On the Complexity of Probabilistic Inference in Singly Connected Bayesian Networks

Dan Wu[1] and Cory Butz[2]

[1] School of Computer Science
University of Windsor
Windsor Ontario
Canada N9B 3P4
[2] Department of Computer Science
University of Regina
Regina Saskatchewan
Canada S4S 0A3

Abstract. In this paper, we revisit the consensus of computational complexity on exact inference in Bayesian networks. We point out that even in singly connected Bayesian networks, which conventionally are believed to have efficient inference algorithms, the computational complexity is still NP-hard.

1 Introduction

Bayesian networks (BNs) have gained popularity in the last decade as a successful framework for processing uncertainty using probability. A Bayesian network [7] consists of two components: a *directed acyclic graph* (DAG) and a set of *conditional probability distributions* (CPDs). The product of these CPDs defines a *joint probability distribution* (JPD). One of the most important tasks for a BN is to perform *probabilistic inference*, which simply means computing the posterior probability distribution for a set of variables given the evidence that some other variables in the network are taking specific values. There are two kinds of probabilistic inference that can be performed, namely, *exact* inference and *approximate* inference. Exact inference means computing the exact posterior probability distribution. Approximate inference produces an inexact, bounded solution, but guarantees that the exact solution is within those bounds. In this paper, we will only comment on the computational complexity of exact inference.

During the development of various probabilistic inference algorithms, the following statements regarding computational complexity of exact inference in BNs are made. Singly connected BNs have linear time algorithm in the number of nodes in a network or the size of the network for exact inference. On the other hand, multiply connected Bayesian networks do not admit efficient algorithms for exact inference in the worst case. Finally, exact probabilistic inference in the general case is NP-hard because it was proved that exact inference in multiply

connected BNs is NP-hard [1] [3]. Based on the above remarks, it has come to the consensus that the singly connected BNs are favorable and tractable while the multiply connected BNs are intractable (at least in the worst case) and should be blamed for causing the exact inference in BNs to be NP-hard.

In this paper, we revisit the consensus of computational complexity on exact inference in BNs. It seems that the consensus is somewhat misleading. Our main argument is that inference in a singly connected BN can be exponential in the worst case. More specifically, we adapt the proof in [1] to demonstrate that exact inference in singly connected BNs can also be NP-hard. That is to say, the hardness of exact inference in BNs should have nothing to do with the topological structure of the DAG of a BN.

The paper is organized as follows. In Section 2, we introduce pertinent background material and notation. We review the current consensus on exact inference in BNs in Section 3. In Section 4, we point out an inconsistency in the consensus. We investigate the inconsistency in Section 5. We discuss the implication of our investigation and conclude the paper in Section 6.

2 Background

We use $R = \{x_1, \ \ldots, \ x_n\}$ to represent a set of discrete variables. Each x_i takes value from a finite domain denoted V_{x_i}. We use capital letters such as X to represent a subset of R and its domain is denoted by V_X. By XY we mean $X \cup Y$. We write $x_i = \alpha$, where $\alpha \in V_{x_i}$, to indicate that the variable x_i is instantiated to the value α. Similarly, we write $X = \beta$, where $\beta \in V_X$, to indicate that X is instantiated to the value β. For convenience, we write $p(x_i)$ to represent $p(x_i = \alpha)$ for all $\alpha \in V_{x_i}$. Similarly, we write $p(X)$ to represent $p(X = \beta)$ for all $\beta \in V_X$.

Definition 1. Let $R = \{x_1, \ \ldots, \ x_n\}$ be a set of discrete variables. A Bayesian network (BN) defined over R consists of two components:(i) a *directed acyclic graph* (DAG) $\mathcal{D}$ whose nodes correspond one-to-one to the variables in R, and (ii) a set $\{p(x_i|\pi_{x_i}) \mid 1 \leq i \leq n\}$ of CPDs where π_{x_i} denotes the parents of x_i in $\mathcal{D}$. The product of the CPDs define a unique joint distribution $p(R)$ as: $p(R) = \prod_{1 \leq i \leq n} p(x_i|\pi_{x_i})$.

