Scalable, Efficient and Correct Learning of Markov Boundaries under the Faithfulness Assumption

Jose M. Peña[1], Johan Björkegren[2] and Jesper Tegnér[1,2]

[1] Computational Biology, Department of Physics and Measurement Technology,
Linköping University, Sweden
[2] Center for Genomics and Bioinformatics, Karolinska Institutet, Sweden

Abstract. We propose an algorithm for learning the Markov boundary of a random variable from data without having to learn a complete Bayesian network. The algorithm is correct under the faithfulness assumption, scalable and data efficient. The last two properties are important because we aim to apply the algorithm to identify the minimal set of random variables that is relevant for probabilistic classification in databases with many random variables but few instances. We report experiments with synthetic and real databases with 37, 441 and 139352 random variables showing that the algorithm performs satisfactorily.

1 Introduction

Probabilistic classification is the process of mapping an assignment of values to some random variables $\mathbf{F}$, the features, into a probability distribution for a distinguished random variable C, the class. Feature subset selection (FSS) aims to identify the minimal subset of $\mathbf{F}$ that is relevant for probabilistic classification. The FSS problem is worth of study for two main reasons. First, knowing which features are relevant and, thus, which are irrelevant is important in its own because it provides insight into the domain at hand. Second, if the probabilistic classifier is to be learnt from data, then knowing the relevant features reduces the dimension of the search space.

In this paper, we are interested in solving the FSS problem following the approach proposed in [9, 10, 11]: Since the Markov boundary of C, $MB(C)$, is defined as any minimal subset of $\mathbf{F}$ such that C is conditionally independent of the rest of $\mathbf{F}$ given $MB(C)$, then $MB(C)$ is a solution to the FSS problem. Under the faithfulness assumption, $MB(C)$ can be obtained by first learning a Bayesian network (BN) for $\{\mathbf{F}, C\}$: In such a BN, $MB(C)$ is the union of the parents and children of C and the parents of the children of C [6]. Unfortunately, the existing algorithms for learning BNs from data do not scale to databases with thousands of features [3, 10, 11] and, in this paper, we are interested in solving the FSS problem for databases with thousands of features but with many less instances. Such databases are common in bioinformatics and medicine.

In this paper, we propose an algorithm for learning MBs from data and prove its correctness under the faithfulness assumption. Our algorithm scales to databases with thousands of features because it does not require learning a complete

BN. Furthermore, our algorithm is data efficient because the tests of conditional independence that it performs are not conditioned on unnecessarily large sets of features. In Section 3, we review other existing scalable algorithms for learning MBs from data and show that they are either data inefficient or incorrect. We describe and evaluate our algorithm in Sections 4 and 5, respectively. We close with some discussion in Section 6. We start by reviewing BNs in Section 2.

2 Preliminaries on BNs

The following definitions and theorems can be found in most books on BNs, e.g. [6, 8]. We assume that the reader is familiar with graph and probability theories. We abbreviate if and only if by iff, such that by st, and with respect to by wrt.

Let $\mathbf{U}$ denote a nonempty finite set of discrete random variables. A Bayesian network (BN) for $\mathbf{U}$ is a pair (G, θ), where G is an acyclic directed graph (DAG) whose nodes correspond to the random variables in $\mathbf{U}$, and θ are parameters specifying a conditional probability distribution for each node X given its parents in G, $p(X|Pa_G(X))$. A BN (G, θ) represents a probability distribution for $\mathbf{U}$, $p(\mathbf{U})$, through the factorization $p(\mathbf{U}) = \prod_{X \in \mathbf{U}} p(X|Pa_G(X))$. In addition to $Pa_G(X)$, two abbreviations that we use are $PC_G(X)$ for the parents and children of X in G, and $ND_G(X)$ for the non-descendants of X in G.

Any probability distribution p that can be represented by a BN with DAG G, i.e. by a parameterization θ of G, satisfies certain conditional independencies between the random variables in $\mathbf{U}$ that can be read from G via the d-separation criterion, i.e. if $d\text{-}sep_G(\mathbf{X}, \mathbf{Y}|\mathbf{Z})$, then $\mathbf{X} \perp\!\!\!\perp_p \mathbf{Y}|\mathbf{Z}$ with $\mathbf{X}$, $\mathbf{Y}$ and $\mathbf{Z}$ three mutually disjoint subsets of $\mathbf{U}$. We say that $d\text{-}sep_G(\mathbf{X}, \mathbf{Y}|\mathbf{Z})$ holds when for every undirected path in G between a node in $\mathbf{X}$ and a node in $\mathbf{Y}$ there exits a node Z in the path st either (i) Z does not have two incoming edges in the path and $Z \in \mathbf{Z}$, or (ii) Z has two incoming edges in the path and neither Z nor any of its descendants in G is in $\mathbf{Z}$. The d-separation criterion in G enforces the local Markov property for any probability distribution p that can be represented by a BN with DAG G, i.e. $X \perp\!\!\!\perp_p (ND_G(X) \setminus Pa_G(X))|Pa_G(X)$. A probability distribution p is said to be faithful to a DAG G when $\mathbf{X} \perp\!\!\!\perp_p \mathbf{Y}|\mathbf{Z}$ iff $d\text{-}sep_G(\mathbf{X}, \mathbf{Y}|\mathbf{Z})$.

