The Journal of Open Source Software

TensorInference: A Julia package for tensor-based probabilistic inference

Martin Roa-Villescas 1* and **Jin-Guo Liu** 2*

1 Eindhoven University of Technology **2** Hong Kong University of Science and Technology (Guangzhou)
***** These authors contributed equally.

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Open Journals
Reviewers:

- @openjournals

Submitted: 01 January 1970
Published: unpublished

Statement of need

A major challenge in developing intelligent systems is the ability to reason under uncertainty, a challenge that appears in many real-world problems across various domains, including artificial intelligence, medical diagnosis, computer vision, computational biology, and natural language processing. Reasoning under uncertainty involves calculating the probabilities of relevant variables while taking into account any information that is acquired. This process, which can be thought of as drawing global insights from local observations, is known as *probabilistic inference*.

Probabilistic graphical models (PGMs) provide a unified framework to perform probabilistic inference. These models use graphs to represent the joint probability distribution of complex systems in a concise manner by exploiting the conditional independence between variables in the model. Additionally, they form the foundation for various algorithms that enable efficient probabilistic inference.

However, even with the representational aid of PGMs, performing probabilistic inference remains an intractable endeavor on many real-world models. The reason is that performing probabilistic inference involves complex combinatorial optimization problems in very high dimensional spaces. To tackle these challenges, more efficient and scalable inference algorithms are needed.

As an attempt to tackle the aforementioned challenges, we present `TensorInference.jl`, a Julia package for probabilistic inference that combines the representational capabilities of PGMs with the computational power of tensor networks. By harnessing the best of both worlds, `TensorInference.jl` aims to enhance the performance of probabilistic inference, thereby expanding the tractability spectrum of exact inference for more complex, real-world models.

Summary

Probabilistic inference entails the process of drawing conclusions from observed data through the axioms of probability theory. Inference algorithms fall into two broad categories: *exact* and *approximate* methods. The main challenge in applying exact inference to real-world problems is its NP-hard computational complexity tied to the model's *treewidth*, a metric of network connectivity. This has prompted a research shift to approximate methods like *Markov chain Monte Carlo* and *variational* inference. Prominent examples of packages that implement such algorithms include `Stan` (Carpenter et al., 2017), `PyMC3` (Oriol et al., 2023), `Turing.jl` (Ge et al., 2018), and `RxInfer.jl` (Bagaev et al., 2023). However, while these methods offer superior scalability, they do not provide formal guarantees of accuracy — a challenge that is, in itself, NP-hard to address. Consequently, exact inference methods are gaining renewed interest for their promise of higher accuracy.

`TensorInference.jl` is a Julia (Bezanson et al., 2017) package designed for performing exact probabilistic inference in discrete graphical models. Capitalizing on the recent advances in the field of tensor networks (Orús, 2014, 2019; Robeva & Seigal, 2019), `TensorInference.jl` offers high-performance solutions for prevalent inference problems. Specifically, it provides methods to:

1. calculate the partition function (also known as the probability of evidence).
2. compute the marginal probability distribution over each variable given evidence.
3. find the most likely assignment to all variables given evidence.
4. find the most likely assignment to a set of query variables after marginalizing out the remaining variables.
5. draw samples from the posterior distribution given evidence (Cheng et al., 2019; Han et al., 2018).

A *tensor* is a mathematical object that generalizes scalars, vectors, and matrices to higher dimensions. In essence, it is a multi-dimensional array of numbers, often used for representing complex data structures in physics, engineering, computer science, and data analytics. A *tensor network* consists of a set of tensors in which some or all indices are contracted according to a specific pattern (Jutho et al., 2023). The term *contraction* refers to the summation over all the possible values along one or more dimensions of a set of tensors. These networks excel at capturing the correlations of different states in complex systems.

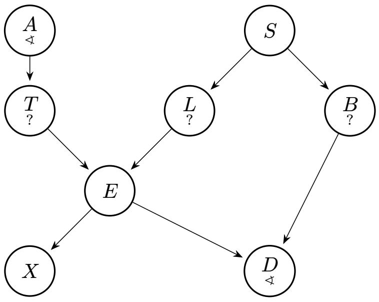
The order in which tensor indices are contracted plays a pivotal role in computational efficiency. Different contraction sequences can produce the same mathematical outcome, but the computational costs can vary by orders of magnitude. Since tensor network methods frequently involve multiple contractions, optimizing the contraction order becomes crucial.

