A Decomposition Algorithm for Learning Bayesian Network Structures from Data

Yifeng Zeng and Jorge Cordero Hernandez

{yfzeng,jorgecordero}@cs.aau.dk
Dept. of Computer Science, Aalborg University, DK-9220 Aalborg, Denmark

Abstract. It is a challenging task of learning a large Bayesian network from a small data set. Most conventional structural learning approaches run into the computational as well as the statistical problems. We propose a decomposition algorithm for the structure construction without having to learn the complete network. The new learning algorithm firstly finds local components from the data, and then recover the complete network by joining the learned components. We show the empirical performance of the decomposition algorithm in several benchmark networks.

Key words: Bayesian Networks, Graphical Models, Structure Learning

1 Introduction

Bayesian networks (BN) [1, 2] are widely used to represent probabilistic relationship among random variables. They have been successfully applied in many domains such as medical diagnosis, gene data analysis, and hardware troubleshooting. Over the last decade, much progress has been made regarding structural learning in Bayesian networks including both the score-based and the constraint-based learning methods [3–6]. The score-based method tries to optimize a scoring function by means of a search strategy. Since finding the optimal Bayesian networks was shown to be an NP-complete problem [7], one has to resort to some heuristic search strategy. The other is constraint-based and infers structures through conditional independency tests. The constraint-based is generally faster than the score-based method and gives a trustworthy result provided there are sufficient data. We focus on the constraint-based learning methods in this paper. Currently, with the mass-throughput data in biomedical informatics, data analysis demands more powerful learning algorithms that could handle data sets having thousands of variables but with limited sample sizes. Most conventional learning methods run into the computational and statistical problems in such a domain. They either can't complete the learning process, or produce a poor structure even when the learning is done. Hence, in this paper, we propose a decomposition algorithm for learning a large Bayesian network from a small amount of data.

The decomposition learning algorithm adopts the divide-and-conquer strategy and contains several procedures to complete the learning task. We discover a set of clusters from a dependency graph built directly from the data. Each

cluster is expected to represent a local domain structure. We establish connection between clusters and learn each cluster separately. The learned clusters are joined together to compose the complete targeted network.

The novel algorithm reduces the computational complexity since it learns clusters instead of the complete network directly. In addition, the algorithm learns clusters in a separated way so that the structural error(due to conditional independency tests) that occurs in the learning of clusters does not influence the global structure learning. Hence, the algorithm avoids the cascading effect of incorrect statistical test results in the structural learning. We experiment the proposed algorithm on several benchmark Bayesian networks and compare with other typical constraint-based learning algorithms. The empirical results show that the decomposition algorithm achieves good performance regarding both structure learning accuracy and run times.

This paper is organized as follows. In Section 2, we discuss some related works on structural learning. In Section 3, we present the decomposition learning algorithm by illustrating embeded procedures. Then, we show comparison results in Section 4. Finally, in Section 5, we conclude the paper with some hint on future work.

2 Related Work

The divide-and-conquer strategy has served as the technique of many learning algorithms that aim at recovering a large Bayesian network structure from data [8–10]. The foundation of this strategy is to identify some appropriate components in a large model. For example, in the approach of learning module networks [9], a module is defined as a set of variables that have similar behavior. All variables in a module share both the same parents and the same conditional probability distribution. It seems that the module formulation is quite strict. However, the formulation is well consistent with some domain concepts such as genes in a cell, stocks in a stock sector, and so on. The sparse candidate algorithm [8] recovers Bayesian networks by specifying the maximum number of parents of variables in the learning process, which significantly reduces the learning complexity. Both the module learning and the sparse candidate approaches orient score-based learning.

The most relevant work is the block learning algorithm [10] that already shows the ability of learning a large Bayesian network from limited data. Similarly, the block learning algorithm recovers structures through procedures of identifying blocks and combining the learned blocks. However, the block identification procedure is incomplete since the block is composed of nodes that have at most two-length distance from block centers. The searching largely depends on the topology of a dependency graph and probably leads to disconnected blocks in some domains. In addition, as shown in the previous work, a large amount of run times are spent in learning overlapping structures. However, the extra procedure does not have much benefit to the final (heuristic) structural combination. Our new algorithm improves the block learning algorithm by designing more robust and appropriate procedures.