Theorem 1. *If a probability distribution p is faithful to a DAG G, then (i) for each pair of nodes X and Y in G, X and Y are adjacent in G iff $X \not\perp\!\!\!\perp_p Y|\mathbf{Z}$ for all $\mathbf{Z}$ st $X, Y \notin \mathbf{Z}$, and (ii) for each triplet of nodes X, Y and Z in G st X and Y are adjacent to Z but X and Y are non-adjacent, $X \rightarrow Z \leftarrow Y$ is a subgraph of G iff $X \not\perp\!\!\!\perp_p Y|\mathbf{Z}$ for all $\mathbf{Z}$ st $X, Y \notin \mathbf{Z}$ and $Z \in \mathbf{Z}$.*

Let p denote a probability distribution for $\mathbf{U}$. The Markov boundary of a random variable $X \in \mathbf{U}$, $MB_p(X)$, is defined as any minimal subset of $\mathbf{U}$ st $X \perp\!\!\!\perp_p (\mathbf{U} \setminus (MB_p(X) \cup \{X\}))|MB_p(X)$.

Theorem 2. *If a probability distribution p is faithful to a DAG G, then $MB_p(X)$ for each node X is unique and is the union of $PC_G(X)$ and the parents of the children of X in G.*

We denote $MB_p(X)$ by $MB_G(X)$ when p is faithful to a DAG G.

Table 1. *IAMB*

```
   $IAMB(T, D)$

   /* add true positives to $MB$ */
1  $MB = \emptyset$
2  repeat
3    $Y = arg\,max_{X \in \mathbf{U} \setminus (MB \cup \{T\})} dep_D(X, T|MB)$
4    if $Y \not\perp\!\!\!\perp_D T|MB$ then
5      $MB = MB \cup \{Y\}$
6  until $MB$ does not change
   /* remove false positives from $MB$ */
7  for each $X \in MB$ do
8    if $X \perp\!\!\!\perp_D T|(MB \setminus \{X\})$ then
9      $MB = MB \setminus \{X\}$
10 return $MB$
```

3 Previous Work on Scalable Learning of MBs

In this section, we review two algorithms for learning MBs from data that Tsamardinos et al. introduce in [9, 10, 11, 12], namely the incremental association Markov blanket ($IAMB$) algorithm and the max-min Markov blanket ($MMMB$) algorithm. To our knowledge, these are the only algorithms that have been experimentally shown to scale to databases with thousands of features. However, we show that $IAMB$ is data inefficient and $MMMB$ incorrect. In the algorithms, $X \not\perp\!\!\!\perp_D Y|\mathbf{Z}$ ($X \perp\!\!\!\perp_D Y|\mathbf{Z}$) denotes conditional (in)dependence wrt a learning database D, and $dep_D(X, Y|\mathbf{Z})$ is a measure of the strength of the conditional dependence wrt D. In particular, the algorithms run a test with the G^2 statistic in order to decide on $X \not\perp\!\!\!\perp_D Y|\mathbf{Z}$ or $X \perp\!\!\!\perp_D Y|\mathbf{Z}$ [8], and use the negative p-value of the test as $dep_D(X, Y|\mathbf{Z})$. Both algorithms are based on the assumption that D is faithful to a DAG G, i.e. D is a sample from a probability distribution p faithful to G.

3.1 IAMB

Table 1 outlines $IAMB$. The algorithm receives the target node T and the learning database D as input and returns $MB_G(T)$ in MB as output. The algorithm works in two steps. First, the nodes in $MB_G(T)$ are added to MB (lines 2-6). Since this step is based on the heuristic at line 3, some nodes not in $MB_G(T)$ may be added to MB as well. These nodes are removed from MB in the second step (lines 7-9). Tsamardinos et al. prove the correctness of $IAMB$ under some assumptions.

Theorem 3. *Under the assumptions that the learning database D is an independent and identically distributed sample from a probability distribution p faithful to a DAG G and that the tests of conditional independence and the measure of conditional dependence are correct, the output of $IAMB(T, D)$ is $MB_G(T)$.*

The assumption that the tests of conditional independence and the measure of conditional dependence are correct should be read as follows: $X \perp\!\!\!\perp_D Y|\mathbf{Z}$