BNs are usually classified into three categories, according to the topological structures of their respective DAGs.

Definition 2. A BN is called a *tree structure* BN if for each node in the DAG of the BN except the root, there is only one parent node.

Definition 3. A BN is called a *singly connected* BN (also known as *polytree*) if there exists at most one (undirected) path between any two nodes in the DAG of the BN. Obviously, tree structure BNs are special cases of singly connected BNs.

[3] It is perhaps worth mentioning that approximate inference in BNs was also proved NP-hard [8].

Definition 4. A BN is called a *multiply connected* BN if there exists more than one (undirected) path between at least two nodes in the DAG of the BN.

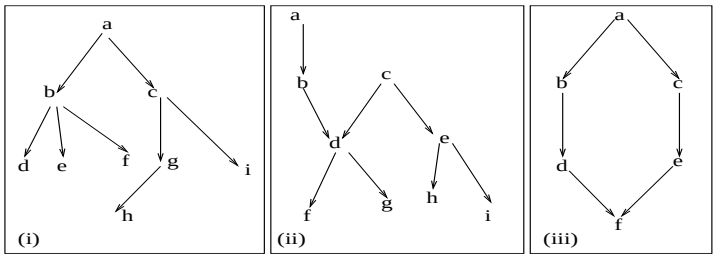

Fig. 1. (i) The DAG of a tree structure BN. (ii) The DAG of a singly connected BN. (iii) The DAG of a multiply connected BN.

Example 1. By definition, the BN in Fig. 1(i) is a tree structure BN. The BN in Fig. 1(ii) is a singly connected BN. The BN in Fig. 1(iii) is a multiply connected BN.

The classification of tree structure, singly connected, and multiply connected BNs, resulted to some extent from the historical development of different algorithms for exact inference, which are discussed in the next section.

3 Computational Complexity of Exact Inference: the Consensus

The key problem in BNs is to perform *probabilistic inference*, which means computing $p(X)$ or $p(X|Y = \beta)$, where $X \cap Y = \emptyset$, and $\beta \in V_Y$. The fact that Y is instantiated to β, i.e., $Y = \beta$, is called the *evidence*.

Algorithms for tree structure BNs and singly connected BNs were designed first. In 1982, Pearl first developed an algorithm featuring message-passing for carrying out probabilistic inference in tree structure BNs. The following year Kim and Pearl extended the algorithm to singly connected BNs. These results were summarized in [6]. For tree structure BNs, Pearl gave a complexity result for exact inference in [6]. For an m-ary tree with n values in the domain for each node in the tree structure BN, one needs to store $n^2 + mn + 2n$ real numbers and perform $2n^2 + mn + 2n$ multiplications per update for inference. Obviously, both storage and computation are efficient in tree structure BNs. For singly connected BNs, it is commonly written that the time and space complexity of exact inference in singly connected BNs is linear in the size of the networks. Here, the size is defined as the number of CPD entries. Further more, if the number of parents of each node is *bounded* by a constant, then the complexity will also be linear in the number of nodes [8].

For multiply connected BNs, the algorithms developed for singly connected BNs can be adapted to process multiply connected BNs through conditioning [8]. Nevertheless, the predominant algorithm so far is the so-called local computation method [5].

The local computation method first transforms the DAG of a BN into a secondary structure called *junction tree* through the moralization and triangulation procedures. A formal treatment on triangulation and building junction trees can be found in [8]. After constructing the junction tree, a potential (a nonnegative function) $\phi(C_i)$ is formed for each clique C_i in the junction tree. We say the size of a clique C_i is the cardinality of its domain, that is, $|V_{C_i}|$. It is easy to see that the bigger the size of a clique, the more expensive the computation will be whenever $\phi(C_i)$ is engaged in the computation for inference.

