Faster-BNI: Fast Parallel Exact Inference on Bayesian Networks

Jiantong Jiang, Zeyi Wen, Atif Mansoor, and Ajmal Mian

Abstract—Bayesian networks (BNs) have recently attracted more attention, because they are interpretable machine learning models and enable a direct representation of causal relations between variables. However, exact inference on BNs is time-consuming, especially for complex problems, which hinders the widespread adoption of BNs. To improve the efficiency, we propose a fast BN exact inference named Faster-BNI on multi-core CPUs. Faster-BNI enhances the efficiency of a well-known BN exact inference algorithm, namely the junction tree algorithm, through hybrid parallelism that tightly integrates coarse- and fine-grained parallelism. Moreover, we identify that the bottleneck of BN exact inference methods lies in recursively updating the potential tables of the network. To reduce the table update cost, Faster-BNI employs novel optimizations, including the reduction of potential tables and re-organizing the potential table storage, to avoid unnecessary memory consumption and simplify potential table operations. Comprehensive experiments on real-world BNs show that the sequential version of Faster-BNI outperforms existing sequential implementation by 9 to 22 times, and the parallel version of Faster-BNI achieves up to 11 times faster inference than its parallel counterparts. Faster-BNI source code is freely available at https://github.com/jjiantong/FastPGM.

Index Terms—Bayesian Networks, Exact Inference, Junction Tree Algorithm.

1 Introduction

Bayesian networks (BNs) [1] are probabilistic graphical models that use directed acyclic graphs (DAGs) to compactly represent a set of random variables and their conditional dependencies. The graphical nature of BNs makes them suitable for a wide range of applications, including fault diagnosis, healthcare, environmental modeling and image analysis [2], [3], [4], [5]. BNs have also attracted much research attention with the recent growing demand for interpretable machine learning models [6], [7].

One crucial aspect of building BNs is to use Bayesian models for *inference*, which is to calculate the conditional probability of certain *query variables*, given some values of other variables called *evidence* of the BN. Figure 1a shows the network structure of an example "Asia" BN with eight variables. One may have interest in whether a patient has tuberculosis (τ) and bronchitis (β), given the observations that the patient has dyspnea ($\delta = 1$) and is a smoker ($\sigma = 1$). In this example, the query variables include τ and β and the evidence is $\{\delta = 1, \sigma = 1\}$. Inference on BNs can be *exact* or *approximate* and both are proven to be NP-hard [8], [9]. Approximate inference often takes less time than exact inference at the cost of accuracy. However, with the demand for the high precision in many applications, simulation methods in approximate inference algorithms become more expensive, and the advantages of approximation fade away. Hence, exact inference becomes increasingly essential [10].

One of the most prominent BN exact inference algorithms is the *junction tree* (JT) algorithm [11]. It is implemented in various mainstream BN libraries [12], [13], [14], and many variants are also based on the improved versions of the JT algorithm [15], [16], [17], [18], [19]. The main steps of JT are as follows. It first converts a BN into a secondary structure called junction tree, as shown in Figure 1b, where each node (called *clique*) and edge (called *separator*) in the junction tree contains a subset of random variables and maintains a *potential table* over the random variables. It then passes *messages* (i.e., functions over variables) along the tree structure and updates all the potential tables. The conditional probabilities of all the variables in the BN can be determined once the potential table updating is completed. The inference can be easily achieved with the conditional probabilities of all the variables.

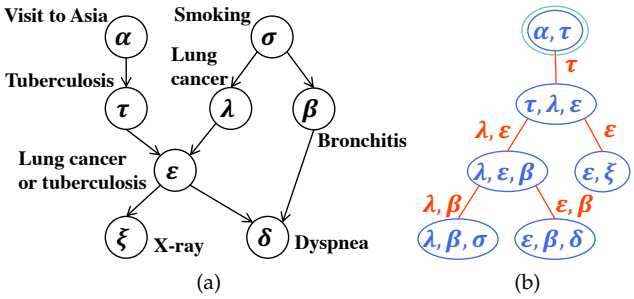


Fig. 1: The example "Asia" BN. (a) The underlying DAG of the BN and (b) the corresponding junction tree generated by the BN. Cliques are in blue, separators are in red, and the double circle indicates the root of the tree.

However, the computational complexity of the JT algorithm increases dramatically with the clique sizes (i.e., the potential table sizes of the cliques), which hinders the use of BNs in large or complex problems where timely inference is

Zeyi Wen is the corresponding author.

- *Jiantong Jiang, Atif Mansoor and Ajmal Mian are with The University of Western Australia, Crawley, WA, 6009, Australia. E-mail: {jiantong.jiang@research., atif.mansoor@, ajmal.mian@}uwa.edu.au*
- *Zeyi Wen is with the Data Science and Analytics Thrust, Hong Kong University of Science and Technology (Guangzhou), and also with Hong Kong University of Science and Technology, China. E-mail: wenzeyi@ust.hk*

required. There are two main types of approaches to accelerate the JT algorithm on multi-core CPUs. The first type of approaches uses coarse-grained inter-clique parallelism that parallelizes the message passing of different cliques [10], [20], [21], [22]. 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 which introduce extra overhead caused by the rerooting [20], [21] or merging operation [22]. The second type is fine-grained intra-clique parallelism that parallelizes the potential table operations inside each clique [23], [24]. However, this approach 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 of them can only be more efficient under certain junction tree structures.

To address the efficiency issues, we propose Faster-BNI, a fast BN inference solution with hybrid inter- and intra-clique parallelism. At the inter-clique level, we develop a junction tree traversal method to exploit parallelization opportunities across the tree structures and a root selection strategy to construct a more balanced tree structure. At the intra-clique level, we parallelize three dominant potential table operations including potential table marginalization, extension and reduction. Moreover, we further optimize Faster-BNI by (i) reducing potential tables to avoid unnecessary potential table operations and (ii) re-organizing the potential table storage to save memory consumption and simplify the potential table computations. To summarize, the main contributions of this paper are as follow:

- We propose Faster-BNI that accelerates the BN exact inference on multi-core CPUs. Faster-BNI uses a hybrid inter- and intra-clique parallelism, which solves the efficiency issues in the existing implementations and suits various network structures.
- We propose new techniques to further enhance Faster-BNI. First, we propose to reduce the potential tables when loading evidence to avoid subsequent unnecessary potential table operations. Second, we carefully reorganize the potential table to reduce memory usage and simplify the potential table computations in the message passing procedure.
- We conduct extensive experiments to study the effectiveness of Faster-BNI and the impact of the proposed optimizations. Experimental results show that Faster-BNI outperforms existing works by up to 22 times. The practical parallelization speedups of Faster-BNI approach the theoretical speedups.

This paper is an extension of our conference paper for Fast-BNI [25]. The major differences from [25] are summarized in the following three aspects. Firstly, we elaborate on the technical details of our proposed hybrid parallelism and provide a detailed analysis of its advantages over previous parallelization schemes (cf. Section 3.2), which were not provided in our previous work. Secondly, we propose two new optimizations (cf. Section 3.3), which lead to about two times speedup over our previous implementation in Fast-BNI. Thirdly, we conduct additional detailed experiments to study (i) the comparison between Faster-BNI and Fast-BNI, (ii) the impact of our new optimizations, (iii) the speedup of hybrid parallelism, and (iv) the comparison between different parallelization schemes.

2 BACKGROUND

In this section, we provide the key terminologies and definitions related to exact inference on Bayesian networks (BNs), and then review the junction tree (JT) algorithm. Finally, we discuss relevant work for BN exact inference.

2.1 Bayesian Networks and Exact Inference

BNs are graphical models that represent a joint distribution over a set of random variables $\mathcal{V} = \{V_0, V_1, ..., V_{n-1}\}$ in $\mathbb{G}$ via a directed acyclic graph (DAG). Typically, one variable corresponds to one feature in machine learning problems. In this paper, we focus specifically on discrete BNs, where the random variables take on a finite set of discrete values.

A BN is defined as $B = (\mathbb{G}, \mathbb{P})$, where $\mathbb{G} = (\mathcal{V}, \mathcal{E})$ is a DAG and $\mathbb{P}$ is the parameters of the network. Figure 1a illustrates the DAG $\mathbb{G}$ of the "Asia" network with eight (0,1)-valued variables and eight edges, where each node in $\mathcal{V}$ is associated with one variable and each edge in $\mathcal{E}$ indicates conditional dependency between two variables. V_j is called a parent of V_i if there exists a directed edge from V_j to V_i in $\mathbb{G}$. For example, variable σ is the parent of variable λ; the direct edge between them indicates whether smoking impacts the likelihood of a patient suffering from lung cancer. The parameter set $\mathbb{P}$ represents a set of conditional probability distributions (CPDs) that describes the probability of possible values of each variable given its possible parent configurations. The joint probability of BN can be decomposed into the product of the CPDs of each variable:

$$P(\mathcal{V}) = P(V_0, V_1, ..., V_{n-1}) = \prod_{i=0}^{n-1} P(V_i|\mathbf{Par}(V_i)) \quad (1)$$

where n is the number of variables, $\mathbf{Par}(V_i)$ is the set of parents of V_i, and $P(V_i|\mathbf{Par}(V_i))$ is the CPD of variable V_i.

