LA-UR-16-23891 (Accepted Manuscript)

Order priors for Bayesian network discovery with an application to malware phylogeny

Oyen, Diane Adele
Anderson, Blake
Sentz, Kari
Anderson-Cook, Christine Michaela

To be published in: Statistical Analysis and Data Mining: The ASA Data Science Journal

DOI to publisher's version: 10.1002/sam.11364

Permalink to record: http://permalink.lanl.gov/object/view?what=info:lanl-repo/lareport/LA-UR-16-23891

Order Priors for Bayesian Network Discovery with an Application to Malware Phylogeny

Diane Oyen
Los Alamos National Laboratory
doyen@lanl.gov

Blake Anderson
Cisco Systems Inc.

Kari Sentz
Los Alamos National Laboratory

Christine Anderson-Cook
Los Alamos National Laboratory

August 26, 2017

Abstract

Bayesian networks have been used extensively to model and discover dependency relationships among sets of random variables. We learn Bayesian network structure with a combination of human knowledge about the *partial ordering* of variables and statistical inference of conditional dependencies from observed data. Our approach leverages complementary information from human knowledge and inference from observed data; to produce networks that reflect human beliefs about the system as well as fitting observed data. Applying prior beliefs about partial orderings of variables is a distinctly different approach from existing methods that incorporate prior beliefs about direct dependencies (or edges) in a Bayesian network. We provide an efficient implementation of the partial-order prior in a Bayesian structure discovery learning algorithm, as well as an edge prior, showing that both priors meet the local modularity requirement necessary for an efficient Bayesian discovery algorithm. In benchmark studies, the partial-order prior improves the accuracy of Bayesian network structure learning as well as the edge prior, even though order priors are more general. Our primary motivation is in characterizing the evolution of families of malware, to aid cybersecurity analysts. For the problem of malware phylogeny discovery, we find that our algorithm, compared to existing malware phylogeny algorithms, more accurately discovers true dependencies that are missed by other algorithms.

Keywords: Bayesian networks, probabilistic graphical models, malware, cybersecurity

1 Introduction

Bayesian networks have emerged as a natural model for combining human knowledge and statistical data [1, 2]. The discovery of structural features in Bayesian networks [3, 4], as learned from data, is of great interest in domains such as cyber-security, bioinformatics and neuroscience. The learned structures give insight into the evolutionary development of malware, the networking of gene interaction pathways or functional brain networks; as just a few examples [5, 6, 7]. The structure (the edges) of a Bayesian network represents conditional dependencies, and are often of particular interest to domain scientists.

A key goal of the malware phylogeny problem is to understand local relationships between different versions of the malware. If insights into the likely ordering or sequence of development of these codes can be quantified, then it is helpful for the software reverse engineer as they consider the adversary's strategy in trying different approaches. By understanding likely sequences of attempts to compromise a network or account, there is an improved opportunity to anticipate future directions and threats. In these applications there are often potentially informative supplementary information, such as first observed occurrences of malware, that are helpful but not foolproof. Leveraging this knowledge and other subject matter expertise in characterizing the evolution of the code can provide insights that should be included in the phylogeny summary.

Yet, several Bayesian network structures may represent the data nearly equally well; therefore, inferring a single structure from data alone often produces a learned model that is confusing to a human domain expert when another structure that is more intuitive may explain the data equally well. One way to address this problem is to use a Bayesian discovery algorithm to estimate the expectation of the presence of each potential edge in the Bayesian network thereby giving a summary of all likely structures [3, 4]. The term *Bayesian discovery* means that local edges are estimated by taking a Bayesian average over all full structure models, to discover dependency relationship patterns when the true underlying structure is not known. We take a different approach building on the advantages of Bayesian discovery and further focusing the solution space on structures that reflect prior beliefs of the user. In particular, the ordering of variables, which determines edge direction, can be difficult or impossible to discern from data, yet this information may be

readily available from a human expert.

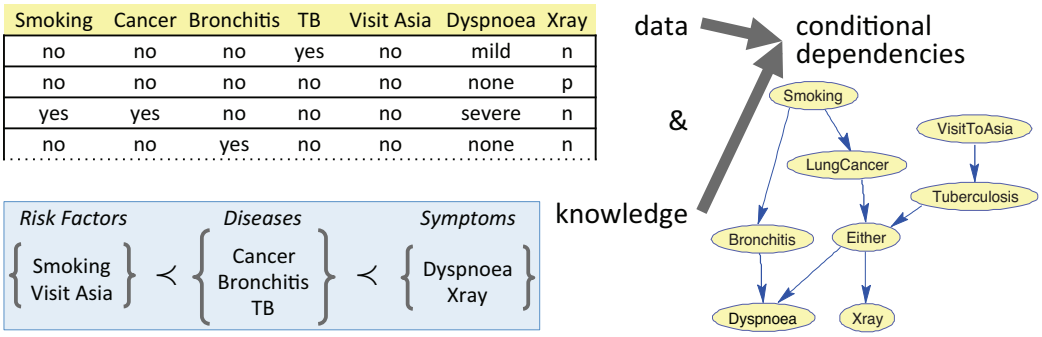


Figure 1: Learning a Bayesian network from observed data and expert knowledge about the partial ordering of variables.

We introduce Bayesian network discovery with order priors, an approach to combine human knowledge about the ordering of variables into a statistical learning algorithm for Bayesian structure discovery. Previous approaches to include human knowledge in Bayesian network structure learning use prior beliefs on edges, rather than orders [8, 7]. Yet, the information in an edge prior is often overly restrictive given the knowledge that the person can provide, and the more general order prior would be more appropriate. Take for example the problem in Fig. 1. In this case, we would like to include the knowledge that *symptoms* depend on *disease* and *disease* depends on *risk factors*. If edge priors were the only option for imposing beliefs, then the prior would take a form such as "smoking is a likely predictor of cancer". However, that is not a prior that we want to use because we should not impose a prior bias toward the dependency between smoking and cancer. Instead, the prior we want is that if the *evidence indicates a statistical dependency* between smoking and cancer, then we believe smoking is a risk factor (not a symptom), and therefore the *preferred direction* of the dependency is that cancer is conditionally dependent on smoking.

We incorporate our informative prior into a Bayesian network structure discovery algorithm, which calculates the expectation of structural features. Our primary contribution is a partial-order prior that is specified as a collection of orderings on pairs of variables. We show that our prior has the property of local modularity and therefore can be incorporated into a Bayesian structure discovery algorithm efficiently. For comparison, we also demonstrate a prior on edges (rather than orders) with local modularity. Benchmark re-

sults indicate that the partial-order prior helps to identify true edges in the network with less data compared to uninformative priors, and reduces false-positive edge identification compared to edge priors even though order priors are more general than edge priors.

We apply our approach to the problem of learning the evolution of malware programs. Cybersecurity analysts and software reverse-engineers need to understand the function of malware in order to evaluate new threats. Malware is typically developed the same way as any software: it is evolved by modifying and combining existing malware [9, 10]. We take a collection of related malware and benign programs and construct a Bayesian network to model the phylogeny of the programs. Experts provide knowledge about the believed order in which these programs were developed, providing a prior on the partial-order of variables; while features extracted from the programs themselves are the observed data from which we discover conditional dependencies. Importantly, we find that including beliefs about variable order as a prior, rather than post-hoc, helps to discover correct edges that are otherwise missed entirely. In other words, the order prior does more than just fix edge directions, it also improves the accuracy of edge detection for the whole Bayesian network. The malware phylogeny application provides a helpful illustration of the new methods and how they can be applied to a complex process to provide clearer insights and understanding of likely evolutionary paths. Unlike other software development phylogeny problems, the adversary developing different instances of the code may seek to obscure the development evolution and there may be less reliable information about the timeline of development.

2 Related Work

Methods for combining human expertise and statistical data in the context of learning Bayesian networks have been proposed that start with a human-engineered Bayesian network and then use observed data to refine the model [1, 11]. However, these methods first require the labor-intensive process of knowledge engineering to produce a prior model. The candidate parents algorithm [12] which imposes a constraint on potential parents for each variable in a Bayesian network could be used to incorporate knowledge about partial orders, but it is a hard constraint rather than a Bayesian prior. Verma and Pearl [13] show theoretically that edge direction can be inferred from data alone, but only for certain

structures. Empirically we see that inferring edge direction is usually difficult, leading to methods of inferring partially directed acyclic graphs [14], though these learned models may be difficult to interpret by humans. Others have considered Bayesian network discovery algorithms with priors over edges obtained from domain knowledge [8, 7]. These approaches set a specific prior on the statistical dependencies represented by the edge. Eaton and Murphy [15] consider a hybrid approach of structure-space MCMC using order-space solutions in the proposal distribution. While these methods explore structure priors, none make use of order priors. To our knowledge, using a prior on the partial ordering of variables is novel.