and $dep_D(X,Y|\mathbf{Z}) = -1$ if $X \perp\!\!\!\perp_p Y|\mathbf{Z}$, and $X \not\perp\!\!\!\perp_D Y|\mathbf{Z}$ and $dep_D(X,Y|\mathbf{Z}) = 0$ otherwise. In order to maximize accuracy in practice, $IAMB$ performs a test if it is reliable and skips it otherwise. Following the approach in [8], $IAMB$ considers a test to be reliable when the number of instances in D is at least five times the number of degrees of freedom in the test. This means that the number of instances required by $IAMB$ to identify $MB_G(T)$ is at least exponential in the size of $MB_G(T)$, because the number of degrees of freedom in a test is exponential in the size of the conditioning set and some tests will be conditioned on at least $MB_G(T)$. However, depending on the topology of G, it can be the case that $MB_G(T)$ can be identified by conditioning on sets much smaller than $MB_G(T)$, e.g. if G is a tree (see Sections 3.2 and 4). Therefore, $IAMB$ is data inefficient because its data requirements can be unnecessarily high. Note that this reasoning applies not only to the G^2 statistic but to any other statistic as well. Tsamardinos et al. are aware of this drawback and describe some variants of $IAMB$ that alleviate it, though they do not solve it, while still being scalable and correct: The first and second steps can be interleaved ($interIAMB$), and the second step can be replaced by the PC algorithm [8] ($interIAMBnPC$). Finally, as Tsamardinos et al. note, $IAMB$ is similar to the grow-shrink (GS) algorithm [5]. In fact, the only difference is that GS uses a simpler heuristic at line 3: $Y = arg\,max_{X \in \mathbf{U} \setminus (MB \cup \{T\})} dep_D(X,T|\emptyset)$. GS is correct under the assumptions in Theorem 3, but it is data inefficient for the same reason as $IAMB$.

3.2 MMMB

$MMMB$ aims to reduce the data requirements of $IAMB$ while still being scalable and correct. $MMMB$ identifies $MB_G(T)$ in two steps: First, it identifies $PC_G(T)$ and, second, it identifies the rest of the parents of the children of T in G. $MMMB$ uses the max-min parents and children ($MMPC$) algorithm to solve the first step. Table 2 outlines $MMPC$. The algorithm receives the target node T and the learning database D as input and returns $PC_G(T)$ in PC as output. $MMPC$ is similar to $IAMB$, with the exception that $MMPC$ considers any subset of the output as the conditioning set for the tests that it performs and $IAMB$ only considers the output. Tsamardinos et al. prove that, under the assumptions in Theorem 3, the output of $MMPC$ is $PC_G(T)$. We show that this is not always true. The flaw in the proof is the assumption that if $X \notin PC_G(T)$, then $X \perp\!\!\!\perp_p T|\mathbf{Z}$ for some $\mathbf{Z} \subseteq PC_G(T)$ and, thus, any node not in $PC_G(T)$ that enters PC at line 7 is removed from it at line 11. This is not always true for the descendants of T. This is illustrated by running $MMPC(T,D)$ with D faithful to the DAG (a) in Table 2. Neither P nor R enters PC at line 7 because $P \perp\!\!\!\perp_p T|\emptyset$ and $R \perp\!\!\!\perp_p T|\emptyset$. Q enters PC because $Q \not\perp\!\!\!\perp_p T|\mathbf{Z}$ for all $\mathbf{Z}$ st $Q, T \notin \mathbf{Z}$. S enters PC because $S \not\perp\!\!\!\perp_p T|\emptyset$ and $S \not\perp\!\!\!\perp_p T|Q$. Then, $PC = \{Q, S\}$ at line 9. Neither Q nor S leaves PC at line 11. Consequently, the output of $MMPC$ includes S which is not in $PC_G(T)$ and, thus, $MMPC$ is incorrect.

Table 2 outlines $MMMB$. The algorithm receives the target node T and the learning database D as input and returns $MB_G(T)$ in MB as output. The algorithm works in two steps. First, PC and MB are initialized with $PC_G(T)$

Table 2. $MMPC$ and $MMMB$

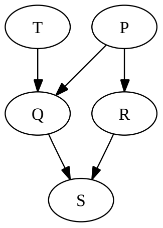
```
$MMPC(T, D)$

   /* add true positives to $PC$ */
 1 $PC = \emptyset$
 2 repeat
 3   for each $X \in \mathbf{U} \setminus (PC \cup \{T\})$ do
 4     $Sep[X] = arg\ min_{\mathbf{Z} \subseteq PC} dep_D(X, T|\mathbf{Z})$
 5   $Y = arg\ max_{X \in \mathbf{U} \setminus (PC \cup \{T\})} dep_D(X, T|Sep[X])$
 6   if $Y \not\perp\!\!\!\perp_D T|Sep[Y]$ then
 7     $PC = PC \cup \{Y\}$
 8 until $PC$ does not change
   /* remove false positives from $PC$ */
 9 for each $X \in PC$ do
10   if $X \perp\!\!\!\perp_D T|\mathbf{Z}$ for some $\mathbf{Z} \subseteq PC \setminus \{X\}$ then
11     $PC = PC \setminus \{X\}$
12 return $PC$
```

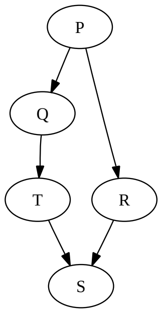
```
$MMMB(T, D)$

   /* add true positives to $MB$ */
 1 $PC = MMPC(T, D)$
 2 $MB = PC$
 3 $CanMB = PC \cup_{X \in PC} MMPC(X, D)$
   /* add more true positives to $MB$ */
 4 for each $X \in CanMB \setminus PC$ do
 5   find any $\mathbf{Z}$ st $X \perp\!\!\!\perp_D T|\mathbf{Z}$ and $X, T \notin \mathbf{Z}$
 6   for each $Y \in PC$ do
 7     if $X \not\perp\!\!\!\perp_D T|\mathbf{Z} \cup \{Y\}$ then
 8       $MB = MB \cup \{X\}$
 9 return $MB$
```