Exact inference on BNs is to calculate the posterior marginal distributions of certain variables of interest, called the *query variables*. And in most cases, we have some observed variables and the observed values are called *evidences* of the network. In the example BN in Figure 1a, suppose we have the observations that the patient has dyspnea ($\delta = 1$) and is a smoker ($\sigma = 1$). The variable of interest could be bronchitis (β) and thus the computation is the posterior marginal distribution over β given the evidence of $\{\delta = 1, \sigma = 1\}$, i.e., $P(\beta|\delta = 1, \sigma = 1)$.

2.2 Junction Tree Algorithm

The JT algorithm is an essential exact inference algorithm. It first converts the original BN into a junction tree, then computes probabilities on the resulting junction tree.

The steps involved in converting the BN to a junction include moralization, triangulation, clique identification and junction tree construction [11]. The basic idea is to turn highly-connected nodes in the BN into cliques of the junction tree. We omit the details as it is not the primary focus

Operation 1: Procedure of junction tree updating.

Input: Junction tree J
$C_{root} \leftarrow$ Arbitrarily select a clique from J as the root
$J \leftarrow$ Load evidence to J
Collection (J, C_{root})
Distribution (J, C_{root})

of this paper. This is mainly because generating a junction tree takes a tiny part of the total inference time and is only done offline once for multiple queries. Figure 1b shows a possible junction tree generated from "Asia". Each node C_i of the junction tree is called a *clique* which contains a subset of the random variables as its *related variables* (i.e., $\mathcal{U}_i \subseteq \mathcal{V}$). Each edge S_{ij} between C_i and C_j is called a *separator* which contains all shared variables of C_i and C_j (i.e., $\mathcal{U}_{ij} = \mathcal{U}_i \cap \mathcal{U}_j$). In the discrete BNs, there is a *potential table* ϕ_{C_i} (resp. $\phi_{S_{ij}}$) associated with each clique C_i (resp. separator S_{ij}). The potential table is a function over the related variables of its associated clique (resp. separator).

When new evidence comes, two steps are required to update the junction tree, as shown in Operation 1. The first step is to *load evidence* to update the tree. It checks every clique of the junction tree to find whether there are some potential table entries of the clique that contain the observed variables but have different instantiations from the evidence. Such conflict entries are set to zero. The second step is *message passing*, which contains two recursive phases *collection* and *distribution*. In this step, all the potential tables are recursively updated by *messages* that are computed from other potential tables. For the collection phase, messages are collected along the tree from the leaves to the root, while messages are distributed from the root to the leaves for the distribution phase. Note that the root clique can be an arbitrary clique in the junction tree. Formally, a message passed from C_i to C_j can be written as:

$$\phi^*_{S_{ij}} = \sum_{\mathcal{U}_i \setminus \mathcal{U}_{ij}} \phi_{C_i}, \quad \phi^*_{C_j} = \phi_{C_j} \frac{\phi^*_{S_{ij}}}{\phi_{S_{ij}}} \tag{2}$$

where $\phi^*_{S_{ij}}$ and $\phi^*_{C_j}$ represent the updated potential tables of separator S_{ij} and clique C_j, respectively. The computational complexity depends on the clique sizes, which are the potential table sizes of the cliques in the junction tree.

Once the message passing ends, we can get potentials of all the cliques. Thus, we can easily compute the marginal distribution of any query variable V_i from a clique C_i that contains the variable (i.e. $V_i \in \mathcal{U}_i$), by marginalizing out all the other variables in the clique.

$$P(V_i) = \sum_{\mathcal{U}_i \setminus V_i} P(\mathcal{U}_i) \tag{3}$$

In this work, we aim to accelerate the JT algorithm with a focus on evidence loading and message passing.

2.3 Related Work

BNs are powerful models for learning and reasoning under uncertainty in artificial intelligence. This study mainly focuses on improving the efficiency of a key exact inference algorithm: JT algorithm. In what follows, we categorize the most relevant work into two categories: algorithms for BN exact inference and studies dedicated to accelerating JT.

Exact inference on BNs. In the early 1980s, Pearl proposed an efficient message passing inference algorithm called polytree [26], [27] that works only for singly connected networks. The JT algorithm [11] extended it to multiply connected networks by first transforming the graph into a junction tree. The algorithm can be further divided into Shenoy-Shafer algorithm [17] and Hugin algorithm [18], [19]. There are many other exact inference algorithms, such as arc reversal [28], variable elimination [29], symbolic probabilistic inference [30], [31] and differential approach [32]. This paper mainly focuses on improving the efficiency of the JT algorithm using parallel techniques, as the JT algorithm is used in mainstream implementations.

Accelerating JT. Kozlov and Singh [10] proposed a parallel implementation that mainly considers the topology of the tree structure, and thus the performance is dependent on the structure of the network. Some other methods also focus on the coarse-grained inter-clique parallelism by using the pointer jumping technique [20], [21], [22]. However, studies [20], [21] exhibit limited performance for multiple evidences, since rerooting is required when the evidences appear at different cliques. Although work [22] does not required rerooting, it may introduce extra overhead when some cliques appear at different chains at the same time. On the other hand, some approaches parallelize inside cliques [23], [24], which use a fine-grained intra-clique parallelism. Several research studies also utilize intra-clique parallelism to accelerate JT on other platforms like GPUs [33], [34]. However, intra-clique parallelism may have a large parallelization overhead, which decreases the efficiency.

3 Our Proposed Faster-BNI

This section elaborates the technical details of our proposed Faster-BNI, a parallel solution for exact inference on BNs.

3.1 Design Overview

Here, we provide an overview of our proposed Faster-BNI, illustrated in Figure 2. Overall, Faster-BNI takes the junction tree as input, executes evidence loading and message passing procedures, and finally outputs the posterior probability distributions of the non-evidence variables. In what follows, we elaborate on detailed structure of Faster-BNI.

Initially, the junction tree is stored in the form of potential tables for clique and separator layers. During this phase, we employ the root selection strategy and junction traversal method to construct a more balanced junction tree, in order to exploit inter-clique parallelization opportunities (cf. Section 3.2.1). Subsequently, the evidence loading procedure is performed, utilizing a table reduction optimization to reduce the potential table size and simplify the subsequent computations (cf. Section 3.3.1). After that, the critical procedure of message passing is executed, comprising collection and distribution phases. Within this procedure, we identify and parallelize two pivotal potential table operations at the intra-clique parallelization level: potential table marginalization within the separator layers, and potential table extension within the clique layers (cf. Section 3.2.2).

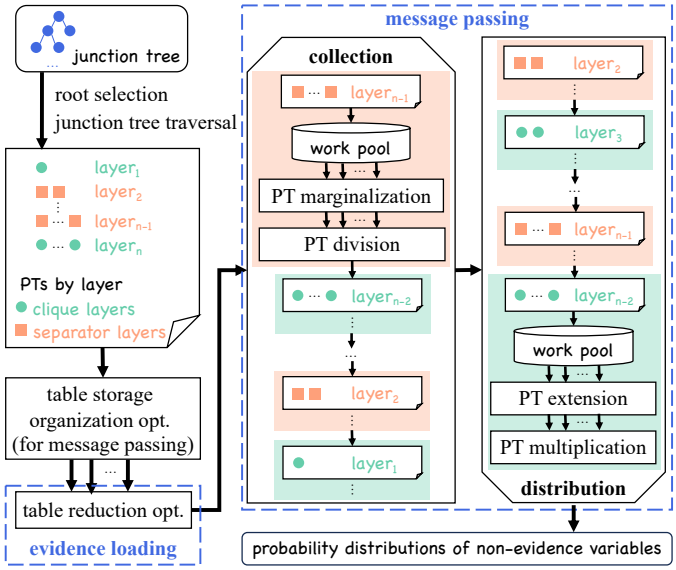


Fig. 2: The overview of Faster-BNI. "PT" stands for "potential table".

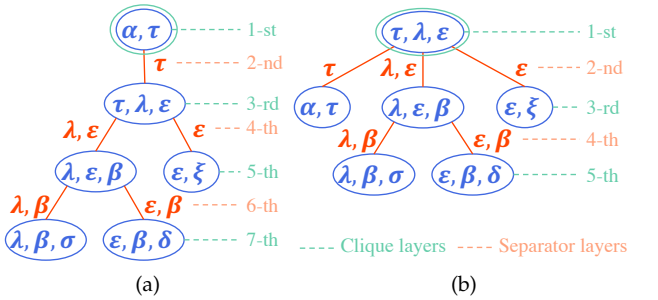


Fig. 3: Two possible junction trees generated by the "Asia" BN with layers marked. Tree (b) is more balanced than tree (a). The double circles indicate the roots of the junction trees.

