BIOINFORMATICS

Vol. 00 no. 00 2005
Pages 1–7

Growing Bayesian Network Models of Gene Networks from Seed Genes

J. M. Peña [a,*], J. Björkegren [b], J. Tegnér [a,b]

[a]Computational Biology, Department of Physics and Measurement Technology, Linköping University, 581 83 Linköping, Sweden, [b]Center for Genomics and Bioinformatics, Karolinska Institutet, 171 77 Stockholm, Sweden

ABSTRACT

Motivation: For the last few years, Bayesian networks (BNs) have received increasing attention from the computational biology community as models of gene networks, though learning them from gene expression data is problematic: Most gene expression databases contain measurements for thousands of genes, but the existing algorithms for learning BNs from data do not scale to such high-dimensional databases. This means that the user has to decide in advance which genes are included in the learning process, typically no more than a few hundreds, and which genes are excluded from it. This is not a trivial decision. We propose an alternative approach to overcome this problem.

Results: We propose a new algorithm for learning BN models of gene networks from gene expression data. Our algorithm receives a seed gene S and a positive integer R from the user, and returns a BN for those genes that depend on S such that less than R other genes mediate the dependency. Our algorithm grows the BN, which initially only contains S, by repeating the following step $R+1$ times and, then, pruning some genes: Find the parents and children of all the genes in the BN and add them to it. Intuitively, our algorithm provides the user with a window of radius R around S to look at the BN model of a gene network without having to exclude any gene in advance. We prove that our algorithm is correct under the faithfulness assumption. We evaluate our algorithm on simulated and biological data (Rosetta compendium) with satisfactory results.

Contact: jmp@ifm.liu.se

1 INTRODUCTION

Much of a cell's complex behavior can be explained through the concerted activity of genes and gene products. This concerted activity is typically represented as a network of interacting genes. Identifying this gene network is crucial for understanding the behavior of the cell which, in turn, can lead to better diagnosis and treatment of diseases.

For the last few years, Bayesian networks (BNs) (Neapolitan, 2003; Pearl, 1988) have received increasing attention from the computational biology community as models of gene networks, e.g. (Badea, 2003; Bernard and Hartemink, 2005; Friedman *et al*., 2000; Hartemink *et al*., 2002; Ott *et al*., 2004; Pe'er *et al*., 2001; Peña, 2004). A BN model of a gene network represents a probability distribution for the genes in the network. The BN minimizes the number of parameters needed to specify the probability distribution by taking advantage of the conditional independencies between the genes. These conditional independencies are encoded in an acyclic directed graph (DAG) to help visualization and reasoning. Learning BN models of gene networks from gene expression data is problematic: Most gene expression databases contain measurements for thousands of genes, e.g. (Hughes *et al*., 2000; Spellman *et al*., 1998), but the existing algorithms for learning BNs from data do not scale to such high-dimensional databases (Friedman *et al*., 1999; Tsamardinos *et al*., 2003). This implies that in the papers cited above, for instance, the authors have to decide in advance which genes are included in the learning process, in all the cases less than 1000, and which genes are excluded from it. This is not a trivial decision. We propose an alternative approach to overcome this problem.

In this paper, we propose a new algorithm for learning BN models of gene networks from gene expression data. Our algorithm receives a seed gene S and a positive integer R from the user, and returns a BN for those genes that depend on S such that less than R other genes mediate the dependency. Our algorithm grows the BN, which initially only contains S, by repeating the following step $R+1$ times and, then, pruning some genes: Find the parents and children of all the genes in the BN and add them to it. Intuitively, our algorithm provides the user with a window of radius R around S to look at the BN model of a gene network without having to exclude any gene in advance.

The rest of the paper is organized as follows. In Section 2, we review BNs. In Sections 3, we describe our new algorithm. In Section 4, we evaluate our algorithm on simulated and biological data (Rosetta compendium (Hughes *et al*., 2000)) with satisfactory results. Finally, in Section 5, we discuss related works and possible extensions to our algorithm.

