Refinement of Bayesian Network Structures upon New Data

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
yanping_xiang@yahoo.com.cn

Saulius Pacekajus
Dept. of Computer Science
Aalborg University, Denmark
saulius@cs.aau.dk

Abstract

Refinement of Bayesian network structures using new data becomes more and more relevant. Some work has been done there; however, one problem has not been considered yet - what to do when new data has fewer or more attributes than the existing model. In both cases data contains important knowledge and every effort must be made in order to extract it. In this paper, we propose a general merging algorithm to deal with situations when new data has different set of attributes. The merging algorithm updates sufficient statistics when new data is received. It expands the flexibility of Bayesian network structure refinement methods. The new algorithm is evaluated in extensive experiments, and its applications are discussed at length.

1 Introduction

Bayesian network (BN) [1] is a directed acyclic graph (DAG) where nodes represent attributes or variables of a subject of matter, and arcs between the nodes describe the causal relationship of attributes or variables. It is widely used in the medical, biological domains, and so on. The use of BN is a logical and natural way to represent the joint probability distribution over the variables.

A lot have been written on how to construct a BN structure from data and most of the writing has been focused on learning the structure using batch algorithms [2]. Batch algorithms go through all of the data and then structure a corresponding BN. This is an interesting field of study but in many cases the whole data is not available i.e. new data is expected sometime in the future. It is neither efficient nor in some cases possible to store all the data met in order to run batch algorithms that would learn the structure from scratch, when new data is available. To address this kind of problems in structural learning, algorithms need to be incremental, and use both previous experience and new data to refine the structure of the network. They must rely on the previous structure to speed up the refinement process, but not too heavily so that they could make corrections to the old structure.

One way is to use the structure of the old network as prior probability. Some work has been done in this line [3]. This however can have the unwanted consequences that the new network is biased to the old structure. Another approach is to use *sufficient statistics* ($SUFF$) or combination of these two approaches [4]. There have been some works in the field of incremental or sequential refinement of network structures. A good review could be found in [5].

The problems that have not gotten enough attention are when new data has different set of attributes. It can have partial attributes, it can have some new attributes or both partial and new attributes. When new data arrives, a network needs to be refined in order to represent the domain more accurately than before. The most accurate approach would be to learn the network from scratch but this of course is not always feasible. The main focus of this paper is on these problems and the solution proposed is to update the $SUFF$ when receiving new data. The basic idea is to add the new attributes to the $SUFF$ in the right proportion.

We present the merging algorithm that orients operations to configurations of all common attributes in old and new $SUFF$, and does the sampling in a reasonable way. The new approach is pretty straightforward, and performs well in extensive experiments.

2 Relevant works

Most of the widely used batch algorithms take data (and some extra information, if necessary) as an input, and learn a Bayesian network structure. In order to be able to update the model, new data must join the old data that has been kept so far, and the batch algorithm needs to start its work from scratch. Obviously this is neither memory nor time efficient, therefore some better methods need to be invented.

A desired method should be able to iteratively use new data instead of starting everything from scratch. In other words a desired method should be intelligent enough to combine new information with its previous experience to update its knowledge. A response to these requirements is incremental methods. These are Buntine (B) [6], Friedman and Goldszmidt(FG) [4] algorithms, and so on. Both of them use $SUFF$ of data that only contains counts of different entries in data instead of data entries. They only require constant time to update $SUFF$ when new records arrive; the record itself can be discarded after $SUFF$ is updated.

When the FG and Bun algorithms are designed to be incremental, such popular batch algorithms like the B, K2 [7] and HCMC [8] algorithms can also be turned into incremental ones. In [5], J. Roure has much discussion on the transformation. After combinng $SUFF$ with reduced search space (RSS), J. Roure introduced new algorithms iK2, iB and iHCMC that are all incremental [5]. In fact they are quite similar to the original FG and Bun algorithms. J. Roure even proposed special heuristics to store only $SUFF$ of the best structure candidates, just as in slightly different styles the FG and Bun algorithms do.

A heuristic to incrementally learn Bayesian network structures even when new data instances have different set of attributes needs to be invented. It would also be nice, if a desired heuristic could solve the problem generally and in an elegant way, so that every or at least most learning algorithms could be improved in a similar fashion and still preserve their original efficiency and quality.