For the discovery of malware phylogeny, most existing techniques rely on bifurcating tree-based approaches [10, 16]. These methods do not identify ancestral relationships in the data, but do give similarity relationships. However, it is more informative to know the set of parents from which a given sample's functionality is derived [9]. Most similar to this paper is the method of [17] using the graphical lasso, an undirected graphical model, with post-hoc assignment of edge direction by a domain expert. We instead use a Bayesian network learning approach because it naturally models the parent-child relationships that we are interested in as a directed acyclic graph. Empirical results indicate that our method provides more accurate learned structures using human knowledge as a prior rather than post-hoc modification of the learned structure.

3 Preliminaries

This section provides necessary background on Bayesian networks and key model assumptions in Bayesian network structure discovery algorithms.

3.1 Bayesian Networks

A Bayesian network is a directed acyclic graph (DAG) with nodes to represent random variables and edges to represent direct influences or dependencies between variables. Stemming from the hierarchical structure of a directed graph, nodes are often referred to in familial terms of *parents*, *children*, and *descendants*. Nodes of a Bayesian network satisfy a

local Markov property where each node is conditionally independent of its non-descendants given its parents.

The edges or influences represent relationships between the nodes. The strengths of these influences are expressed by conditional probabilities (the probability of the states of a child node given the states of the parent(s) node). In particular, to specify the distribution for a network, the prior probabilities for all of the root nodes (nodes without a predecessor) must be specified as well as the conditional probabilities of all non-root nodes given all possible combinations of their direct predecessors. The network encodes a set of conditional independencies and conditional dependencies amongst the variables. This distinction enables the factorization of the calculation of the joint distribution into products of marginal distributions as follows [18, 1]. A Bayesian network $B = \{G, \theta\}$ describes the joint probability distribution over p random variables $\mathbf{X} = [X_1, X_2, \ldots, X_p]$, where G is a directed acyclic graph (DAG). An edge (X_j, X_i) in G means that the child X_i is conditionally independent of all non-descendants given its parents $\pi(X_i)$ where $X_j \in \pi(X_i)$. Given the DAG structure, the joint probability distribution $P(\mathbf{X})$ factors into the product of the conditional probability distributions for each child given its parents as parameterized by θ:

$$P(\mathbf{X}) = \prod_{i=1}^{p} P(X_i|\pi_i, \theta_i). \tag{1}$$

We often use the shorthand notation π_i to mean $\pi(X_i)$, the set of variables that are parents of variable i.

Bayesian networks provide a way of representing the relationships between variables and allow for the factoring of conditional independencies between variables. This factorization provides a computationally feasible mechanism to calculate marginal probabilities based on the dependency relationship without having to calculate the full joint probability distribution. This is at the crux of the message passing algorithms where probabilistic inference can be achieved without the necessity of enumerating all possible variable states [2].

The joint probability distribution, $P(\mathbf{X})$, in Equation 1 holds under different configurations of the network. So the different versions of the Bayesian network can represent the same joint distribution. The difference between the two networks involves the specifics of the conditional probabilities and the semantics of the network. It is possible to transform

one network to another via the joint distribution and Bayes rule. A transformation may require additional edges [19, 20].

The *structure*, or set of edges, of the network, G, is of particular interest in many domains as it is easy to interpret and gives valuable information about the interaction of variables. The structure of a Bayesian network can be estimated from observed data D, a matrix of $p \times n$ dimension with n samples, by selecting the network that maximizes the posterior probability $P(G|D)$,

$$P(G|D) = \frac{P(G)P(D|G)}{P(D)} \propto P(G)P(D|G), \tag{2}$$

where $P(G)$ is a prior on the structure, $P(D|G) = \prod_{s=1}^{n} P(\mathbf{X} = D_s)$ is the data likelihood, and $P(D)$ is a normalization constant that need not be calculated; and estimating the parameters, θ_i of the conditional probabilities to maximize:

$$P(\theta_i|D) \propto P(\theta_i) \prod_{s=1}^{n} P(X_i = D_{i,s}|X_{\pi_i} = D_{\pi_i,s}, \theta_i), \tag{3}$$

where the form of $P(X_i|\pi_i, \theta_i)$ is typically a multinomial distribution for discrete data.

3.2 Bayesian Discovery of Bayesian Networks

Finding the optimal network structure is NP-Hard, therefore instead of finding a single high-scoring network, we use the method of Bayesian structure discovery, as described here. Given a large number of candidate networks and a limited amount of data, the posterior probability of any network may be quite small. However, summary statistics regarding structural features of networks may have high posterior even with limited data [3]. Structural features (such as an edge) can be described by an indicator function f such that for $f(G) = 1$ the feature exists in graph G, otherwise $f(G) = 0$. The posterior probability of the feature given observed data, D, is equivalent to the expectation of its indicator:

$$E_{\text{DAG}}(f|D) = \frac{1}{P(D)} \sum_{G} P(G)P(D|G)f(G). \tag{4}$$

However, this sum can be intractable, as the number of DAGs is super-exponential in the number of variables.

An important insight to making the expectation tractable is that we could fix the *order* of the variables. An order, $\prec$, is a permutation on the indices of the variables $X_{\prec(1)}, X_{\prec(2)}, \ldots, X_{\prec(p)}$ such that parents must precede children in the order, i.e. X_j cannot be a parent of X_i if $\prec(j) \geq \prec(i)$. Given an order, learning optimal parents for each child factors into local calculations, and summing over DAGs consistent with the order is tractable [11, 21]. Taking this one step further, we can condition on a node order, and then obtain the unconditional posterior by summing over orders [3]:

$$E_{\text{ord}}(f|D) = \frac{1}{P(D)} \sum_{\prec} P(\prec) \sum_{G \subseteq \prec} P(G|\prec) P(D|G) f(G). \tag{5}$$

Importantly, these two formulations for $E(f|D)$ are not the same, as most DAGs, G, will be consistent with multiple orders, $\prec$. Typically, the formulation of Eq. (5) produces an acceptable bias in favor of simpler structures.

Koivisto and Sood [4] give an efficient dynamic programming method for calculating Eq. 5. The approach is rather involved, so we summarize only the key points here. All efficient Bayesian network algorithms require that calculations are modular meaning that they factor into local calculations. We describe the modularity assumptions which are critical for determining which priors we can use:

1. **Parameter modularity**: Modularity of the Bayesian network parameters must hold,
   $P(\theta|G) = \prod_{i=1}^{p} P(\theta_i|\pi_i)$ and
   $P(X = x|G) = \prod_{i=1}^{p} P(x_i|x_{\pi_i}, \theta_i)$.

2. **Structure prior modularity**: The network model prior, $P(G,\prec)$, is the product of a prior on the order, $P(\prec)$, and a prior on the structure, $P(G|\prec)$. The model prior must be modular so that $P(G,\prec) = c \prod_{i=1}^{p} q_i(U_i) q_i'(\pi_i)$, where U_i is the set of variables preceding X_i in the order $\prec$ (potential parents of X_i) and c is a normalization constant. $q_i(U_i)$ is a local prior on the set of variables preceding X_i in the order; and $q_i'(\pi_i)$ is a local prior on the parent set of X_i. These model priors are typically uninformative (uniform) or favor small parent-set sizes.

3. **Feature modularity**: The features must be modular, $f(G) = \prod_{i=1}^{p} f_i(\pi(X_i))$ where $\pi(X_i)$ is the parent set of variable i. f_i is a local feature, defined such that if the parent

set of X_i is inconsistent with feature f, then $f_i(\pi(X_i)) = 0$; otherwise, $f_i(\pi(X_i)) = 1$. Therefore, feature f is modular if the product of local features f_i gives $f(G) = 1$ if the graph has feature f and $f(G) = 0$ otherwise.