2 BAYESIAN NETWORKS

The following definitions and theorem can be found in most books on Bayesian networks, e.g. (Neapolitan, 2003; Pearl, 1988). 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 non-empty finite set of 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 $\mathbf{U}$, $p(\mathbf{U})$,

[*] To whom correspondence should be addressed.

through the factorization $p(\mathbf{U}) = \prod_{X \in \mathbf{U}} p(X|Pa_G(X))$. Hereafter, $PC_G(X)$ denotes the parents and children of X in G, and $ND_G(X)$ 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 $\mathbf{U}$. The statement $d\text{-}sep_G(\mathbf{X}, \mathbf{Y}|\mathbf{Z})$ is true when for every undirected path in G between a node in $\mathbf{X}$ and a node in $\mathbf{Y}$ there exits a node W in the path st either (i) W does not have two parents in the path and $W \in \mathbf{Z}$, or (ii) W has two parents in the path and neither W nor any of its descendants in G is in $\mathbf{Z}$. A probability distribution p is said to be faithful to a DAG G when $\mathbf{X} \perp\!\!\!\perp_p \mathbf{Y}|\mathbf{Z}$ iff $d\text{-}sep_G(\mathbf{X}, \mathbf{Y}|\mathbf{Z})$.

The nodes W, X and Y form an immorality in a DAG G when $X \rightarrow W \leftarrow Y$ is the subgraph of G induced by W, X and Y. Two DAGs are equivalent when they represent the same d-separation statements. The equivalence class of a DAG G is the set of DAGs that are equivalent to G.

Theorem 1. *Two DAGs are equivalent iff they have the same adjacencies and the same immoralities.*

Two nodes are at distance R in a DAG G when the shortest undirected path in G between them is of length R. $G(X)^R$ denotes the subgraph of G induced by the nodes at distance at most R from X in G.

3 GROWING PARENTS AND CHILDREN ALGORITHM

A BN models a gene network by equating each gene with a random variable or node. We note that the DAG of a BN model of a gene network does not necessarily represent physical interactions between genes but conditional (in)dependencies. We aim to learn BN models of gene networks from gene expression data. This will help us to understand the probability distributions underlying the gene networks in terms of conditional (in)dependencies between genes.

Learning a BN from data consists in, first, learning a DAG and, then, learning a parameterization of the DAG. Like the works cited in Section 1, we focus on the former task because, under the assumption that the learning data contain no missing values, the latter task can be efficiently solved according to the maximum likelihood (ML) or maximum a posteriori (MAP) criterion (Neapolitan, 2003; Pearl, 1988). To appreciate the complexity of learning a DAG, we note that the number of DAGs is super-exponential in the number of nodes (Robinson, 1973). In this section, we present a new algorithm for learning a DAG from a database D. The algorithm, named growing parents and children algorithm or $AlgorithmGPC$ for short, is based on the faithfulness assumption, i.e. on the assumption that D is a sample from a probability distribution p faithful to a DAG G. $AlgorithmGPC$ receives a seed node S and a positive integer R as input, and returns a DAG that is equivalent to $G(S)^R$. $AlgorithmGPC$ grows the DAG, which initially only contains S, by repeating the following step $R+1$ times and, then, pruning some nodes: Find the parents and children of all the nodes in the DAG and add them to it. Therefore, a key step in $AlgorithmGPC$ is the identification of $PC_G(X)$ for a given node X in G. The functions $AlgorithmPCD$ and $AlgorithmPC$ solve this step. We have previously introduced these two functions in (Peña *et al.*, 2005) to learn Markov boundaries from high-dimensional data. They are correct versions of an incorrect function proposed in (Tsamardinos *et al.*, 2003).

Hereafter, $X \not\perp\!\!\!\perp_D Y|\mathbf{Z}$ ($X \perp\!\!\!\perp_D Y|\mathbf{Z}$) denotes conditional (in)dependence wrt the learning database D, and $dep_D(X, Y|\mathbf{Z})$ is a measure of the strength of the conditional dependence wrt D. In order to decide on $X \not\perp\!\!\!\perp_D Y|\mathbf{Z}$ or $X \perp\!\!\!\perp_D Y|\mathbf{Z}$, $AlgorithmGPC$ runs a χ^2 test when D is discrete and a Fisher's z test when D is continuous and, then, uses the negative p-value of the test as $dep_D(X, Y|\mathbf{Z})$. See (Spirtes *et al.*, 1993) for details on these tests.

Table 1 outlines $AlgorithmPCD$. The algorithm receives the node S as input and returns a superset of $PC_G(S)$ in PCD. The algorithm tries to minimize the number of nodes not in $PC_G(S)$ that are returned in PCD. The algorithm repeats the following three steps until PCD does not change. First, some nodes not in $PC_G(S)$ are removed from $CanPCD$, which contains the candidates to enter PCD (lines 4-8). Second, the candidate most likely to be in $PC_G(S)$ is added to PCD and removed from $CanPCD$ (lines 9-11). Since this step is based on the heuristic at line 9, some nodes not in $PC_G(S)$ may be added to PCD. Some of these nodes are removed from PCD in the third step (lines 12-16). The first and third steps are based on the faithfulness assumption. $AlgorithmPCD$ is correct under some assumptions. See Appendix A for the proof.