Exact inference in multiply connected BNs was developed as follows. Lauritzen and Spiegelhalter [4] first proposed the local computation method for exact inference on junction trees (also called clustering method) and showed that their method can be implemented in a computationally *feasible* manner in some real-life expert systems. The authors were concerned with the size of the clique in the junction tree (transformed from the DAG of a BN), and they realized that their method would not be computational *feasible* if a large clique is present in the junction tree. Different architectures were developed to implement the local computation method. Jensen et al. [3] provided an object-oriented version of the computational scheme in [4]. This extension forms the core of the renowned Hugin architecture. Consequently, the Hugin architecture has the same concern as the Lauritzen-Spiegelhalter architecture, namely, the size of the clique in a junction tree. The Shafer-Shenoy architecture [9] used a different propagation scheme and used hypertree and Markov tree (junction tree) to describe the architecture. In [9], it was repeatedly emphasized that the efficiency and feasibility of their architecture depends on the size of the clique in a junction tree. Cooper formally confirmed these concerns by showing that exact probabilistic inference in BNs is NP-hard [1].

To summarize, the above discussion gives rise to the consensus that singly connected BNs have efficient inference, while multiply connected BNs do not. Thus, multiply connected BNs are the core of the inference problem.

4 Inconsistency in the Consensus

Although the local computation method was originally developed with the intention to solve the problem of exact inference in multiply connected BNs, it is important to realize that it is also applicable to singly connected BNs. In other words, given a singly connected BN, besides the specifically designed algorithms in [6], one can also apply the local computation architecture to solve the inference problem in a given singly connected BN. Regarding inference in singly connected BNs, an inconsistency arises when we compare the *specifically designed algorithms* [6] with the *local computation method* [5].

Consider an application involving a singly connected BN. On one hand, if one applies the specifically designed algorithms [6], results will be returned in time *linear* to the size of the network [8]. On the other hand, if one applies the local computation architecture, one has to be cautious that the size of the cliques in the constructed junction tree should be feasible. If a large clique is present in the junction tree and the size of the clique is not feasible, then the task of inference would *not* be computational feasible even given a singly connected BN.

In other words, there are then two different claims regarding inference in a singly connected BN. One is very positive and says this is definitely efficient in time linear to the size of the network. The other is rather conservative and says this may not be computational feasible if the junction tree transformed from the given *singly connected* BN contains a large clique. These two claims are seemingly inconsistent and are more carefully examined in the next section.

5 Exploring the Inconsistency

The concern of the local computation architecture pertains to the presence of a large clique size in the junction tree, which renders the local computations intractable. When transforming a singly connected BN into a junction tree, is it possible to create a large clique? The answer is definitely yes as the following example shows.

Example 2. Consider the singly connected DAG $\mathcal{D}$ in Fig. 2 (i). Note that the node x in $\mathcal{D}$ has n parents, i.e., $y_1, \ \ldots, \ y_n$. The junction tree constructed from $\mathcal{D}$ is shown in Fig. 2 (ii). As one may notice that one of the cliques contains variables $x, \ y_1, \ \ldots, \ y_n$. If n is large, even assuming all the variables are binary, storing the potential $\phi(x, \ y_1, \ \ldots, \ y_n)$ or engaging it in any computation will not be feasible as the storage and computation will be exponential with respect to the number of variables involved.

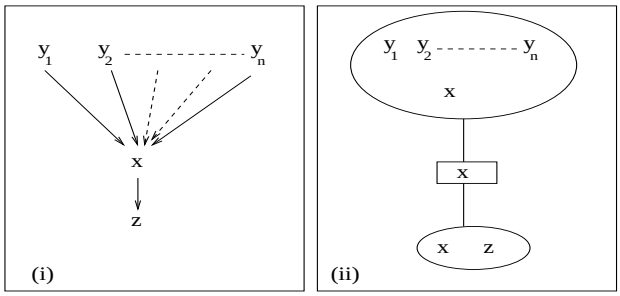

Fig. 2. (i) A singly connected DAG $\mathcal{D}$, where variable x has a large number of parents. (ii) The constructed junction tree has a large clique.

