CORE
Metadata, citation and similar papers at core.ac.uk
Provided by Teeside University's Research Repository

Learning Local Components to Understand Large Bayesian Networks

Yifeng Zeng
Dept. of Computer Science
Aalborg University
Denmark
yfzeng@cs.aau.dk

Yanping Xiang
Dept. of Computer Science
Uni. of Electronic Sci. and Tech. of China
P. R. China
yanping_xiang@yahoo.com.cn

Jorge Cordero H. and Yujian Lin
Dept. of Computer Science
Aalborg University
Denmark
corde,yjlin@cs.aau.dk

***Abstract*—Bayesian networks are known for providing an intuitive and compact representation of probabilistic information and allowing the creation of models over a large and complex domain. Bayesian learning and reasoning are nontrivial for a large Bayesian network. In parallel, it is a tough job for users (domain experts) to extract accurate information from a large Bayesian network due to dimensional difficulty. We define a formulation of *local components* and propose a clustering algorithm to learn such local components given complete data. The algorithm groups together most inter-relevant attributes in a domain. We evaluate its performance on three benchmark Bayesian networks and provide results in support. We further show that the learned components may represent local knowledge more precisely in comparison to the full Bayesian networks when working with a small amount of data.**

I. Introduction

Bayesian network (BN) [1] is a directed acyclic graph where nodes represent variables (or attributes) of a subject of matter, and arcs between the nodes describe the causal relationship of variables (or attributes). It is a tedious job for domain experts to construct a BN from domain knowledge. Instead, they resort to possible methods for learning the BN if data is available in the domain. More about the specific methods is discussed and summarized in an experimental comparison regarding their learning ability and capability [2]. It shows that building a large Bayesian network is still a piece of tough work in a complex domain. The large domain presents much difficulty for the determination of causal relationships among the variables. Matters are more serious when there are relatively few data since the data are insufficient to structure a reliable and accurate network.

On other aspect, even having a large BN that has been successfully learned from the data, users (or domain experts) still find it hard to analyze the BN due to familiar dimensional difficulty. Some users are often lost in a large and complex network. More often, they choose to study each portion of the large BN that is a small size of BN representing specialized local domain knowledge. By doing so, they would not be interfered by other irrelevant (or weakly relevant) variables in the large network. In some cases, they may be interested in a particular portion hereby it is **not** necessary to learn a full BN from the data. For instance, some users are only interested in either the left ulnaris or right ulnaris in the MUNIN network (the full network consists of thousands of nodes) [3]. It would be more useful and efficient to present them the specified portion of the MUNIN network instead of exposing them the full network that must be learned using computation intensive learning techniques. Hence, the twin problems, limitations of conventional learning methods and complex representation of a full network, arise of learning a small portion of network that would provide a more proper way on understanding a large BN.

In this paper, we first define a portion of BN as a *local component*, and then propose a clustering algorithm to learn local components from the data. We discuss two properties of local components that project a sufficient representation of local domain knowledge. The property proposal makes it possible to learn local components automatically from the data.

We do not intend to learn the full BN, but propose to find local components automatically in the learning process. We inspire the clustering algorithm from the identification of local structures in a general complex network [4], and adapt the star discovering approach in the BN decomposition learning algorithm [5]. The research on complex networks reflects that most network structures are not random and most relevant nodes are close and reside in a neighboring position. We may discover a hidden, but natural, local structure from a constructed graph through the connectivity analysis of networks.

Following the same vein, the clustering algorithm first finds a set of clusters from an initial dependency graph in an iterative way. The dependency graph is an expansion of a tree structure and is built directly from the data. It structures most relevant variables in a regular way. Given the detected cluster variables, the algorithm utilizes any of BN learning approaches to construct the final local components into small BNs. We show experimental results on three benchmark networks and demonstrate the algorithm performance regarding the learning and reasoning accuracy. More importantly, we verify that the learned local component is sufficient to represent local domain knowledge in a large network.