3 New data with different set of attributes

The incremental approaches have two common properties. The first one is that they all use sufficient statistics, and at iteration they repeat the search path by traversing the reduced DAG space. The later property is also known as RSS heuristic. While RSS does not really care about the attributes of data, $SUFF$ is where the main knowledge is stored. To understand how a problem of new data with different set of attributes can be solved it is therefore necessary to understand $SUFF$.

3.1 Sufficient statistics

To store all the data that is constantly arriving in incremental algorithms is not feasible because of finite memory. On the other hand, not everything in the data is useful for learning a Bayesian network structure. Actually a data entry alone does not give any information. It is obvious that the scoring needs to be explored in order to find out what in the data is necessary for the learning process and what is not, since only a scoring function deals with the data directly.

The well known BDe scoring function [7] is described in Eq. 1.

$$P(B_S|D) = P(B_S)\prod_{i=1}^{n}\prod_{j=1}^{q_i}\frac{(r_i-1)!}{(N_{ij}+r_i-1)!}\prod_{k=1}^{r_i}N_{ijk}! \tag{1}$$

where B_S is a structure of Bayesian network, D is a data set, $P(B_S)$ is the prior probability of a network structure B_S, that can be used to express any prior knowledge of a structure. n is the number of variables, r_i is the arity of a variable X_i, q_i is the number of possible parent ($\mathbf{Pa_i}$) configurations, N_{ijk} is a number of entries in D where variable X_i is set to its k^{th} value and its parents are instantiated to their j^{th} configuration and $N_{ij} = \sum_{k=1}^{r_i} N_{ijk}$.

Hence only the counts of entries satisfying certain attribute configurations are necessary. Let $\mathbf{X}$ denote a set of random variables and $\mathbf{x}$ be the value assignments for the variables in $\mathbf{X}$. Let the vector $\hat{N}_{\mathbf{X}}^{D}$ contain the counts of entries in the data D where $\mathbf{X} = \mathbf{x}$ for every possible $\mathbf{x}$. A set $\hat{N}_{\mathbf{X}}^{D}$ is called *sufficient statistics* of variables $\mathbf{X}$. Then, given the decomposability of the scoring function, $\hat{N}_{X_i,\mathbf{Pa_i}}^{D}$ for all $X_i \in \mathbf{X}$ and all possible parent sets for X_i is all the information necessary to learn the Bayesian network structure for the data D.

$$|SUFF(B_S)| = n \times \sum_{i=0}^{p} \left(\begin{array}{c} n \\ i \end{array} \right) \times arity(X)^i \tag{2}$$

The memory used for $SUFF$ can be described by its cardinality in Eq. 2, where n is the number of variables, p the maximum number of parents and $arity(X)$ is the arity of variable X. It is assumed for simplicity that all the variables have the same arity. As it can be seen, cardinality of $SUFF$ is more than exponential. When $p = n-1$ the cardinality is extremely large, but, fortunately, in most applications p is rather small as well as an average arity of a variable. One interested in implementing memory and time efficient $SUFF$ should consider such data structures like AD-trees introduced in [9] or an even more optimised dynamic AD-trees discussed in [10].

Having a compact $SUFF$ it is possible to keep the history of all the data entries seen while using constant amount of data, therefore all the experience of incremental structure learning algorithms is accumulated there. When new data with different set of attributes arrives, it can be considered equivalent to a problem where the new or old (or both) data has missing values. Neither the Minimum Description Length function [11] nor the BDe scoring function can deal with missing values without special improvements. Experiments showed that the BDe scoring function can even sense

5 entries with missing values in a data set containing 20000 items. The resulted BN structure was different from the one obtained when those 5 entries were not removed or reasonable values were specified (or even randomly, if the number of entries with missing data is relatively small to the size of the data set). Although incremental algorithms use $SUFF$ instead of data directly, the discrepancy of counts in the data (which is the reason that scoring functions work incorrect) still exists since it is transferred to $SUFF$. In addition, if AD-trees together with most common value (MCV) [5] pruning are used for $SUFF$, the entries with missing values would eventually be identified as entries satisfying MCV configurations, which in fact would not be true.

3.2 Merging data with different sets of attributes

Since the same problem holds for both data sets and $SUFF$ of those data sets it would be nice to find a general way to solve it. The proposal is pretty straightforward. While merging two data sets (or $SUFF$) the missing attribute values should be sampled with respect to the distribution of the same attribute values in the set they are present. For the sake of a general representation, the merging algorithm is presented in Fig. 1 in the form of two data sets. It can be used with $SUFF$ instead of data sets with minor technical changes.