The use of a tensor network-based infrastructure (Jutho et al., 2023) offers several advantages when dealing with complex computational tasks. Firstly, it simplifies the process of computing gradients by employing differentiable programming (Liao et al., 2019), a critical operation for the aforementioned inference tasks. Secondly, it supports generic element types without a significant compromise on performance. This feature enables the solution of a variety of problems using the same tensor network contraction algorithm, simply by varying the element types used. This allowed us to seamlessly implement solutions for several of the inference tasks described above (Liu et al., 2021, 2022). Thirdly, it allows users to define a hyper-optimized contraction order, which is known to have a significant impact on the computational performance of contracting tensor networks (Gao et al., 2021; Markov & Shi, 2008; Pan & Zhang, 2022). `TensorInference.jl` provides a predefined set of state-of-the-art contraction ordering methods, each identified by a specific name for ease of reference. These methods include a *local search-based method*, denoted as `TreeSA` (Kalachev et al., 2022), two methods based on *min-cut algorithms*, denoted as `SABipartite` and `KaHyParBipartite` (Gray & Kourtis, 2021); as well as a *greedy algorithm*, denoted as `GreedyMethod`. Lastly, `TensorInference.jl` leverages the cutting-edge developments commonly found in tensor network libraries, including a highly optimized set of BLAS routines (Blackford et al., 2002) and GPU technology.

`TensorInference.jl` succeeds `JunctionTrees.jl` (Roa-Villescas et al., 2022, 2023), a Julia package implementing the Junction Tree Algorithm (JTA) (Jensen et al., 1990; Lauritzen & Spiegelhalter, 1988). While the latter employs tensor-based technology to optimize the computation of individual sum-product messages within the JTA context, `TensorInference.jl` takes a different route. It adopts a holistic tensor network approach, which opens new doors for optimization opportunities and significantly reduces the algorithm's complexity compared to the JTA. Other prominent examples of exact inference packages for probabilistic inference include `libDAI` (Mooij, 2010), `Merlin` (Marinescu, 2022), and `toulbar2` (Hurley et al., 2016). For a performance comparison of `TensorInference.jl` against these alternatives, please see the Performance evaluation section in the documentation of `TensorInference.jl`.

Usage example

The graph below corresponds to the *ASIA network* (Lauritzen & Spiegelhalter, 1988), a simple Bayesian network (Pearl, 1985) used extensively in educational settings. It describes the probabilistic relationships between different random variables which correspond to possible diseases, symptoms, risk factors and test results.



Random variable	Meaning
A	Recent trip to Asia
T	Patient has tuberculosis
S	Patient is a smoker
L	Patient has lung cancer
B	Patient has bronchitis
E	Patient has T and/or L
X	Chest X-Ray is positive
D	Patient has dyspnoea

Figure 1: The ASIA network: a simplified example of a Bayesian network from the context of medical diagnosis (Lauritzen & Spiegelhalter, 1988).

In the example, a patient has recently visited Asia and is now experiencing dyspnea. These conditions serve as the evidence for the observed variables (A and D). The doctor's task is to assess the likelihood of various diseases — tuberculosis, lung cancer, and bronchitis — which constitute the query variables in this scenario (T, L, and B).

We now demonstrate how to use `TensorInference.jl` for conducting a variety of inference tasks on this toy example. Please note that as the API may evolve, we recommend checking the examples directory of the official `TensorInference.jl` repository for the most up-to-date version of this example.

```
# Import the TensorInference package, which provides the functionality needed
# for working with tensor networks and probabilistic graphical models.
In [1]: using TensorInference

# Load the ASIA network model from `asia.uai` in the examples directory.
# Refer to the package documentation for a description of the format of this file.
model = read_model_file(pkgdir(TensorInference, "examples", "asia", "asia.uai"))

# Create a tensor network representation of the loaded model.
tn = TensorNetworkModel(model)

Out [1]: TensorNetworkModel{Int64, OMEinsum.DynamicNestedEinsum{Int64}, Array{Float64}}
         variables: 1, 2, 3, 4, 5, 6, 7, 8
         contraction time = 2^6.044, space = 2^2.0, read-write = 2^7.098

# Calculate the partition function. Since the factors in this model are
# normalized, the partition function is the same as the total probability, 1.
In [2]: probability(tn) |> first

Out [2]: 1.0000000000000002

# Calculate the marginal probabilities of each random variable in the model.
In [3]: marginals(tn)
```