II. Related Work

The idea of using small BNs to represent local domain knowledge is not new on a large BN reasoning. Xiang [6] provided an early piece work on multiply sectioned Bayesian network (MSBN). The MSBN is a large BN that contains a set of connected local BNs. Each local BN is formulated carefully to model local knowledge so that an exact propagation is guaranteed in the large BN. Currently, an example of MSBN is constructed manually by domain experts. Similar work includes network fragments in multi-entity BN [7]. Another branch work proposed mixture component densities to approximate BN so as to achieve tractable inference [8]. Most of the above work does not refer to data-driven construction of BN.

Druzdzel [9] used a local model, called pICI model, to improve the BN parameter learning whenever there are large conditional probability tables. However, the local model is formulated by partitioning a given network structure. We also notice that local structures were examined to improve the quality of learning BN structures [10]. It shows that the learning requires fewer parameters while resulting in a more complex network structure. In a parallel line, Eran *et al.* [11] proposed a formulation of *module* in a large BN for a special learning task. A module contains a set of variables that exhibit similar behavior. More precisely, the module variables must share the same parents and conditional probability distribution, and the module may not be equivalent to local models. The restriction makes it possible for learning a large BN of thousands of variables.

One additional relevance is the k-modes algorithm on the attribute clustering [12]. The algorithm is one of the most efficient methods on clustering attributes. Similar to the k-means, it is subject to local optima due to a random selection of initial modes. Current work shows that the star discovering procedure outperforms the k-modes algorithm in Bayesian domains [13]; thereby, we adapt the star discovering procedure in this paper.

III. Learning Local Components

We start with the property of local components and move to an approach for learning local components from data.

A. Local Components

A local component is a small size of Bayesian network that represents local domain knowledge. A full Bayesian network may contain several local components that are disjoint or share a set of common nodes. To ease the illustration, we denote a local component as $B = \{G, P\}$ where $G = \{V, E\}$ is a directed acyclic graph having a set of nodes V connected by directed arcs E [1], and P is the probability distribution over V. Moreover, we need some guideline to facilitate the learning of local components from the data. Formally, a local component shall satisfy the following two properties.

Property 1: Local Dependency. The variables within a local component have a strong inter-dependency.

The dependency is weighted by a correlation function such as mutual information [14]. Assume that the local component B_i has the component center o_i, the weight sum is defined: $W^{B_i} = \sum_{v_j \in V_i/o_i} w_{o_i,v_j}$. Given the complete data $D = (d_{1,l}, \cdots, d_{n,l})$ where $d_{i,l}$ represents the sample indexed by l for attribute d_i [2], we aim to find a set of local components $B = (B_1, \cdots, B_m)$ that maximize the weight sum over the set of components: $\sum_{B_i \in B} W^{B_i}$.

Property 2: Sufficiency. A local component is sufficient to learn local domain knowledge without querying other components.

The second property examines the goodness of a local component. The sufficient representation could be evaluated by investigating a Markov blanket of component variables and querying component variables given specific evidences in the local component. The Markov blanket of a variable v_i is the set consisting of the parents of v_i, the children of v_i, and the variables sharing a child with v_i [1]. Given its Markov blanket, the variable v_i is conditional independent from other variables in a BN.

We notice that the first property points out the basic principle for constructing a local component. The resulted local components may expect to fulfill the second property partially. The two properties suggest our new algorithm in the next section.

B. The Learning Algorithm

The main approach we propose in this paper is the algorithm for learning local components. The basic idea adopts mutual information between pairs of discrete random variables as a correlation function in order to group variables into a set of clusters. Then, a local component is learned given cluster variables and domain data. The learning component algorithm consists of three main phases: *Capturing Dependency*, *Clustering Variables*, and *Recovering Components*.