(a)



(b)



and $CanMB$ with $PC_G(T) \cup_{X \in PC_G(T)} PC_G(X)$ by calling $MMPC$ (lines 1-3). $CanMB$ contains the candidates to enter MB. Second, the parents of the children of T in G that are not yet in MB are added to it (lines 4-8). This step is based on the following observation. The parents of the children of T in G that are missing from MB at line 4 are those that are non-adjacent to T in G. These parents are in $CanMB \setminus PC$. Therefore, if $X \in CanMB \setminus PC$ and $Y \in PC$, then X and T are non-adjacent parents of Y in G iff $X \not\perp\!\!\!\perp_p T|\mathbf{Z} \cup \{Y\}$ for any $\mathbf{Z}$ st $X \perp\!\!\!\perp_p T|\mathbf{Z}$ and $X, T \notin \mathbf{Z}$. Note that $\mathbf{Z}$ can be efficiently obtained at line 5: $MMPC$ must have found such a $\mathbf{Z}$ and could have cached it for later retrieval. Tsamardinos et al. prove that, under the assumptions in Theorem 3, the output of $MMMB$ is $MB_G(T)$. We show that this is not always true even if $MMPC$ were correct. The flaw in the proof is the observation that motivates the second step of $MMMB$, which is not true. This is illustrated by running $MMMB(T, D)$ with D faithful to the DAG (b) in Table 2. Let us assume that $MMPC$ is correct. Then, $MB = PC = \{Q, S\}$ and $CanMB = \{P, Q, R, S, T\}$ at line 4. P enters MB at line 8 if $\mathbf{Z} = \{Q\}$ at line 5, because $P \in CanMB \setminus PC$, $S \in PC$, $P \perp\!\!\!\perp_p T|Q$ and $P \not\perp\!\!\!\perp_p T|\{Q, S\}$. Consequently, the output of $MMMB$ can include P which is not in $MB_G(T)$ and, thus, $MMMB$ is incorrect even if $MMPC$ were correct.

In practice, $MMMB$ performs a test if it is reliable and skips it otherwise.

Table 3. *AlgorithmPCD*, *AlgorithmPC* and *AlgorithmMB*

```
AlgorithmPCD(T, D)

1  $PCD = \emptyset$
2  $CanPCD = \mathbf{U} \setminus \{T\}$
3  repeat
      /* remove false positives from $CanPCD$ */
4     for each $X \in CanPCD$ do
5        $Sep[X] = arg\,min_{\mathbf{Z} \subseteq PCD} dep_D(X, T|\mathbf{Z})$
6     for each $X \in CanPCD$ do
7        if $X \perp\!\!\!\perp_D T|Sep[X]$ then
8           $CanPCD = CanPCD \setminus \{X\}$
      /* add the best candidate to $PCD$ */
9     $Y = arg\,max_{X \in CanPCD} dep_D(X, T|Sep[X])$
10    $PCD = PCD \cup \{Y\}$
11    $CanPCD = CanPCD \setminus \{Y\}$
      /* remove false positives from $PCD$ */
12    for each $X \in PCD$ do
13       $Sep[X] = arg\,min_{\mathbf{Z} \subseteq PCD \setminus \{X\}} dep_D(X, T|\mathbf{Z})$
14    for each $X \in PCD$ do
15       if $X \perp\!\!\!\perp_D T|Sep[X]$ then
16          $PCD = PCD \setminus \{X\}$
17 until $PCD$ does not change
18 return $PCD$
```

```
AlgorithmPC(T, D)

1  $PC = \emptyset$
2  for each $X \in AlgorithmPCD(T, D)$ do
3     if $T \in AlgorithmPCD(X, D)$ then
4        $PC = PC \cup \{X\}$
5  return $PC$
```

```
AlgorithmMB(T, D)

   /* add true positives to $MB$ */
1  $PC = AlgorithmPC(T, D)$
2  $MB = PC$
   /* add more true positives to $MB$ */
3  for each $Y \in PC$ do
4     for each $X \in AlgorithmPC(Y, D)$ do
5        if $X \notin PC$ then
6           find $\mathbf{Z}$ st $X \perp\!\!\!\perp_D T|\mathbf{Z}$ and $X, T \notin \mathbf{Z}$
7           if $X \not\perp\!\!\!\perp_D T|\mathbf{Z} \cup \{Y\}$ then
8              $MB = MB \cup \{X\}$
9  return $MB$
```

$MMMB$ follows the same criterion as $IAMB$ to decide whether a test is reliable or not. If $MMMB$ were correct, then it would be data efficient because the number of instances required to identify $MB_G(T)$ would not depend on the size of $MB_G(T)$ but on the topology of G.

