POSTER: Fast Parallel Exact Inference on Bayesian Networks

Jiantong Jiang[†], Zeyi Wen[‡,§], Atif Mansoor[†], Ajmal Mian[†]
[†]The University of Western Australia
[‡]Hong Kong University of Science and Technology (Guangzhou); [§]Hong Kong University of Science and Technology
jiantong.jiang@research.uwa.edu.au,wenzeyi@ust.hk,{atif.mansoor,ajmal.mian}@uwa.edu.au

Abstract

Bayesian networks (BNs) are attractive, because they are graphical and interpretable machine learning models. However, exact inference on BNs is time-consuming, especially for complex problems. To improve the efficiency, we propose a fast BN exact inference solution named Fast-BNI on multi-core CPUs. Fast-BNI enhances the efficiency of exact inference through hybrid parallelism that tightly integrates coarse- and fine-grained parallelism. We also propose techniques to further simplify the bottleneck operations of BN exact inference. Fast-BNI source code is freely available at https://github.com/jjiantong/FastBN.

CCS Concepts: • **Computing methodologies** → **Parallel algorithms**; **Machine learning algorithms**.

Keywords: Bayesian Networks, Inference, Junction Tree

1 Introduction

Bayesian networks (BNs) are probabilistic graphical models. BNs use directed acyclic graphs (DAGs), which are often learned from data [2], to represent random variables and their conditional dependencies. *Exact inference* on BNs is an essential task that calculates the conditional probability of certain *query variables*, given some values of other variables called *evidence* as knowledge to the BN.

Junction Tree (JT) is one of the most prominent BN exact inference algorithms. The key idea is to first convert a BN into a secondary structure called junction tree, where each node (called *clique*) and edge (called *separator*) in the tree contains a subset of variables and maintains a *potential table* over the variables. It then passes *messages* (i.e., functions over variables) along the tree structure and updates all the potential tables.

However, exact inference on BNs is proven to be NP-hard. The complexity of JT increases dramatically with the clique sizes (i.e., the potential table sizes of the cliques), which hinders the use of BNs in complex problems. There are two main types of approaches to accelerate JT on multi-core CPUs. The first type uses coarse-grained inter-clique parallelism that parallelizes the message passing of different cliques [3]. However, inter-clique parallelism is load unbalanced, because the workloads for various cliques are highly different. Some approaches in this category utilize pointer jumping techniques, but introduce additional overhead caused by the rerooting or merging operation. The second type of approaches is fine-grained intra-clique parallelism that parallelizes the potential table operations inside each clique [4]. Zheng [5] accelerated JT on GPUs using the similar idea. However, this type of approaches has efficiency issues from the large parallelization overhead since the table operations are invoked frequently. Moreover, inter-clique parallelism exhibits limited performance for the trees with a small number of cliques, and intra-clique parallelism has efficiency issues on the trees with many small cliques. Therefore, both can only be more efficient under certain junction tree structures.

2 Our Fast-BNI Design

To address the efficiency issues of directly using only inter- or intra-clique parallelism, we propose Fast-BNI, a fast and parallel BN exact inference solution with hybrid inter- and intra-clique parallelism.

For the inter-clique parallelism, we first develop a breadth-first search based *traversal method* to exploit parallelization opportunities across the tree structures. Our traversal method views all the cliques and separators as nodes of the tree and marks the layer where each of them is located. Secondly, we employ a *root selection strategy* to construct a more balanced tree with the minimal number of layers to reduce the total number of parallelization invocations.

For the intra-clique parallelism, we identify and parallelize three dominant potential table operations, including potential table *marginalization*, *extension* and *reduction*. The key step to the potential table operations is to find the index mappings between the original and the updated tables. Accordingly, the complexity of these operations depend on the potential table size, which increases dramatically with the number of random variables in the clique (or separator) and

PPoPP '23, February 25-March 1, 2023, Montreal, QC, Canada
© 2023 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0015-6/23/02.
https://doi.org/10.1145/3572848.3577476

Table 1. Comparison of Fast-BNI with other implementations, and speedup of Fast-BNI over each compared implementation.

BN	Sequential implementation			Parallel implementation						
	Execution time (sec)		Speedup	Execution time (sec)				Speedup		
	UnBBayes	Fast-BNI-seq	UnBBayes	Dir.	Prim.	Elem.	Fast-BNI-par	Dir.	Prim.	Elem.
Hailfinder	28.3	4.0	7.1	3.0	3.2	4.0	2.5	1.2	1.3	1.6
Pathfinder	319.2	68.9	4.6	40.5	23.6	27.8	11.1	3.6	2.1	2.5
Diabetes	90961	6944	13.1	3016	2311	3316	558.6	5.4	4.1	5.9
Pigs	43714	3729	11.7	3353	1068	2380	221.7	15.1	4.8	10.7
Munin2	3054	2643	1.2	1951	934.7	1638	241.7	8.1	3.9	6.8
Munin4	258194	34198	7.6	20364	10348	21398	3021	6.7	3.4	7.1

the number of states of the variables. We develop the corresponding intra-clique primitives that parallelize the index mapping computations of different potential table entries.