Theorem 2. *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 are correct, the output of $AlgorithmPCD(S)$ includes $PC_G(S)$ but does not include any node in $ND_G(S) \setminus Pa_G(S)$.*

The assumption that the tests of conditional independence are correct means that $X \perp\!\!\!\perp_D Y|\mathbf{Z}$ iff $X \perp\!\!\!\perp_p Y|\mathbf{Z}$.

The output of $AlgorithmPCD(S)$ must be further processed in order to obtain $PC_G(S)$, because it may contain some descendants of S in G other than its children. These nodes can be easily identified: If X is in the output of $AlgorithmPCD(S)$, then X is a descendant of S in G other than one of its children iff S is not in the output of $AlgorithmPCD(X)$. $AlgorithmPC$, which is outlined in Table 1, implements this observation. The algorithm receives the node S as input and returns $PC_G(S)$ in PC. $AlgorithmPC$ is correct under some assumptions. See Appendix A for the proof.

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 are correct, the output of $AlgorithmPC(S)$ is $PC_G(S)$.*

Finally, Table 1 outlines $AlgorithmGPC$. The algorithm receives the seed node S and the positive integer R as input, and returns a DAG in DAG that is equivalent to $G(S)^R$. The algorithm works in two phases based on Theorem 1. In the first phase, the adjacencies in $G(S)^R$ are added to DAG, which initially only contains S, by repeating the following step $R+1$ times and, then, pruning some nodes: $PC_G(X)$ is obtained by calling $AlgorithmPC(X)$ for each node X in DAG (lines 3-4) and, then, $PC_G(X)$ is added to DAG by calling

Table 1. $AlgorithmPCD$, $AlgorithmPC$ and $AlgorithmGPC$

```
    $AlgorithmPCD(S)$

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

```
    $AlgorithmPC(S)$

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

```
    $AlgorithmGPC(S, R)$

 1  $DAG = \{S\}$
 2  for $1, \ldots, R+1$ do
 3    for each $X \in DAG$ do
 4      $PC[X] = AlgorithmPC(X)$
 5    for each $X \in DAG$ do
 6      $AddAdjacencies(DAG, X, PC[X])$
 7  $Prune(DAG)$
 8  $AddImmoralities(DAG)$
 9  return $DAG$
```

$AddAdjacencies(DAG, X, PC_G(X))$ for each node X in DAG (lines 5-6). The function $AddAdjacencies(DAG, X, PC_G(X))$ simply adds the nodes in $PC_G(X)$ to DAG and, then, links each of them to X with an undirected edge. In practice, $AlgorithmPC$ and $AddAdjacencies$ are not called for each node in DAG but only for those they have not been called before for. Since lines 3-6 are executed $R+1$ times, the nodes at distance $R+1$ from S in G are added to DAG, though they do not belong to $G(S)^R$. These nodes are removed from DAG by calling $Prune(DAG)$ (line 7). In the second phase of $AlgorithmGPC$, the immoralities in $G(S)^R$ are added to DAG by calling $AddImmoralities(DAG)$ (line 8). For each triplet of nodes W, X and Y st the subgraph of DAG induced by them is $X - W - Y$, the function $AddImmoralities(DAG)$ adds the immorality $X \rightarrow W \leftarrow Y$ to DAG iff $X \not\!\perp\!\!\!\perp_D Y|\mathbf{Z} \cup \{W\}$ for any $\mathbf{Z}$ st $X \perp\!\!\!\perp_D Y|\mathbf{Z}$ and $X, Y \notin \mathbf{Z}$. In practice, such a $\mathbf{Z}$ can be efficiently obtained: $AlgorithmPCD$ must have found such a $\mathbf{Z}$ and could have cached it for later retrieval. The function $AddImmoralities(DAG)$ is based on the faithfulness assumption.

We note that the only directed edges in DAG are those in the immoralities. In order to obtain a DAG, the undirected edges in DAG can be oriented in any direction as long as neither directed cycles nor new immoralities are created. Therefore, strictly speaking, $AlgorithmGPC$ returns an equivalence class of DAGs rather than a single DAG (Theorem 1). $AlgorithmGPC$ is correct under some assumptions. See Appendix A for the proof.

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 are correct, the output of $AlgorithmGPC(S, R)$ is the equivalence class of $G(S)^R$.*

Though the assumptions in Theorem 4 may not hold in practice, correctness is a desirable property for an algorithm to have and, unfortunately, most of the existing algorithms for learning BNs from data lack it.

4 EVALUATION

In this section, we evaluate $AlgorithmGPC$ on simulated and biological data (Rosetta compendium (Hughes *et al.*, 2000)).