The Journal of Open Source Software

```
Out [3]: Dict{Vector{Int64}, Vector{Float64}} with 8 entries:
         [8] => [0.435971, 0.564029]
         [3] => [0.5, 0.5]
         [1] => [0.01, 0.99]
         [5] => [0.45, 0.55]
         [4] => [0.055, 0.945]
         [6] => [0.064828, 0.935172]
         [7] => [0.11029, 0.88971]
         [2] => [0.0104, 0.9896]

# Set the evidence to assume that the 'X-ray' result (variable 7) is negative.
# Recompute the contraction order of the tensor network, as setting the evidence
# may affect it.
In [4]: tn = TensorNetworkModel(model, evidence = Dict(7 => 0))

Out [4]: TensorNetworkModel{Int64, OMEinsum.DynamicNestedEinsum{Int64}, Array{Float64}}
         variables: 1, 2, 3, 4, 5, 6, 7 (evidence → 0), 8
         contraction time = 2^6.0, space = 2^2.0, read-write = 2^7.066

# Calculate the maximum log-probability among all configurations.
In [5]: maximum_logp(tn)

Out [5]: 0-dimensional Array{Float64, 0}:
         -3.6522217920023303

# Generate 10 samples from the posterior distribution.
In [6]: sample(tn, 10)

Out [6]: 10-element TensorInference.Samples{Int64}:
         [1, 1, 0, 0, 1, 0, 0, 0]
         [1, 1, 1, 1, 1, 1, 0, 1]
         [1, 0, 1, 1, 1, 0, 0, 0]
         [1, 1, 0, 1, 1, 1, 0, 1]
         [1, 1, 0, 0, 1, 0, 0, 0]
         [1, 1, 0, 0, 1, 0, 0, 0]
         [1, 1, 0, 1, 0, 1, 0, 0]
         [1, 1, 0, 0, 0, 0, 0, 0]
         [1, 1, 0, 0, 1, 0, 0, 1]
         [1, 1, 0, 1, 1, 1, 0, 0]

# Retrieve both the maximum log-probability and the most probable configuration
In [7]: logp, cfg = most_probable_config(tn)

Out [7]: (-3.6522217920023303, [1, 1, 0, 0, 0, 0, 0, 0])

# Compute the most probable values for a subset of variables (e.g., 4 and 7)
# while marginalizing over the others. This process is known as Maximum a
# Posteriori (MAP) estimation.
In [8]: mmap = MMAPModel(model, evidence=Dict(7=>0), queryvars=[4,7])

Out [8]: MMAPModel{Int64, Array{Float64}}
         variables: 4, 7 (evidence → 0)
         query variables: [[1, 2, 6, 5, 3, 8]]
         contraction time = 2^6.0, space = 2^2.0, read-write = 2^7.0

# Get the most probable configurations for variables 4 and 7.
In [9]: most_probable_config(mmap)

Out [9]: (-2.8754627318176693, [1, 0])
```

```
# Compute the total log-probability of having lung cancer. The results suggest
# that the probability is roughly half.
In [10]: log_probability(mmap, [1, 0]), log_probability(mmap, [0, 0])

Out [10]: (-2.8754627318176693, -2.920624801067186)
```

Acknowledgments

This work is partially funded by the Netherlands Organization for Scientific Research and the Guangzhou Municipal Science and Technology Project (No. 2023A03J0003). We extend our gratitude to Madelyn Cain and Patrick Wijnings for their insightful discussions on the intersection of tensor networks and probabilistic graphical models.

References

Bagaev, D., Podusenko, A., & Vries, B. de. (2023). RxInfer: A Julia package for reactive real-time Bayesian inference. *Journal of Open Source Software*, *8*(84), 5161. https://doi.org/10.21105/joss.05161

Bezanson, J., Edelman, A., Karpinski, S., & Shah, V. (2017). Julia: A fresh approach to numerical computing. *SIAM Review*, *59*(1), 65–98. https://doi.org/10.1137/141000671

Blackford, L. S., Petitet, A., Pozo, R., Remington, K., Whaley, R. C., Demmel, J., Dongarra, J., Duff, I., Hammarling, S., Henry, G., & others. (2002). An updated set of basic linear algebra subprograms (BLAS). *ACM Transactions on Mathematical Software*, *28*(2), 135–151. https://doi.org/10.1145/567806.567807

Carpenter, B., Gelman, A., Hoffman, M. D., Lee, D., Goodrich, B., Betancourt, M., Brubaker, M., Guo, J., Li, P., & Riddell, A. (2017). Stan: A probabilistic programming language. *Journal of Statistical Software*, *76*(1).