Prior to presenting the learning algorithm, we explain some denotations. We introduce a distance function, $Dist(v_i, v_j)$, to measure the length between a pair of nodes, v_i and v_j, in a graph. For instance, $Dist(v_i, v_j)$ is equal to 1 if v_i and v_j are adjacent and linked by the edge $e_{i,j}$. The $Deg(v_i)$ function returns the degree of the node v_i. The algorithm is detailed in Fig. 1.

Phase 1. We construct and expand a maximum spanning tree [15] to build an initial dependency graph (lines 1-6). We use mutual information $MI(v_i, v_j)$ to evaluate the dependency between two variables v_i and v_j. The mutual

[1] Later, we may abuse E for a set of undirected edges in other graphs.

[2] In this paper, both attributes d_i in data and nodes or vertices v_i in graphs represent random variables in the domain. They are not further distinguished.

Learning Local Components
Input: Data $D = (d_{1,l}, \cdots, d_{n,l})$, θ
Output: $B = (B_1, \cdots, B_m)$

Phase 1: Capturing Dependency

1: Compute a complete graph $CG = (V^{CG}, E^{CG})$ with weights $W^{CG} = (w_{i,j}^{CG} = MI(v_i, v_j) | i, j = 1, \cdots, n$ and $i \neq j)$
2: Construct a maximum spanning tree $M = (V^M, E^M)$ with weights W^M
3: FOR each $v_i \in V^M$ **DO**
4: Compute $AW(v_i) = \frac{\sum_{v_j \in V^M} w_{i,j}}{Deg(v_i)}$ ▷ $AW(v_i)$: average ▷ weight for v_i
5: FOR each $v_i \in V^{CG}$ **DO**
6: IF $w_{i,j}^{CG} > AW(v_i)$ **THEN**
7: Add $w_{i,j}^{CG}$ and $e_{i,j}^{CG}$ into W^M and E^M respectively

Phase 2: Clustering Variables

8: WHILE $V^S \neq \emptyset$ **DO**
9: FOR each $v_i \in V^M$ **THEN** ▷ Generate a star ▷ $S_i = (V^{S_i}, E^{S_i})$ with the weight sum W^{S_i}
10: Add v_i into the set V^{S_i} ▷ Initialize S_i with ▷ the star center $o_i = v_i$
11: Add v_j into the set V^{S_i} iff $Dist(o_i, v_j) \leq 2$ ▷ $v_j = (v_{j_1}, v_{j_2} | Dist(o_i, v_{j_1}) = 1$, ▷ $Dist(o_i, v_{j_2}) = 2)$
12: Add v_h into the set V^{S_i} iff $Deg(v_h) = 1$ and $Dist(v_h, v_{j_2}) = 1$
13: Add e_{i,j_1}, e_{j_1,j_2} and e_{h,j_2} into the set E^{S_i}
14: Compute the weight sum for S_i: $W^{S_i} = \sum(w_{i,j_1} + w_{j_1,j_2} + w_{h,j_2})$
15: Find a cluster $C_k \leftarrow V^{S_i}$ iff $S_i = \underset{S_i \in S}{\text{argmax}}(W^{S_i} \in W^S)$
16: Remove star edges: $E^M \leftarrow (E^M - e_{i,j_1})$ iff $e_{i,j_1} \in E^S$
17: Compose a set of clusters $C \overset{\cup}{\leftarrow} C_k$
18: $V^S \leftarrow (V^S - C)$
19: IF $\frac{|C_i \cap C_j|}{|C_i|} \geq \theta$ **THEN**
20: Combine C_i and C_j, $C \leftarrow (C - C_i)$

Phase 3: Recovering Components

21: FOR each $C_i \in C$ **THEN**
22: Learn B_i using any BN learning method
23: Compose a set of local components $B \overset{\cup}{\leftarrow} B_i$

Figure 1. The learning local component algorithm contains three phases. The first phase outputs the dependency graph that is an expansion of the maximum spanning tree. Subsequently, a set of clusters are discovered from the graph and constructed into a set of local components.

information measures an average reduction in uncertainty about v_i that results from learning values of v_j. We compute $MI(v_i, v_j)$ for all pairs of variables and build a complete graph in which each edge $e_{i,j}$ connecting two variables, v_i and v_j, has weight $w_{i,j}$ (or $w_{i,j}^{CG}$)[3] (line 1). Given the complete graph, we build the maximum spanning tree M using a modified version of the Kruskal's algorithm [16] (the original Kruskal's algorithm for finding the minimum spanning tree sorts the weights increasingly instead of decreasingly) (line 2). The construction results in $n-1$ edges in the tree M.