As it can be seen, the algorithm runs through all common attribute ($A_1 \cap A_2$) configurations (value assignments) that are met in both data sets (line 8). It is because only from the common attributes the missing attribute values can be sampled reasonably. If the common value configuration (a_s) is present in both data sets - the one that has missing attributes and the other that has them present (combinations of lines 11 and 14 as well as 19 and 22) - the algorithm can sample reasonable values for the missing attributes according to their distribution in the set they are present (lines 16 and 24). In case some common attribute configurations are in data set with missing attribute values but not within the data set with those attributes present (lines 12 and 20), it is impossible to make a reasonable sampling, hence, all the entries satisfying that configuration are deleted (lines 13 and 21). This choice is made in order to save precision.

The argument for deleting configuration has two. First, configurations present in one data set and missing in the other occur because they have a low probability; hence, their contribution to the general relationships is not strong, and removing them does not have a big impact. The consequence of deleting entries is that it makes the algorithm more sensitive to the size of data. If the size of data is too small with respect to the number of attributes, configurations have lower chances to be included since the number of configurations increases exponentially to the number of attributes. The more attributes the data set has, the bigger chunks of new data should be accumulated in order to learn relationships between new and old attributes without losing much knowledge about their relationships. Secondly, experiments have shown if the values are generated randomly the bias is introduced. Randomly sampling for missing attribute values with configurations that are not in the data set with those available attributes was tried. The bias is small enough to be ignored as long as the number of this kind of configurations is small. However, the side effects (extra relations are introduced or some of them disappear) become more and more visible when the number of configurations increases. This results in new relationships that eventually affect a network structure.

```
Merging Algorithm
Require: $D_1, D_2$ data sets
Ensure: a new data set $D$ which is a result of merging $D_1$ with
    $D_2$
 1: $A_1 \leftarrow D_1.getAttributes()$
 2: $A_2 \leftarrow D_2.getAttributes()$
 3: if $A_1 = A_2$ then
 4:     return $D \leftarrow D_1 + D_2$
 5: else
 6:     $D'_1 \leftarrow D_1.copy()$
 7:     $D'_2 \leftarrow D_2.copy()$
 8:     for all $A_1 \cap A_2$ value assignments $a_s$ in $D_1 \cup D_2$ do
 9:         $c_1 \leftarrow D_1.getCount(A_1 \cap A_2 = a_s)$
10:         $c_2 \leftarrow D_2.getCount(A_1 \cap A_2 = a_s)$
11:         if $A_2 \backslash A_1 \neq \emptyset$ and $c_1 > 0$ then
12:             if $c_2 = 0$ then
13:                 $D'_1.deleteEntries(A_1 \cap A_2 = a_s)$
14:             else
15:                 $E \leftarrow D'_1.selectEntries(A_1 \cap A_2 = a_s)$
16:                 $E.sampleValues(A_2 \backslash A_1, D_2.entriesDistribution(A_2 \backslash A_1))$
17:             end if
18:         end if
19:         if $A_1 \backslash A_2 \neq \emptyset$ and $c_2 > 0$ then
20:             if $c_1 = 0$ then
21:                 $D'_2.deleteEntries(A_1 \cap A_2 = a_s)$
22:             else
23:                 $E \leftarrow D'_2.selectEntries(A_1 \cap A_2 = a_s)$
24:                 $E.sampleValues(A_1 \backslash A_2, D_1.entriesDistribution(A_1 \backslash A_2))$
25:             end if
26:         end if
27:     end for
28:     return $D \leftarrow D'_1 + D'_2$
29: end if
```

Figure 1: The pseudo code of merging algorithm

4 Experiments

The proposed algorithm is evaluated in extensive experiments. Before experimental results are presented, it is necessary to notice that the problem can be divided into three cases as shown in Fig. 2):

- The first case is when new data (or $SUFF$ of data) D_2 introduces new attributes (Figure 2a). This could happen, when some new knowledge about a subject is available and new variables into a model need to be introduced.

- In the second case, new data (or $SUFF$ of data) D_2 has only partial attributes (Figure 2b). Sometimes new data that only has part of model attributes is available. In spite of that, it still contains useful knowledge about present attributes and their relationship, that could be used to strengthen the model.