We find some shortcomings of accelerating the JT algorithm using only one of the two granularities of parallelism, such as load unbalancing between threads, high parallelization overhead and structure-dependent parallel performance. To remedy the shortcomings, Fast-BNI utilizes a hybrid parallelism that closely integrates inter- and intra-clique parallelism by flattening the nested operations. At the beginning of each layer, all the potential table entries corresponding to this layer are packed to constitute one of the parallel tasks. The tasks are then distributed to the parallel threads to perform concurrently. To summarize, the proposed parallelism has three main advantages, including (i) workload balancing, (ii) smaller parallelization overhead and (iii) adaptability to various structures.

3 Evaluation

We conducted experiments on a Linux machine with two 26-core 2GHz Intel Xeon Platinum 8167M CPUs and 768GB main memory. Our algorithm was tested on six real-world BNs[1] with different sizes, where the last four are considered as large-scale BNs. We randomly generated 2,000 test cases from each network, each with 20% of the observed variables.

Table 1 shows the execution time comparison of sequential and parallel implementations of Fast-BNI with the existing implementations. Specifically, we compared the sequential version of Fast-BNI (i.e., Fast-BNI-seq) with the JT implementation in the UnBBayes library[2] [1]; we also compared Fast-BNI parallel version (i.e., Fast-BNI-par) with parallel implementations [3–5]. Implementation [3] (denoted by *Direct*, *Dir.*) uses a *direct* coarse-grained parallelism; implementation [4] (denoted by *Primitive*, *Prim.*) proposes four fine-grained node-level *primitives* for JT; implementation [5] (denoted by *Element*, *Elem.*) utilizes fine-grained *element*-wise parallelism. For comparing the parallel implementations, we varied the number of OpenMP threads t from 1 to 32 and chose the one with the shortest execution time.

[1]https://www.bnlearn.com/bnrepository/
[2]UnBBayes: https://sourceforge.net/projects/unbbayes/.

As can be seen from the "Speedup" columns of Table 1, the sequential implementation of Fast-BNI can be 1.2 to 13.1 times faster than UnBBayes. When comparing the parallel implementations, Fast-BNI-par can run 1.2 to 15.2 times faster than the counterparts. It is worth noting that Fast-BNI has more advantages over existing implementations on larger networks. For some small-scale networks, the speedups of Fast-BNI to other parallel implementations are relatively small, because they require short execution time for Bayesian inference (e.g., less than 4 seconds for *Hailfinder*) and the parallelization overhead of the small-scale networks accounts for a large proportion. Another observation is that Fast-BNI always achieves its shortest execution time when $t = 32$ on large BNs. *Munin4* is a very large BN with more than 1,000 nodes and edges. The experiment on *Munin4* is the task that takes the longest time to complete. This task ran almost three days using UnBBayes, and spent 3 to 6 hours using the existing parallel implementations, while the execution time is significantly reduced to less than one hour using the proposed Fast-BNI.

Acknowledgments

Professor Ajmal Mian is the recipient of an Australian Research Council Future Fellowship Award (project number FT210100268) funded by the Australian Government. Zeyi Wen is the corresponding author.

References

[1] Rommel Carvalho, KB Laskey, Paulo Costa, Marcelo Ladeira, Laécio Santos, and Shou Matsumoto. 2010. UnBBayes: modeling uncertainty for plausible reasoning in the semantic web. *Semantic web* (2010), 953–978.

[2] Jiantong Jiang, Zeyi Wen, and Ajmal Mian. 2022. Fast Parallel Bayesian Network Structure Learning. In *International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 617–627.

[3] Alexander V Kozlov and Jaswinder Pal Singh. 1994. A parallel Lauritzen-Spiegelhalter algorithm for probabilistic inference. In *Supercomputing'94: Proceedings of the 1994 ACM/IEEE Conference on Supercomputing*. IEEE, 320–329.

[4] Yinglong Xia and Viktor K. Prasanna. 2007. Node Level Primitives for Parallel Exact Inference. *SBAC-PAD* (2007), 221–228.

[5] Lu Zheng. 2013. *Parallel Junction Tree Algorithm on GPU*. Ph. D. Dissertation. Carnegie Mellon University.