These operations are performed throughout the message passing procedure, as depicted in Figure 2. To further improve the efficiency of message passing, we introduce a table storage organization optimization, significantly simplifying the potential table computations (cf. Section 3.3.2).

Faster-BNI employs hybrid parallelism that closely integrates inter- and intra-clique parallelism, which can remedy the shortcomings of using only one of the two granularities, including (i) load unbalancing among threads, (ii) high parallelization overhead and (iii) structure-dependent parallel performance (cf. Section 3.2.3).

3.2 Faster-BNI Parallelization Scheme

This section provides the design of our hybrid inter- and intra-clique parallelization scheme as well as its advantages over other parallelization schemes.

3.2.1 *Inter-clique Parallelism*

We consider inter-clique parallelism in Faster-BNI, which is coarse-grained parallelism that takes the independence in topology of the junction tree into consideration.

The message passing procedure is inherently sequential due to the dependencies between the passing functions. Specifically, for the collection procedure, one clique C_i can pass the message to its parent only when C_i has received all the messages from its children. Similarly, C_i can only pass the message to its children when it has received the message from its parent in the distribution procedure. But some cliques can still be performed in parallel since they are computationally independent. In order to exploit the parallelization opportunities in the tree structure, we propose a junction tree traversal method based on breadth-first search. Our traversal method views all the cliques and separators as nodes of the tree, and marks the layer where each clique and separator is located. Figure 3a shows an example that marks the layers of the junction tree shown in Figure 1b. In the collection procedure, computations are performed layer by layer from the leaves to the root, while distribution operates from the root to the leaves. The potential table operations inside each layer can be parallelized because the computations in the same layer are independent. In each layer, inter-clique parallelism distributes evenly all the cliques (or separators) in the layer to different threads.

The efficiency of inter-clique parallelism is better when the number of cliques (or separators) inside each layer is large enough. Accordingly, instead of selecting an arbitrary clique as the root, we select the root clique that minimizes the number of layers. This generates a more balanced tree that has a smaller number of layers and a larger average number of cliques (or separators) per layer. The root selection strategy is implemented by checking each clique in the tree, finding its longest distance to other cliques and minimizing the distance. Figure 3 shows two possible junction trees with different root cliques. We can see that the tree in Figure 3b is more balanced with a smaller number of layers (i.e., 5) and a larger average number of cliques (or separators) per layer (i.e., 2.2).

The root selection strategy and junction tree traversal method take less than 1% of the total inference time, and are performed offline before loading evidence. They only require to perform once for multiple queries. Therefore, their execution time is negligible.

3.2.2 *Intra-clique Parallelism*

We consider intra-clique parallelism in Faster-BNI, which is fine-grained parallelism that focuses on the expensive potential table operations inside each clique (or separator). We first identify two essential potential table operations that require high computational overhead in space and time.

Potential table marginalization. According to Equation (2), potential table marginalization operation is used in the message passing procedure to update the potential table of a separator (i.e., $\phi^*_{S_{ij}}$) from the potential table of a clique (i.e., ϕ_{C_i}), which is used in the separator layers. Figures 4a and 4b show the procedure of marginalizing X from the potential table with related variables $\{X, Y, Z\}$. In this example, variable X has three possible states $\{x_1, x_2, x_3\}$, while Y and Z have two possible states. Here, marginalization sums over all possible states of X while retaining the states of Y and Z. For example, to update the entry (y_2, z_2), we add the probabilities of all the entries in the original table that contain (y_2, z_2) without

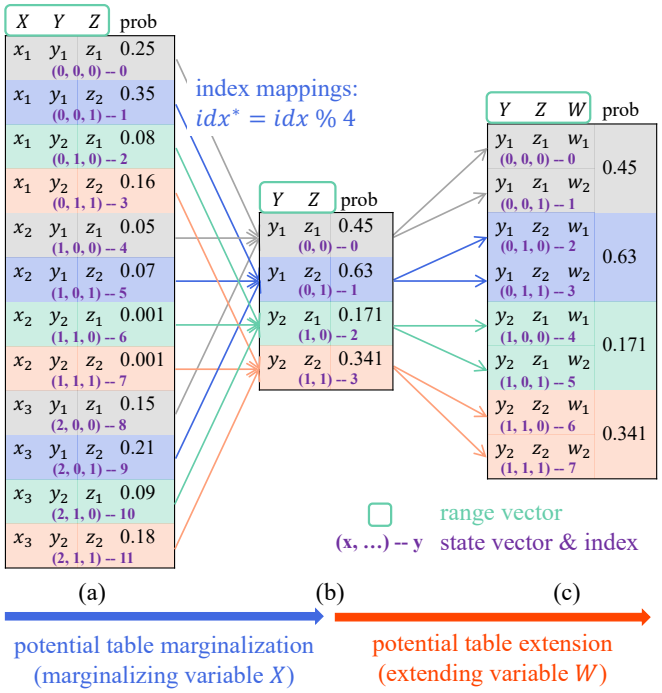


Fig. 4: Example of potential table marginalization and extension. (a) to (b): marginalizing variable X from $\{X, Y, Z\}$; (b) to (c): extending variable W from $\{Y, Z\}$. Due to the table storage organization optimization in Section 3.3.2, the range vector of each potential table and the index mappings from (a) to (b) are also provided.

considering the states of X. Therefore, this entry is updated to $0.16 + 0.001 + 0.18 = 0.341$.

Potential table extension. According to Equation (2), we need to perform multiplication in the message passing procedure when updating the potential table of a clique (i.e., $\phi^*_{C_j}$) using the message from a separator (i.e., $\phi^*_{S_{ij}}/\phi_{S_{ij}}$). Potential table extension is a pre-computation of multiplication that makes the two factors of multiplication have an equal size. It is used in the clique layers. Figures 4b and 4c show the procedure of extending the potential table from related variables $\{Y, Z\}$ to $\{Y, Z, W\}$, where variable W has two possible states w_1 and w_2. This operation may be performed for the case that we need to compute the Cartesian product of the potential table in Figure 4b and another potential table with related variables $\{Y, Z, W\}$, and the potential table extension operation is performed before the multiplication. For the example shown in Figures 4b and 4c, the entry (y_2, z_2) is extended to two entries (y_2, z_2, w_1) and (y_2, z_2, w_2), each of which has a probability 0.341 obtained from the entry (y_2, z_2).

To summarize, the key step to the above potential table operations is to find the table index mappings between the original and the updated tables. Accordingly, the complexity of these potential table 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. Since the potential table operations are performed frequently during the message passing procedure, intra-clique parallelism is to parallelize each entry of the potential table operations, so as to reduce the time consumption in the potential table operations. The computations of finding the index mappings for different entries are independent and thus can be processed in parallel. Besides potential table marginalization and extension, potential table multiplication and division, as well as the potential table updating on evidence loading, are also parallelized. Intra-clique parallelism distributes the potential table entries evenly to the threads and hence achieves good load balancing among threads.

3.2.3 *Hybrid Inter- and Intra-clique Parallelism*

We find some limitations in accelerating the JT algorithm using only inter- or intra-clique parallelism.

Limitations of inter-clique parallelism. Inter-clique parallelism has the following two major limitations. First, inter-clique parallelism is load unbalanced, because the workloads of potential table operations for different cliques (or separators) can be highly different. The workload of the potential table operation of a clique (resp. separator) depends on the potential table size of the clique (resp. separator). If multiple cliques (resp. separators) in the same layer having different potential table size, the execution time is determined by the time of the largest clique (resp. separator). Second, the dependencies among cliques (or separators) can be the bottleneck. Therefore, this scheme can be efficient for the tree structure where there are a large number of cliques inside each layer, but has limited parallelization opportunities if the number of cliques for each layer is small.

Limitations of intra-clique parallelism. The main limitation of intra-clique parallelism is the high parallelization overhead. On the one hand, the potential table operations are performed frequently. On the other hand, the workloads for each potential table entry can be relatively small. Therefore, this scheme is more efficient for the junction trees with large cliques (or separators), so that each thread has a larger amount of workload to amortize parallel overhead (e.g., frequent thread invocations). However, it may exhibit limited performance for a tree structure with many small cliques (or separators).

