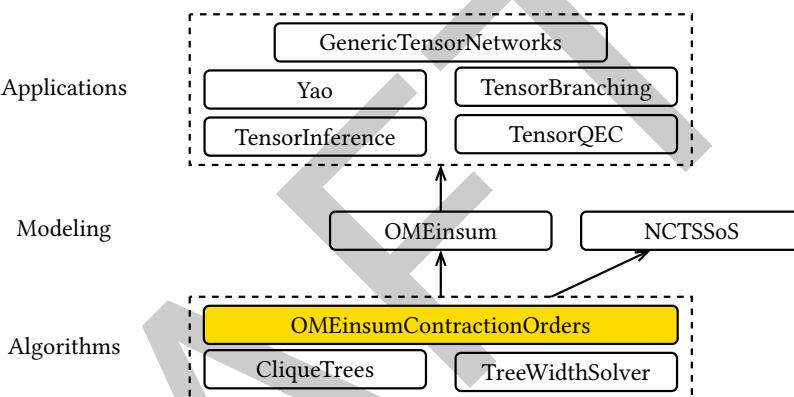
$$\mathcal{L} = w_t \cdot tc + w_s \cdot \max(0, sc - sc_{target}) + w_{rw} \cdot rwc,$$

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

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

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

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

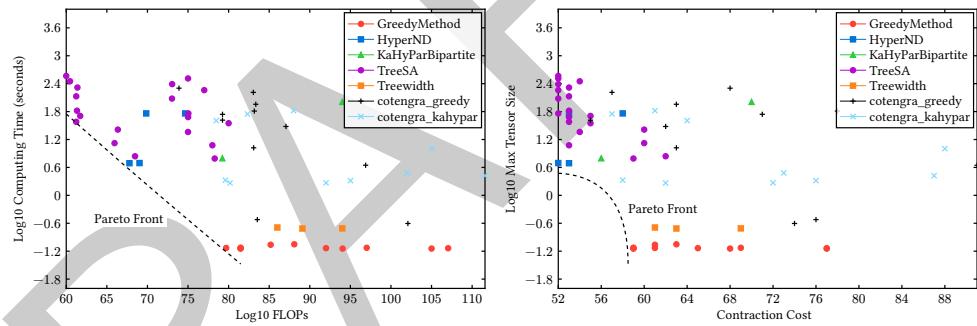
## 51 Features and benchmarks

<sup>52</sup> The major feature of OMECO is contraction order optimization. OMECO provides several  
<sup>53</sup> algorithms with complementary performance characteristics that can be simply called by the  
<sup>54</sup> 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

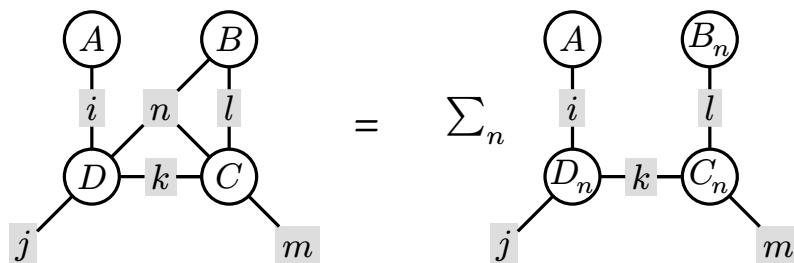
Optimizer	Description
ExactTreewidth	Exact algorithm with exponential runtime ( <a href="#">Bouchitté &amp; Todinca, 2001</a> ), based on TreeWidthSolver
Treewidth	Clique tree elimination methods from CliqueTrees package

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 bench-  
 60 mark results for the Sycamore quantum supremacy circuit, highlighting the Pareto front  
 61 where contraction order quality is balanced with optimization runtime. Real-world examples  
 62 demonstrating applications to quantum circuit simulation, combinatorial optimization, and  
 63 probabilistic inference are available in the [OMEinsumContractionOrdersBenchmark](#) repository.  
 64 Optimizer performance is highly problem-dependent, with no single algorithm dominating  
 65 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.

66 [JG: TODO: Please also cite CliqueTree paper.]  
 67 Optimizers prefixed with cotengra\_ are from the Python package cotengra ([Gray & Kourtis, 2021](#)); all others are OMECO implementations. For both optimization objectives (minimizing  
 68 time and space complexity), OMECO optimizers dominate the Pareto front. Given suffi-  
 69 cient optimization time, TreeSA consistently achieves the lowest time and space complexity.  
 70 GreedyMethod provides the fastest optimization but yields suboptimal contraction orders, while  
 71 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 ([Liu et al., 2023](#)) and TensorBranching (TODO: add citation) for combinatorial optimization,  
 75 TensorInference ([Roa-Villegas & Liu, 2023](#)) for probabilistic inference, and TensorQEC for  
 76 quantum error correction. This infrastructure is expected to benefit other applications requiring  
 77 tree or path decomposition, such as polynomial optimization ([Magron & Wang, 2021](#)).