- The combination of the first two cases is the third one. New data (or $SUFF$ of data) D_2 has some new attributes as well as some old attributes of D_1 are missing (Figure 2c).

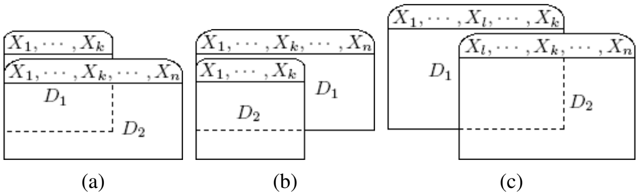

$X_1, \cdots, X_k$
$X_1, \cdots, X_k, \cdots, X_n$
D_1
D_2
(a)
$X_1, \cdots, X_k, \cdots, X_n$
$X_1, \cdots, X_k$
D_1
D_2
(b)
$X_1, \cdots, X_l, \cdots, X_k$
$X_l, \cdots, X_k, \cdots, X_n$
D_1
D_2
(c)

Figure 2: (a): D_2 introduces new attributes; (b): D_2 has partial attributes; (c): D_2 has partial attributes as well as introduces some new.

Experiments showed there is a large difference between the first two cases and the third case. While in the first two cases all the attributes can be related because they are present in at least one data set or $SUFF$, in the third case it is really complicated to relate attributes that are only in D_1 with those only in D_2. Hence it was necessary to conduct different experiments for the first two cases and the third case respectively.

4.1 Cases 1 and 2

The experiments for the first two cases were conducted with three networks Asia (8 nodes) [12], Studfarm (12 nodes) and Boblo (22 nodes) provided by Hugin. The two data sets containing 5000 entries were generated from the three networks. All the data sets were checked whether the K2 structure learning algorithm [9] can learn the original network from them. The measures taken were three: the logarithmic BDe score of the original network given the merged data set, the score of the network learned using the K2 algorithm given the merged data set, and the average time used to merge the two data sets (the experiments were conducted 6 times each and then average was taken). The experiments were run in this way: at the beginning the first data set is the one that has all the attributes available and the attributes of the second one are gradually removed every step until only one or two are left; then, the data sets are switched in the second stage to be sure that the algorithm can work correctly with different data sets. The results for all three experiments are shown in Fig. 3(a), Fig. 3(b) and Fig. 3(c) respectively.

Experimental results show that the merging algorithm works well with data generated from the Asia and Studfarm networks. In Fig. 3(a) and Fig. 3(b), the score of networks leant from the merged data sets is the same as that of the original networks in any situation. It indicates all resulted networks best fit the data. Furthermore, according to the resulted structures (not shown here), all of them learned from the merged data sets by the K2 algorithm are identical to the original network structure. Hence, the algorithm can be considered reliable. The average time used by the algorithm is reasonable and scales slower than the time spent for learning when the number of attributes increases.

On the other hand, results are not so nice with the data generated from the Boblo network. Fig. 3(c) shows the resulted networks fit data well, even better than the original network, and the algorithm uses reasonable time. However, half of the network structures learned from the merged data differs from the original one.

It indicates there is some information lost or some new relationships introduced that change the data distribution. A reasonable explanation could be that there is too little data. Knowing that the number of configurations of attributes grows exponentially to the number of attributes it is pretty naive to expect the same amount of data is sufficient for data sets with 8, 12 and 22 attributes. Hence a decision to conduct experiments with 10000 entries data sets generated from the Boblo network was made.

Experimental results in Fig. 3(d) show that the idea to increase the amount of data is good. All the structures learned from the merged data by the K2 algorithm are identical to the original one, and time does not increase dramatically. Hence it is necessary to conclude that one using the merging algorithm should have data sets with a reasonable size (with respect to the number of attributes).

The experimental results conclude the merging algorithm is reliable, effective and efficient (it takes similar time in spite of the number of attributes missing) when at least one data set has all the attributes (cases 1 and 2).