To overcome the limitations, a natural idea is to integrate the two granularities of parallelism. A simple strategy is to dynamically schedule the tasks. The tasks of the cliques (or separators) within the same layer are continuously generated. When processing the inner potential table operations, the tasks for different potential table entries are also generated and processed by the idle threads. However, this simple integration strategy fails to get good speedups. The main reason is that the efficiency of inter- and intra-clique parallelism is structure-dependent, making it hard to find the structure suitable for both. For example, junction trees that contain a large number of cliques benefit from inter-clique parallelism. However, such type of structures may have many small cliques, which limits the efficiency of intra-clique parallelism. Furthermore, the scheme leads to more cost for scheduling and it is also inefficient to invoke parallel threads in a nested way due to the thread creation overhead.

To tackle the shortcomings of accelerating the JT algorithm using inter-clique parallelism, intra-clique parallelism and the simple integration strategy, we propose hybrid inter- and intra-clique parallelism that closely integrates the

two granularities of parallelism and avoids the nested invocations of parallel threads. In particular, at the beginning of each layer, all the potential table entries corresponding to this layer are pushed into a work pool and constitute one of the parallel tasks. The pre-processing before parallelization also includes the decisions of whether the tables need to perform the corresponding potential table operation, and the organization of continuous memory for all the table entries to be operated in this layer to store the data required for the operation. Then the tasks in the work pool can be distributed to the parallel threads to perform the potential table operations on different table entries concurrently. This process is performed layer by layer.

The proposed hybrid parallelism has the following three main advantages. **(i) Workload balancing.** Compared with inter-clique parallelism, the workloads of different potential table entries can be evenly distributed to each thread with hybrid parallelism. Since the entries executed in parallel are from the same layer, they do the same operation and have the same amount of computations. Thus, Faster-BNI solves the issue of load unbalancing. **(ii) Lower parallel overhead.** Compared with intra-clique parallelism, all the potential tables within the same layer are considered at the same time in hybrid parallelism. Therefore, the parallel invocation overhead is significantly reduced. On the other hand, each thread has a more reasonable amount of workload as the work pool for one parallel invocation contains more tasks, which can also amortize the parallelization overhead. **(iii) Adaptability to various structures.** Faster-BNI does not require some specific junction tree structures to achieve good efficiency, since the inter- and intra-clique parallelism are tightly integrated. More specifically, Faster-BNI does not require the junction trees with many cliques inside each layer to fit inter-clique parallelism, or the junction trees with large cliques to fit intra-clique parallelism. The proposed hybrid parallelism can adapt to various tree structures.

3.3 Further Enhancing Faster-BNI

In this section, we aim to tackle two crucial challenges to enhance Faster-BNI. Firstly, in evidence loading, it is inefficient to only change the values of the conflict potential table entries but maintain the potential table size. Secondly, there are many large potential tables in one junction tree, which consume much memory and lead to complicated computations in the message passing procedure.

3.3.1 Table Reduction in Evidence Loading

During the evidence loading procedure, some potential tables in the junction tree are updated to reflect the evidence information. This is the potential table reduction operation, as shown in Figures 5a and 5b. Specifically, the potential table contains three related variables X, Y and Z at the beginning. Assuming we observe $Z = z_1$, then all the table entries that conflict with the evidence $Z = z_1$ are set to zero. However, the potential table size is not changed after this operation as shown in Figures 5a and 5b, which is inefficient because the complexity of all the potential table operations depends on the potential table size.

We can observe from Figure 5b that after loading evidence $Z = z_1$, the related variables of the table are the same

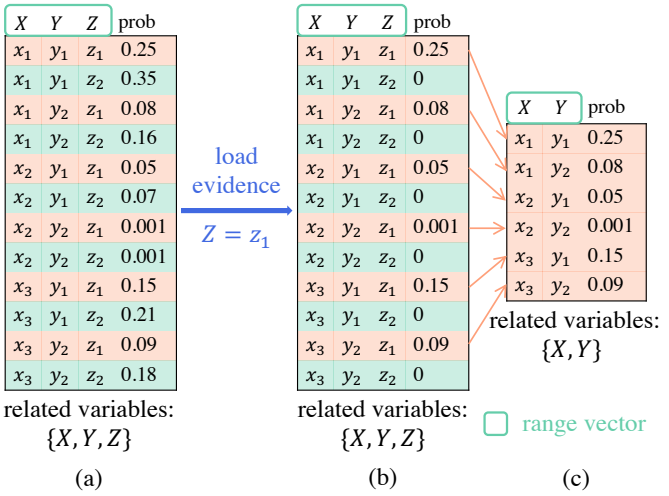


Fig. 5: Example of potential table reduction. (a) the original potential table before potential table reduction, (b) the table after naive potential table reduction and (c) the table after optimized potential table reduction.

as before (i.e., $\{X, Y, Z\}$), but they should become $\{X, Y\}$ because the values of Z in all the non-zero entries are fixed to z_1. Therefore, we can directly delete the conflict entries instead of setting them to zero, and remove the evidence variables from the related variables of the clique at the same time, as shown in Figure 5c. Note that the potential table reduction operation is also parallelized in Faster-BNI. Like potential table marginalization and extension, the key step to potential table reduction is also to find the table index mappings between the original and the updated tables. Therefore, the parallelization scheme of Faster-BNI can be applied to the potential table reduction operation.

By the optimized potential table reduction, the size of potential table can be reduced by a factor of f, where f equals the product of the number of states of all the evidence variables. For the simple example in Figure 5, the size of the table is reduced by half, since there is one evidence variable Z with two states originally. As a result, the table reduction optimization brings a nice cascading effect on the complexity of the subsequent potential table operations in the message passing procedure, such as the time-consuming operations of potential table marginalization and extension, because the complexity of all the potential table operations depends on the corresponding potential table size.

3.3.2 Table Storage Organization in Message Passing

In large problems, the number of variables and the number of states of each variable are often large, leading to high complexity of the potential table operations. We analyze the complexity using potential table marginalization as an example. To marginalize the potential table ϕ_{C_i} of clique C_i to get the potential table $\phi^*_{S_{ij}}$ of separator S_{ij}, the traversal of ϕ_{C_i} and $\phi^*_{S_{ij}}$ is required. The size of ϕ_{C_i} equals the number of possible states of the related variables $\mathcal{U}_i$ of C_i, which is $\prod_{V_k \in \mathcal{U}_i} r_{V_k}$, where r_{V_k} is the number of states of V_k. Similarly, the size of $\phi^*_{S_{ij}}$ is $\prod_{V_k \in \mathcal{U}_{ij}} r_{V_k}$. For each entry to be marginalized, we need to get and compare the values of all the related variables $\mathcal{U}_{ij}$ of S_{ij}. Additionally, the algorithm

proceeds every clique during message passing. Therefore, the overall complexity is $O(\sum_{l=1}^{nc}(|\mathcal{U}_{ij}| \times (\prod_{V_k \in \mathcal{U}_i} r_{V_k} + \prod_{V_k \in \mathcal{U}_{ij}} r_{V_k})))$, where nc is the number of cliques. On the other hand, large potential tables also result in more memory consumption, since we need to store the configurations of the variables and the corresponding probability for each entry of the table. To conclude, the potential table operations require high computational overhead in space and time.

To alleviate the high cost of time and space, Faster-BNI uses a potential table data structure. Suppose that clique C_i contains q_i variables. It corresponds to a potential table ϕ_{C_i} and we use a q_i-dimensional *range vector* $\mathbf{rv}_i$ to represent the variables contained in its potential table. Each configuration of the table can be compiled into a q_i-dimensional *state vector* according to the order of $\mathbf{rv}_i$. We use an *index* for each table entry to avoid storing the configurations, because each index can be transformed to and from a state vector. The transformations can be performed using a q_i-dimensional auxiliary array of the clique that stores the cumulative numbers of possible states starting from the rightmost element of $\mathbf{rv}_i$.

The key to the potential table operations is to find the index mappings between the two involved tables. Typically, the transformations between indexes and state vectors of the two tables are required for finding the mappings. In order to further reduce the computational burden of such transformations, we propose to constrain the order of the range vector as follows: for each non-root clique C_i, its parent clique C_j and the separator S_{ij} between them, those variables stored on the left of $\mathbf{rv}_i$ are the variables not in S_{ij}, while those on the right are the variables contained in S_{ij}. In this way, we can obtain the mappings of the involved tables without checking the state vector for each entry. This simplified index mapping computation can be applied to the following two cases in the message passing procedure: (i) the potential table marginalization operations in the collection procedure, and (ii) the potential table extension operations in the distribution procedure.