Example 2 explicitly demonstrates that a node in a singly connected BN with a large number of parent nodes must result in a large clique in the transformed junction tree. That is, it is entirely possible for a singly connected BN to have a large size clique. The presence of a large size clique will cause not only a storage problem for the corresponding CPD $p(x|y_1, \ \ldots, \ y_n)$, but also the problem of engaging $p(x|y_1, \ \ldots, \ y_n)$ in any computation during inference.

The exponential computational complexity entailed by large size cliques occurring in a junction tree hints that exact inference might be NP-hard. Cooper [1] successfully proved that the exact inference is NP-hard by transforming a well known NP-complete problem, namely, the *3SAT* problem [2], into a decision problem version of exact inference in multiply connected BNs. Since a singly connected BN, as demonstrated in Example 2, may also induce large cliques, it is thus worth exploring whether exact inference in a singly connected BN is also NP-complete.

In the following, we first demonstrate that a variant of the 3SAT problem is itself also a NP-complete problem. We then further show that this variant can be transformed into a *singly connected* BN in order to show that exact inference in singly connected BNs is also NP-hard.

5.1 A Variant of the 3SAT problem

The 3SAT problem includes a collection $C = \{c_1, c_2, \ldots, c_m\}$ of clauses on a finite set U of n Boolean variables. If u is a variable in U, then u and $\neg u$ are *literals* over U. Each clause c_i contains a disjunction of three literals over U, for example, $(\neg u_2 \vee u_6 \vee \neg u_8)$. A *truth assignment* for U is an assignment which assigns either T (true) or F (false) to each variable in U. The literal u is true if and only if the variable u is assigned T. The literal $\neg u$ is false if and only if the variable u is assigned F. Given a truth assignment, a clause is satisfied (or evaluated true) if at least one literal is true. The clause $(\neg u_2 \vee u_6 \vee \neg u_8)$ is satisfied (i.e., true) unless $u_2 = T$, $u_6 = F$ and $u_8 = T$. A collection C of clauses over U is satisfiable if and only if there exists some truth assignment for U that simultaneously satisfies all of the clauses in C. The 3SAT decision problem involves determining whether there is a truth assignment for U that satisfies all of the clauses in C. We denote an instance of the 3SAT problem as $I = (U, \ C)$.

Example 3. Consider an instance $I = (U, \ C)$ of the problem 3SAT in which $U = \{u_1, u_2, u_3, u_4\}$ and $C = \{(u_1 \vee u_2 \vee u_3), (\neg u_1 \vee \neg u_2 \vee u_3), (u_2 \vee \neg u_3 \vee u_4)\}$. One satisfying truth assignment is given by $u_1 = T$, $u_2 = F$, $u_3 = F$ and $u_4 = T$. Thus, this instance of 3SAT decision problem has the answer "yes" in this example. This example will be called $3SAT_{ex}$.

We now introduce a variant of the 3SAT problem, which will be referred to as the 3SATV problem. We then prove that the 3SATV problem is NP-complete.

Very much similar to the 3SAT problem, the 3SATV problem also includes a collection $C^{'}$ of clauses on a finite set $U^{'}$ of n Boolean variables. The only

difference between 3SAT and 3SATV is that each variable in $U^{'}$ is denoted u_i^j with not only the subscript i but also a superscript j. The 3SATV decision problem involves determining whether there is a truth assignment for $U^{'}$ that satisfies all of the clauses in $C^{'}$ and all the variables in $U^{'}$ with the same subscript are assigned the same truth value. We denote an instance of the 3SATV problem as $I^{'} = (U^{'},\ C^{'})$.