4.1 Simulated Data

We consider databases sampled from two discrete BNs that have been previously used as benchmarks for algorithms for learning BNs from data, namely the Alarm BN (37 nodes and 46 edges) (Herskovits, 1991) and the Pigs BN (441 nodes and 592 edges) (Jensen, 1997). We also consider databases sampled from Gaussian networks (GNs) (Geiger and Heckerman, 1994), a class of continuous BNs. We generate random GNs as follows. The DAG has 50 nodes, the number of edges is uniformly drawn from [50, 100], and the edges link uniformly drawn pairs of nodes. Each node follows a Gaussian distribution whose mean depends linearly on the value of its parents. For each node, the unconditional mean, the parental linear coefficients and the conditional standard deviation are uniformly drawn from [-3, 3], [-3, 3] and [1, 3], respectively. We consider three sizes for the databases sampled, namely 100, 200 and 500 instances. We do not claim that the databases sampled resemble gene expression databases, apart from the number of instances. However, they make it possible to compare the output of $AlgorithmGPC$ with the DAGs of the BNs sampled. This will provide us with some insight into the performance of $AlgorithmGPC$ before we turn our attention to gene expression data in the next section. Since we

Table 2. Adjacency precision and recall of $AlgorithmGPC$

Data	Size	R	Precision	Recall	Precision$_5$	Recall$_5$
Alarm	100	1	0.88±0.06	0.52±0.05	0.89±0.06	0.34±0.04
Alarm	100	2	0.87±0.08	0.33±0.05	0.87±0.10	0.23±0.04
Alarm	200	1	0.94±0.04	0.64±0.06	0.97±0.06	0.45±0.05
Alarm	200	2	0.93±0.05	0.44±0.06	0.96±0.04	0.35±0.06
Alarm	500	1	0.97±0.03	0.78±0.03	0.99±0.03	0.57±0.07
Alarm	500	2	0.97±0.04	0.63±0.03	0.99±0.01	0.49±0.04
Pigs	100	1	0.70±0.01	0.75±0.02	0.85±0.03	0.55±0.02
Pigs	100	2	0.58±0.02	0.53±0.02	0.68±0.03	0.36±0.02
Pigs	200	1	0.85±0.02	0.93±0.01	0.96±0.01	0.81±0.01
Pigs	200	2	0.80±0.02	0.78±0.02	0.87±0.02	0.63±0.03
Pigs	500	1	0.88±0.01	1.00±0.00	0.96±0.02	1.00±0.00
Pigs	500	2	0.85±0.01	1.00±0.00	0.90±0.01	1.00±0.00
GNs	100	1	0.86±0.06	0.51±0.10	0.91±0.09	0.38±0.09
GNs	100	2	0.84±0.07	0.29±0.10	0.88±0.10	0.20±0.08
GNs	200	1	0.88±0.05	0.60±0.12	0.92±0.09	0.43±0.12
GNs	200	2	0.85±0.06	0.38±0.15	0.88±0.10	0.26±0.12
GNs	500	1	0.88±0.05	0.67±0.10	0.91±0.06	0.51±0.14
GNs	500	2	0.85±0.07	0.46±0.13	0.86±0.09	0.33±0.14

Table 3. Immorality precision and recall of $AlgorithmGPC$

Data	Size	R	Precision	Recall	Precision$_5$	Recall$_5$
Alarm	100	1	0.82±0.14	0.28±0.09	0.00±0.00	0.00±0.00
Alarm	100	2	0.79±0.12	0.18±0.06	0.56±0.31	0.06±0.04
Alarm	200	1	0.80±0.11	0.46±0.07	1.00±0.00	0.03±0.04
Alarm	200	2	0.78±0.10	0.29±0.05	0.52±0.11	0.12±0.03
Alarm	500	1	0.90±0.07	0.62±0.04	1.00±0.00	0.19±0.12
Alarm	500	2	0.92±0.05	0.46±0.06	0.82±0.13	0.24±0.05
Pigs	100	1	0.65±0.02	0.46±0.03	0.49±0.05	0.35±0.05
Pigs	100	2	0.55±0.02	0.41±0.03	0.59±0.06	0.27±0.02
Pigs	200	1	0.83±0.02	0.76±0.03	0.69±0.08	0.64±0.04
Pigs	200	2	0.76±0.02	0.71±0.02	0.73±0.04	0.58±0.03
Pigs	500	1	0.90±0.01	0.97±0.02	0.89±0.05	0.94±0.04
Pigs	500	2	0.83±0.02	0.95±0.02	0.82±0.04	0.94±0.02
GNs	100	1	0.59±0.22	0.15±0.09	0.41±0.32	0.04±0.07
GNs	100	2	0.59±0.22	0.09±0.07	0.55±0.28	0.05±0.08
GNs	200	1	0.70±0.17	0.25±0.12	0.52±0.32	0.09±0.11
GNs	200	2	0.70±0.17	0.17±0.11	0.59±0.29	0.08±0.07
GNs	500	1	0.67±0.14	0.34±0.13	0.56±0.29	0.19±0.17
GNs	500	2	0.68±0.14	0.24±0.13	0.61±0.21	0.13±0.11