The maximum spanning tree is the smallest graph that optimally approximates the probability distribution between the variables. However, some strong dependency may be lost since the tree structure needs to be preserved in the construction process. To retrieve such dependency, we expand the tree by adding more edges into the already built tree M (lines 3-7). We compute the average weight $AW(v_i)$ for every node v_i in the tree. It is the ratio of the weight sum of edges ($w_{i,j}$ connecting v_i to its adjacent nodes v_j in M) to v_i's degree (line 4). The average weight becomes the lower bound when we are adding possible edges. We consider all edges $e_{i,j}^{CG}$ that link v_i to other nodes in the complete graph CG. If the edge $e_{i,j}^{CG}$ has a larger weight than the computed average weight, it is retrieved and added into M (line 6-7). Consequently, the expansion ensures most of the largely weighted edges to be kept for each variable in the dependency graph. The resulted graph M contains n nodes, $V^M = (v_1, \cdots, v_n)$, and generally more than $n-1$ edges each of which is weighted by the mutual information $MI(v_i, v_j)$. The dependency graph captures the most relevant connections among n variables.

Most computation occurs in constructing the complete graph. The complexity takes the order of $O(n^2)$. For building the maximum spanning tree, we use an union-finder data structure and a sorted list in the modified Kruskal's algorithm and the complexity is in the order of $O(n \log n)$.

Phase 2. Given the resulted dependency graph, we group the domain variables into a set of clusters. Each cluster consists of a subset of domain variables that have strong dependency. This phase is an iterative process on composing the set of clusters. Each iteration examines whether the established clusters have already contained all domain variables (line 8). In the beginning, we build n stars, $S = (S_1, \cdots, S_n)$. Each star is a graph, $S = \{V^{S_i}, E^{S_i}\}$, that contains nodes V^{S_i} and edges E^{S_i} connecting them (lines 9-14). A star has the selected node v_i as the star center o_i (line 10). Then, we expand the star by adding two types of nodes: one is within the distance of **2** from the star center $o_i (= v_j)$ (line 11) and the other is a leaf node($Deg(v_h) = 1$) connected to the nodes already included in the star (line 12). For convenience, we denote the nodes as v_{j_1} and v_{j_2} that are away from the start center with the distances of 1 and 2 respectively. We choose the distance value ($Dist(v_i, v_j) \leq 2$) considering that v_i has the largest distance of two from other nodes v_j within v_i's Markov blanket. We include all potential nodes in a greedy way. In addition, we compute the weight W^{S_i} for each star S_i by summing up all edge weights (line 14). The weight reflects the dependency among star variables.

We select the star as a cluster that has the largest weight

[3]The superscript denotes the holder of variables such as a complete graph CG, a maximum spanning tree M, and later a star S_i, and is ignored if the indication is already clear in the text.

in the set of stars (line 15). Note that a cluster consists of only nodes without edges. Once one star becomes the new cluster, we remove edges e_{i,j_1} from the dependency graph M that connect the center of the elected star to its adjacent nodes (line 16). This step is necessary since we need to weaken the impact of the established cluster on the selection of a new cluster in the next iteration. We do not remove other star edges because they may connect cluster outliers and relate to future clusters. The reduced dependency graph enters a new iteration in which a new cluster emerges from the selection of stars. The process terminates until all nodes are exhaustively clustered.