The most common feature to look for is a directed edge $u \to v$ s.t. $f = 1$ if $X_u \in \pi_v$, which is clearly modular. If these modularity assumptions hold, then the likelihood over order space factors into local calculations as shown in Eq. 6 [4]:

$$E(f|D) = \sum_{\prec} \prod_{i=1}^{p} q_i(U_i) \left[\sum_{\pi_i \subseteq U_i} q_i'(\pi_i) P(x_i|\pi_i) f_i(\pi_i) \right], \tag{6}$$

where $\pi_i = \pi(X_i)$ is the parent set of variable i.

The unconditional posterior for the features is obtained by summing over orders, using the following steps:

1. Calculate family scores: $\beta_i(\pi_i) = q_i'(\pi_i) P(x_i|\pi_i) f_i(\pi_i)$ for each node i and potential set of parents π_i.

2. Calculate local contribution of each subset $U \subseteq \mathcal{X} - \{i\}$ of potential parents of i: $\alpha_i(U) = \sum_{\pi_i \subseteq U} \beta_i(\pi_i)$, where $\mathcal{X}$ is the set of scalar variables that make up the data vector $\mathbf{X} = [X_1, X_2, \ldots, X_p]$.

3. Sum over the subset lattice of the various U_i to obtain the sum over orders $\prec$.

The maximum number of parents allowed for any child is typically fixed to a small natural number, r. The computational complexity for calculating the exact expectations of all features is $O(p2^p + K_n p^{r+1})$, where K_n is the cost of calculating each family score from n observed samples. For large networks, roughly $p > 30$, the exponential term is intractable. In these cases, MCMC simulations give an approximation to the expectations, so that $E(f|D) \approx \frac{1}{T} \sum_{t=1}^{T} P(\prec_t) P(f|D, \prec_t)$ for $\prec_t$ sampled from order space [3] or partial orders [22].

4 Bayesian Discovery with Informative Priors

We propose two methods for incorporating prior knowledge into order-space Bayesian discovery algorithms. The first method is to place a prior on individual child-parent rela-

tionships (or edges), as has been explored in structure-space algorithms previously. The second, preferred, method is to place a prior on pairwise order relationships (partial orders). Partial order priors have not been previously studied, yet there are three advantages. Partial order priors are easy to implement in the efficient order-space discovery algorithm; and such a prior allows a person to give the kind of prior knowledge that can be difficult to discover from the data; and cannot be captured directly by structure priors.

4.1 Structure Priors

Following Werhli and Husmeier [7] we consider a prior on structures based on expert knowledge about the parents of a particular variable:

$$q_i'(\pi_i) = \frac{1}{Z_i'} e^{-\lambda' \left(\sum_{j=1}^{p} |I(j \in \pi_i) - B(f_{ij})| \right)} , \tag{7}$$

where $\lambda' \geq 0$ is a hyper-parameter, often called the temperature, that determines the strength of the prior. As $\lambda' \to 0$, the prior distribution in the equation becomes flat and uninformative on the structure of the network. As $\lambda' \to \infty$ the network structure increasingly favors the expert knowledge-based prior information. Z_i' is the normalization term as described below. $B(f_{ij}) \in [0, 1]$ represents prior knowledge about interactions between nodes through a directed edge $i \to j$, where:

$$\begin{aligned} &B(f_{ij}) < 0.5, \quad \text{if } i \text{ is believed to not be a parent of } j; \\ &B(f_{ij}) = 0.5, \quad \text{if no prior information available; and,} \\ &B(f_{ij}) > 0.5, \quad \text{if } i \text{ is believed to be a parent of } j. \end{aligned} \tag{8}$$

The normalization term Z_i', also known as the partition function, is calculated by summing over all possible parent sets for a given variable i,

$$Z_i' = \sum_{\pi \subseteq \mathcal{X} - \{i\}} e^{-\lambda' \left(\sum_{j=1}^{p} |I(j \in \pi_i) - B(f_{ij})| \right)}. \tag{9}$$

This structure prior is clearly modular and therefore can be efficiently calculated as part of a Bayesian discovery algorithm. A similar prior was previously used within a structure-space Bayesian discovery algorithm [7]; and we are showing here how to incorporate this prior into an order-space Bayesian discovery algorithm. The computational complexity of

this sum is on the order of $\binom{p}{r} = O(p^r)$, where r is a user-specified maximum number of parents allowed per child, typically a small integer like 4.

Incorporating this structure prior into the order-space dynamic programming algorithm, happens during the calculation of the family scores:

$$\begin{aligned}\beta_i(\pi_i) &= q_i'(\pi_i)p(x_i|x_{\pi_i})f_i(\pi_i) \\ &= \frac{1}{Z_i'}p(x_i|x_{\pi_i})f_i(\pi_i)e^{-\lambda'\left(\sum_{j=1}^{p}|I(j\in\pi_i)-B(f_{ij})|\right)},\end{aligned} \tag{10}$$

as shown in Algorithm 1. There are p^{r+1} of these β family scores to calculate. Each of these scores takes some constant time to calculate that depends on the number of observed samples of data, which we call K_n; plus the calculation of the normalization term, $O(p^r)$. Note that calculating the family scores is the first step of Bayesian network learning, with or without prior feature beliefs. Therefore, the overall computational complexity of these β family scores is $O((r + K_n)p^{r+2})$, as opposed to $O(K_n p^{r+1})$ without the structure prior, representing a minor increase in computational complexity.

4.2 Partial Order Priors

In Bayesian network learning, if the variables are constrained to a fixed ordering, $\prec$, then the parents of a node i must precede i in the order, $\prec(\pi_i) < \prec(i)$. Order-space Bayesian discovery algorithms take advantage of the efficient computation over fixed orders by calculating summary statistics over order-space. Instead of placing a prior over parents themselves or constraining structures to adhere to a full ordering, we prefer to place a prior on pairwise order relationships. This translates into a prior over partial orders, that is, identifying which variables are likely to precede a particular child in the order. Partial-order priors only put a preference on the direction of the relationship, not on whether the relationship exists. In contrast to a structure prior that assigns a prior to a specific parent child relationship, the partial order prior takes advantage of weaker information and allows for priors on more distant relationships between variables, but it also allows the prior to be "ignored" if no statistical dependency exists between two variables.

Pairwise order relationships are used as a prior because it is manageable for a human to specify and the prior is modular. Specifically, the pairwise order prior knowledge is captured through, $B(u_{ij})$, are specified as follows:

Algorithm 1 Bayesian Network Family Scores with Optional Edge Priors

procedure BNFamilyEdgePrior($D, B(f), \lambda'$) ▷ Calculate family scores $\beta_i(\pi_i)$
  for all $i \in 1, 2, \ldots, p$ **do** ▷ Loop over variables
    if $B(f)$ given **then** ▷ Calculate family priors from edge prior beliefs
      for all $\pi_i \subseteq \mathcal{X} - \{i\}$ s.t. $|\pi_i| \leq r$ **do** ▷ Loop over potential parent sets
        $q \leftarrow e^{-\lambda'(\sum_{j=1}^{p} |I(j \in \pi_i) - B(f_{ij})|)}$ ▷ Unnormalized edge priors
        $Z'_i \leftarrow Z'_i + q$ ▷ Running sum of partition function
      end for
      for all $\pi_i \subseteq \mathcal{X} - \{i\}$ s.t. $|\pi_i| \leq r$ **do**
        $q'_i(\pi_i) \leftarrow q / Z'_i$ ▷ Normalized edge priors
        $\beta_i(\pi_i) \leftarrow q'_i(\pi_i) P(x_i|\pi_i) f_i(\pi_i)$ ▷ Family scores with edge priors
      end for
    else ▷ No edge beliefs given
      for all $\pi_i \subseteq \mathcal{X} - \{i\}$ s.t. $|\pi_i| \leq r$ **do**
        $q'_i(\pi_i) \leftarrow (\sum_{r'=0}^{r} \binom{p-1}{r'})^{-1}$ ▷ Uninformative edge priors
        $\beta_i(\pi_i) \leftarrow q'_i(\pi_i) P(x_i|\pi_i) f_i(\pi_i)$ ▷ Family scores
      end for
    end if
  end for
  return β ▷ The family scores are β which are the input to Algorithm 2
end procedure

$$
\begin{aligned}
B(u_{ij}) &< 0.5, \quad \text{if } i \text{ is believed not to precede } j; \\
B(u_{ij}) &= 0.5, \quad \text{if no prior information available; and,} \\
B(u_{ij}) &> 0.5, \quad \text{if } i \text{ is believed to precede } j.
\end{aligned} \tag{11}
$$

Using the same style of energy potential function as used previously, we assign this partial-order prior:

$$
q_i(U_i) = \frac{1}{Z_i} e^{-\lambda\left(\sum_{j=1}^{p} |I(j \in U_i) - B(u_{ij})|\right)}, \tag{12}
$$

where $\lambda \geq 0$ is a hyper-parameter which determines the strength of the prior; U_i is the set of variables that precede i (including parents and more distant relationships); $|U_i|$ is the number of elements in set U_i; and the normalization term is

$$
Z_i = \sum_{U \subseteq \mathcal{X} - \{i\}} e^{-\lambda\left(C_i + |U| - 2\sum_{j \in U} B(u_{ij})\right)}. \tag{13}
$$

This prior is modular and therefore can be readily incorporated into the dynamic programming method, as detailed in Algorithm 2. The Bayesian discovery algorithm (Algorithm 2) takes as input the family scores previously computed using Algorithm 1.