3 Decomposition Learning Algorithms

A constraint-based approach learns a network structure by using some statistical hypothesis tests to detect dependency or (conditional) independency among variables or attributes in a data set. The results of several tests are combined by the constraint-based approach in order to construct a Bayesian network structure. The test results might be incorrect especially when insufficient data are provided. Since various test results might depend on each other in some unknown manner, the error of the induced structure is not under control and spread in a global way. In addition, a large number of variables lead to an increasing order of conditional independency tests, which makes the learning intractable. To circumvent these shortcomings, we propose the decomposition algorithm that enhances the learning ability of conventional learning techniques by specifying a modular framework.

We briefly describe the decomposition learning algorithm in Fig. 1. The algorithm receives the data set D of size l [1] and the parameter ε used to control the cluster expansion (line 4). We firstly construct the dependency graph M directly from the data through the procedure of building a dependency graph (BDG)(line 2). The first procedure also produces two sets of edge weights, W^M and W^G, for the dependency graph M and the complete graph G respectively. Then, we partition M into several disjoint clusters using the procedure of star discovery (SD)(line 3). The procedure is highly motivated by current research work on complex network [11, 12] and is rather reliable to generate consistent clusters. The disjoint clusters reveal some local components that shall be connected in the domain. Hence, we expand the clusters into a set of overlapping clusters by discovering a high correlation between inter-cluster memberships. The procedure of cluster expansion (CE) uses the parameter ϵ to control the proportion of overlapping variables in the truly correlated clusters (line 4). We proceed to learn a Bayesian knot for each overlapping cluster separately by structuring the relation of cluster variables. The procedure of learning Baysian knots (LBK) may utilize any of available structural learning algorithms (line 5). Finally, we use structural rules to combine the learned Bayesian knots and recover the complete Bayesian network structure B in which a set of nodes V^B are connected with directed edges E^B.

3.1 Build a Dependency Graph

Bayesian network structures exhibit the dependency among variables in a data set. A strong dependency always gathers the variables into one local component. In other words, the tightly linked variables are potential nodes that will be enclosed in the same cluster. We expect to build a representative dependency graph from which some sound clusters could be discovered. The graph must be able to characterize a strong dependency of domain variables through their connectivity. We select the maximum spanning tree [13] as the dependency graph

[1] Both the attribute x_i in data set and the node or vertex v_i in graphs represent variables in the domain. They are interchangable and not further distinguished in this paper.

Decomposition Learning Algorithm (DL)

Input: $D = \{x_{1,l}, \cdots, x_{n,l}\}$ and parameter ϵ
Output: A Bayesian network structure $B = (V^B, E^B)$
1: Load the data D
2: Build the dependency graph M: $(M, W^M, W^G) = BDG(D)$
3: Partition M into a set of disjoint clusters C: $C = SD(M, W^M)$
4: Expand C into a set of overlapping clusters OC: $OC = CE(C, W^G, \epsilon)$
5: Learn a set of Bayesian Knots BK: $BK = LBK(OC, D)$
6: Compose the final Bayesian network structure B: $B = CBK(BK)$

Fig. 1. The decomposition learning framework includes multiple procedures that will be illustrated subsequently.

M since the tree is the smallest connected graph that *optimally* approximates the joint distribution of domain variables. The dependency graph $M = (V^M, E^M)$ is a tree in which each edge, $e_{i,j} \in E^M$, connects a pair of nodes v_i and v_j (v_i, $v_j \in V^M$) and the edge has the weight $w_{i,j}$ ($w_{i,j} \in W^M$) measured by the mutual information $MI(v_i, v_j)$. We show the procedure of building a dependency graph (BDG) in Fig. 2.

Build a Dependency Graph (BDG)

Input: Data $D = \{x_{1,l}, \cdots, x_{n,l}\}$
Output: $M = (V^M, E^M)$, W^M, W^G
1: Compute the complete Graph $G = \{V^G, E^G\}$ with weights $W^G = \{w_{i,j} = MI(v_i, v_j) | i, j = 1, \cdots, n \text{ and } i \neq j\}$
2: $k = 0$, $E^M \leftarrow \emptyset$, $W^M \leftarrow \emptyset$ $\triangleright Initialization$
3: Sort E^G decreasingly according to the weight $w_{i,j} \in W^G$
4: **For** $e_{i,j} \in E^G \wedge k < |V^M|$ **do**
5: **If**$(\{e_{i,j}\} \cup E^M)$ do not create a cycle in M **Then**
6: $E^M \overset{\cup}{\leftarrow} \{e_{i,j}\}$, $W^M \overset{\cup}{\leftarrow} \{w_{i,j}\}$, $k = k + 1$

Fig. 2. The BDG procedure builds the maximum spanning tree as the dependency graph M from the data D.