**Figure 3:** Illustration of index slicing. Looping over an index, such as  $n$  in the figure, decomposes a large tensor network contraction into a series of smaller ones.

79 The second feature of OMECO is index slicing, which is a technique to trade time complexity  
 80 for reduced space complexity by looping over a subset of indices directly, as shown in [Figure 3](#).  
 81 In OMECO, the interface to perform index slicing is `slice_code`, and currently we only support  
 82 one Slicer, TreeSASlicer, which is implemented the dynamic slicing algorithm based on  
 83 TreeSA.

## 84 Acknowledgments

85 This work was partially funded by Google Summer of Code 2024 and the Open Source Promotion  
 86 Plan (OSPP 2023). We thank Feng Pan for insightful discussions and code contributions on  
 87 the slicing technique.

## 88 References

- 89 Bezanson, J., Karpinski, S., Shah, V. B., & Edelman, A. (2012). Julia: A fast dynamic language  
 90 for technical computing. *arXiv:1209.5145 [Cs]*. <https://doi.org/10.48550/arXiv.1209.5145>
- 91 Bishop, C. M., & Nasrabadi, N. M. (2006). *Pattern recognition and machine learning* (Vol.  
 92 4). Springer.
- 93 Bouchitté, V., & Todinca, I. (2001). Treewidth and minimum fill-in: Grouping the minimal  
 94 separators. *SIAM Journal on Computing*, 31(1), 212–232.
- 95 Gray, J., & Kourtis, S. (2021). Hyper-optimized tensor network contraction. *Quantum*, 5, 410.  
 96 <https://doi.org/10.22331/q-2021-03-15-410>
- 97 Haegeman, J., Lubich, C., Oseledets, I., Vandereycken, B., & Verstraete, F. (2016). Unifying  
 98 time evolution and optimization with matrix product states. *Physical Review B*. <https://doi.org/10.1103/PhysRevB.94.165116>
- 100 Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau,  
 101 D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van  
 102 Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., ...  
 103 Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, 585(7825), 357–362.  
 104 <https://doi.org/10.1038/s41586-020-2649-2>
- 105 Kalachev, G., Panteleev, P., & Yung, M.-H. (2021). *Recursive multi-tensor contraction for  
 106 XEB verification of quantum circuits*. <https://arxiv.org/abs/2108.05665>
- 107 Liu, J.-G., Gao, X., Cain, M., Lukin, M. D., & Wang, S.-T. (2023). Computing solution  
 108 space properties of combinatorial optimization problems via generic tensor networks. *SIAM  
 109 Journal on Scientific Computing*, 45(3), A1239–A1270.
- 110 Luo, X.-Z., Liu, J.-G., Zhang, P., & Wang, L. (2020). Yao. Jl: Extensible, efficient framework

- 111 for quantum algorithm design. *Quantum*, 4, 341.
- 112 Magron, V., & Wang, J. (2021). TSSOS: A julia library to exploit sparsity for large-scale  
113 polynomial optimization. [arXiv:2103.00915](https://arxiv.org/abs/2103.00915).
- 114 Markov, I. L., & Shi, Y. (2008). Simulating Quantum Computation by Contracting Tensor  
115 Networks. *SIAM Journal on Computing*, 38(3), 963–981. <https://doi.org/10.1137/050644756>
- 117 Piveteau, C., Chubb, C. T., & Renes, J. M. (2024). Tensor-Network Decoding Beyond 2D.  
118 *PRX Quantum*, 5(4), 040303. <https://doi.org/10.1103/PRXQuantum.5.040303>
- 119 Qing, Y., Li, K., Zhou, P.-F., & Ran, S.-J. (2024). *Compressing neural network by tensor  
120 network with exponentially fewer variational parameters* (No. arXiv:2305.06058). arXiv.  
121 <https://doi.org/10.48550/arXiv.2305.06058>
- 122 Roa-Villegas, M., Gao, X., Stuijk, S., Corporaal, H., & Liu, J.-G. (2024). Probabilistic  
123 Inference in the Era of Tensor Networks and Differential Programming. *Physical Review  
124 Research*, 6(3), 033261. <https://doi.org/10.1103/PhysRevResearch.6.033261>
- 125 Roa-Villegas, M., & Liu, J.-G. (2023). TensorInference: A julia package for tensor-based  
126 probabilistic inference. *Journal of Open Source Software*, 8(90), 5700.
- 127 Villegas, M. R., Liu, J.-G., Wijning, P. W. A., Stuijk, S., & Corporaal, H. (2023). Scaling  
128 Probabilistic Inference Through Message Contraction Optimization. *2023 Congress in  
129 Computer Science, Computer Engineering, & Applied Computing (CSCE)*, 123–130. <https://doi.org/10.1109/CSCE60160.2023.00025>