

TensorInference: A Julia package for probabilistic inference through tensor-based technology




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](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Published: unpublished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](#)).

Summary

TensorInference.jl is a Julia ([J. Bezanson et al., 2017](#)) library designed for performing probabilistic inference (JG: some text book level reference like PRML) in discrete graphical models. It leverages the recent explosion of advances in the field of tensor networks to provide high-performance solutions for common inference tasks. These tasks include calculating: 1) the partition function or probability of evidence, 2) the marginal probability distribution over each variable given evidence, 3) the most likely assignment to all variables given evidence, 4) the most likely assignment to the query variables after marginalizing out the remaining variables and 5) samples generated from the variable probability distribution given evidence. The infrastructure based on tensor networks Liu et al. ([2022](#)) allows users to define the contraction ordering method, which is known to have a significant impact on the computational performance Pan & Zhang ([2021](#)) of these algorithms. A predefined set of state-of-the-art contraction ordering methods is made available to users. These methods include the *recursive multi-tensor contraction method* (TreeSA) ([Kalachev et al., 2022](#)), the *hyper-optimized tensor network contraction method* (KaHyParBipartite) ([Gray & Kourtis, 2021](#)) and its simulated annealing variant SABipartite, and a *greedy-based memory minimization method* GreedyMethod. Finally, TensorInference.jl harnesses the latest developments in computational technology, including a highly optimized set of BLAS routines and GPU technology.

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 drawing global insights from local observations, a process known as *probabilistic inference*.

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

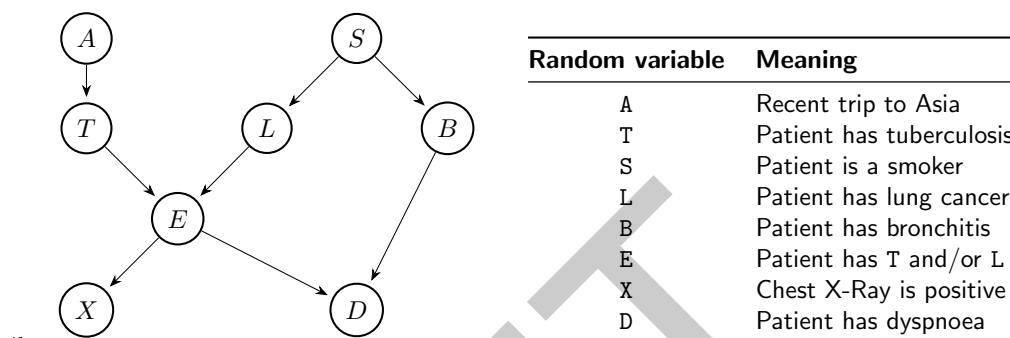
However, performing probabilistic inference on many real-world problems remains intractable due to the intrinsic complexity of performing combinatorial optimization tasks (JG: why combinatorial optimization?) in high dimensional spaces. To tackle these challenge, more efficient and scalable inference algorithms are needed.

We present TensorInference.jl, a Julia ([Jeff Bezanson et al., 2012](#); [J. Bezanson et al., 2017](#)) package for probabilistic inference. This package combines the representational capabilities of PGMs with the computational power of tensor networks. By harnessing the best of both worlds,

41 TensorInference.jl aims to enhance the performance of probabilistic inference, thereby
42 expanding the tractability spectrum of exact inference for more complex models.

43 Usage example

44 The graph below corresponds to the *ASIA network*, a simple Bayesian model used extensively
45 in educational settings. It was introduced in (Lauritzen & Spiegelhalter, 1988).



46
47 We now demonstrate how to use TensorInference.jl for conducting a variety of inference
48 tasks on this toy example.

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

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

# Create a tensor network representation of the loaded model.
# The variable 7 is the variable of interest, which will be retained in the output.
tn = TensorNetworkModel(instance; openvars=[7])

# Calculate the partition function for each assignment of variable 7.
probability(tn)

# Calculate the marginal probabilities of each random variable in the model.
marginals(tn)

# Retrieve the variables associated with the tensor network model.
get_vars(tn)

# Assume that the "X-ray" result (variable 7) is positive.
# Since setting an evidence may affect the contraction order of the tensor
# network, recompute it.
tn = TensorNetworkModel(instance; evidence=Dict{7 => 0})

# Calculate the maximum log-probability among all configurations.
maximum_logp(tn)

# Generate 10 samples from the probability distribution represented by the
```

```
# model.
sample(tn, 10)

# Retrieve both the maximum log-probability and the most probable
# configuration. In this configuration, the most likely outcomes are that the
# patient smokes (variable 3) and has lung cancer (variable 4).
logp, cfg = most_probable_config(tn)

# Compute the most probable values of certain variables (e.g., 4 and 7) while
# marginalizing over others. This is known as Maximum a Posteriori (MAP)
# estimation.
mmap = MMAPModel(instance, queryvars=[4, 7])

# Get the most probable configurations for variables 4 and 7.
most_probable_config(mmap)

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

Acknowledgments

This work is partially funded by the Netherlands Organization for Scientific Research. The authors want to thank Madelyn Cain for helpful advice.

References

- 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>
- Bezanson, Jeff, 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>
- Gray, J., & Kourtis, S. (2021). Hyper-optimized tensor network contraction. *Quantum*, 5, 410. <https://doi.org/10.22331/q-2021-03-15-410>
- Kalachev, G., Panteleev, P., & Yung, M.-H. (2022). *Multi-tensor contraction for XEB verification of quantum circuits*. <https://arxiv.org/abs/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.
- 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>
- 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>
- Pan, F., & Zhang, P. (2021). *Simulating the sycamore quantum supremacy circuits*. <https://arxiv.org/abs/2103.03074>
- 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/iy009>