

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




Jin-Guo Liu<sup>1\*</sup> and Martin Roa-Villescas<sup>2\*</sup>

<sup>1</sup> Hong Kong University of Science and Technology (Guangzhou) <sup>2</sup> Eindhoven University of Technology

\* 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Published: unpublished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 for performing probabilistic inference in discrete graphical models. It leverages the recent explosion of advances in the field of tensor networks to offer 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 all variables given evidence, 3) the most likely assignment to all variables given evidence, and 4) the most likely assignment to the query variables after marginalizing out the remaining variables. The infrastructure based on tensor networks allows users to define the contraction ordering technique, which is known to have a substantial impact on the computational performance of these algorithms. A predefined set of state-of-the-art contraction ordering techniques are available, which include the *recursive multi-tensor contraction method* (TreeSA) (Kalachev et al., 2022), the *hyper-optimized tensor network contraction method* (KaHyParBipartite) (Gray & Kourtis, 2021), the *hierarchical partitioning with dynamic slicing method* (SABipartite) (Pan & Zhang, 2021), and a *greedy-based memory minimization method* (GreedyMethod) (Liu et al., 2022). Finally, TensorInference.jl leverages the latest developments in computational technology, including the 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*.

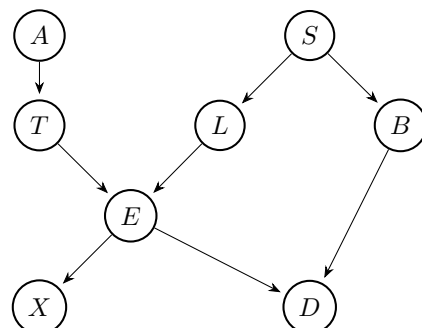
*Probabilistic graphical models* (PGMs) provide a unified framework to address these challenges. These models use graphs to concisely represent the joint probability distribution of complex systems by exploiting the conditional independence between variables in the model. Additionally, they form the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

However, performing probabilistic inference on many real-world problems remains intractable due to the high dimensional spaces and lack of structure in their data. To address these problems, 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 that combines the representational capabilities of PGMs with the computational power of tensor networks.

## Usage example

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



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

We now demonstrate how to use the `TensorInference.jl` package for conducting a variety of inference 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.
tn = TensorNetworkModel(instance)

# Calculate the log10 partition function
probability(tn) |> first |> log10

# 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)

# Set an evidence: Assume that the "X-ray" result (variable 7) is positive.
set_evidence!(instance, 7 => 0)

# Since setting an evidence may affect the contraction order of the tensor
# network, recompute it.
tn = TensorNetworkModel(instance)

# 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.
set_query!(instance, [4, 7])
mmap = MMAPModel(instance)

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