The partition function, Z_i, must calculate a sum over $\sum_{j=0}^{p-1} \binom{p-1}{j} = O(2^{p-1})$ possible sets of ancestors. The overall computational complexity including this partial order prior is $O(p^2 2^p + K_n p^{r+2} 2^{p-1})$. Compared with the complexity without the prior, $O(p2^p + K_n p^{r+1})$, the feature expectations are significantly more expensive to calculate, but the overall complexity remains exponential in p. Note that this is still significantly less computationally expensive than the structure-space calculation.

5 Experiments on Benchmark Data

Experiments on data generated from known networks show that partial-order priors provide valuable information to the network discovery algorithm that is different from using edge prior information. Using data generated from known benchmark networks, we compare four approaches; 1) informative priors on structure using order-space expectation, as described in the section "Structure Priors"; 2) informative priors on structure using structure-space

Algorithm 2 Bayesian Network Discovery with Optional Order Priors

procedure BNDiscoveryOrderPrior($\beta, B(U), \lambda$) ▷ Calculate $E(f|D)$
  for all $i \in 1, 2, \ldots, p$ **do** ▷ Loop over variables
    for all $U_i \subseteq \mathcal{X} - \{i\}$ **do** ▷ Calculate $\alpha_i(U_i)$ functions
      $\alpha_i(U_i) \leftarrow \sum_{\pi_i \subseteq U_i} \beta_i(\pi_i)$ ▷ Sum contributions of parents to partial order U_i
    end for
    if $B(U)$ given **then** ▷ Calculate order priors from order prior beliefs
      for all $U_i \subseteq \mathcal{X} - \{i\}$ **do** ▷ Loop over potential partial orders
        $q \leftarrow e^{-\lambda(\sum_{j=1}^{p} |I(j \in U_i) - B(u_{ij})|)}$ ▷ Unnormalized partial order priors
        $Z_i \leftarrow Z_i + q$ ▷ Running sum of partition function
      end for
      for all $U_i \subseteq \mathcal{X} - \{i\}$ **do**
        $q_i(U_i) \leftarrow q / Z_i$ ▷ Normalized partial order priors
      end for
    else ▷ No order beliefs given
      for all $U_i \subseteq \mathcal{X} - \{i\}$ **do**
        $q_i(U_i) \leftarrow 2^{-(p-1)}$ ▷ Uninformative partial order priors
      end for
    end if
  end for
  for all $\prec$ = permutation of $(1, 2, \ldots, p)$ **do** ▷ Loop over full orders (or MCMC)
    $s(\prec) \leftarrow \prod_{i=1}^{p} q_i(U_i)\alpha_i(U_i)$ ▷ Combine order priors and family scores
    $E(f|D) \leftarrow E(f|D) + s(\prec)$ ▷ Running sum over full orders
  end for
  return $E(f|D)$ ▷ The expectation of structure features are $E(f|D)$
end procedure

expectation, a competing algorithm; 3) informative priors on partial orders, as described in the section "Partial Order Priors"; and 4) uninformative priors, a baseline method using the typical uninformative uniform prior. For all order-space expectations, we use the BeanDisco C++ implementation [22][1]. The code has been modified to implement the informative priors as described above. For comparison, the same prior information was given to the structure-space MCMC algorithm as implemented in the Matlab/C package BDAGL[2] for structure-space expectations. We impose a limit on the size of parent sets, as described for each data set. In all presented results on the Benchmark data, we use $\lambda = \lambda' = 1$, but other values could be chosen to enforce a stronger or weaker prior.

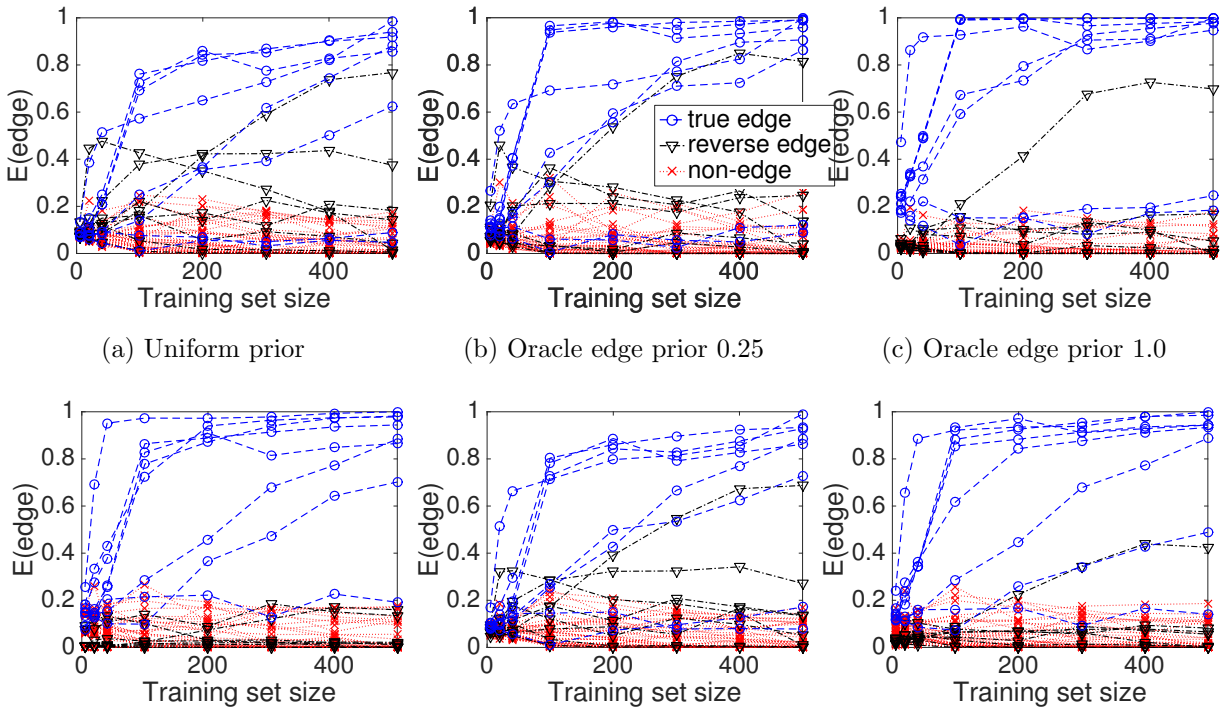
5.1 Benchmark Data

We use benchmark data generated from known networks so that we can evaluate how well the ground truth structure is recovered. To test the effectiveness of prior knowledge, we create prior knowledge matrices: $B(f)$ for structure priors and $B(U)$ for order priors. The *Asia* network has 8 discrete variables, from which we generate data samples. We provide one type of domain knowledge that is relevant to the *Asia* data. The variables in this dataset can be split into three sets (as shown in Fig. 1), representing *risk factors*, $\mathcal{R} =$ {Smoking, VisitToAsia}; *diseases*, $\mathcal{D} =$ {LungCancer, Tuberculosis, Bronchitis, Either}; and *symptoms*, $\mathcal{S} =$ {Dyspnoea, Xray}. We provide prior information that risk factors are likely ancestors of diseases by setting $B_{\mathcal{RD}} = 1$ for all variables in $\mathcal{R}$ and $\mathcal{D}$; diseases are likely ancestors of symptoms by setting $B_{\mathcal{DS}} = 1$ for all variables in $\mathcal{D}$ and $\mathcal{S}$; risk factors are unlikely to have any ancestors by setting $B_{i\mathcal{R}} = 0$ for all variables i; and symptoms are unlikely ancestors of anything by setting $B_{\mathcal{S}i} = 0$ for all variables i.