Example 4. Consider an instance $I^{'} = (U^{'},\ C^{'})$ of the 3SATV problem, where $U^{'} = \{u_1^1,\ u_1^2,\ u_2^1,\ u_2^2,\ u_2^3,\ u_3^1,\ u_3^2,\ u_3^3,\ u_4^1\}$ and $C^{'} = \{(u_1^1 \vee u_2^1 \vee u_3^1), (\neg u_1^2 \vee \neg u_2^2 \vee u_3^2), (u_2^3 \vee \neg u_3^3 \vee u_4^1)\}$. We want to determine whether there exists an truth assignment for $U^{'}$ that satisfies all of the clauses in $C^{'}$, furthermore, we require that variables u_1^1, u_1^2, u_1^3 are assigned the same truth value; variable u_2^1, u_2^2, u_2^3 are assigned the same truth value; u_3^1, u_3^2, u_3^3 are assigned the same truth value. One satisfying truth assignment is given by $u_1^j = T$ where $j = 1,\ 2,\ 3$, $u_2^j = F$ where $j = 1,\ 2,\ 3$, $u_3^j = F$ where $j = 1,\ 2,\ 3$ and $u_4^1 = T$. Thus, this instance of the 3SATV decision problem has the answer "yes". This example will be called $3SATV_{ex}$.

In the following, we will prove that the 3SATV problem is also NP-complete. We first demonstrate how one can polynomially transform any instance of a known NP-complete problem, for example, the 3SAT problem, to an instance of the 3SATV problem. We use an example to illustrate this transformation.

Consider the instance $I = (U,\ C)$ in $3SAT_{ex}$ in Example 3 and the instance $I^{'} = (U^{'},\ C^{'})$ in $3SATV_{ex}$ in Example 4. We demonstrate how one can transform $I = (U,\ C)$ into $I^{'} = (U^{'},\ C^{'})$. If we rewrite the clause set C from $3SAT_{ex}$ and the clause set $C^{'}$ from $3SATV_{ex}$ together below, one may immediately realize that the transformation is straightforward.

$$C = \{(u_1 \vee u_2 \vee u_3), (\neg u_1 \vee \neg u_2 \vee u_3), (u_2 \vee \neg u_3 \vee u_4)\},$$
$$C^{'} = \{(u_1^1 \vee u_2^1 \vee u_3^1), (\neg u_1^2 \vee \neg u_2^2 \vee u_3^2), (u_2^3 \vee \neg u_3^3 \vee u_4^1)\}.$$

The clause set $C^{'}$ is obtained by transforming each clause in C to a clause in $C^{'}$. More specifically, we transform one-to-one a clause c (in C) to a clause $c^{'}$ (in $C^{'}$) by adding a superscript j to each variable u_i occurring in the clause c to obtain the variable u_i^j which will appear in the transformed clause $c^{'}$. The superscript j indicates that the original variable u_i appears for the jth time in the clause set C. For example, consider the clause $(\neg u_1 \vee \neg u_2 \vee u_3)$ in C above. When transforming this clause to a corresponding clause in $C^{'}$, we add superscript to each variable occurring in it, namely, u_1, u_2, and u_3. Since u_1 now appears for the second time (variable u_1 appears for the first time in the clause $(u_1 \vee u_2 \vee u_3)$), it is then transformed into the variable u_1^2. Similarly, u_2 and u_3 are transformed into u_2^2 and u_3^3, respectively. Once we obtain the transformed clause set $C^{'}$, the set of Boolean variable $U^{'}$ is just the union of all the variables occurring in each clause in $C^{'}$. Obviously, this process can be generalized to be applied to any instance of the 3SAT problem in polynomial time.

Besides showing that one can transform polynomially any instance of the 3SAT problem to an instance of the 3SATV problem, in order to prove that the

3SATV problem is NP-complete, we also need to show that any instance I of the 3SAT problem is satisfiable if and only if the transformed instance $I^{'}$ of the 3SATV problem is also satisfiable.

Suppose instance I is satisfiable, that means there exists a truth assignment to the variables in U such that all the clauses in C are evaluated true. For the instance $I^{'}$, we now demonstrate a truth assignment to the variables in $U^{'}$ such that all the clauses in $C^{'}$ are evaluated true as well. For each variable u_i in U, there are variables u_i^j in $U^{'}$ which are constructed from the multiple occurrence of the variable u_i among the clauses in C. We assign the same truth value of u_i to those variables u_i^j in $U^{'}$. In other words, it u_i is assigned T(F), then u_i^j are all assigned T(F). For any clause c in C consisting of variables u_i, u_j, and u_k, according to the construction process of $I^{'}$, there is a corresponding clause $c^{'}$ in $C^{'}$ consisting of variable u_i^l, u_j^m, and u_k^n. Since u_i^l, u_j^m, and u_k^n are assigned the same truth values as those of u_i, u_j, and u_k, respectively, and clause c is satisfiable, it then follows that the clause $c^{'}$ is also satisfiable. Therefore, every clause in $C^{'}$ is satisfiable.