use $R = 1, 2$ in the next section, it seems reasonable to use $R = 1, 2$ in this section as well.

The comparison between the output of $AlgorithmGPC$ and the DAGs of the BNs sampled should be done in terms of adjacencies and immoralities (Theorem 1). Specifically, we proceed as follows for each database sampled from a BN with DAG G. We first run $AlgorithmGPC$ with each node in G as the seed node S and $R = 1, 2$ and, then, report the average adjacency (immorality) precision and recall for each value of R. Adjacency (immorality) precision is the number of adjacencies (immoralities) in the output of $AlgorithmGPC$ that are also in $G(S)^R$ divided by the number of adjacencies (immoralities) in the output. Adjacency (immorality) recall is the number of adjacencies (immoralities) in the output of $AlgorithmGPC$ that are also in $G(S)^R$ divided by the number of adjacencies (immoralities) in $G(S)^R$. We find important to monitor wether the performance of $AlgorithmGPC$ is sensitive or not to the degree of S. For this purpose, we also report the average adjacency (immorality) precision and recall over the nodes in G with five or more parents and children (four nodes in the Alarm BN and 39 nodes in the Pigs BN). The significance level for the tests of conditional independence is the standard 0.05.

Table 2 summarizes the adjacency precision and recall of $AlgorithmGPC$. The columns Precision and Recall show the average adjacency precision and recall, respectively, over all the nodes. The columns Precision$_5$ and Recall$_5$ show the average adjacency precision and recall, respectively, over the nodes with five or more parents and children. Each row in the table shows average and standard deviation values over 10 databases of the corresponding size for the Alarm and Pigs BNs, and over 50 databases for the GNs. We reach two conclusions from the table. First, the adjacency precision of $AlgorithmGPC$ is high in general, though it slightly degrades with R. Second, the adjacency recall of $AlgorithmGPC$ is lower than the adjacency precision, and degrades with both the degree of S and R. This is not surprising given the small sizes of the learning databases.

Table 3 summarizes the immorality precision and recall of $AlgorithmGPC$. The main conclusion that we obtain from the table is that $AlgorithmGPC$ performs better for learning adjacencies than for learning immoralities. This is particularly noticeable for GNs. The reason is that learning adjacencies as in $AlgorithmGPC$ is more robust than learning immoralities. In other words, learning immoralities as in $AlgorithmGPC$ is more sensitive to any error previously made than learning adjacencies. This problem has been previously noted in (Badea, 2003, 2004; Spirtes *et al.*, 1993). A solution to it has been proposed in (Badea, 2003, 2004). We plan to implement it in a future version of $AlgorithmGPC$.

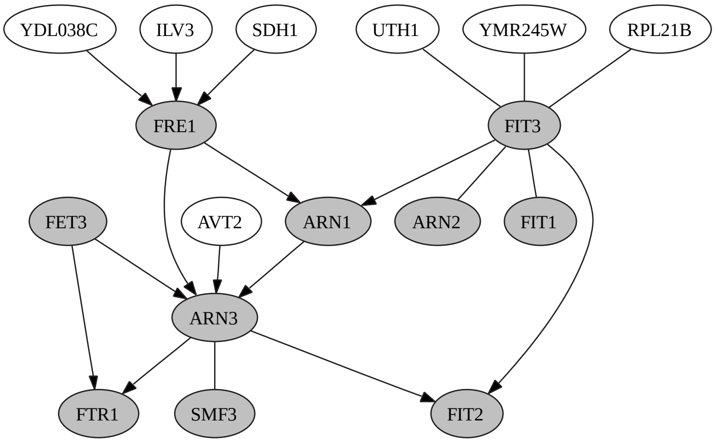

Fig. 1. BN model of the iron homeostasis pathway learnt by $AlgorithmGPC$ from the Rosetta compendium with ARN1 as the seed gene S and $R = 2$. Gray-colored genes are related to iron homeostasis according to (Jensen and Culotta, 2002; Lesuisse *et al*., 2001; Philpott *et al*., 2002; Protchenko *et al*., 2001), while white-colored genes are not known to us to be related to iron homeostasis.