We compute $MI(v_i, v_j)$ for all pairs of variables (for l-size samples) and construct the complete graph G (line 1). We use the hash table that shows efficient computation. The complexity of this task is in the order of $O(n^2)$. We slightly modify the Kruskal's algorithm to build the tree M (the original Kruskal's algorithm [14] finds the minimum spanning tree by sorting weights decreasingly instead of increasingly) (lines 3-6). We use an union-finder data structure and a sorted list for adding arcs into M. The complexity is in the order of $O(n \log n)$.

3.2 Discover Local Components

The output M is a minimal description of dependency among the variables. We opt for this dependency graph because it represents the most significant interactions in a topology that could be clustered (recall that variable clustering in complex graphs is an NP-hard problem). Many clustering methods [15, 16]

have appeared and shown competitive results in some domains. However, most of them aim for different optimization problems. Moreover, they can't generate consistent clusters due to random selection of initial cluster modes. We are interested in offering a robust algorithm that clusters a set of truly dependent variables by examining a graph topology together with edge weights.

We aim to find a set of clusters C (each cluster C_i contains a set of vertices V^{C_i} in which one vertex v_j is called as cluster center node o_j) that maximize the function in Eq. 1. In other words, we want to maximize the sum of depenency weights (between cluster variables v_i and cluster nodes o_j) over multiple clusters.

$$C = \underset{C}{\operatorname{argmax}} \sum_{C_i \in C} \sum_{v_i \in (V^{C_i} - \{o_j\})} w_{i,j} \tag{1}$$

where o_j is the center node v_j in the cluster C_i and $w_{i,j} \in W^G$.

We use some sound graph operations to maximize Eq. 1 and show the Star Discovery (SD) procedure in Fig. 3. The idea is motivated by current research results on complex networks and evolves from the spanning star in the scale-free networks [17]. The research characterizes domain patterns in terms of connectivity of nodes, densities of clusters of nodes, and so on. It indicates that nodes of strong relations are always close and reside in a neigboring position. It suggests some hidden, but natural, domain patterns could be discovered by investigating the constructed graph topology.

Star Discovery Procedure (SD)

Input: $M = (V^M, E^M)$, W^M
Output: $C = \{C_1, C_2, \cdots, C_k\}$
1: For each $v_i \in V^M$
2: $o_i \leftarrow v_i$
3: $Adj(v_i) \overset{\cup}{\leftarrow} \{v_j\}$ iff $e_{i,j} \in E^M$
4: $Leaf(v_j) \overset{\cup}{\leftarrow} \{v_h\}$ iff $e_{h,j} \in E^M \wedge v_j \in Adj(v_i) \wedge e_{h,*} \notin (E^M - \{e_{h,j}\})$
5: $V^{S_i} \overset{\cup}{\leftarrow} \{o_i\} \cup Adj(v_i) \cup Leaf(v_j)$
6: $E^{S_i} \overset{\cup}{\leftarrow} \{e_{i,j}\} \cup \{e_{h,j}\}$
7: $W^{S_i} = \sum(w_{i,j} + w_{h,j})$ ▷ *Star weights*
8: While $V^S \neq \emptyset$
9: $C_k \leftarrow V^{S_i}$ iff $S_i = \underset{S_i \in S}{\operatorname{argmax}}(W^{S_i} \in W^S)$
10: $S \leftarrow (S - S_i - S_j)$, $W^S \leftarrow (W^S - W^{S_i} - W^{S_j})$ iff $v_j \in S_i \wedge v_j = (o_j \in S_j)$
11: $C \overset{\cup}{\leftarrow} C_k$, $V^S \leftarrow (V^S - C)$

Fig. 3. The SD procedure finds a set of disjoint clusters C from M through the building of star graph S and avoids random initialization of clusters.