Some of the established clusters may have a set of overlapping nodes. We proceed to merge two clusters into a larger one if any of them has at least a θ percentage of common nodes (line 19-20). It was empirically found that setting θ to 0.5 produced a reasonable amount of clusters, and a setting of $\theta = 1$ provided the highest number of clusters. Formally speaking: Let $|C_i|$ be the number of nodes in clusters C_i, the percentage of common nodes between C_i and C_j for $|C_i|$ is at least θ iff $\frac{|C_i \cap C_j|}{|C_i|} \geq \theta$.

The complexity of phase 2 is dominated by the iterative construction of stars and cluster selection in each iteration. Assume having k numbers of clusters built iteratively, we need to take $O(kn^3)$ operations searching for all nodes within a certain distance.

Phase 3. Each cluster has a subset of local domain variables and will be constructed into a local component. The second phase finds most relevant variables for each local component, and then we need to structure the variables in the local component. Note that the expanded tree structure (using the measurement of mutual information in the first phase) is only utilized to find a set of clusters in the initial dependency graph and will not function in this phase.

We use any of available BN learning methods to learn each local component (line 22). It includes both the structure and parameter learning. The structure learning links component variables using directed arcs E while the parameter learning provides conditional probability distributions P in the local component. The complexity of this phase depends on the selected learning technique. For example, regarding the structural learning method, if the PC algorithm is used, the complexity is in the order of $O(mq^r)$, where r is the largest size of parents for a node, q is the largest component size. In general $q \ll n$, the complexity of learning local components is trivial in comparison with learning the full network.

IV. Experimental Results

We take several benchmark networks to evaluate the performance of the local component learning algorithm. Three of them are simply described in Table 1. Table 1 depicts the number of variables for each domain and all sample sizes.

Table I
Domains, number of variables and different sample sizes used in the experiments.

Domain	$\|V\|$	Sample Sizes
HeparII	70	210∼20K
Win95PTS	76	228∼20K
Andes	223	669∼20K

As for the sufficiency of local components, we demonstrate that the algorithm learns an accurate structure of local components representing local domain knowledge. It shows that structures of local components are even more representative, using the measurement of Markov blankets, than the full network when working with a small sample size. More importantly, we show local components response quite well in the reasoning task when queries are proposed within a single component.

A. Experiment 1: Structural Tests

The experiments compared both the local component and the full BN structure learned from the same sample size against the *true* BN [4]. We use the PC algorithm to learn both the local component (phase 3 in Fig. 1) and the full BN structures. Note that we learn the full BN directly from the data without local components.

The evaluation targets at the property of local components on the sufficient knowledge representation. We consider the Markov blanket for the comparison measurement and define two evaluation criteria, $\lambda_1(v_i)$ and $\lambda_2(v_i)$, in Eq. 1.

$$\lambda_1(v_i) = \frac{|MBL(v_i) \cap MBT(v_i)|}{|MBT(v_i)|}$$
$$\lambda_2(v_i) = \frac{|MBL(v_i) \cap MBT(v_i)|}{|MBL(v_i)|} \quad (1)$$

where $MBL(v_i)$ denotes the *learned* Markov blanket of v_i in the local component (or in the full BN if the full network is learned directly from the data), and $MBT(v_i)$ the *true* Markov blanket of v_i in the true BN [5]. For the case when the variable v_i resides in different local components, we take the Markov blanket that has the largest size among all the local components.

As shown in Eq. 1, $\lambda_1(v_i)$ measures the ability of the learning algorithm to identify the Markov blanket in the true Bayesian network. The second criterion $\lambda_2(v_i)$ is the ratio of the true Markov blankets to all of the Markov blankets found in the local components (or in the full BN if the full network is measured). It evaluates the accuracy of the learning algorithm to identify proper Markov blankets. We compute the average values of $\lambda_1(v_i)$ and $\lambda_2(v_i)$ respectively for all variables $v_i \in V$, and denote them as λ_1 and λ_2.