For the *Asia* network, we use a *domain* prior in which domain knowledge is used as described above. The other type of prior knowledge is called the *oracle* prior because it is based on the true structure (note that it is not realistic to expect to have such prior knowledge in real applications, but we use it for comparing algorithms). We start by setting every element of the prior matrix to 0.5, representing "no prior information" about

[1] http://www.cs.helsinki.fi/u/tzniinim/BEANDisco

[2] http://www.cs.ubc.ca/~murphyk/Software/BDAGL



(a) Uniform prior (b) Oracle edge prior 0.25 (c) Oracle edge prior 1.0

(d) Domain order prior (e) Oracle order prior 0.25 (f) Oracle order prior 1.0

Figure 2: Learning curves for *Asia* from order-space expectation using different priors.

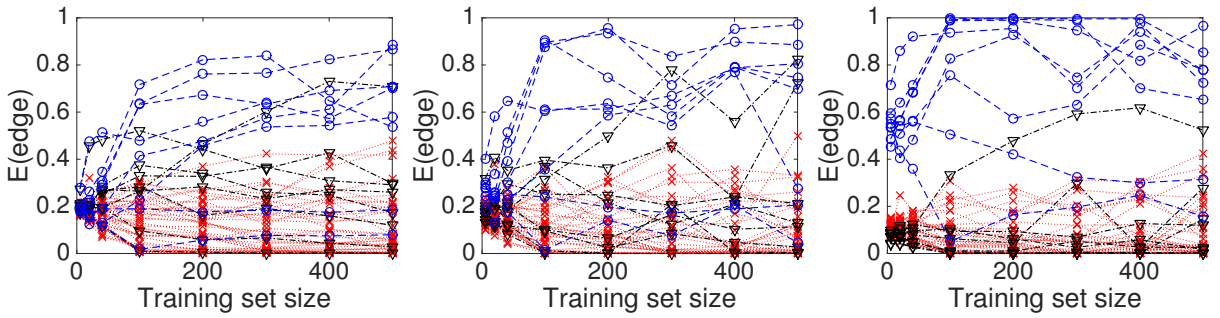
the feature or partial order. For each ordered pair of variables, we randomly select some fraction, $\{0\%, 10\%, 25\%, 50\%, 75\%, 100\%\}$, of features, f, and set $B(f_{ij}) = 1$ if the edge e_{ij} exists in the true network; and $B(f_{ij}) = 0$ if the edge e_{ij} does not exist in the true network; for that same pair, if the edge exists, we set $B(u_{ij}) = 1$ and $B(u_{ji}) = 0$. When setting the partial-order priors, if the ordered pair represents a non-edge in the true network (neither e_{ij} nor e_{ji} exists), then no prior knowledge is included for that pair. Thus the order priors provide less information than the structure priors. For each ground truth network and each fraction of prior knowledge, we repeat the random selection of prior knowledge and the sampling of data 5 times to produce 5 different trials of observed data with prior knowledge.

5.2 Benchmark Results

We first look at results from the small benchmark network *Asia* [23]. Fig. 2 shows the $E(e)$ learning curves for each ordered pair (i, j) on 8 variables. In the first of the plots, we

see the learning curve using order-space exact calculation of the edge expectation without any prior knowledge. As the amount of data increases, the expectations of true edges (blue circles) tend to go toward 1 while the expectations of non-edges (red exes) tend to go toward 0. We also indicate "reverse" edges (black triangles) which are non-edges (and therefore the expectation should go toward 0) that have the same end points as a true edge, but in the wrong order. The algorithm clearly improves as the amount of data increases. It is interesting to note that some of the probabilistic dependencies are so subtle that it would take tens of thousands of samples to identify them. In the *Asia* benchmark, $P(\text{Tuberculosis} = \text{true}|\text{VisitToAsia} = \text{true}) < 0.0001$. In Fig. 2 Tuberculosis $\rightarrow$ VisitToAsia is the edge that is a false-negative in all of the plots. We know from experimentation that it takes about 30,000 samples to identify that particular edge (not shown in Fig. 2). The other false-negative in many of the plots of Fig. 2, is the mirror-edge of the false-positive "reverse" edge which happens to be Smoking $\rightarrow$ LungCancer. Edges like this one that are higher in the parental structure are more likely to be identified in reverse than edges lower in the structure because it is easier to flip the direction on edges that do not have a lot of ancestry above. This is because if an edge is flipped, many of the edges in its ancestry may also need to be reversed according to Bayes' rule to represent the same joint distribution; yet flipping an edge does not affect the (original) descendants.

If we add prior knowledge about known edges and non-edges, we see that the learning curves improve in Fig. 2(b) and (c); it takes less data to correctly identify true edges and non-edges with *oracle* prior knowledge on 25% of the potential edges or all of the potential edges. Interestingly, when using edge priors, the data can overwhelm the prior and in this case identify one of the true edges in reverse of the correct direction. The addition of *domain* prior knowledge about the partial orders of *risk factors*, *diseases*, and *symptoms* improves the learning curve as shown in Fig. 2(d). This domain knowledge effectively fixes the problem of learning high expectations for reverse edges. We also show the *oracle* prior knowledge on 25% and 100% of ordered-pairs as a partial-order prior. Note that the *oracle* partial-order prior gives no prior information about pairs of variables that have no true edge between them, and so there is less information than given by the *domain* order prior. Comparing the *oracle* order prior to the *oracle* edge prior, we see that at small amounts



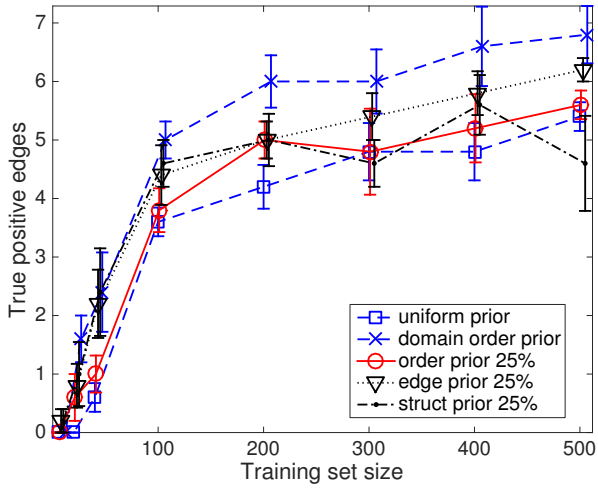
(a) Uniform prior (b) Oracle edge prior 0.25 (c) Oracle edge prior 1.0

Figure 3: Learning curves for *Asia* using structure-space MCMC.

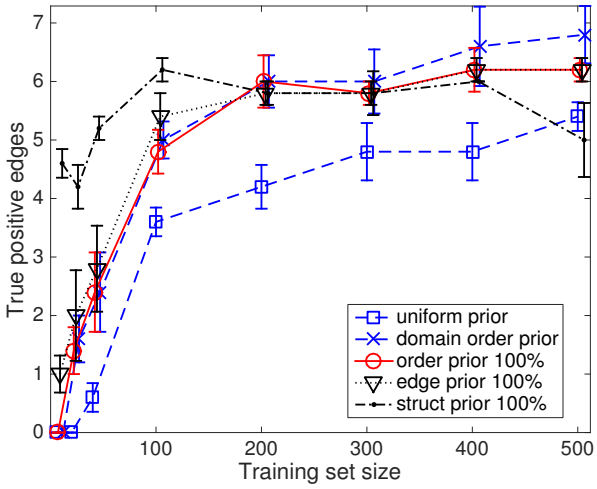
of data, the edge priors can help to boost the expectation of true edges. Yet, at larger sample sizes, the order prior is actually more effective than the edge prior at reducing the expectation of some reverse edges. Other values for the fraction of prior knowledge were investigated and show similar trends.