We start by building a set of stars $S = \{S_1, \cdots, S_n | S_i = (V^{S_i}, E^{S_i})\}$ (lines 2-7). Each star S_i is not a single node, but a connected sub-graph in the dependency

graph. We initialize each node v_i as the star center node o_i (line 2). The center node o_i, together with its adjacent nodes $Adj(v_i)$ and leaf nodes $Leaf(v_j)$ next to the adjacent nodes v_j ($v_j \in Adj(v_i)$), composes the initial n stars (lines 3-6). In addition, we compute the star weight W^{S_i} that is the sum of weights for all edges in S_i (line 7). Then, we find a set of clusters C from the set of stars S and each cluster C_k contains only vertices V^{C_i} without edges (lines 8-11) [2]. The star S_i that has the largest star weight W^{S_i} of all remained stars is chosen as the cluster (line 9). When the star S_i becomes a cluster it will be removed from the set S together with the stars S_j that have the center node o_j residing in the selected star S_i (line 10). Afterwards, we select the star of the second largest weight as a new cluster. Hence, we get a set of k clusters in an iterative way without having to specify the cluster number k in the initialization. The SD complexity is dominated by the building of stars and takes $O(n^3)$ operations searching for all adjacent and leaf nodes.

The SD procedure maximizes Eq. 1 through finding clusters that contain nodes close to cluster centers in the dependency graph. We notice the SD procedure avoids random initialization of clusters since it builds clusters by selecting the star that has the largest weight among all remained stars. Consequently, we do not need to specify the cluster number k and get consistent clusters upon one data set. This is significantly different from other clustering methods that need to assume a number of initial clusters at random.

3.3 Cluster Expansion

A cluster contains a set of most correlated variables that may compose a local component in the domain. Since the SD procedure may result in disjoint clusters we may lose some local correlations that link variables in separated clusters. In addition, we need to recover the complete network structure by joining local cluster structures. The interdependency of clusters will provide foundation in the combination phase. Hence, we proceed to expand disjoint clusters into overlapping clusters by discovering cluster interdependency.

We present the Cluster Expansion (CE) procedure in Fig. 4. The basic idea is to expand clusters by including outlier variables that have most strong dependency with cluster memberships. The procedure uses two phases, cluster expansion (lines 1-6) and region expansion (lines 7-14), to generate a set of overlapping clusters. In the first phase, we use the parameter ϵ to control the number of overlapping variables for possibly expanded clusters (line 3). For each cluster C_i, we identify $\lceil |C_i| * \epsilon \rceil$ (the ceil function $\lceil \cdot \rceil$) numbers of outlier variables v_j that have the most strong dependency with cluster variables v_i by measuring their weights (line 4), and include these outlier varliables into the targeted cluster (line 6). The complexity of this phase is governed by the searching of relevant variables in k disjoint clusters and is in the order of $O(ksn)$ where s is the maximal cardinality of any given cluster C_i.

In the first phase, clusters are expanded through absorbing a limited number of outlier variables that have strong dependency with cluster memberships.

[2] Since a cluster contains only vertices we sometimes use C_i as V^{C_i} depending on the context.

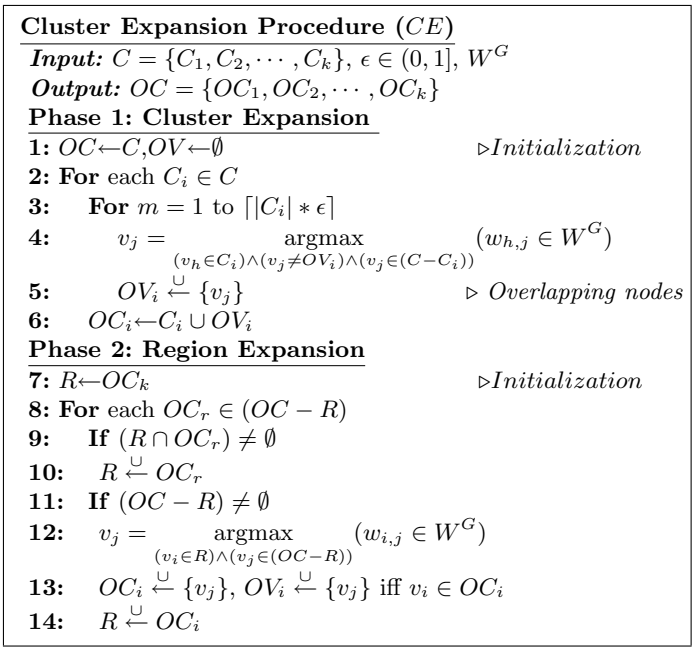

Fig. 4. The CE procedure expands disjoint clusters by absorbing most relevant outlier variables into targeted clusters.