[4] We take the benchmark networks as the true BN.

[5] We defined: $\lambda_1(v_i) = 1$ and $\lambda_2(v_i) = 1$ if both $|MBL(v_i)|$ and $|MBT(v_i)|$ are equal to 0; $\lambda_1(v_i) = 0$ and $\lambda_2(v_i) = 0$ if either $|MBL(v_i)|$ or $|MBT(v_i)|$ is equal to 0, but not both.

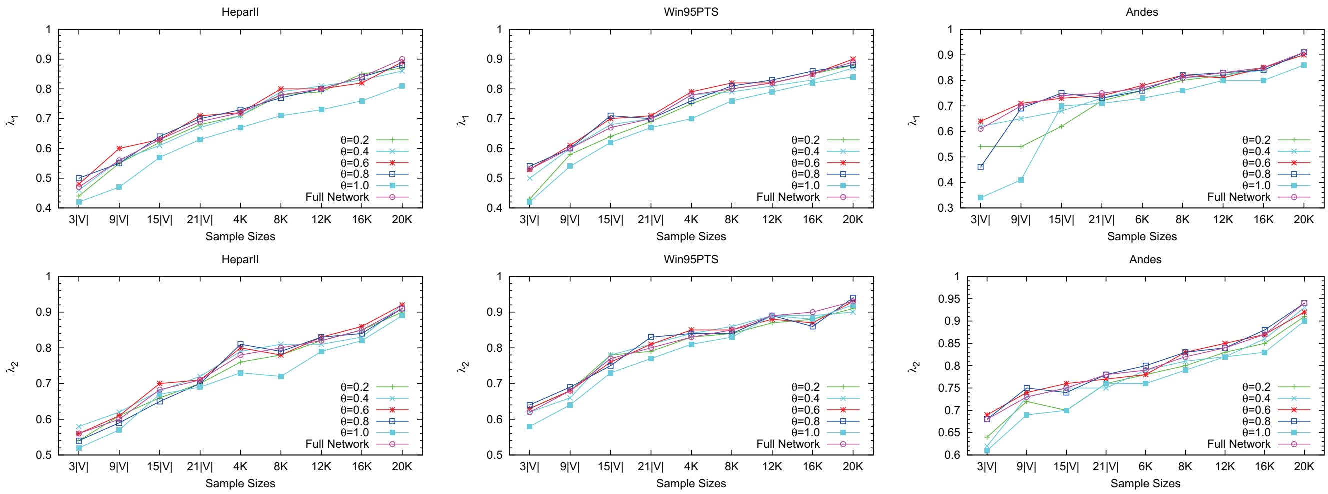

Figure 2. Performance of λ_1 and λ_2. The crowd curves indicate the learned local component has almost the same ability of accurately representing local domain knowledge as the full Bayesian network. A good selection of the θ value produces better results, which is particularly true when a large Bayesian network (e.g. Andes domain) is expected to be learned from a small sample size.

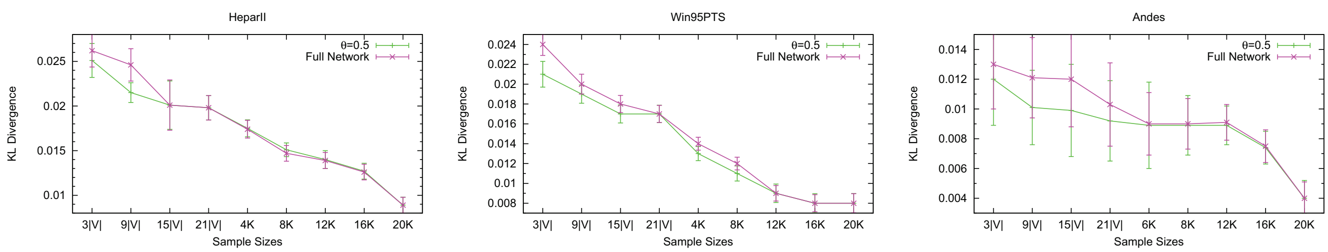

Figure 3. Performance of KL divergence. Both representations have relatively low KL values (with small deviations) and the local components achieve better reasoning results on a small sample size.