In short, the most noteworthy feature of $AlgorithmGPC$ is its high adjacency precision. This is an important feature because it implies that the adjacencies returned are highly reliable, i.e. there are few false positives among them.

4.2 Biological Data

We use the Rosetta compendium (Hughes *et al*., 2000) in order to illustrate the usefulness of $AlgorithmGPC$ to learn biologically coherent BN models of gene networks from gene expression data. The Rosetta compendium consists of 300 full-genome expression profiles of the yeast *Saccharomyces cerevisiae*. In other words, the learning database consists of 300 instances and 6316 continuous random variables.

Iron is an essential nutrient for virtually every organism, but it is also potentially toxic to cells. We are interested in learning about the iron homeostasis pathway in yeast, which regulates the uptake, storage, and utilization of iron so as to keep it at a non-toxic level. According to (Lesuisse *et al*., 2001; Philpott *et al*., 2002; Protchenko *et al*., 2001), yeast can use two different high-affinity mechanisms, reductive and non-reductive, to take up iron from the extracellular medium. Genes FRE1, FRE2, FTR1 and FET3 control the reductive mechanism, while genes ARN1, ARN2, ARN3 and ARN4 control the non-reductive mechanism. Genes FIT1, FIT2 and FIT3 facilitate iron transport. The iron homeostasis pathway in yeast has been previously used in (Margolin *et al*., 2004; Pe'er *et al*., 2001) to evaluate the accuracy of their algorithms for learning models of gene networks from gene expression data. Specifically, both papers report models of the iron homeostasis pathway learnt from the Rosetta compendium, centered at ARN1 and with a radius of two. Therefore, we run $AlgorithmGPC$ with ARN1 as the seed gene S and $R = 2$. The significance level for the tests of conditional independence is the standard 0.05. The output of $AlgorithmGPC$ is depicted in Figure 1. Gray-colored genes are related to iron homeostasis, while white-colored genes are not known to us to be related to iron homeostasis. The gray-colored genes include nine of the 11 genes mentioned above as related to iron homeostasis, plus SMF3 which has been proposed to function in iron transport in (Jensen and Culotta, 2002). If $R = 1$, then the output involves four genes, all of them related to iron homeostasis. If $R = 2$, then the output involves 17 genes, 10 of them related to iron homeostasis. Therefore, the output of $AlgorithmGPC$ is rich in genes related to iron homeostasis. We note that all the genes related to iron homeostasis are dependent one on another, and that any node that mediates these dependencies is also related to iron homeostasis. This is consistent with the conclusions drawn in Section 4.1, namely that the adjacencies returned by $AlgorithmGPC$ are highly reliable. Regarding running time, $AlgorithmGPC$ takes 6 minutes for $R = 1$ and 37 minutes for $R = 2$ (C++ implementation, not particularly optimized for speed, and run on a Pentium 2.4 GHz, 512 MB RAM and Windows 2000). Roughly speaking, we expect the running time of $AlgorithmGPC$ to be exponential in R. However, R will usually be small because we will usually be interested in those genes that depend on S and none or few other genes mediate the dependency. This is also the case in (Margolin *et al*., 2004; Pe'er *et al*., 2001).

In comparison, the model of the iron homeostasis pathway in (Margolin *et al*., 2004) involves 26 genes (16 related to iron homeostasis), while the model in (Pe'er *et al*., 2001) involves nine genes (six related to iron homeostasis). Further comparison with the latter paper which, unlike the former, learns BN models of gene networks makes clear the main motivation of our work. In order for their algorithm to be applicable, Pe'er *et al*. focus on 565 relevant genes selected in advance and, thus, exclude the remaining 5751 genes from the learning process. On the other hand, $AlgorithmGPC$ produces a biologically coherent output, only requires identifying a single relevant gene in advance, and no gene is excluded from the learning process.

5 DISCUSSION

We have introduced $AlgorithmGPC$, an algorithm for growing BN models of gene networks from seed genes. We have evaluated it on synthetic and biological data with satisfactory results. In (Hashimoto *et al*., 2004), an algorithm for growing probabilistic Boolean network models of gene networks from seed genes is proposed. Our work can be seen as an extension of the work by Hashimoto *et al*. to BN models of gene networks. However, there are some other significant differences between both works. Unlike them, we have proved the correctness of our algorithm. Their algorithm requires binary data, while ours can learn from both discrete and continuous data. They report results for a database with only 597 genes, while we have showed that our algorithm can deal with databases with thousands of genes. Other work that is related to ours, though in a less degree, is (Tanay and Shamir, 2001) where an algorithm that suggests expansions to a given gene pathway is presented.