4.2 Case 3

Different experiments need to be conducted for case 3 when a data set has some attributes that are not in the other one. In this situation attributes that are only in the first data set cannot be directly related with attributes only in the second data set. Nevertheless it is expected, that it should be possible to obtain the relationship between them through the

Vars Removed	Orig. Net. Score	Result Net. Score	Time
0	-22516	-22516	0s
1	-22505	-22505	8s
2	-22529	-22529	7.5s
4	-22556	-22556	8s
6	-22716	-22716	8s
7	-22680	-22680	8.5s

(a)Results of merging two 5000 entries data sets generated from network Asia with 8 nodes

Vars Removed	Orig. Net. Score	Result Net. Score	Time
0	-3695	-3695	1s
2	-3675	-3675	11s
4	-3720	-3720	8s
6	-3732	-3732	8.5s
8	-3692	-3692	8.5s
10	-3728	-3728	8.5s
11	-3710	-3710	8.5s

(b)Results of merging two 5000 entries data sets generated from network Studfarm with 12 nodes

Vars Removed	Orig. Net. Score	Result Net. Score	Time
0	-47020	-47020	1s
4	-45940	-45940	24s
8	-46159	-46159	25s
12	-46907	-46908	20s
16	-47105.3	-47105.8	19s
18	-47136	-47143	20s
20	-47036	-47036	19s

(c)Results of merging two 5000 entries data sets generated from network Boblo with 22 nodes

Vars Removed	Orig. Net. Score	Result Net. Score	Time
0	-90863	-90863	4s
4	-91321	-91321	49s
8	-91863	-91863	50s
12	-92943	-92943	30s
16	-93270	-93270	27s
18	-93561	-93561	33s
20	-93577	-93577	30s

(d)Results of merging two 10000 entries data sets generated from network Boblo with 22 nodes

Figure 3: Experimental Results from Cases 1 and 2.

common attributes.

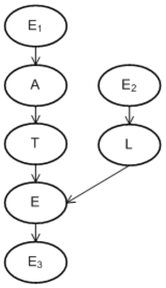


Figure 4: The original network in case 3

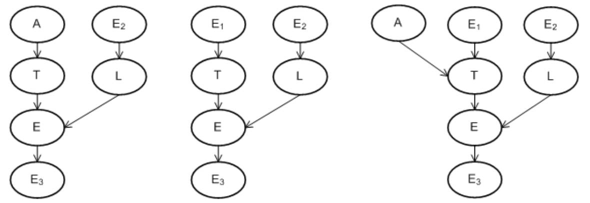
It is supposed that it should be safe to have missing attributes if they are conditionally independent. To check how conditional dependency properties manage to subsist when different attributes are not available in two data sets an experiment was made. A Markov blanket of node T was taken from the Asia network, and then three extra nodes E_1, E_2 and E_3 are added in all three possible different positions, so they do not get involved in T's Markov blanket. The experimental network is shown in Fig. 4. Just as the experiments were done in the first two cases, different attributes were removed from the two original data sets containing 5000 entries each; then, they were merged and a network was built using the K2 learning algorithm.

Since there are four nodes $\{A, E, L, T\}$ in T's Markov blanket(here it is supposed to include T) and three extra nodes $\{E_1, E_2, E_3\}$, 21 different cases have been conducted as well as 21 network structures best fitting them have been obtained. There are 3 groups in 21 cases that need to be commented:

Group 1 (12 cases). One extra attribute ($\{E_1, E_2, E_3\}$) is removed from the first data set and one of T's Markov blanket nodes ($\{A, E, L, T\}$) from the second data set. As expected this does not change T's Markov blanket conditional dependency properties with some exceptions in two cases shown in Fig. 5 and Fig. 6 respectively. Other 10 cases could output the same structure as the original one.

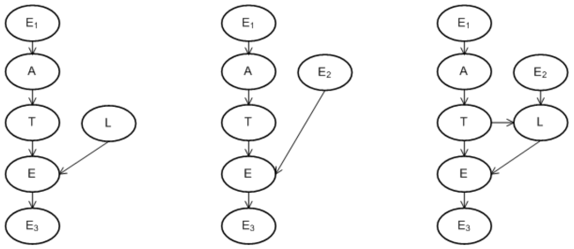
The results of the two exception cases could be explained in this way. In both of them, extra attributes and T's Markov blanket attributes that have direct relations were removed from different data sets, which causes the original relation to be lost. Hence the arcs between E_1 and T (in Fig. 5(b)) and between E_2 and E (in Fig 6(b)) are introduced when intermediate variables are removed. In Fig. 5(c), the arc between nodes E_1 and T is still kept since they have a strong relation that is encoded in the second data set (D_2) and is not compromised in the merged data set ($D_1 \cup D_2$). Hence the resulted network in Fig. 5(c) does not show nodes E_1 and T are independent given node A. Similarly, in Fig. 6(c), the introduced arc between T and L preserves the relation between E_2 and E. The result follows the logical combination of the two network structures that tries to preserve dependent and conditional independent relations encoded in these two networks.