We use the potential table marginalization in collection as an example to analyze the advantage of the optimization on computation simplification, while the potential table extension operations in distribution work in a similar way. Suppose that Figure 4a is the potential table ϕ_{C_i} of clique C_i, and Figure 4b is the updated potential table $\phi_{S_{ij}}$ of separator S_{ij}. Through the proposed data structure, finding an index mapping between the two tables (e.g., $idx = 6$ of old table ϕ_{C_i} to $idx^* = 2$ of updated table $\phi_{S_{ij}}$) is initially realized by (i) performing an index-to-configuration mapping for ϕ_{C_i} to get the configuration $cfg = \{x_2, y_2, z_1\}$, (ii) getting the configuration $cfg^* = \{y_2, z_1\}$ of $\phi_{S_{ij}}$ according to cfg and the range vector indices of the variables to be marginalized (e.g., index of X is 0), and (iii) performing a configuration-to-index mapping for $\phi_{S_{ij}}$ to obtain $idx^* = 2$. With the help of the table storage organization optimization, variables $\{Y, Z\}$ are guaranteed to be the right part of $\mathbf{rv}_i$ because they are contained in S_{ij}. As a result, the mappings between the indexes in ϕ_{C_i} and the ones in $\phi_{S_{ij}}$ are intuitive from Figure 4. Each index idx^* of the new table can be easily calculated from the index idx of the old table by $idx\%4$, where $\%$ is the modulo operator, and 4 is the product of the number of states of Y and Z. In this way, the overall complexity significantly reduces to $O(\sum_{l=1}^{nc} \prod_{V_k \in \mathcal{U}_i} r_{V_k})$.

3.4 Potential Applications and Generalization

Here, we briefly discuss the potential applications and generalization of our proposed techniques. Faster-BNI is able to generalize to a broader range of probabilistic graphical models (PGMs), such as Markov Random Fields (MRFs). While BNs and MRFs exhibit different graphical structures, i.e., directed graphs for BNs and undirected graphs for MRFs, the underlying principle of message passing remains consistent across both models. More specifically, both the JT algorithm in BNs and belief propagation-based inference algorithms in MRFs involve passing messages between nodes in a graph, iteratively updating beliefs, and converging to the desired inference results.

Message passing constitutes the majority of the execution time for the JT algorithm. Faster-BNI enhances message passing through optimizations such as hybrid parallelization and table storage organization. By recognizing the shared principle of message passing in the PGMs, the optimization techniques proposed in Faster-BNI can be adapted and extended to accelerate inference tasks in other PGMs.

4 Experimental Evaluation

In this section, we extensively evaluate our proposed techniques and compare the results with existing methods.

4.1 Experimental Setup

Baselines and platform. We implemented Faster-BNI using OpenMP in C++ for exact inference on BNs and compared its performance with the existing methods. Specifically, we compared Faster-BNI with the sequential JT implementation in the open-source library UnBBayes [12]. We also compared Faster-BNI with previous parallel implementations [10], [23], [34]. Implementation [10] (denoted by *Direct*, *Dir.*) uses a *direct* coarse-grained parallelism that mainly considers the topological dependencies of the tree structure; implementation [23] (denoted by *Primitive*, *Prim.*) proposes four fine-grained node-level *primitives* to accelerate the JT algorithm; implementation [34] (denoted by *Element*, *Elem.*) accelerates the message passing by utilizing fine-grained *element*-wise parallelism and arithmetic parallelism. All the experiments were conducted on a Linux machine with a 32-core AMD EPYC 7502 CPU and 64GB main memory. The query results of Faster-BNI are exactly the same as the other implementations, since all of them are implementations of the same exact inference algorithm.

Data sets. We used six real-world BNs[1] as shown in Table 1. The BNs are from various fields and have different sizes. These BNs have been widely used for comparative studies in the community, and the last four are considered as large-scale in the community. They also have various structures and state spaces, and thus lead to very different junction trees, as listed in Table 1. The junction trees generated from the *Diabetes* and *Munin4* BNs have very large average clique sizes, which are more than 50,000, and the maximum clique sizes of them even exceed 3,500,000. We generated 1,000 test cases from each network. Following the common settings [40], [41], [42], each test case has 20 observed variables.

1. https://www.bnlearn.com/bnrepository/

TABLE 1: Information of reference BNs and the resulting junction trees.

Data set	Reference BN information			Resulting JT information			
	# of nodes	# of edges	# of parameters	# of cliques	# of layers	Ave. clique size	Max clique size
Hailfinder [35]	56	66	2656	43	15	232	3267
Pathfinder [36]	109	195	77155	91	13	2007	32256
Diabetes [37]	413	602	429409	336	79	59071	3534300
Pigs [38]	441	592	5618	368	29	11842	1594323
Munin [39]	1041	1397	80592	872	35	19526	1000000
Munin4 [39]	1038	1388	80352	882	31	54358	3528000

TABLE 2: Execution time comparison of Faster-BNI with other implementations under both sequential and parallel setting. Speedup of Faster-BNI over each compared implementation is also reported.

Data set	Sequential implementation			Parallel implementation						
	Execution time (sec)		Speedup	Execution time (sec)				Speedup		
	UnBBayes	Faster-BNI-seq	UnBBayes	Dir.	Prim.	Elem.	Faster-BNI-par	Dir.	Prim.	Elem.
Hailfinder	13.7	0.9	14.8	1.7	1.9	2.2	0.9	1.9	2.0	2.4
Pathfinder	175.8	16.6	10.6	28.9	19.4	21.9	5.2	5.6	3.7	4.2
Diabetes	48842	2223	22.0	2347	1661	2610	438.4	5.4	3.8	6.0
Pigs	24530	1878	13.1	2769	1338	2275	271.0	10.2	4.9	8.4
Munin	41486	4660	8.9	6398	3081	5372	600.2	10.7	5.1	8.9
Munin4	200501	16280	12.3	14776	7036	15957	2173	6.8	3.2	7.3

TABLE 3: Detailed comparison of the parallel and sequential versions of Faster-BNI with UnBBayes and Prim. on the *Hailfinder* and *Pathfinder* BNs. "-seq" and "-par" represent sequential and parallel implementation, respectively.

Hailfinder	L1-cache accesses	L1-cache misses (rate)	LL-cache accesses	LL-cache misses (rate)	CPU utilization
Faster-BNI-par	5.1×10^9	$1.8 \times 10^8 (3.61\%)$	5.6×10^7	$2.8 \times 10^6 (4.85\%)$	12
Faster-BNI-seq	4.6×10^9	$1.2 \times 10^8 (2.52\%)$	4.5×10^5	$3.6 \times 10^3 (0.81\%)$	1
Prim.	5.0×10^9	$4.6 \times 10^8 (9.21\%)$	1.5×10^8	$3.1 \times 10^7 (19.71\%)$	12
UnBBayes	3.6×10^{10}	$1.5 \times 10^9 (4.27\%)$	8.9×10^7	$7.4 \times 10^6 (8.33\%)$	1
Pathfinder	**L1-cache accesses**	**L1-cache misses (rate)**	**LL-cache accesses**	**LL-cache misses (rate)**	**CPU utilization**
Faster-BNI-par	8.0×10^{10}	$2.0 \times 10^9 (2.49\%)$	1.3×10^8	$3.6 \times 10^4 (0.03\%)$	12
Faster-BNI-seq	7.9×10^{10}	$1.8 \times 10^9 (2.29\%)$	8.6×10^7	$7.0 \times 10^4 (0.08\%)$	1
Prim.	7.1×10^{10}	$3.4 \times 10^9 (4.80\%)$	4.2×10^8	$1.7 \times 10^6 (0.4\%)$	12
UnBBayes	6.8×10^{11}	$1.7 \times 10^{10} (2.52\%)$	3.7×10^8	$1.4 \times 10^8 (37.74\%)$	1

4.2 Overall Comparison

In this section, we study the improvement of Faster-BNI over Fast-BNI and other existing work.

4.2.1 Comparison with Existing Work

We compare the execution time of both sequential and parallel implementations of Faster-BNI with the existing implementations for 1,000 queries. Specifically, we compare the sequential version of Faster-BNI (i.e., Faster-BNI-seq) with UnBBayes [12] and the parallel version of Faster-BNI (i.e., Faster-BNI-par) with Dir. [10], Prim. [23] and Elem. [34]. For comparing the parallel implementations, we vary the number of OpenMP threads t from 1 to 32 and chose the one with the shortest execution time.

The experimental results are summarized in Table 2. As can be seen from the "Speedup" columns of the table, the sequential implementation of Faster-BNI can be 8.9 to 22.0 times faster than UnBBayes. When comparing the parallel implementations, Faster-BNI-par can run 1.2 to 15.2 times faster than the counterparts. It is worth noting that Faster-BNI has more advantages over existing implementations on larger networks. For some small-scale networks, the speedups are relatively small, because they require short execution time for inference (e.g., less than 1 seconds for *Hailfinder*) and the parallelization overhead of the small-scale networks accounts for a large proportion. Another observation is that Faster-BNI always achieves its shortest execution time when $t = 32$ on large networks. The experiment on *Munin4* is the task that takes the longest time to complete. This task ran more than two days using UnBBayes, and spent 2 to 4 hours using the existing parallel implementations, while the execution time is significantly reduced to less than one hour using Faster-BNI.