4 Scalable, Efficient and Correct Learning of MBs

In this section, we present a new algorithm for learning MBs from data that scales to databases with thousands of features. Like $IAMB$ and $MMMB$, our algorithm is based on the assumption that the learning database D is a sample from a probability distribution p faithful to a DAG G. Unlike $IAMB$, our algorithm is data efficient. Unlike $MMMB$, our algorithm is correct under the assumptions in Theorem 3. Our algorithm identifies $MB_G(T)$ in two steps: First, it identifies $PC_G(T)$ and, second, it identifies the rest of the parents of the children of T in G. Our algorithm, named *AlgorithmMB*, uses *AlgorithmPCD* and *AlgorithmPC* to solve the first step. $X \not\perp\!\!\!\perp_D Y|\mathbf{Z}$, $X \perp\!\!\!\perp_D Y|\mathbf{Z}$ and $dep_D(X, Y|\mathbf{Z})$ are the same as in Section 3. Table 3 outlines *AlgorithmPCD*. The algorithm receives the target node T and the learning database D as input and returns a superset of $PC_G(T)$ in PCD as output. The algorithm tries to minimize the number of nodes not in $PC_G(T)$ that are returned in PCD. The algorithm repeats three steps until PCD does not change. First, some nodes not in $PC_G(T)$ are removed from $CanPCD$, which contains the candidates to enter PCD (lines 4-8). This step is based on the observation that $X \in PC_G(T)$ iff $X \not\perp\!\!\!\perp_p T|\mathbf{Z}$ for all $\mathbf{Z}$ st $X, T \notin \mathbf{Z}$. Second, the candidate most likely to be in $PC_G(T)$ is added to PCD and removed from $CanPCD$ (lines 9-11). Since this step is based on the heurististic at line 9, some nodes not in $PC_G(T)$ may be added to PCD as well. Some of these nodes are removed from PCD in the third step (lines 12-16). This step is based on the same observation as the first step.

Theorem 4. *Under the assumptions that the learning database D is an independent and identically distributed sample from a probability distribution p faithful to a DAG G and that the tests of conditional independence and the measure of conditional dependence are correct, the output of $AlgorithmPCD(T, D)$ includes $PC_G(T)$ and does not include any node in $ND_G(T) \setminus Pa_G(T)$.*

Proof. First, we prove that the nodes in $PC_G(T)$ are included in the output PCD. If $X \in PC_G(T)$, then $X \not\perp\!\!\!\perp_p T|\mathbf{Z}$ for all $\mathbf{Z}$ st $X, T \notin \mathbf{Z}$ (Theorem 1). Consequently, X enters PCD at line 10 and does not leave it thereafter.

Second, we prove that the nodes in $ND_G(T) \setminus Pa_G(T)$ are not included in the output PCD. It suffices to study the last time that lines 12-16 are executed. At line 12, $Pa_G(T) \subseteq PCD$ (see paragraph above). Therefore, if PCD still contains some $X \in ND_G(T) \setminus Pa_G(T)$, then $X \perp\!\!\!\perp_p T|\mathbf{Z}$ for some $\mathbf{Z} \subseteq PCD \setminus \{X\}$ (local Markov property). Consequently, X is removed from PCD at line 16. □

The output of $AlgorithmPCD$ must be further processed in order to obtain $PC_G(T)$, because it may contain some descendants of T in G other than its children. These nodes can be easily identified: If X is in the output of $AlgorithmPCD(T, D)$, then X is a descendant of T in G other than one of its children iff T is not in the output of $AlgorithmPCD(X, D)$. $AlgorithmPC$, which is outlined in Table 3, implements this observation. The algorithm receives the target node T and the learning database D as input and returns $PC_G(T)$ in PC as output. We prove that $AlgorithmPC$ is correct under some assumptions.

Theorem 5. *Under the assumptions that the learning database D is an independent and identically distributed sample from a probability distribution p faithful to a DAG G and that the tests of conditional independence and the measure of conditional dependence are correct, the output of $AlgorithmPC(T, D)$ is $PC_G(T)$.*

Proof. First, we prove that the nodes in $PC_G(T)$ are included in the output PC. If $X \in PC_G(T)$, then $T \in PC_G(X)$. Therefore, X and T satisfy the conditions at lines 2 and 3, respectively (Theorem 4). Consequently, X enters PC at line 4.

Second, we prove that the nodes not in $PC_G(T)$ are not included in the output PC. Let $X \notin PC_G(T)$. If X does not satisfy the condition at line 2, then X does not enter PC at line 4. On the other hand, if X satisfies the condition at line 2, then X must be a descendant of T in G other than one of its children and, thus, T does not satisfy the condition at line 3 (Theorem 4). Consequently, X does not enter PC at line 4. □

Finally, Table 3 outlines $AlgorithmMB$. The algorithm receives the target node T and the learning database D as input and returns $MB_G(T)$ in MB as output. The algorithm works in two steps. First, MB is initialized with $PC_G(T)$

by calling $AlgorithmPC$ (line 2). Second, the parents of the children of T in G that are not yet in MB are added to it (lines 3-8). This step is based on the following observation. The parents of the children of T in G that are missing from MB at line 3 are those that are non-adjacent to T in G. Therefore, if $Y \in PC_G(T)$, $X \in PC_G(Y)$ and $X \notin PC_G(T)$, then X and T are non-adjacent parents of Y in G iff $X \not\perp\!\!\!\perp_p T|\mathbf{Z} \cup \{Y\}$ for any $\mathbf{Z}$ st $X \perp\!\!\!\perp_p T|\mathbf{Z}$ and $X, T \notin \mathbf{Z}$. Note that $\mathbf{Z}$ can be efficiently obtained at line 6: $AlgorithmPCD$ must have found such a $\mathbf{Z}$ and could have cached it for later retrieval. We prove that $AlgorithmMB$ is correct under some assumptions.