Most of the previous works on learning BN models of gene networks from gene expression data, e.g. (Badea, 2003; Bernard and Hartemink, 2005; Hartemink *et al*., 2002; Ott *et al*., 2004; Peña, 2004), do not address the poor scalability of the existing algorithms for learning BNs from data. They simply reduce the dimensionality of the gene expression data in advance so that the existing algorithms are applicable. To our knowledge, (Friedman *et al*., 2000; Pe'er *et al*., 2001) are the only exceptions to this trend. These works build upon the algorithm in (Friedman *et al*., 1999) which, in order to scale to high-dimensional data, restricts the search for the parents of each node to a small set of candidate parents that are heuristically selected in advance. Unfortunately, they do not report results for databases with more than 800 genes. Moreover, the performance of their algorithm heavily depends on the number of candidate parents allowed for each node, which is a user-defined parameter, and on the heuristic for selecting them. For instance, if the user underestimates the number of parents of a node, then the node will lack some of its parents in the final BN and, even worse, these errors may propagate to the rest of the BN. $AlgorithmGPC$ does not involve any heuristic or parameter that may harm performance. Instead, it copes with high-dimensional data by learning a local BN around the seed node rather than a global one.

We are currently extending $AlgorithmGPC$ with the following two functionalities. In order to release the user from having to specify the radius R, we are developing an automatic criterion to decide when to stop growing the BN. In order to assist the user in the interpretation of the BN learnt, we are implementing the methods in (Friedman *et al*., 2000; Pe'er *et al*., 2001; Peña, 2004) to assess the confidence in the BN learnt.

ACKNOWLEDGEMENT

This work is funded by the Swedish Foundation for Strategic Research (SSF) and Linköping Institute of Technology.

APPENDIX A: PROOFS OF THE THEOREMS

For any probability distribution p that can be represented by a BN with DAG G, the d-separation criterion enforces the local Markov property, i.e. $X \perp\!\!\!\perp_p ND_G(X) \setminus Pa_G(X) | Pa_G(X)$ (Neapolitan, 2003; Pearl, 1988). Therefore, $X \perp\!\!\!\perp_p Y | Pa_G(X)$ for all $Y \in ND_G(X) \setminus Pa_G(X)$ due to the decomposition property (Pearl, 1988).

PROOF OF THEOREM 2. First, we prove that the nodes in $PC_G(S)$ are included in the output PCD. If $X \in PC_G(S)$, then $X \not\perp\!\!\!\perp_p S|\mathbf{Z}$ for all $\mathbf{Z}$ st $X, S \notin \mathbf{Z}$ due to the faithfulness assumption. Consequently, X enters PCD at line 10 and does not leave it thereafter due to the assumption that the tests of conditional independence are correct.

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

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

Second, we prove that the nodes not in $PC_G(S)$ are not included in the output PC. Let $X \notin PC_G(S)$. 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 S in G other than one of its children and, thus, S does not satisfy the condition at line 3 due to Theorem 2. Consequently, X does not enter PC at line 4. □

PROOF OF THEOREM 4. We have to prove that the output DAG has the same adjacencies and the same immoralities as $G(S)^R$ due to Theorem 1. It is immediate that DAG has the same adjacencies as $G(S)^R$ at line 8 due to Theorem 3. Likewise, it is immediate that DAG has the same immoralities as $G(S)^R$ at line 9 due to the faithfulness assumption and the assumption that the tests of conditional independence are correct. □

REFERENCES

Badea,L. (2003) Inferring Large Gene Networks from Microarray Data: A Constraint-Based Approach. In *Proceedings of the Workshop on Learning Graphical Models for Computational Genomics at the Eighteenth International Joint Conference on Artificial Intelligence*.

Badea,L. (2004) Determining the Direction of Causal Influence in Large Probabilistic Networks: A Constraint-Based Approach. In *Proceedings of the Sixteenth European Conference on Artificial Intelligence*, 263-267.

Bernard,A. and Hartemink,A.J. (2005) Informative Structure Priors: Joint Learning of Dynamic Regulatory Networks from Multiple Types of Data. In *Proceedings of the Pacific Symposium on Biocomputing*, 459-470.

Friedman,N., Linial,M., Nachman,I. and Pe'er,D. (2000) Using Bayesian Networks to Analyze Expression Data. *Journal of Computational Biology*, **7**, 601-620.

Friedman,N., Nachman,I. and Pe'er,D. (1999) Learning Bayesian Network Structure from Massive Datasets: The "Sparse Candidate" Algorithm. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, 206-215.