Consequently, some isolated regions that contain a set of connected clusters may appear. For example, through the CE procedure, four disjoint clusters (C_1, C_2, C_3, and C_4) may result in two isolated regions, $(OC_1 \cup OC_2)$ and $(OC_3 \cup OC_4)$. The cluster C_1 locates $\lceil |C_1| * \epsilon \rceil$ most relevant variables all of which reside in the cluster C_2, and the cluster C_2 finds all the most relevant variables in the cluster C_1; so do the clusters C_3 and C_4. We need to remedy the cluster expansion phase to ensure the cluster reachability (direct or undirect) if it is necessary.

In the same vein as cluster expansion phase, the second phase expands isolated regions by including (region) outlier variables that have the most dependency with region variables. We compose the region R by connecting the clusters (from the first phase) that have already shared some overlapping variables (lines 9-10). Then, we detect possible isolated regions (line 11). If such regions exist we need to connect them by adding the most relevant outlier variables v_j into the targeted cluster (lines 12-13). We also get the byproduct of a set of overlapping nodes OV_i (line 13). The complexity of the second phase is dominated by the searching of relevant nodes in possibly isolated regions and is in the order of $O(nm)$ where m is the maximum number of variables within one region.

3.4 Recover Bayesian Network Structures

The CE procedure expands the disjoint clusters so that each cluster is connected to at least one of other clusters. We proceed to learn a set of Bayesian

Knots (BK) by structuring relations of variables in clusters. We describe the learning Bayesian knots (LBK) procedure in Fig. 5. The procedure receives the input of the data set D and a set of overlapping clusters OC (line 1). We apply any of available structual learning algorithms to construct a Bayesian knot (BK) that is a directed acyclic graph (line 3). Each BK_i contains a set of nodes V^{BK_i} connected by directed edges E^{BK_i} and could be viewed as a local structure in the domain. The procedure complexity relies on the selected learning algorithm (line 3). For example, if the PC algorithm is used the complexity is in the order of $O(kr^q)$ for learning k clusters where q is the maximum number of parents for a node and r is the largest cluster size. In general, a cluster contains a small subset of domain variables ($r \ll n$). The complexity is relatively low comparing with the order of $O(n^q)$ for learning the complete network directly.

Learning Bayesian Knots(LBK)
Input: Data D, Clusters OC
Output: $BK = \{BK_1, BK_2, \cdots, BK_k\}$
1: Load the data D
2: For each $OC_i \in OC$
3: Construct BK_i using any structural learning algorithm
4: $BK \overset{\cup}{\leftarrow} BK_i$

Fig. 5. The LBK procedure learns Bayesian knots (much smaller than the complete network) using any of available structural learning algorithms and recovers local domain structures.

The final procedure is to complete the learning task by joining the learned Bayesian knots that share common variables. We show the procedure of combining Bayesian knots in Fig. 6. The procedure takes some rules to address conflicting structural problems and to avoid global directed cycles in the network. The conflict occurs when the direction of arcs connecting overlapping nodes differs in linked knots.

Combine Bayesian Knots(CBK)
Input: $BK = \{BK_1, BK_2, \cdots, BK_k\}$
Output: $B = (V^B, E^B)$
1: Start the global skeleton of a complete network $B = \{V^B, E^B\}$
2: $E^B \leftarrow (E^B - \{e_{ij}\})$ iff $e_{ij} \notin E^{BK_i}$
3: For each $BK_i \in BK$
4: Orient $e_{ij} \in E^B$ iff $e_{ij} \in E^{BK_i} \wedge e_{ij} \notin (E^{BK} - E^{BK_i})$
5: For all undirected edges $e_{ij} \in E^B$
6: If $v_i \rightarrow v_j$, v_j and v_h are adjacent, and v_i, v_h are not adjacent, then orient $v_j - v_h$ as $v_j \rightarrow v_h$
7: If there is a directed path from v_i to v_j, and v_i, v_j are adjacent, then orient $v_i - v_j$ as $v_i \rightarrow v_j$
8: Otherwise, orient $v_i - v_j$ at random

Fig. 6. The CBK procedure joins Bayesian knots into the complete network through structural rules without furthering (in)dependency tests.