Fig. 2 presents the results for the λ_1 and λ_2 estimates in terms of the θ values and sample sizes. For a specific θ value, we tested the learning algorithm across small and sufficient sample sizes. We find that the larger the sample size, the better the performance of the λ_1 and λ_2 estimates. This is true for both the learned local component and the learned full BN. In most cases, the local components contain the same Markov blankets as the full BN learned directly from the data. For a large sample size, there is no significant difference on the structural estimation of both the local components and the full BN. For a small sample size, we find a more accurate Markov blanket when a proper value of θ is selected in the algorithm. This is because a small sample size is not statistic enough to detect the real conditional independence for learning the full BN. A larger θ value generates more local components each of which is a relatively small corresponding to the sample size. Hence, as for the sufficient statistic, learning local components results in better local structures than the learned full BN regarding a small sample size. We shall note that the selection of θ values may affect the performance. For instance, the setting of $\theta = 1$ leads to too many clusters each of which may contain a small amount number of nodes. Consequently, it is difficult to recover a correct Markov blanket.

We carefully make some conclusions after having analyzed the experimental results: The first one is that in most of the cases the increment in the sample size logically produces better structures. The second one is that for every sample size, the learned local components achieve a similar representation of local knowledge comparing with the learned full BN. In some cases, the local components may have a more accurate representation of local domain knowledge than the full network. The approach of learning local components has more advantages for recovering network structures from a small sample size.

B. Experiment 2: Reasoning Comparison

In this experiment,we empirically investigate how well the reasoning is performed in the learned local components comparing with the full BN. Similar to experiment 1, we use the PC algorithm to learn the structures of both the local components (using $\theta = 0.5$) and the full BN. Then, we learn their parameters through the maximum likelihood estimation [17].

We randomly choose a set of evidence nodes, $EN^{BN} = \{EN^{B_1}, \cdots, EN^{B_m}\}$, that are scattered in all of the resulted local components, and perform the propagation after a certain evidence is entered into the selected nodes. We use the junction tree algorithm [1] for the reasoning, and get the posterior probability, $Pr_{v_i \in (V^{B_i}/EN^{B_i})}(v_i|EN^{B_i})$, for each of the rest nodes conditioned on the evidence in each local component. We may get different probabilities for some of the rest nodes since the nodes may appear in different local components. In this case, we return their average probability values.

We do the same thing (selecting the same evidence nodes and evidence) in the true BN. By doing so, for each node v_i, we may obtain two (different) posterior probability values: the one, $Pr_{v_i \in (V^{B_i}/EN^{B_i})}(v_i|EN^{B_i})$, is computed from the learned local components, and the other, $Pr_{v_i \in (V^{BN}/EN^{BN})}(v_i|EN^{BN})$, from the true BN. We compute the Kullback-Liebler (KL) divergence between these two probabilities, and get the average KL values for all of the rest nodes. We repeat the selection and propagation for 10 times in both the local components and the true BN, and report the average of the average KL values.

Similarly, we get the average KL divergence between the posterior probabilities in the full BN and those in the true BN. We show the comparison in Fig. 3.

For all three domains, the KL divergence is lower than 0.03 (an insignif cant number when thinking about KL estimates) over different sample sizes. In general, the reasoning results in the local component prove to be at least as accurate as the ones in the full BN. For a small sample size ($3|V|$ to $21|V|$), the local components have a lower discrepancy with respect to the true BN than the full BN. This may be resulted from the cascading effect of errors in both the parameter and structure learning of the full BN when the BN contains a large number of nodes. For a suff cient sample size the local components perform the propagation as well as the full BN. We conclude that the local components are suff cient to provide accurate and reliable answers to initiated queries. It is not necessary to learn the full BN and then perform the inference in the large network, which is often a time-consuming task .

