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

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

We find some shortcomings of accelerating the JT algo-

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 imple-
mentation in the UnBBayes library[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; implementa-
tion [4] (denoted by Primitive, Prim) proposes four fine-
gained node-level primitives for JT; implementation [5] (de-
noted by Element, Elem) utilizes fine-grained element-wise 
parallelism. For comparing the parallel implementations, we 
varied the number of OpenMP threads from 1 to 32 and 
chose the one with the shortest execution time.

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 im-
plementations, 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 par-
allelization 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 Re-
search Council Future Fellowship Award (project number 
FT210100268) funded by the Australian Government. Zeyi 
Wen is the corresponding author.

### References

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

Network Structure Learning. In International Parallel and Distributed 

Spiegelhalter algorithm for probabilistic inference. In Supercomput-
ing ’94: Proceedings of the 1994 ACM/IEEE Conference on Supercomput-
ing. IEEE, 320–329.

Parallel Exact Inference. SBAC-PAD (2007), 221–228.

[5] Lu Zheng. 2013. Parallel Junction Tree Algorithm on GPU. Ph. D. Disser-

---

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

<table>
<thead>
<tr>
<th>BN</th>
<th>Sequential implementation</th>
<th>Parallel implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Execution time (sec)</td>
<td>Speedup</td>
</tr>
<tr>
<td>Hailfinder</td>
<td>28.3</td>
<td>4.0</td>
</tr>
<tr>
<td>Pathfinder</td>
<td>319.2</td>
<td>68.9</td>
</tr>
<tr>
<td>Diabetes</td>
<td>90961</td>
<td>6944</td>
</tr>
<tr>
<td>Pigs</td>
<td>43714</td>
<td>3729</td>
</tr>
<tr>
<td>Munin2</td>
<td>3054</td>
<td>2643</td>
</tr>
<tr>
<td>Munin4</td>
<td>258194</td>
<td>34198</td>
</tr>
</tbody>
</table>
```

---