We start the Bayesian network B with a complete undirected graph (line 1). Then, we remove edges from the complete graph that do not exist in any of the learned Bayesian knots (line 2). All the remained edges must be directed in at least one of the Bayesian knots. We direct those edges that have already been oriented in at most one Bayesian knot (line 4). Subsequently, we use three rules to orient the rest undirected edges since the edges are directed differently in overlapping Bayesian knots (lines 5-8). The first rule is to avoid new v-structures (line 6). In most constraint-based learning methods, directions of edges participating in v-structures are uncovered using independency tests, rather than through structural rules afterwards. The second rule avoids directed cycles by forcing the arc direction (line 7). Finally, if both rules can't be applied we orient edges randomly following directions in one Bayesian knot (line 8). The combination procedure aims for the arc orientation using structural rules instead of expensive independency tests.

4 Experimental Results

We demonstrate the empirical performance of the decomposition learning algorithm on several benchmarks : ALARM (37 nodes), Hailfinder (56 nodes), HeparII (70 nodes), Pathfinder (109 nodes), and Andes (223 nodes). We also compare the performance with two typical constraint-based learning methods. One is the basic learning method of the PC algorithm [3] and the other is three phase dependency analysis ($TPDA$) [18] algorithm that is the winner of 2001 KDD cup. In addition, we compare with the block learning algorithm. We generate several data sets (ranging from small to large sample sizes) and compute the Euclidean distance (of the *sensitivity* and *specificity* from the perfect score 1) [19] between the learned structures and the benchmarks. The Euclidean distance is defined in Eq. 2.

$$distance = \sqrt[2]{(1 - sensitivity)^2 + (1 - specificity)^2} \tag{2}$$

where the *sensitivity* of the algorithm is the ratio of correctly identified edges (undirected arcs) over the total number of edges in the real network while the *specificity* is the ratio of edges correctly identified as not belonging in the graph over the true number of edges not present in the real network.

In most cases, we show that the decomposition learning algorithm outperforms other learning algorithms and achieves lower distance values. In particular, the new algorithm keeps a good quality structure even when the data set is reduced. Furthermore, we obtain computational savings from using the decomposition algorithm as indicated by the low run times.

We show the performance of the decomposition learning algorithm in Fig. 7 [3]. Each data point is the average of 10 runs for different data sets of same size [4]. Both the decomposition and the block learning algorithms that are equipped

[3] We specify $\epsilon \in [0.40,0.60]$ concerning the tradeoff between the cluster size and the overlapping set.

[4] We only count successful runs of the BL algorithm when it produces connected local structures.