For comparison with the structure-space expectation, we use MCMC over structure space with a burn-in of 1000 steps, collected 1000 MCMC samples sub-sampled at 100-step intervals. At these settings, each MCMC chain took approximately 108 seconds compared with 3.2 seconds on average for the exact calculation over order space. This is an approximation to the true expectation over structure-space, as the exact calculation is not practical beyond $p = 5$. Fig. 3 shows the results of structure-space MCMC. The estimates are clearly more variable than those obtained from the exact computation over order-space. The expectations calculated over structure space do not separate the true edges from false edges as quickly as the order-space calculation does. Prior information can increase the estimated expectation of true edges, but it appears to require more prior information to have a positive effect on the estimate.

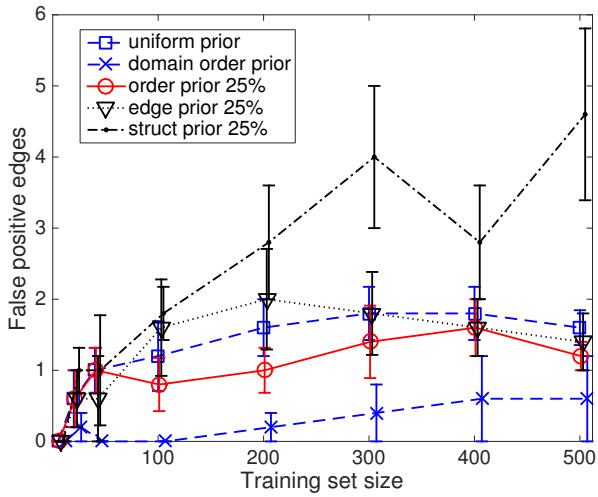
Our goal is to be able to distinguish true edges from non-edges, and so for direct comparison of the methods, we look at how well the various algorithms and priors can identify true edges. To quantify this, we calculate the true positive (TP) edges identified and the false positive (FP) non-edges and reverse-edges identified under the various algorithms



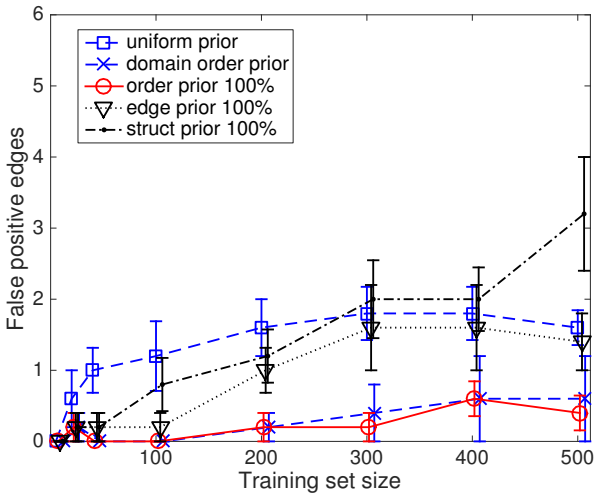
(a) TP curves for 0.25 priors (up is good)



(b) TP curves for 1.0 priors (up is good)



(c) FP curves for 0.25 priors (down is good)



(d) FP curves for 1.0 priors (down is good)

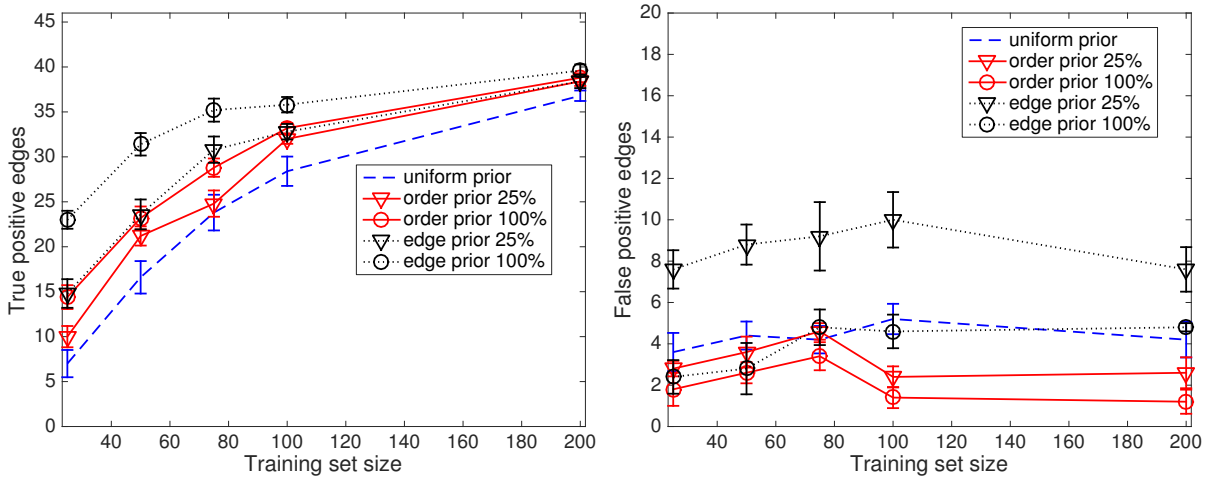
Figure 4: Comparison of edge identification learning curves for *Asia* with various priors. Top row shows true positives out of 7 true edges (up is good). Bottom row shows false positives (down is good). In these plots, the MCMC algorithm with *oracle* edge priors is labeled as the "struct prior". In both the left and right plots, we include the uniform prior as a baseline, and we include the *domain* order prior as our recommended approach.

and priors:

$$\begin{aligned} \text{TP} &= \sum_{e \in \{e_{\text{true}}\}} I(E(e) \geq 0.5) \\ \text{FP} &= \sum_{e \in \{e_{\text{non}} \cup e_{\text{rev}}\}} I(E(e) \geq 0.5), \end{aligned} \tag{14}$$

where I is an indicator function such that $I(\text{pred}) = 1$ if pred is true and 0 otherwise. The true positive learning curves are shown in the top row of Fig. 4. There are too many algorithms to display in a single plot, and so the results are split into two plots, with the left plot including the *oracle* priors at 25% and the right plot including the *oracle* priors at 100%. The true positive learning curves confirm our observation that prior knowledge enables the algorithms to identify significantly more true edges with less data than possible without the prior. Generally, the more prior knowledge, the more true edges are identified at small training set sizes. In particular, we can see that the *domain* order prior is particularly effective across all training set sizes, even though it avoids placing a bias directly on edge estimation. In contrast, the edge priors are stronger assertions that make it possible to identify true edges with less data, but this type of prior knowledge may not always be available. True positives should always be evaluated in the context of the corresponding false positives. We see that the false positive learning curves are relatively flat, but can get worse with increasing amounts of data due to the data overwhelming the priors. The *domain* order prior and the *oracle* order prior at 100% of pairs specified are the most effective priors at reducing false positives. This is due to the ability of order priors to eliminate the identification of reverse edges, as we can see in the edge estimation curves of Fig. 2.