To further investigate why Faster-BNI is faster, we use perf Linux profiler to obtain the detailed measurements for Faster-BNI-par, Faster-BNI-seq, UnBBayes and Prim. on two examples *Hailfinder* and *Pathfinder*. We choose the two BNs, because UnBBayes requires much time on the larger BNs when profiling. We choose Prim. as a parallel implementation representative, because it is the best one among the three parallel baselines. The results are shown in Table 3. For the parallel implementations, we set the number of threads t to 12 because both Faster-BNI-par and Prim. can achieve a relatively good efficiency under this setting. We can observe that the parallel version of Faster-BNI increases CPU utilization. Moreover, Faster-BNI have fewer accesses to the L1 cache and last level (LL) cache due to the proposed optimizations for reducing the potential table operations. Faster-BNI also decreases the rate of cache misses.

4.2.2 Comparison with Our Previous Implementation [25]

In this section, we compare Faster-BNI with our previous implementation Fast-BNI [25].

Figure 6 compares memory efficiency of Faster-BNI and Fast-BNI. The memory consumption for Fast-BNI serves as the baseline, denoted as 100% for each dataset, and the

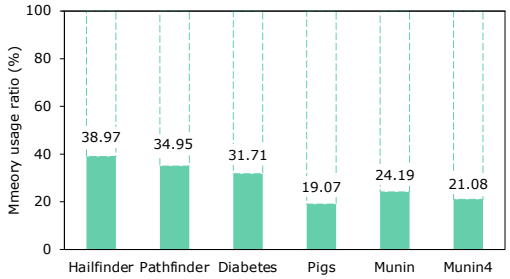


Fig. 6: Comparison of memory usage between Faster-BNI and Fast-BNI. Memory usage of Fast-BNI is set as 100%.

TABLE 4: Execution time comparison between Fast-BNI and Faster-BNI under sequential and parallel settings.

Data set	Sequential implementation		
	Execution time (sec)		Speedup
	Fast-BNI	Faster-BNI	
Hailfinder	1.9	0.9	2.1
Pathfinder	38.2	16.6	2.3
Diabetes	3728	2223	1.7
Pigs	2951	1878	1.6
Munin	7408	4660	1.6
Munin4	26382	16280	1.6
Data set	**Parallel implementation**		
	Execution time (sec)		Speedup
	Fast-BNI	Faster-BNI	
Diabetes	527.2	438.4	1.2
Pigs	362.2	271.0	1.3
Munin	789.9	600.2	1.3
Munin4	2952	2173	1.4

memory usage of Faster-BNI is depicted as the percentage relative to Fast-BNI. Notably, Faster-BNI has significant memory savings, ranging from 61% to 81%. The benefit is mainly due to the adoption of our proposed potential table data structure (cf. Section 3.3.2). The degree of memory-saving efficiency varies across different datasets, owing to diverse junction tree structures, such as different numbers of cliques, variables within cliques, clique sizes, etc.

Table 4 compares the execution time of Faster-BNI with Fast-BNI under both sequential and parallel settings. We can observe that our new implementation consistently outperforms our previous one. The sequential implementation of Faster-BNI is about 2 times faster than that of Fast-BNI, and the parallel implementation of Faster-BNI can bring up to 36% improvement. The improvement is mainly due to our careful optimizations to reduce the potential table computations. These optimizations include (i) *table reduction*: reducing the potential table in the evidence loading procedure (cf. Section 3.3.1), and (ii) *table storage organization*: organizing potential table storage in the message passing procedure (cf. Section 3.3.2). These general optimizations can be applied to both sequential and parallel implementations. The parallel results on *Hailfinder* and *Pathfinder* are omitted in Table 4, because they already require short inference time for the sequential implementations by the proposed optimizations.

Ablation study. To get a better understanding of the efficiency improvements, we study the impacts of our proposed optimizations on the execution time of four key components in the message passing procedure: (i) table marginalization on collection ("col. mar."), (ii) table marginalization on distribution ("dis. mar."), (iii) table extension on collection ("col. ext."), and (iv) table extension on distribution ("dis. ext."). These components represent the most complicated potential table operations in the message passing procedure (cf. Section 3.2.2). The execution time of the other components of JT is denoted by "others". Figure 7 shows a breakdown of the execution time. We observe that Faster-BNI reduces the time cost for the four key components in Fast-BNI. To investigate where the efficiency improvement originates from, we first enable table reduction optimization under the "Fast-BNI", and report the results in the "Fast-BNI + table reduction" bars. Then, we enable table storage re-organization optimization, and report the results in the "Faster-BNI" bars. Figure 7, we extract the following three key conclusions. Firstly, table reduction optimization reduces the time cost for all the complicated potential table operations in message passing. Since this optimization can reduce the size of some potential tables when loading evidence, it reduces the computational complexity of the subsequent potential table operations. Secondly, table storage organization significantly impacts the execution time of table marginalization operations on collection and table extension operations on distribution. With the help of table

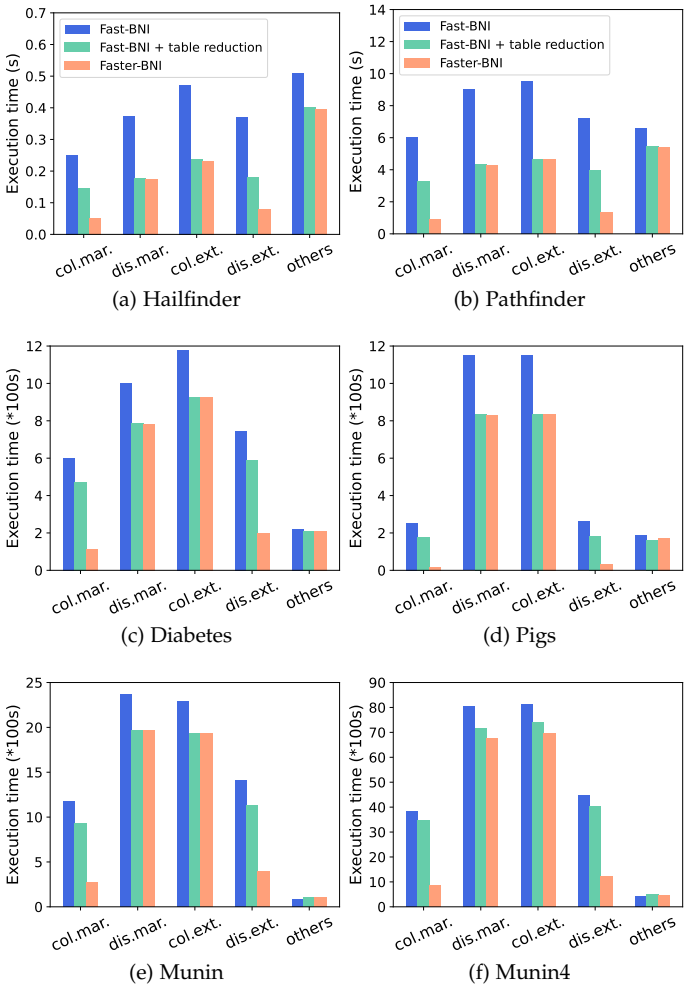


Fig. 7: Execution time breakdown on four key components of the JT algorithm under different settings.

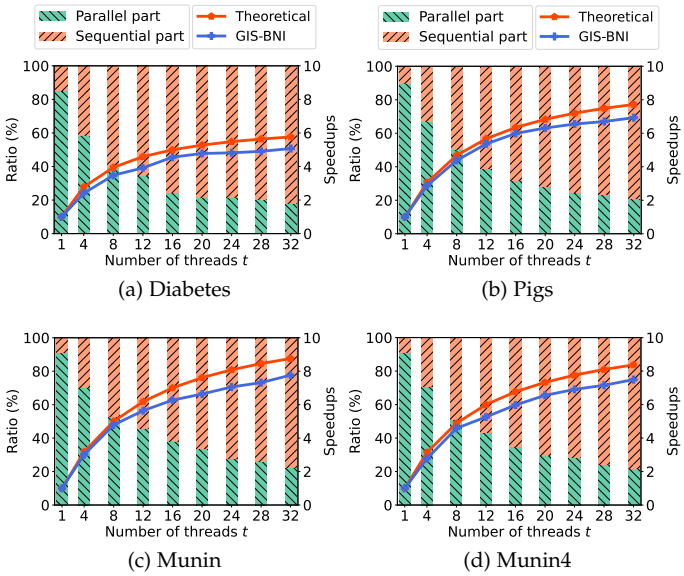


Fig. 8: Theoretical speedup computed by Amdahl's law and practical speedup of the proposed Faster-BNI under different number of threads. The ratios of parallel part and sequential part of the program are also provided.