Cheng, S., Wang, L., Xiang, T., & Zhang, P. (2019). Tree tensor networks for generative modeling. *Physical Review B*, *99*(15), 155131. https://doi.org/10.1103/PhysRevB.99.155131

Gao, X., Kalinowski, M., Chou, C.-N., Lukin, M. D., Barak, B., & Choi, S. (2021). *Limitations of linear cross-entropy as a measure for quantum advantage*. https://doi.org/10.48550/arXiv.2112.01657

Ge, H., Xu, K., & Ghahramani, Z. (2018). Turing: A language for flexible probabilistic inference. *International Conference on Artificial Intelligence and Statistics, AISTATS 2018, 9-11 April 2018, Playa Blanca, Lanzarote, Canary Islands, Spain*, 1682–1690. http://proceedings.mlr.press/v84/ge18b.html

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

Han, Z.-Y., Wang, J., Fan, H., Wang, L., & Zhang, P. (2018). Unsupervised generative modeling using matrix product states. *Physical Review X*, *8*(3), 031012. https://doi.org/10.1103/PhysRevX.8.031012

Hurley, B., O'Sullivan, B., Allouche, D., Katsirelos, G., Schiex, T., Zytnicki, M., & Givry, S. de. (2016). Multi-language evaluation of exact solvers in graphical model discrete optimization. *Constraints*, *21*(3), 413–434. https://doi.org/10.1007/s10601-016-9245-y

Jensen, F. V., Lauritzen, S. L., & Olesen, K. G. (1990). Bayesian updating in causal probabilistic networks by local computations. *Computational Statistics Quarterly*, *4*, 269–282.

Jutho, Lukas, ho-oto, maartenvd, getzdan, Liu, J.-G., Aluthge, D., Florian, Lyon, S., Morley, A., Privett, A., Brann, D., Iouchtchenko, D., Saba, E., Otto, F., Garrison, J., Bhattacharya, J., Feist, J., TagBot, J., … jemiryguo. (2023). *Jutho/TensorOperations.jl: v4.0.0* (Version v4.0.0). Zenodo. https://doi.org/10.5281/zenodo.8166121

Kalachev, G., Panteleev, P., & Yung, M.-H. (2022). *Multi-tensor contraction for XEB verification of quantum circuits*. https://doi.org/10.48550/arXiv.2108.05665

Lauritzen, S. L., & Spiegelhalter, D. J. (1988). Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society: Series B (Methodological)*, *50*(2), 157–194. https://doi.org/10.1111/j.2517-6161.1988.tb01721.x

Liao, H.-J., Liu, J.-G., Wang, L., & Xiang, T. (2019). Differentiable programming tensor networks. *Physical Review X*, *9*(3), 031041. https://doi.org/10.1103/PhysRevX.9.031041

Liu, J.-G., Gao, X., Cain, M., Lukin, M. D., & Wang, S.-T. (2022). *Computing solution space properties of combinatorial optimization problems via generic tensor networks*. arXiv. https://doi.org/10.48550/ARXIV.2205.03718

Liu, J.-G., Wang, L., & Zhang, P. (2021). Tropical tensor network for ground states of spin glasses. *Physical Review Letters*, *126*(9). https://doi.org/10.1103/physrevlett.126.090506

Marinescu, R. (2022). *Merlin*.

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

Mooij, J. M. (2010). LibDAI: A free and open source C++ library for discrete approximate inference in graphical models. *Journal of Machine Learning Research*, *11*, 2169–2173. http://www.jmlr.org/papers/volume11/mooij10a/mooij10a.pdf

Oriol, A.-P., Virgile, A., Colin, C., Larry, D., J., F. C., Maxim, K., Ravin, K., Jupeng, L., C., L. C., A., M. O., Michael, O., Ricardo, V., Thomas, W., & Robert, Z. (2023). PyMC: A modern and comprehensive probabilistic programming framework in Python. *PeerJ Computer Science*, *9*, e1516. https://doi.org/10.7717/peerj-cs.1516

Orús, R. (2014). A practical introduction to tensor networks: Matrix product states and projected entangled pair states. *Annals of Physics*, *349*, 117–158. https://doi.org/10.1016/j.aop.2014.06.013

Orús, R. (2019). Tensor networks for complex quantum systems. *Nature Reviews Physics*, *1*(9), 538–550. https://doi.org/10.1038/s42254-019-0086-7

Pan, F., & Zhang, P. (2022). Simulation of quantum circuits using the big-batch tensor network method. *Phys. Rev. Lett.*, *128*, 030501. https://doi.org/10.1103/PhysRevLett.128.030501

Pearl, J. (1985). Bayesian networks: A model of self-activated memory for evidential reasoning. *Proc. Of Cognitive Science Society (CSS-7)*.

Roa-Villescas, M., 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)*.

Roa-Villescas, M., Wijnings, P. W. A., Stuijk, S., & Corporaal, H. (2022). Partial evaluation in junction trees. *2022 25th Euromicro Conference on Digital System Design (DSD)*, 429–437. https://doi.org/10.1109/DSD57027.2022.00064

Robeva, E., & Seigal, A. (2019). Duality of graphical models and tensor networks. *Information and Inference: A Journal of the IMA*, *8*(2), 273–288. https://doi.org/10.1093/imaiai/iay009