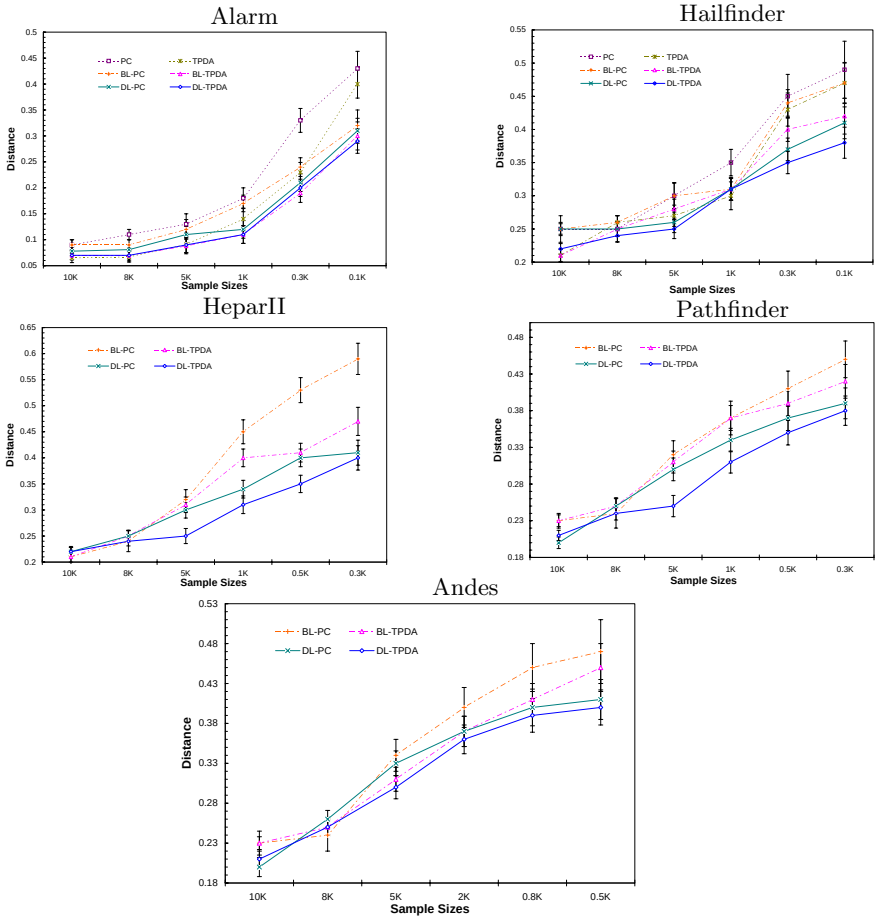

Fig. 7. Performance profile of the decomposition learning algorithm comparing with other learning algorithms. The dotted lines with different colors denote the PC and $TPDA$ learning algorithms, the dashed lines, $BL-PC$ and $BL-TPDA$, denote the block learning algorithms configured by the PC and $TPDA$ learning engines respectively, and the solid lines, $DL-PC$ and $DL-TPDA$, denote the decomposition learning algorithms equipped with the PC and $TPDA$ learning techniques (in the LBK procedure) respectively.

with learning engines have better performance than the PC and $TPDA$ learning algorithms regarding the distance measure. This remains true for a range of data sets. For small domains, such as Alarm and Hailfinder networks, both the decomposition learning algorithm and the block learning algorithm exhibit similar performance of low distance. However, the decomposition algorithm has significantly better results on the rest three large networks, especially for small data sets. We report only the performance of the BL and DL learning algorithms on the three larger networks since both the PC and $TPDA$ algorithms fail.

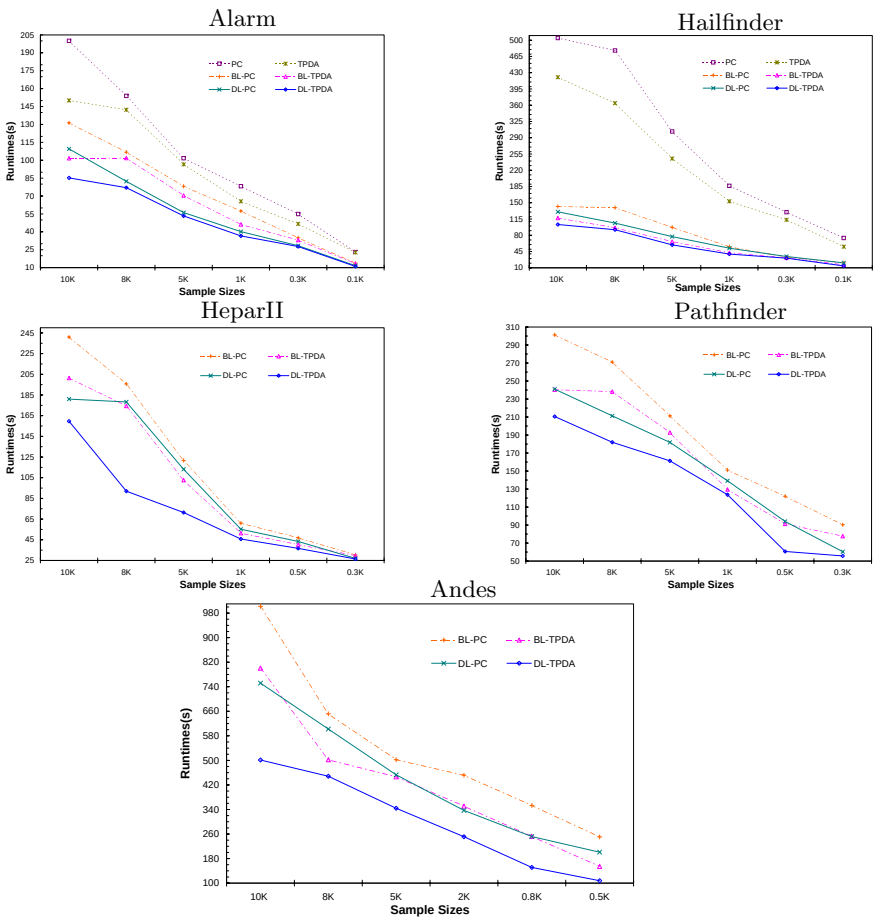

Fig. 8. Runtimes comparison (3GHz, 2GB RAM). The decomposition algorithm is scalable in learning large domains.