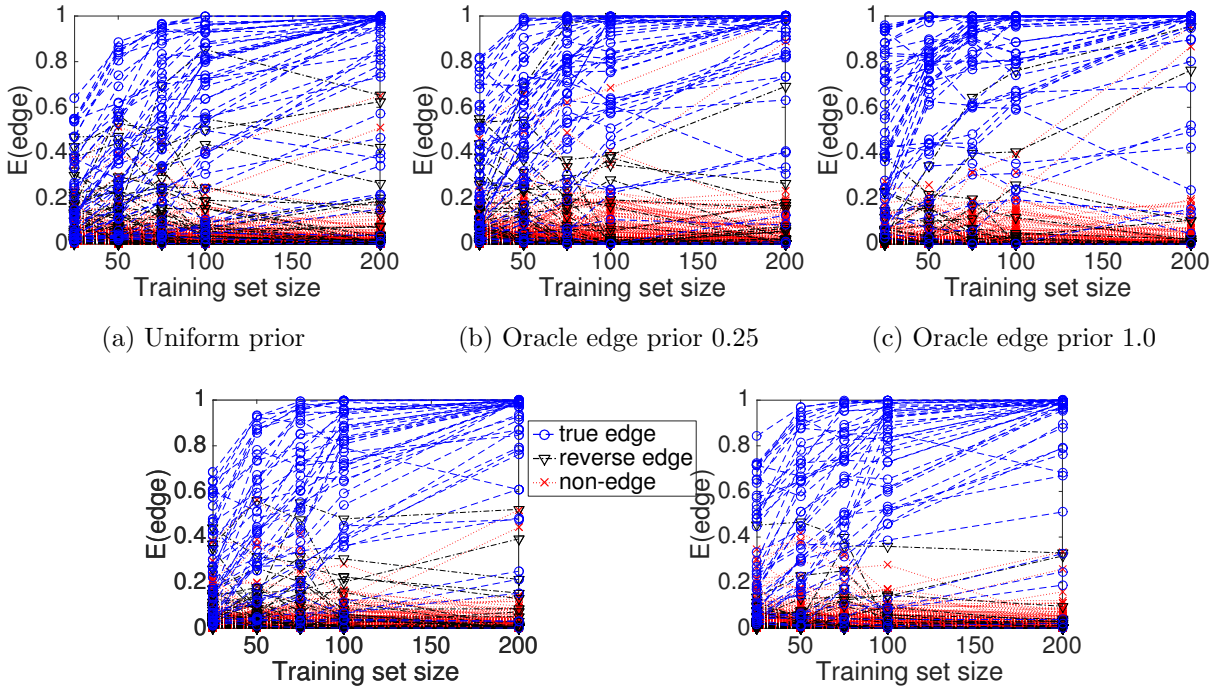
We next look at results from a larger benchmark, the *alarm* network with $p = 37$ variables [24]. This network is too large for the structure-space BDAGL software which is limited to data with $p \leq 20$. To calculate the order-space expectation, we use MCMC over partial-orders as implemented in BeanDisco with 1000 burn-in steps, 100 MCMC samples collected at 10-step intervals. We set maximum parent set size $r = 4$ and bucket size to 10. Fig. 5 shows the summary learning curves of true positives and false positives calculated according to Eq. 14. The true positives plot (Fig. 5(a)) indicates that using the prior information as an edge prior enables the learning algorithm to identify true edges with less



(a) True positives (up is good)

(b) False positives (down is good)

Figure 5: Comparison of learning curves on *Alarm* for various priors. True positives (left) are out of 46 true edges.



(a) Uniform prior

(b) Oracle edge prior 0.25

(c) Oracle edge prior 1.0

(d) Oracle order prior 0.25

(e) Oracle order prior 1.0

Figure 6: Learning curves for *alarm* from order-space expectation.

data, and that this is also true for prior information about the partial order, although the improvement is less dramatic. The false positive plot (Fig. 5(b)) shows that when using edge priors, the data can overwhelm the prior and in this case incorrectly identify non-edges as edges. The learning curves for the estimation $E(e)$ of each ordered pair (i, j) on 37 variables in Fig. 6 confirm that non-edges (not just reverse edges) are falsely identified as edges when using edge priors but that these same non-edges are not falsely identified using order priors. Overall, we see that, like in the *Asia* benchmark, at small amounts of data the edge priors can boost the identification of true edges. Yet, at all sample sizes, the order prior is more effective than the edge prior at reducing false positives. Other values for the fraction of prior knowledge were investigated and show similar trends.

Overall, these empirical results on benchmark data demonstrate that partial order priors improve the identification of true edges in a Bayesian network. Edge priors are stronger, and therefore should be used when available; however, a prior belief on variable order is often the only type of prior information available, rather than a prior belief about edges. We have demonstrated how to leverage such prior information about variable order in Bayesian network structure learning. Additionally, order information appears to be complementary to observed data in a way that edge priors are not. Large amounts of data can overwhelm edge priors, so that incorrect edges are learned despite correct edge priors. If prior information about edges and partial orders are both available, then these results suggest that combining the two types of priors could provide complementary information in addition to observed data to further improve results.

6 Application to Malware Characterization

Advanced persistent threats (APT) are a serious problem for many corporations and government agencies, in addition to malware aimed at a more general audience. APT is characterized as malware that has been tailored to accomplish a specific goal against a specific target. During the life cycle of APT malware, the authors generally follow standard software engineering principles which naturally leads to a phylogenetic graph, a graph demonstrating the evolutionary relationships among the different software versions [10, 9].

When beginning the process of understanding a new, previously unseen sample of mal-

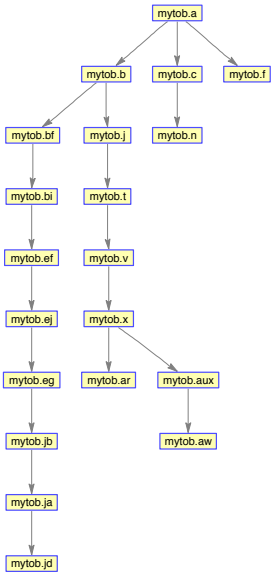


Figure 7: Desired output from a malware phylogeny learner.

ware, it is advantageous to leverage information gained from reverse engineering previously seen members of that instance's *family* because techniques learned from related instances can often be applied to a new program instance. A malware family is a group of related malware instances which share a common codebase and exhibit similar functionality (e.g. different branches in a software repository). We focus on extracting a more detailed picture about the relationships among the malware instances within a given family. Fig. 7 exhibits the proposed output of the ideal phylogenetic algorithm. This figure clearly shows the evolution of the *mytob* worm as a directed graph. The information present in Fig. 7 is invaluable for a reverse engineer tasked with understanding specific instances within a malware family as well as the general evolution of the family.

Experts use several sources of information to come up with a graph depicting the evolution of malware, such as the time the malware was first seen in the wild, the compile-time timestamp, the functionality of the sample, and the obfuscation methods employed in the sample. We investigate the combination of expert knowledge about the order in which programs appeared, along with using the programs themselves as data to infer the ancestry of the programs.

6.1 Data Views and Markov Chain Data Representation

We take advantage of three different types of data with the aim of covering the most popular data views used for malware analysis in the literature. The first data view is the control flow graph of the disassembled binary, which is a static data view (meaning the executable is not run) [25]. The other two views are dynamic data views (meaning the data is collected while the program is being executed). These dynamic data views are the dynamic instruction trace [26] and the dynamic system call trace [27]. We give a brief explanation of how these views are translated into data, but a more in-depth description of the views used is found in Anderson et al. [28].

We transform each data view into a Markov chain representing a summary of the sequential transitions in the program. As an illustrative example, we focus on the dynamic trace data, although this representation is suitable for any sequence-based data view. The dynamic trace data are the instructions the program executes, typically in a virtual machine to reduce the risk of contamination. Given an instruction trace $\mathcal{P}$, we are interested in finding a new representation, $\mathcal{P}'$, such that we can make unified comparisons in graph space while still capturing the sequential nature of the data. We achieved this by transforming the dynamic trace data into a Markov chain which is represented as a weighted, directed graph. A graph, $g = \langle V, E \rangle$, is composed of two sets, V and E. The elements of V are called vertices and the elements of E are called edges. In this representation, the edge weight, w_{ij}, between vertices i and j corresponds to the transition probability from state i to state j in the Markov chain, hence, the edge weights for edges originating at v_i are required to sum to 1, $\sum_{i \to j} w_{ij} = 1$. We use an $m \times m$ adjacency matrix, $\mathcal{A}$, to represent the graph (where $m = |V|$), where each entry in the matrix, $a_{ij} = w_{ij}$ [29].

The nodes of the graph are the instructions the program executes. To find the edges of the graph, we first scan the instruction trace, keeping counts for each pair of successive instructions. After filling in the adjacency matrix with these values, we normalize the matrix such that all of the non-zero rows sum to one. This process of estimating the transition probabilities ensures a well-formed Markov chain. The constructed graphs approximate the pathways of execution of the program, which we exploit as data samples.

The Markov chain graph can be summarized as $g = \langle V, E \rangle$; where V is the vertex

set composed of unique instructions; and, E is the weighted edge set where the weights correspond to the transition probabilities and are estimated from the data.