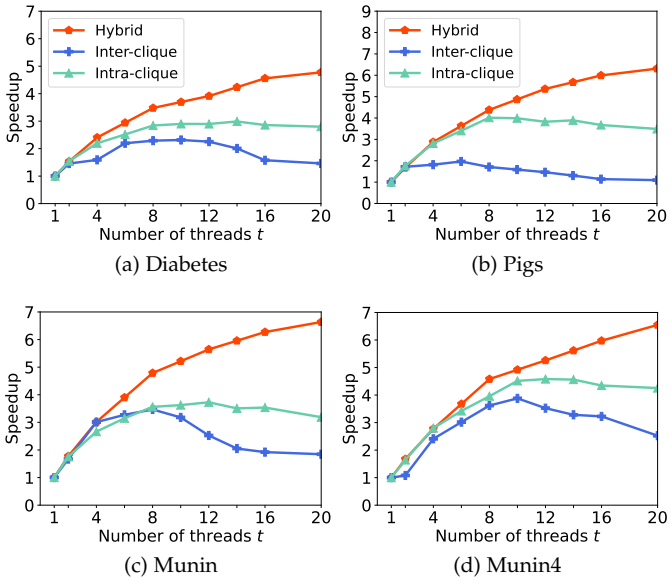


Fig. 9: Execution time of parallel implementations using three different parallel schemes: inter-clique parallelism, intra-clique parallelism and hybrid parallelism.

storage organization, the expensive index mappings in the above potential table operations are avoided by simple modulo calculations. Thirdly, the reason why the speedup of the parallel implementation is not as high as that of the sequential implementation is that the two optimizations reduce the parallel ratio of the algorithm, since they reduce the execution time of the key components.

4.3 Comparison with Theoretical Speedup

Here, we compare the speedups of the parallel version of Faster-BNI to the sequential version with the theoretical speedups. The theoretical speedups are computed by Amdahl's Law [43], which is

$$I(t) = \frac{1}{(1-r) + \frac{r}{t}} \tag{4}$$

where t is the number of parallel threads, r is the ratio of the parallel part of the program, and $I(t)$ means the theoretical speedup under t threads. We obtained the ratio of parallel part r from one sequential run of the program.

Figure 8 shows the theoretical and practical speedups on the four large-scale BNs. We can observe that the speedups of Faster-BNI generally approach the theoretical speedups. Figure 8 also provides the ratios of the parallel and sequential parts of the program respectively. The sequential parts of Faster-BNI mainly come from the pre- and post-computations for the hybrid parallelism, including potential table property calculations, memory management and atomic writing. Junction tree construction is also sequential. As shown in Figure 8c, the *Munin* network has the largest r, which is 91.4%. Therefore, it has the largest theoretical speedup as well as the practical speedup of Faster-BNI among the networks tested. On the other hand, results on the relatively small-scale networks *Hailfinder* and *Pathfinder* are omitted, since Faster-BNI has a relatively small speedup on them due to the small ratio of their parallel part (e.g., $r = 0.5$ for *Hailfinder*). When t reaches 32, the ratios of the parallel parts for all the networks drop by 0.2 or around 0.2, leading to a slower speedup growth.

4.4 Comparing the Different Parallelization Schemes

To investigate the performance of different parallel schemes, we implemented two parallel versions using the schemes of inter- and intra-clique parallelism, and compared them with Faster-BNI that employs the hybrid inter- and intra-clique parallelism. All these parallel versions are based on the optimized version of Faster-BNI.

Figure 9 shows the speedups of the three parallel implementations under the four large networks. We observe that hybrid parallelism always leads to the highest speedups, indicating the effectiveness of the optimizations used in the hybrid parallelism on the large BNs. Overall, the inter-clique parallelism is the worst and the reasons include the limited parallelization opportunities caused by the dependencies among cliques and the issues of workload unbalancing. Especially when increasing threads cannot bring enough efficiency gains, the speedup may decrease since the efficiency gains cannot surpass the parallelization overhead. The speedups of intra-clique parallelism can be better than that of inter-clique parallelism, because these large networks often generate junction trees with many large cliques, from which intra-clique parallelism can benefit. Moreover, the execution time of the intra-clique parallelism can be reduced by more than 50% using the hybrid parallelism which solves its efficiency issue of high parallelization overhead.

4.5 Studies on Different Network Sizes

To better understand Faster-BNI, Figure 10 shows the speedups of the parallel version of Faster-BNI over its

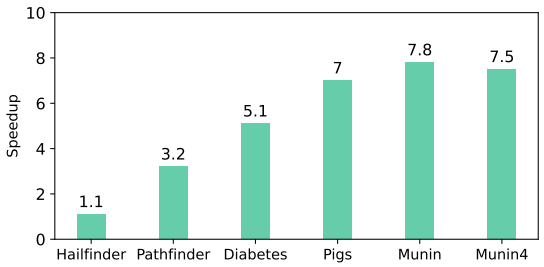


Fig. 10: Studies on speedup of Faster-BNI-par over Faster-BNI-seq under different network sizes.

sequential version on BNs with 2,000 samples. The six BNs tested are of different network sizes and have various structures of the resulting junction trees, as shown in Table 1. We can observe that Faster-BNI achieves high speedups for larger networks, indicating good scalability of the proposed techniques to large networks. For example, Faster-BNI achieves 7.8 times speedup on the *Munin* and 7.5 on *Munin4* BNs, both of which have larger number of nodes and edges in their network structure. On the other hand, the speedups of parallel implementation for the small-scale networks (i.e., *Hailfinder* and *Pathfinder*) are relatively small, because they already require short execution time for the inference on BNs and the parallelization overhead of these small-scale networks accounts for a large proportion. Therefore, with the help of the proposed general optimizations discussed in Section 3.3, our sequential implementation is sufficient for such small-scale networks with a relatively small number of nodes and small clique sizes in the generated junction tree.

5 Conclusion

In this paper, we have proposed a parallel junction tree (JT) algorithm namely Faster-BNI. The challenges of developing a fast solution for exact inference on BNs including addressing load unbalancing issues, amortizing parallelization overhead and generalizing to various network structures. To tackle these challenges, Faster-BNI exploits hybrid parallelism that tightly integrate inter- and intra-clique parallelism. Moreover, Faster-BNI is powered by a series of careful optimizations including (i) junction tree traversal method to exploit parallelization opportunities, (ii) root selection strategy to generate a more balanced tree for better parallelization, (iii) optimized potential table reduction to reduce the subsequent potential table operations and (iv) well-organized potential table storage to reduce memory consumption and simplify the potential table computations. Extensive experimental results have shown that Faster-BNI achieves up to 22 times faster than the existing solutions; the parallel version of Faster-BNI achieves up to 8 times speedup and the practical speedups of Faster-BNI approach the theoretical speedups. Furthermore, Faster-BNI has demonstrated good scalability to the network sizes.

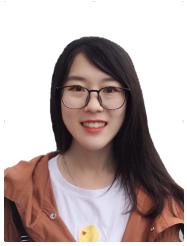
Acknowledgments

This research was funded by ARC Grant number DP190102443. Professor Ajmal Mian is the recipient of an Australian Research Council Future Fellowship Award (project number FT210100268) funded by the Australian Government.