Theorem 6. *Under the assumptions that the learning database D is an independent and identically distributed sample from a probability distribution p faithful to a DAG G and that the tests of conditional independence and the measure of conditional dependence are correct, the output of $AlgorithmMB(T, D)$ is $MB_G(T)$.*

Proof. First, we prove that the nodes in $MB_G(T)$ are included in the output MB. Let $X \in MB_G(T)$. Then, either $X \in PC_G(T)$ or $X \notin PC_G(T)$ but X and T have a common child Y in G (Theorem 2). If $X \in PC_G(T)$, then X enters MB at line 2 (Theorem 5). On the other hand, if $X \notin PC_G(T)$ but X and T have a common child Y in G, then X satisfies the conditions at lines 3-5 (Theorem 5) and at lines 6-7 (Theorem 1). Consequently, X enters MB at line 8.

Second, we prove that the nodes not in $MB_G(T)$ are not included in the output MB. Let $X \notin MB_G(T)$. X does not enter MB at line 2 (Theorem 5). If X does not satisfy the conditions at lines 3-6, then X does not enter MB at line 8. On the other hand, if X satisfies the conditions at lines 3-6, then it must be due to either $T \to Y \to X$ or $T \leftarrow Y \leftarrow X$ or $T \leftarrow Y \to X$. Therefore, X does not satisfy the condition at line 7 (faithfulness assumption). Consequently, X does not enter MB at line 8. □

In practice, $AlgorithmMB$ performs a test if it is reliable and skips it otherwise. $AlgorithmMB$ follows the same criterion as $IAMB$ and $MMMB$ to decide whether a test is reliable or not. $AlgorithmMB$ is data efficient because the number of instances required to identify $MB_G(T)$ does not depend on the size of $MB_G(T)$ but on the topology of G. For instance, if G is a tree, then $AlgorithmMB$ does not need to perform any test that is conditioned on more than one node in order to identify $MB_G(T)$, no matter how large $MB_G(T)$ is. $AlgorithmMB$ scales to databases with thousands of features because it does not require learning a complete BN. The experiments in Section 5 confirm it. Like $IAMB$ and $MMMB$, if the assumptions in Theorem 6 do not hold, then $AlgorithmMB$ may not return a MB but an approximation.

5 Experiments

In this section, we evaluate $AlgorithmMB$ on synthetic and real data. We use $interIAMB$ as benchmark (recall Section 3.1). We would have liked to include

Table 4. Results of the experiments with the Alarm and Pigs databases

Database	Instances	Algorithm	Precision	Recall	Distance	Time
Alarm	100	*interIAMB*	0.85±0.06	0.46±0.03	0.54±0.06	0±0
Alarm	100	*AlgorithmMB*	0.79±0.04	0.49±0.05	0.51±0.04	0±0
Alarm	200	*interIAMB*	0.87±0.04	0.59±0.04	0.42±0.04	0±0
Alarm	200	*AlgorithmMB*	0.94±0.03	0.56±0.05	0.38±0.06	0±0
Alarm	500	*interIAMB*	0.91±0.03	0.73±0.03	0.30±0.04	0±0
Alarm	500	*AlgorithmMB*	0.94±0.01	0.72±0.04	0.25±0.04	0±0
Alarm	1000	*interIAMB*	0.93±0.03	0.80±0.01	0.22±0.02	0±0
Alarm	1000	*AlgorithmMB*	0.99±0.01	0.79±0.01	0.17±0.02	0±0
Alarm	2000	*interIAMB*	0.92±0.04	0.83±0.01	0.21±0.04	0±0
Alarm	2000	*AlgorithmMB*	1.00±0.00	0.83±0.02	0.14±0.02	0±0
Alarm	5000	*interIAMB*	0.92±0.02	0.86±0.01	0.18±0.02	0±0
Alarm	5000	*AlgorithmMB*	1.00±0.00	0.86±0.02	0.11±0.02	1±0
Alarm	10000	*interIAMB*	0.92±0.04	0.90±0.01	0.14±0.03	0±0
Alarm	10000	*AlgorithmMB*	1.00±0.00	0.91±0.02	0.07±0.02	1±0
Alarm	20000	*interIAMB*	0.94±0.00	0.92±0.00	0.10±0.00	1±0
Alarm	20000	*AlgorithmMB*	1.00±0.00	0.92±0.00	0.05±0.00	3±0
Pigs	100	*interIAMB*	0.82±0.01	0.59±0.01	0.48±0.02	0±0
Pigs	100	*AlgorithmMB*	0.83±0.01	0.81±0.02	0.29±0.02	0±0
Pigs	200	*interIAMB*	0.80±0.00	0.82±0.00	0.37±0.00	0±0
Pigs	200	*AlgorithmMB*	0.97±0.01	0.96±0.01	0.07±0.01	1±0
Pigs	500	*interIAMB*	0.82±0.00	0.84±0.00	0.34±0.00	0±0
Pigs	500	*AlgorithmMB*	0.98±0.00	1.00±0.00	0.02±0.00	2±0

interIAMBnPC in the evaluation but we were unable to finish the implementation on time. We will include it in an extended version of this paper. We do not consider *GS* because *interIAMB* outperforms it [10]. We do not consider *MMMB* because we are not interested in incorrect algorithms.