Group 2 (3 cases). Both attributes removed are extra at-



(a) Network learned from D_1 (b) Network learned from D_2 (c) Network learned from $D_1 \cup D_2$

Figure 5: The case when $E1$ and A are removed.

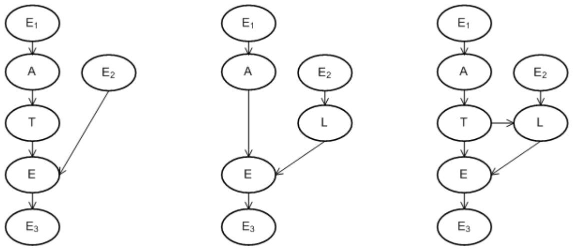


(a) Network learned from D_1 (b) Network learned from D_2 (c) Network learned from $D_1 \cup D_2$

Figure 6: The case when $E2$ and L are removed.

tributes $\{E_1, E_2, E_3\}$. Since they do not have any direct relation, as expected, all three resulted networks are identical to the original one.

Group 3 (6 cases). Both attributes removed are from T's Markov blanket $\{A, E, L, T\}$, therefore some missampling happens. Nevertheless in most of the cases the resulted networks are identical to the original one. The two exception cases are shown in Figs. 7 and 8 respectively. The case in Fig. 7 can be explained in the same way as the second exception case in Fig. 6. However, the case in Fig. 8 is somewhat different. Some extra arcs are again introduced, but the arc from E_2 to E_3 does not follow the logical combination of structures of the two compounding data sets. There is no obvious reason for this.

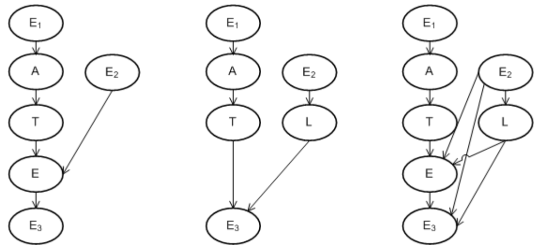


(a) Network learned from D_1 (b) Network learned from D_2 (c) Network learned from $D_1 \cup D_2$

Figure 7: The case when L and T are removed

5 Conclusion

When attributes are added into a network the effects on the existing nodes can be very hard to be judged. The effects can range from no impact on the existing structure, which means the new attributes are only introduced as parents or children of the existing attributes. The effects can also be very dramatic they might really affect the relationships associated with the existing attributes. The algorithm deals with both of these situations. It successuflly copes with Cases 1 and 2 while showing the ability to preserve the underlying structure in Case 3. We are in the future avenue investigating the conditions in a more formal way.



(a) Network learned from D_1 (b) Network learned from D_2 (c) Network learned from $D_1 \cup D_2$

Figure 8: The case when L and E are removed

References

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

[2] Neapolitan, R.E.: Learning Bayesian Networks. Prentice Hall (2003)

[3] Lam, W., Bacchus, F.: Using new data to refine a bayesian network. In: UAI. (1994) 383–390

[4] Friedman, N., Goldszmidt, M.: Sequential update of bayesian network structure. In: UAI. (1997) 165–174

[5] Roure, J.: Incremental methods for bayesian network structure learning. (1995)

[6] Buntine, W.: Theory refinement on bayesian networks. In: UAI. (1991) 52–60

[7] Cooper, G.F., Herskovits, E.: A bayesian method for the induction of probabilistic networks from data. Machine Learning **9** (1992) 309–347

[8] Castelo, R., Kocka, T.: On inclusion-driven learning of bayesian networks. Machine Learning Research **4** (2003) 527–574

[9] Anderson, B., Moore, A.: Ad-trees for fast counting and for fast learning of association rules. In: KDD. (1998) 134–138

[10] Komarek, P., Moore, A.: A dynamic adaptation of ad-trees for efficient machine learning on large data sets. In: ICML. (2000) 495–502

[11] Lam, W., Bacchus, F.: Learning bayesian belief networks - an approach based on mdl principle. Computational Intelligence **10** (1992) 269–293

[12] Lauritzen, S.L., Spiegelhalter, D.J.: Local computations with probabilities on graphical structures and their application to expert systems. Journal of the Royal Statistical Society **50**(2) (1988) 157–224