Geiger,D. and Heckerman,D. (1994) Learning Gaussian Networks. In *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, 235-243.

Hartemink,A.J., Gifford,D.K., Jaakkola,T.S. and Young,R.A. (2002) Combining Location and Expression Data for Principled Discovery of Genetic Regulatory Network Models. In *Proceedings of the Pacific Symposium on Biocomputing*, 437-449.

Hashimoto,R.F., Kim,S., Shmulevich,I., Zhang,W., Bittner,M.L. and Dougherty,E.R. (2004) Growing Genetic Regulatory Networks from Seed Genes. *Bioinformatics*, **20**, 1241-1247.

Herskovits,E.H. (1991) Computer-Based Probabilistic-Network Construction. PhD Thesis, Stanford University.

Hughes,T.R., Marton,M.J., Jones,A.R., Roberts,C.J., Stoughton,R., Armour,C.D., Bennett,H.A., Coffey,E., Dai,H., He,Y.D. *et al*. (2000) Functional Discovery via a Compendium of Expression Profiles. *Cell*, **102**, 109-126.

Jensen,C.S. (1997) Blocking Gibbs Sampling for Inference in Large and Complex Bayesian Networks with Applications in Genetics. PhD Thesis, Aalborg University.

Jensen,L.T. and Culotta,V.C. (2002) Regulation of *Saccharomyces cerevisiae* FET4 by Oxygen and Iron. *Journal of Molecular Biology*, **318**, 251-260.

Lesuisse,E., Blaiseau,P.L., Dancis,A. and Camadro,J.M. (2001) Siderophore Uptake and Use by the Yeast *Saccharomyces cerevisiae*. *Microbiology*, **147**, 289-298.

Margolin,A., Banerjee,N., Nemenman,I. and Califano,A. (2004) Reverse Engineering of Yeast Transcriptional Network Using the ARACNE Algorithm. Submitted.

Neapolitan,R.E. (2003) Learning Bayesian Networks. Prentice Hall.

Ott,S., Imoto,S. and Miyano,S. (2004) Finding Optimal Models for Small Gene Networks. In *Proceedings of the Pacific Symposium on Biocomputing*, 557-567.

Pearl,J. (1988) Probabilistic Reasoning in Intelligent Systems. Morgan Kaufmann.

Pe’er,D., Regev,A., Elidan,G. and Friedman,N. (2001) Inferring Subnetworks from Perturbed Expression Profiles. *Bioinformatics*, **17**, S215-S224.

Peña,J.M. (2004) Learning and Validating Bayesian Network Models of Genetic Regulatory Networks. In *Proceedings of the Second European Workshop on Probabilistic Graphical Models*, 161-168.

Peña,J.M., Björkegren,J. and Tegnér,J. (2005) Scalable, Efficient and Correct Learning of Markov Boundaries under the Faithfulness Assumption. In *Proceedings of the Eighth European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty*, to appear.

Philpott,C.C., Protchenko,O., Kim,Y.W., Boretsky,Y. and Shakoury-Elizeh,M. (2002) The Response to Iron Deprivation in *Saccharomyces cerevisiae*: Expression of Siderophore-Based Systems of Iron Uptake. *Biochemical Society Transactions*, **30**, 698-702.

Protchenko,O., Ferea,T., Rashford,J., Tiedeman,J., Brown,P.O., Botstein,D. and Philpott,C.C. (2001) Three Cell Wall Mannoproteins Facilitate the Uptake of Iron in *Saccharomyces cerevisiae*. *The Journal of Biological Chemistry*, **276**, 49244-49250.

Robinson,R.W. (1973) Counting Labeled Acyclic Digraphs. In *New Directions in Graph Theory*, 239-273.

Spellman,P.T., Sherlock,G., Zhang,M.Q., Iyer,V.R., Anders,K., Eisen,M.B., Brown,P.O., Botstein,D. and Futcher,B. (1998) Comprehensive Identification of Cell Cycle-Regulated Genes of the Yeast *Saccharomyces cerevisiae* by Microarray Hybridization. *Molecular Biology of the Cell*, **9**, 3273-3297.

Spirtes,P., Glymour,C. and Scheines,R. (1993) Causation, Prediction, and Search. Springer-Verlag.

Tanay,A. and Shamir,R. (2001) Computational Expansion of Genetic Networks. *Bioinformatics*, **17**, S270-S278.

Tsamardinos,I., Aliferis,C.F. and Statnikov,A. (2003) Time and Sample Efficient Discovery of Markov Blankets and Direct Causal Relations. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 673-678.