ACKNOWLEDGMENT

The f rst author acknowledges partial support from both the Obel Family Foundation in Denmark and National Natural Science Foundation of China (No. 60974089 and No. 60975052). Yanping Xiang thanks the support from National Natural Science Foundation of China (No. 60974089).

REFERENCES

[1] F. V. Jensen, *An introduction to Bayesian networks*. New Work: Springer, 1996.

[2] I. Tsamardinos, L. E. Brown, and C. F. Aliferis, "The max-min hill-climbing bayesian network structure learning algorithm," *Machine Learning*, vol. 65, no. 1, pp. 31–78, 2006.

[3] K. G. Olesen, U. Kjarulff, F. Jensen, F. V. Jensen, B. Falck, S. Andreassen, and S. K. Andersen:, "A munin network for the median nerve - a case study in loops," *Applied Artif cial Intelligence*, no. 3, pp. 385–404, 1989.

[4] R. Cohen, S. Havlin, and D. ben Avraham, *Structural Properties of Scale Free Networks*, ser. Handbook of graphs and networks, chp. 4. Berlin GmbH: WILEY-VCH, 2002.

[5] Y. Zeng and J. C. Hernandez, "A decomposition algorithm for learning bayesian network structures from data," in *Proceedings of the Twelfth Pacif c-Asia Conference on Knowledge Discovery and Data Mining(PAKDD)*, 2008, pp. 441–453.

[6] Y. Xiang, *Probabilistic Reasoning in Multiagent Systems: A Graphical Models Approach*. Cambridge University Press, 2002.

[7] K. B. Laskey and S. M. Mahoney, "Network fragments: Representing knowledge for constructing probabilistic models," in *Proceedings of the Thirteenth Conference on Uncertainty in Artif cial Intelligence*, 1997, pp. 334–341.

[8] V. Tresp, M. Haft, and R. Hofmann, "Mixture approximations to bayesian networks," in *Proceedings of the Fifteenth Conference on Uncertainty in Artif cial Intelligence*, 1999, pp. 639–646.

[9] A. Zagorecki, M. Voortman, and M. J. Druzdzel, "Decomposing local probability distributions in bayesian networks for improved inference and parameter learning," in *Proceedings of the Nineteenth International Florida Artif cial Intelligence Research Society Conference*, 2006, pp. 860–865.

[10] N. Friedman and M. Goldszmidt, "Learning bayesian networks with local structure," in *Proceedings of the NATO Advanced Study Institute on Learning in graphical models*, 1998, pp. 421–459.

[11] E. Segal, D. Pe'er, and A. Regev, "Learning module networks," in *Proceedings of the Nineteenth Conference on Uncertainty Artif cial Intelligence*, 2003, pp. 525–534.

[12] W.-H. Au, K. Chan, A. Wong, and Y. Wang, "Attribute clustering for grouping, selection, and classif cation of gene expression data," *IEEE Trans. on Computational Biology and Bioinformatics*, vol. 2, pp. 83–101, 2005.

[13] Y. Zeng, J. C. Hernandez, and S. Lin, "Attribute clustering based on heuristic tree partition," in *Proceedings of the Thirteenth Pacif c-Asia Conference on Knowledge Discovery and Data Mining(PAKDD)*, 2009, pp. 681–688.

[14] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, 2nd ed. New York: Wiley-Interscience, 2006.

[15] C. Chow and C. Liu, "Approximating discrete probability distributions with dependence trees," *IEEE Transaction on Information Theory*, no. 12, pp. 462–467, 1968.

[16] R. Sedgewick, *Algorithms in Java, Part 5 Graph Algorithms*. Addison Wesley, 2004.

[17] D. Heckerman, "A tutorial on learning in bayesian networks," *MSR-TR-95-06*, no. 3, pp. 1–57, 1995.