Suppose the instance $I^{'}$ is satisfiable, we now need to show that there exists a truth assignment to the variables in U under which every clause in C is satisfiable. Consider a truth assignment to the variables in $U^{'}$ such that each clause $c^{'}$ in $C^{'}$ is satisfiable. For variables u_i^j in $U^{'}$, they are all assigned the same truth value, we then assign the same truth value assigned to u_i^j to the variable u_i in U. We thus obtain a truth assignment to every variable in U. Suppose the clause $c^{'}$ in $C^{'}$ consists of variable u_i^l, u_j^m, and u_k^n. According to the construction process described early, there is a corresponding clause c in C consisting of variables u_i, u_j, and u_k. Since u_i, u_j, and u_k are assigned the same truth values as those of u_i^l, u_j^m, and u_k^n, respectively, it then follows that the clause c in C is satisfiable. Therefore, every clause c in C is satisfiable.

The above discussion in fact proves the following theorem.

Theorem 1. The 3SATV problem is NP-complete.

5.2 The Complexity of Exact Inference in Singly Connected BNs

To prove that a problem Q' is NP-hard, it is sufficient to transform a known NP-complete problem Q to Q' and to show that this transformation can be done in time that is polynomial in the size of Q. In this subsection, we transform the 3SATV problem to a decision-problem version of probabilistic inference using singly connected BNs (*PISBND*). The transformation from the PISBND decision problem to the probabilistic inference problem, called *PISBN*, will be straightforward. Therefore, we will show that PISBN is NP-hard.

We first show how to polynomially transform 3SATV into PISBND, a decision problem that determines whether $p(Y = T) > 0$ in a given singly connected BN. PISBN returns "yes," if $p(Y = T) > 0$; it returns "no," otherwise.

Let $I^{'} = (U^{'}, C^{'})$ be any instance of the 3SATV problem. We seek to construct a singly connected BN on $U^{'}C^{'}Y$ from any instance of 3SATV in polyno-

mially time, where Y is a new variable, such that $p(Y = T) > 0$ if and only if $C^{'}$ is satisfiable.

The nodes in the constructed singly connected BN are $U^{'} \cup C^{'} \cup \{Y\}$. Each variable $u_i^j \in U^{'}$ is represented as a node u_i in the singly connected BN. Each clause $c_i \in C^{'}$ is represented as a node c_i in the singly connected BN. For each clause $c_i \in C^{'}$, let the three literals in c_i be denoted w_i^1, w_i^2 and w_i^3. For instance, given clause $c_2 = (\neg u_1 \vee \neg u_2 \vee u_3)$, then $w_2^1 = u_1$, $w_2^2 = u_2$, $w_2^3 = u_3$. The edges can now be defined as follows. For each node $c_i \in C^{'}$, there is a directed edge from each of the three literals w_i^1, w_i^2 and w_i^3 to c_i. Finally, there are directed edges from each $c_i \in C^{'}$ to variable Y.

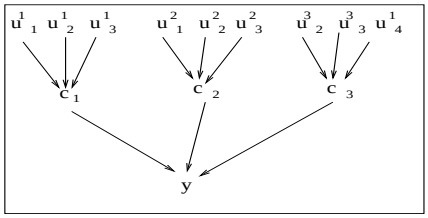

Fig. 3. A singly connected BN transformed from $3SATV_{ex}$.

Example 5. Given the example $3SATV_{ex}$, the DAG of the constructed singly connected BN is shown in Figure 3.