5.1 Synthetic Data

These experiments evaluate the accuracy and data efficiency of *AlgorithmMB* wrt those of *interIAMB*. For this purpose, we consider databases sampled from two known BNs, namely the Alarm BN [4] and the Pigs BN [11]. These BNs have 37 and 441 nodes, respectively, and the largest MB consists of eight and 68 nodes, respectively. We run *interIAMB* and *AlgorithmMB* with each node in each BN as target and, then, report the average precision and recall over all the nodes for each BN. Precision is the number of true positives in the output divided by the number of nodes in the output. Recall is the number of true positives in the output divided by the number of true positives in the BN. We also combine precision and recall as $\sqrt{(1-\text{precision})^2+(1-\text{recall})^2}$ to measure the Euclidean distance from perfect precision and recall. Finally, we also report the running time in seconds. Both algorithms are written in C++ and all the experiments are run on a Pentium 2.4 GHz, 512 MB RAM and Windows 2000. The significance level for the tests of conditional independence is 0.01.

Table 4 summarizes the results of the experiments with the Alarm and Pigs databases for different number of instances. Each entry in the table shows the average and standard deviation values over 10 databases (the same 10 databases for *interIAMB* and *AlgorithmMB*). For the Alarm databases, both algorithms achieve similar recall but $AlgorithmMB$ scores higher precision and, thus, shorter distance than $interIAMB$. Therefore, $AlgorithmMB$ usually returns fewer false positives than $interIAMB$. The explanation is that $AlgorithmMB$ performs more tests than $interIAMB$ and this makes it harder for false positives to enter the output. See, for instance, the heuristic in $AlgorithmPCD$ and the double check in $AlgorithmPC$. For this reason, we expect $interIAMBnPC$ to perform better than $interIAMB$ but worse than $AlgorithmMB$. For the Pigs databases where larger MBs exist, $AlgorithmMB$ outperforms $interIAMB$ in terms of precision, recall and distance. For instance, $AlgorithmMB$ correctly identifies the MB of node 435 of the Pigs BN, which consists of 68 nodes, with only 500 instances, while $interIAMB$ performs poorly for this node (precision=1.00, recall=0.04 and distance=0.96). The explanation is that, unlike $interIAMB$, $AlgorithmMB$ does not need to condition on the whole MB to identify it. Note that $interIAMBnPC$ could not have done better than $interIAMB$ for this node. In fact, $interIAMB$ and $interIAMBnPC$ require a number of instances at least exponential in 68 for perfect precision and recall for this node. Consequently, we can conclude that $AlgorithmMB$ is more accurate and data efficient than $interIAMB$ and, seemingly, $interIAMBnPC$.

5.2 Real Data

These experiments evaluate the ability of $AlgorithmMB$ wrt that of $interIAMB$ to solve a real-world FSS problem involving thousands of features. Specifically, we consider the Thrombin database which was provided by DuPont Pharmaceuticals for the KDD Cup 2001 and is exemplary of the real-world drug design environment [1]. The database contains 2543 instances characterized by 139351 binary features. Each instance represents a drug compound tested for its ability to bind to a target site on thrombin, a key receptor in blood clotting. The features describe the three-dimensional properties of the compounds. Each compound is labelled with one out of two classes, either it binds to the target site or not. The task of the KDD Cup 2001 is to learn a classifier from 1909 given compounds in order to predict binding affinity. The accuracy of the classifier is evaluated wrt the remaining 634 compounds. The accuracy is computed as the average of the accuracy on true binding compounds and the accuracy on true non-binding compounds. The Thrombin database is particularly challenging for two reasons. First, the learning data are extremely imbalanced: Only 42 compounds out of 1909 bind. Second, the testing data are not sampled from the same probability distribution as the learning data, because the compounds in the testing data were synthesized based on the assay results recorded in the learning data. Better than 60 % accuracy is impressive according to [1].

As discussed in Section 1 and in [1], solving the FSS problem for the Thrombin database is crucial due to the excessive number of features. Since the truly relevant features for binding affinity are unknown, we cannot use the same performance criteria for $interIAMB$ and $AlgorithmMB$ as in Section 5.1. Instead, we run each algorithm on the learning data and, then, use only the features in the output to learn a naive Bayesian (NB) classifier [2], whose accuracy on the

Table 5. Results of the experiments with the Thrombin database

Algorithm	**Features**	**Accuracy**	**Time**
Winner KDD Cup 2001 with TANB	4.00	0.68	Not available
Winner KDD Cup 2001 with NB	4.00	0.67	Not available
interIAMB	8.00±0.00	0.52±0.02	3102±69
AlgorithmMB	4.00±1.00	0.60±0.02	8631±915

testing data is our performance criterion: The higher the accuracy the better the features selected and, thus, the algorithm. We also report the number of features selected and the running time of the algorithm in seconds. The rest of the experimental setting is the same as in Section 5.1.