We also observe from Fig. 7 that the decomposition learning algorithm retains a good quality of learned structures when the sample size is noticeably reduced. In addition, the decomposition algorithms have a lower variance than the block learning algorithms. This is due to the DL method has a reliable clustering method SD comparing with the incomplete block identification in BL.

Finally, the run times in Fig. 8 are indicative of the computational savings incurred by using the decomposition learning algorithm. The decomposition algorithm achieves more savings than the block learning algorithm since the latter needs an expensive procedure of learning overlapping structures. Using the decomposition algorithm we were able to learn the three large domains of HeparII, Pathfinder, and Andes, while both the PC and $TPDA$ algorithms run out of memory. We expect similar results of good performance without intensive computation in real applications.

5 Discussion

The decomposition learning algorithm is able to learn a large Bayesian network structure and shows good performance even when insufficient data are provided. It significantly improves the block learning algorithm on the aspects of robust clustering methods and well-defined combination rules. The modular design provides a way to exploit state-of-the-art of both Bayesian network learning and attribute clustering techniques. In addition, the decomposition learning algorithm offers useful intermediate clusters or Bayesian knots that represent local domain structures and may attract interest into further study. Several issues relevant to the decomposition learning algorithm deserves further study. We are currently investigating an adaptive cluster expansion.

References

1. Pearl, J.: Probabilistic reasoning in intelligent systems: networks of plausible inference. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1988)
2. Jensen, F.V.: An introduction to Bayesian networks. Springer, New Work (1996)
3. Spirtes, P., Glymour, C., Scheines, R.: Causation, Prediction, and Search. Springer, New Work, USA (1993)
4. Heckerman, D.: A tutorial on learning in bayesian networks. (1995)
5. Neapolitan, R.E.: Learning Bayesian Networks. Prentice Hall (2003)
6. Tsamardinos, I., Brown, L.E., Aliferis, C.F.: The max-min hill-climbing bayesian network structure learning algorithm. Mach. Learn. **65**(1) (2006) 31–78
7. Chickering, D.M.: Learning bayesian networks is np-complete. In: Proceedings of AI and Statistics, 1995. (1995)
8. Friedman, N., Nachman, I., Peer, D.: Learning of bayesian network structure from massive datasets: the sparse candidate algorithm. In: UAI. (1999) 206–215
9. Segal, E., Pe'er, D., Regev, A.: Learning module networks. In: UAI. (2003) 525–534
10. Zeng, Y., Poh, K.L.: Block learning bayesian network structure from data. In: Proceedings of the Fourth International Conference on Hybrid Intelligent Systems (HIS-04). (2004) 14–19
11. Newman, M.E.J.: The structure and function of complex networks. SIAM Review **45**(2) (1957) 167–256
12. Albert, R.Z., Barabasi, A.L.: Statistical mechanics of complex networks. Modern Physics (74) (2002) 47–97
13. Chow, C., Liu, C.: Approximating discrete probability distributions with dependence trees. IEEE Transaction on Information Theory (12) (1968) 462–467
14. Sedgewick, R.: Algorithms in Java, Part 5 Graph Algorithms. Addison Wesley (2004)
15. Asano, T., Bhattacharya, B., Keil, M., Yao, F.: Clustering algorithms based on minimum and maximum spanning trees. Proceedings of the fourth annual symposium on Computational Geometry (1988) 252–257
16. Grygorash, O., Zhou, Y., Jorgensen., Z.: Minimum spanning tree based clustering algorithms. IEEE International Conference on Tools with AI (2006)
17. Gallian, J.: Dynamic survey of graph labeling. Elec. J. Combin. **14**(6) (2007)
18. Cheng, J., Greiner, R., Kelly, J., Bell, D., Liu, W.: Learning bayesian networks from data: An information-theory based approach. Artificial Intelligence, **137**(1) (2002) 43–90
19. Tsamardinos, I., Aliferis, C., Statnikov, A.: Time and sample efficient discovery of markov blankets and direct causal relations. In: KDD. (2003) 673–678