References

[1] J. Pearl, "Probabilistic reasoning in intelligent systems: networks of plausible inference," 1988.
[2] E. Kyrimi, S. McLachlan, K. Dube, M. R. Neves, A. Fahmi, and N. Fenton, "A comprehensive scoping review of bayesian networks in healthcare: Past, present and future," *Artificial Intelligence in Medicine*, vol. 117, p. 102108, 2021.
[3] B. G. Marcot and T. D. Penman, "Advances in Bayesian network modelling: Integration of modelling technologies," *Environmental modelling & Software*, vol. 111, pp. 386–393, 2019.
[4] M. T. Amin, F. Khan, S. Ahmed, and S. Imtiaz, "A data-driven bayesian network learning method for process fault diagnosis," *Process Safety and Environmental Protection*, vol. 150, pp. 110–122, 2021.
[5] N. R. Nayak, S. Kumar, D. Gupta, A. Suri, M. Naved, and M. Soni, "Network mining techniques to analyze the risk of the occupational accident via bayesian network," *International Journal of System Assurance Engineering and Management*, vol. 13, no. 1, pp. 633–641, 2022.
[6] D. Gunning and D. Aha, "Darpa's explainable artificial intelligence (XAI) program," *AI Magazine*, vol. 40, no. 2, pp. 44–58, 2019.
[7] C. Rudin, "Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead," *Nature Machine Intelligence*, vol. 1, no. 5, pp. 206–215, 2019.
[8] G. F. Cooper, "The computational complexity of probabilistic inference using Bayesian belief networks," *Artificial intelligence*, vol. 42, no. 2-3, pp. 393–405, 1990.
[9] P. Dagum and M. Luby, "Approximating probabilistic inference in Bayesian belief networks is NP-hard," *Artificial intelligence*, vol. 60, no. 1, pp. 141–153, 1993.
[10] A. V. Kozlov and J. P. Singh, "A parallel Lauritzen-Spiegelhalter algorithm for probabilistic inference," in *Supercomputing'94: Proceedings of the 1994 ACM/IEEE Conference on Supercomputing*. IEEE, 1994, pp. 320–329.
[11] S. L. Lauritzen and D. J. Spiegelhalter, "Local computations with probabilities on graphical structures and their application to expert systems," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 50, no. 2, pp. 157–194, 1988.
[12] R. Carvalho, K. Laskey, P. Costa, M. Ladeira, L. Santos, and S. Matsumoto, "UnBBayes: modeling uncertainty for plausible reasoning in the semantic web," *Semantic web*, pp. 953–978, 2010.
[13] A. Franzin, F. Sambo, and B. Di Camillo, "bnstruct: an R package for Bayesian network structure learning in the presence of missing data," *Bioinformatics*, vol. 33, no. 8, pp. 1250–1252, 2017.
[14] K. Murphy *et al.*, "The bayes net toolbox for matlab," *Computing science and statistics*, vol. 33, no. 2, pp. 1024–1034, 2001.
[15] A. L. Madsen and F. V. Jensen, "Lazy propagation: a junction tree inference algorithm based on lazy evaluation," *Artificial Intelligence*, vol. 113, no. 1-2, pp. 203–245, 1999.
[16] J. D. Park and A. Darwiche, "Morphing the Hugin and Shenoy–Shafer architectures," in *European Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty*. Springer, 2003, pp. 149–160.
[17] P. P. Shenoy and G. Shafer, "Propagating belief functions with local computations." *IEEE Expert*, vol. 1, no. 3, pp. 43–52, 1986.
[18] F. V. Jensen, K. G. Olesen, and S. K. Andersen, "An algebra of Bayesian belief universes for knowledge-based systems," *Networks*, vol. 20, no. 5, pp. 637–659, 1990.
[19] F. V. Jensen, S. L. Lauritzen, and K. G. Olesen, "Bayesian updating in causal probabilistic networks by local computations," 1990.
[20] D. M. Pennock, "Logarithmic time parallel bayesian inference," in *Conference in Uncertainty in Artificial Intelligence*, 1998.
[21] V. K. Namasivayam and V. K. Prasanna, "Scalable parallel implementation of exact inference in bayesian networks," *12th International Conference on Parallel and Distributed Systems*, vol. 1, pp. 8 pp.–, 2006.
[22] Y. Xia and V. K. Prasanna, "Junction tree decomposition for parallel exact inference," in *2008 IEEE International Symposium on Parallel and Distributed Processing*. IEEE, 2008, pp. 1–12.
[23] ——, "Node level primitives for parallel exact inference," *19th International Symposium on Computer Architecture and High Performance Computing*, pp. 221–228, 2007.

[24] ——, "Scalable node-level computation kernels for parallel exact inference," *IEEE Transactions on Computers*, vol. 59, no. 1, pp. 103–115, 2009.
[25] J. Jiang, Z. Wen, A. Mansoor, and A. Mian, "Fast parallel exact inference on bayesian networks," in *Proceedings of the 28th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2023.
[26] J. Kim and J. Pearl, "A computational model for causal and diagnostic reasoning in inference systems," in *International Joint Conference on Artificial Intelligence*, 1983, pp. 0–0.
[27] J. Pearl, "Fusion, propagation, and structuring in belief networks," *Artificial intelligence*, vol. 29, no. 3, pp. 241–288, 1986.
[28] R. D. Shachter, "Evidence absorption and propagation through evidence reversals," in *Machine Intelligence and Pattern Recognition*. Elsevier, 1990, vol. 10, pp. 173–190.
[29] N. L. Zhang and D. Poole, "A simple approach to Bayesian network computations," in *Canadian Conference on Artificial Intelligence*, 1994.
[30] R. D. Shachter, B. D'Ambrosio, and B. Del Favero, "Symbolic probabilistic inference in belief networks." in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 90, 1990, pp. 126–131.
[31] Z. Li and B. d'Ambrosio, "Efficient inference in Bayes networks as a combinatorial optimization problem," *International Journal of Approximate Reasoning*, vol. 11, no. 1, pp. 55–81, 1994.
[32] A. Darwiche, "A differential approach to inference in Bayesian networks," *Journal of the ACM*, vol. 50, no. 3, pp. 280–305, 2003.
[33] H. Jeon, Y. Xia, and V. K. Prasanna, "Parallel exact inference on a CPU-GPGPU heterogenous system," in *2010 39th International Conference on Parallel Processing*. IEEE, 2010, pp. 61–70.
[34] L. Zheng, "Parallel junction tree algorithm on GPU," Ph.D. dissertation, Carnegie Mellon University, 2013.
[35] B. Abramson, J. Brown, W. Edwards, A. Murphy, and R. L. Winkler, "Hailfinder: A bayesian system for forecasting severe weather," *International Journal of Forecasting*, vol. 12, no. 1, pp. 57–71, 1996.
[36] D. E. Heckerman, E. J. Horvitz, and B. N. Nathwani, "Toward normative expert systems: Part i the pathfinder project," *Methods of information in medicine*, vol. 31, no. 02, pp. 90–105, 1992.
[37] S. Andreassen, R. Hovorka, J. Benn, K. G. Olesen, and E. R. Carson, "A model-based approach to insulin adjustment," in *Proceedings of the Third Conference on Artificial Intelligence in Medicine*. Springer, 1991, pp. 239–248.
[38] C. Skaanning, "Blocking gibbs sampling for inference in large and complex bayesian networks with applications in genetics," 1997.
[39] S. Andreassen, F. Jensen, S. Andersen, B. Falck, U. Kjrul, M. Woldbye, A. Srensen, A. Rosenfalck, and F. Jensen, "Computer-aided electromyography and expert systems," 1989.
[40] J. Cheng and M. J. Druzdzel, "Ais-bn: An adaptive importance sampling algorithm for evidential reasoning in large bayesian networks," *Journal of Artificial Intelligence Research*, vol. 13, pp. 155–188, 2000.
[41] C. Yuan and M. J. Druzdzel, "An importance sampling algorithm based on evidence pre-propagation," in *Conference in Uncertainty in Artificial Intelligence*, 2003, pp. 624–631.
[42] ——, "Importance sampling algorithms for bayesian networks: Principles and performance," *Mathematical and Computer Modelling*, vol. 43, no. 9-10, pp. 1189–1207, 2006.
[43] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proceedings of the April 18-20, 1967, spring joint computer conference*, 1967, pp. 483–485.



Jiantong Jiang is currently a PhD candidate at The University of Western Australia, after receiving a Master's degree from Northeastern University in China. Before commencing her PhD study, she was a Research Assistant at School of Software Engineering, Zhejiang University, China. Her research interests include high-performance computing and automatic machine learning system.

Zeyi Wen is an Assistant Professor at Hong Kong University of Science and Technology, Guangzhou (HKUST-GZ). Before joining HKUST-GZ, he was a Lecturer at The University of Western Australia, a Research Fellow in National University of Singapore and The University of Melbourne, after receiving his PhD degree from The University of Melbourne. Zeyi's areas of research include high-performance computing, machine learning systems and data mining.

Atif Mansoor is an Assistant Professor at the Department of Computer Science and Software Engineering at The University of Western Australia. Before joining The University of Western Australia, he taught at The National University of Sciences and Technology and The Punjab University in Pakistan. He was Erasmus Mundus scholar at Norwegian University of Science and Technology, Gjovik and Jean Monnet University, Saint-Etienne, France. Atif has published more than 50 papers in international conferences and journals. His research area is Computer Vision, Pattern Classification and Cyber-Physical systems.

Ajmal Mian is a Professor of Computer Science at the University of Western Australia. He is the recipient of three prestigious national level fellowships from the Australian Research Council (ARC) including the recent Future Fellowship award 2022. He is a Fellow of the International Association for Pattern Recognition, Distinguished Speaker of the Association for Computing Machinery and former President of the Australian Pattern Recognition Society. He received the West Australian Early Career Scientist of the Year Award 2012 and the HBF Mid-Career Scientist of the Year Award 2022. He has secured research funding from the ARC, the National Health and Medical Research Council of Australia, US Department of Defence DARPA, and the Australian Department of Defence. He has published around 300 scientific papers. He served as a Senior Editor for IEEE TNNLS and Associate Editor for IEEE TIP and the Pattern Recognition journal. His research interests include computer vision, artificial intelligence, machine learning, 3D point cloud analysis, facial recognition, human action measurement and video analysis.