Table 5 summarizes the results of the experiments with the Thrombin database. The table shows the average and standard deviation values over 10 runs of *interIAMB* and *AlgorithmMB* because the algorithms break ties at random and, thus, different runs can return different MBs. The table also shows the accuracy of the winner of the KDD Cup 2001, a tree augmented naive Bayesian (TANB) classifier [2] with the features 10695, 16794, 79651 and 91839 and only one augmenting edge between 10695 and 16794, as well as the accuracy of a NB classifier with the same features as the winning TANB. We were unable to learn a NB classifier with all the 139351 features. The winning TANB and NB are clearly more accurate than *interIAMB* and *AlgorithmMB*. The explanation may be that the score used to learn the winning TANB, the area under the ROC curve with a user-defined threshold to control complexity, works better than the tests of conditional independence in *interIAMB* and *AlgorithmMB* when the learning data are as imbalance as in the Thrombin database. This question is worth of further investigation, but it is out of the scope of this paper. What is more important in this paper is the performance of *AlgorithmMB* wrt that of *interIAMB*. The former is clearly more accurate than the latter, though it is slower because it performs more tests. It is worth mentioning that, while the best run of *interIAMB* reaches 54 % accuracy, two of the runs of *AlgorithmMB* achieve 63 % accuracy which, according to [1], is impressive. The features selected by *AlgorithmMB* in these two runs are 12810, 28852, 79651, 91839 and either 106279 or 109171. We note that no existing algorithm for learning BNs from data can handle such a high-dimensional database as the Thrombin database.

6 Discussion

We have introduced *AlgorithmMB*, an algorithm for learning the MB of a node from data without having to learn a complete BN. We have proved that *AlgorithmMB* is correct under the faithfulness assumption. We have shown that *AlgorithmMB* is scalable and data efficient and, thus, it can solve the FSS problem for databases with thousands of features but with many less instances. Since there is no algorithm for learning BNs from data that scales to such high-dimensional databases, it is very important to develop algorithms for learning

MBs from data that, like *AlgorithmMB*, avoid learning a complete BN as an intermediate step. To our knowledge, the only work that has addressed the poor scalability of the existing algorithms for learning BNs from data is [3], where Friedman et al. propose restricting the search for the parents of each node to some promising nodes that are heuristically selected. Therefore, Friedman et al. do not develop a scalable algorithm for learning BNs from data but some heuristics to use prior to running any existing algorithm. Unfortunately, Friedman et al. do not evaluate the heuristics for learning MBs from data. It is worth mentioning that learning the MB of each node can be a helpful intermediate step in the process of learning a BN from data [5]. As part of *AlgorithmMB*, we have introduced *AlgorithmPC*, an algorithm that returns the parents and children of a target node. In [7], we have reused this algorithm for growing BN models of gene networks from seed genes.

Acknowledgements

We thank Björn Brinne for his comments. This work is funded by the Swedish Foundation for Strategic Research (SSF) and Linköping Institute of Technology.

References

1. Cheng, J., Hatzis, C., Hayashi, H., Krogel, M. A., Morishita, S., Page, D., Sese, J.: KDD Cup 2001 Report. ACM SIGKDD Explorations **3** (2002) 1–18. See also http://www.cs.wisc.edu/∼dpage/kddcup2001/
2. Friedman, N., Geiger, D., Goldszmidt, M.: Bayesian Network Classifiers. Machine Learning **29** (1997) 131-163
3. Friedman, N., Nachman, I., Peér, D.: Learning Bayesian Network Structure from Massive Datasets: The "Sparse Candidate" Algorithm. UAI (1999) 206–215
4. Herskovits, E. H.: Computer-Based Probabilistic-Network Construction. PhD Thesis, Stanford University (1991)
5. Margaritis, D., Thrun, S.: Bayesian Network Induction via Local Neighborhoods. NIPS (2000) 505–511
6. Neapolitan, R. E.: Learning Bayesian Networks. Prentice Hall (2003)
7. Peña, J. M., Björkegren, J., Tegnér, J.: Growing Bayesian Network Models of Gene Networks from Seed Genes. Submitted (2005)
8. Spirtes, P., Glymour, C., Scheines, R.: Causation, Prediction, and Search. Springer-Verlag (1993)
9. Tsamardinos, I., Aliferis, C. F.: Towards Principled Feature Selection: Relevancy, Filters and Wrappers. AI & Statistics (2003)
10. Tsamardinos, I., Aliferis, C. F., Statnikov, A.: Algorithms for Large Scale Markov Blanket Discovery. FLAIRS (2003) 376-380
11. Tsamardinos, I., Aliferis, C. F., Statnikov, A.: Time and Sample Efficient Discovery of Markov Blankets and Direct Causal Relations. KDD (2003) 673–678
12. Tsamardinos, I., Aliferis, C. F., Statnikov, A.: Time and Sample Efficient Discovery of Markov Blankets and Direct Causal Relations. Technical Report DSL TR-03-04, Vanderbilt University (2003)

This article was processed using the LaTeX macro package with LLNCS style