The CPDs for the singly connected BN are now constructed. For each of the root nodes $u_i^j \in U^{'}$, the CPD $p(u_i^j)$ is $p(u_i^j = T) = 1/2$. For each of the clause nodes $c_j \in C^{'}$, the CPD $p(c_j|w_j^1, w_j^2, w_j^3)$ is defined as follows. If clause c_j is T, then $p(c_j = T|w_j^1, w_j^2, w_j^3) = 1$; otherwise, if clause c_j is F, then $p(c_j = T|w_j^1, w_j^2, w_j^3) = 0$. The last CPD to construct is $p(Y|c_1, c_2, \ldots, c_m)$. If $c_1 = T$, and $c_2 = T$, ..., and $c_m = T$, then $p(Y = T|c_1, c_2, \ldots, c_m) = 1$; otherwise, $p(Y = T|c_1, c_2, \ldots, c_m) = 0$. That is, if at least one clause $c_i = F$, then $p(Y = T|c_1, c_2, \ldots, c_m) = 0$. We now show the claim in the next result.

Theorem 2. Let $I^{'} = (U^{'}, C^{'})$ be any instance of the 3SATV problem. Consider the singly connected BN on $U^{'}C^{'}Y$ constructed as above. Then $C^{'}$ is satisfiable if and only if $p(Y = T) > 0$ in the constructed singly connected BN. [4]

Thus, we have shown that any instance of 3SATV can be polynomially transformed to PISBND. This result implies that PISBN is NP-hard.

6 Concluding Remarks

Our analysis raises the question as to why inference in singly connected BNs is considered to be efficient. The complexity of exact inference in singly connected

[4] Due to page limit, the proof will appear in an extended version of this paper.

BNs was written as $O(N \cdot q^e)$ in [6], where N is the number of variables in the BN, q denotes the cardinality of the domain of each variable, and the number of parents for each variable in the BN is bounded by e. Obviously, if e is bounded and q is fixed, q^e is the coefficient of N and $O(Nq^e)$ could be considered linear and not exponential. However, if the number of parents for each variable in the BN is not bounded by e, then as N grows bigger, e may also grow bigger (e can be as large as $N-1$), then the complexity $O(Nq^e)$ perhaps can not be simply considered as linear(or polynomial) anymore.

Our investigation in the previous sections has showed that the feasibility of exact inference lies with whether the DAG of a BN contains a node with a large number of parent nodes (which causes exponential storage and computation). The presence of a node with a large number of parents can occur in both singly connected and multiply connected BNs. Therefore, in both singly and multiply connected BNs, the computation for exact inference will be exponential in the worst case. On the contrary, the computational cost in tree structure BNs is efficient as characterized by Pearl in [6]. This is a result of the fact that, by definition, every node in a tree structure BN has at most one parent node. Subsequently every clique in the constructed junction tree will contain only two nodes, i.e., no large size clique will ever be created.

References

[1] G.F. Cooper. The computational complexity of probabilistic inference using bayesian belief networks. *Artificial Intelligence*, 42(2-3):393–405, 1990.

[2] M.R. Garey and D.D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York, 1979.

[3] F.V. Jensen, S.L. Lauritzen, and K.G. Olesen. Bayesian updating in causal probabilistic networks by local computation. *Computational Statistics Quarterly*, 4:269–282, 1990.

[4] S.L. Lauritzen and D.J. Spiegelhalter. Local computation with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society*, 50:157–244, 1988.

[5] Vasilica Lepar and Prakash P. Shenoy. A comparison of Lauritzen-Spiegelhalter, Hugin, and Shenoy-Shafer architectures for computing marginals of probability distributions. In Gregory F. Cooper and Serafín Moral, editors, *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence (UAI-98)*, pages 328–337, San Francisco, July 24–26 1998. Morgan Kaufmann.

[6] J. Pearl. Fusion, propagation, and structuring in belief networks. *Artificial Intelligence*, 29:241–288, 1986.

[7] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers, San Francisco, California, 1988.

[8] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (2nd edition)*. Prentice Hall, Englewood Cliffs, New Jersey, 2003.

[9] G. Shafer. *Probabilistic Expert Systems*. Society for Industrial and Applied Mathematics, 1996.