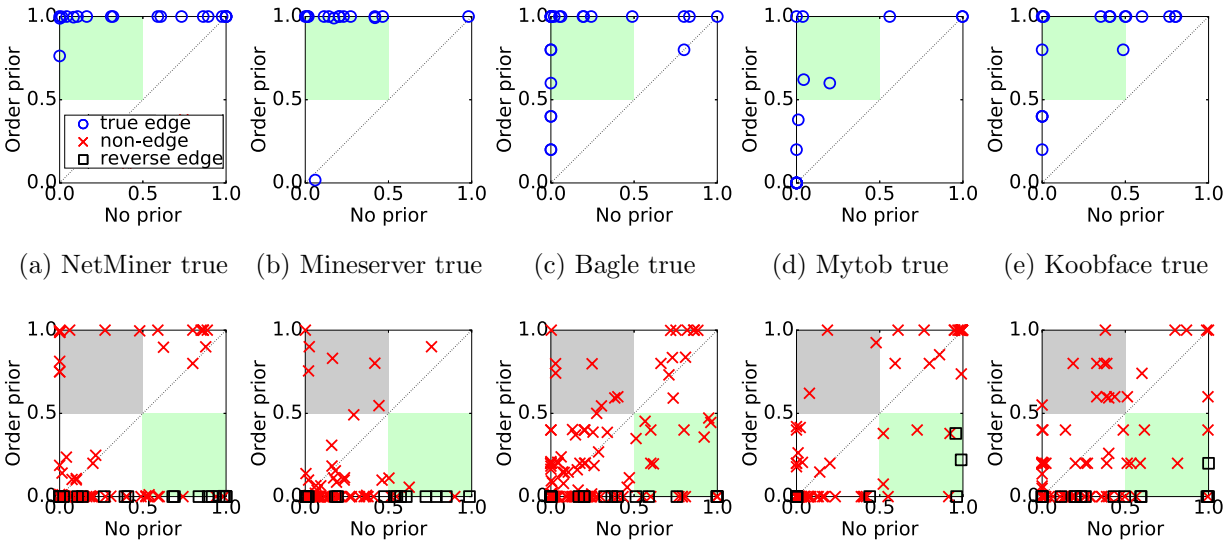
The transition matrices, $\mathcal{A}$, are typically sparse, meaning that many of the values are zero. Therefore, to ease the Bayesian network learning, we binarize the matrices, so that any non-zero probabilities are treated as a possible path, and represented with 1. The matrices are then vectorized to treat each value as an observed sample of the data.

6.2 Learning Graphs with Order Priors

We apply our Bayesian network discovery algorithm to families of malware programs along with expert information about the likely order in which these programs were developed. We investigate three malicious program families: Bagle [30], Koobface [31], and Mytob [32]; using the data features described above. Because the ground truth for malware development is not available, we also investigate two families of benign (non-malware) program families: NetworkMiner [33] and Mineserver [34]. For the benign programs, we have the ground truth phylogenetic graph as reported in their respective open-source repositories. For the malware programs, as a gold-standard best network, we use phylogenetic graphs constructed by domain experts.

The variables of the Bayesian network represent programs and the observed samples are the discretized transition matrix values. We incorporate domain knowledge about the order in which programs may have been developed. For a family with p programs, we construct the $p \times p$ prior matrix, $B(U)$, such that if program X_i is believed to precede program X_j, then $B(u_{ij}) = 1$ and $B(u_{ji}) = 0$; otherwise $B(u_{ij}) = 0.5$ (an uninformative prior).

With informative priors, network discovery should be able to better identify true edges and non-edges than is possible without this knowledge. We compare informative priors on partial orders, and uninformative uniform priors where $B(u_{ij}) = 1 \; \forall \; i, j$. We impose a limit on the size of parent sets to be at most 4. In all presented results on malware phylogeny, we use $\lambda = 1000$. The relatively high value of λ was selected to give greater weight to the perceived value of the domain knowledge captured in the prior and was found through a trial and error process. We perform a 10-fold cross validation to get posterior estimates of the expectations of edges with standard error [35].



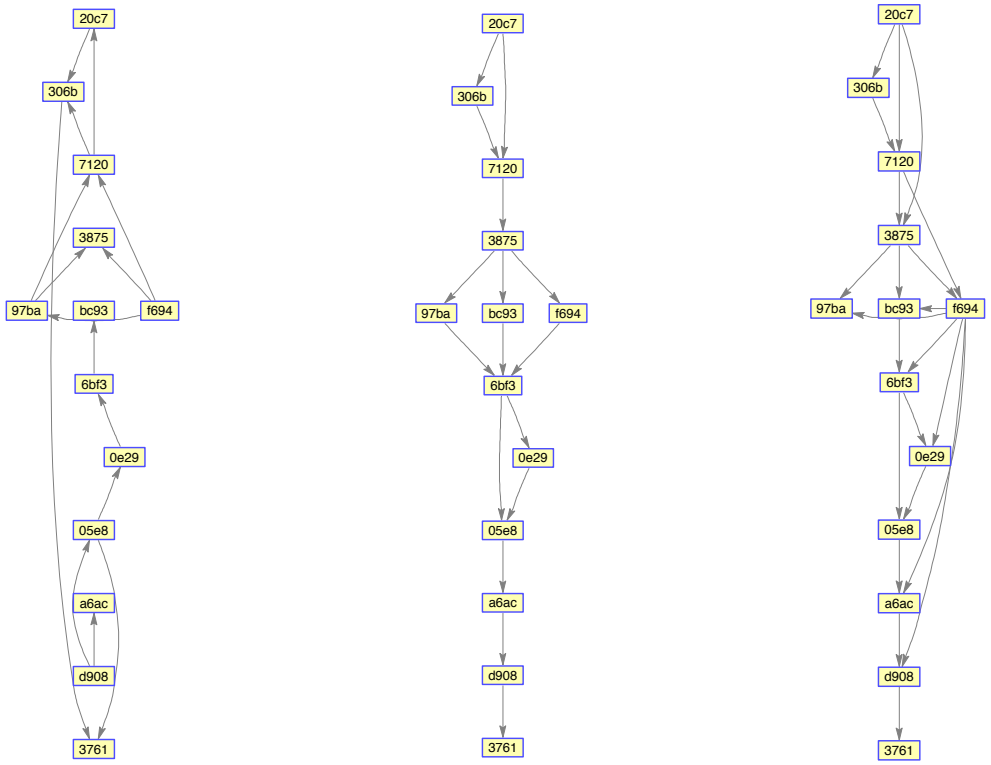
(a) NetMiner true (b) Mineserver true (c) Bagle true (d) Mytob true (e) Koobface true

(f) NetMiner non (g) Mineserver non (h) Bagle non (i) Mytob non (j) Koobface non

Figure 8: Comparison of edge expectations learned with and without the partial-order prior knowledge. Above-the-line indicates that the estimates benefit from the order prior for the true edges in the top row; while below-the-line indicates that the estimates benefit from the order prior for the reverse and non-edges of the bottom row. For a threshold value of 0.5, the pale green shade shows edges that are incorrectly identified by the plain BN discovery algorithm, but correctly identified when prior information is added; while the gray shade (upper left on non-edge plots) shows edges that are correct with plain BN discovery but incorrect when prior information added (assuming a threshold value of 0.5). In the bottom row, reverse edges are plotted as black squares, while all other non-edges are red Xs.

6.3 Phylogeny Learning Results

We investigate whether prior knowledge about pairwise-ordering of some nodes improves the edge expectation calculated by a Bayesian network discovery algorithm. Then we quantitatively compare our algorithm against competing algorithms on benign and malware families of programs.



(a) Uniform prior (b) Gold standard (c) Order prior

Figure 9: Mineserver phylogenetic graphs (a) learned without prior information; (b) gold standard (true network) from repository logs (displayed in the center for ease of comparison to learned networks); and (c) learned with order prior information.

6.3.1 Effect of Prior Knowledge

We first look at results comparing the edge expectation $E(e)$ learned *with* prior information and *without* prior information. We split the results into three different categories: learning the true edges, *reverse* edges and *non-edges*, as in the benchmark experiments. Ideally, the expectations of true edges should be close to 1. Non-edges are edges that do not exist in the ground-truth network, and so the expectations of non-edges should ideally be close to 0. Reverse edges are non-edges that have the same end points as a true edge, but in the wrong direction, and so the expectations of reverse edges should again be close to 0. We separate out the reverse edges because they can be particularly difficult to infer from data alone.

Fig. 8 shows the edge expectations (averaged over 10 cross-folds) estimated with the partial-order prior versus those estimated with an uninformative uniform prior, for each of the three families of malware. In the top row, we see that the estimates for the true edges are generally higher (above the dashed diagonal line) using the prior information than without, indicating that the prior information makes it easier to learn true-positive dependencies. Reverse edges should have low expectation, and the bottom row of the figure indicates that the order prior usually produces lower expectation for reverse edges (black squares). For the non-edges (red Xs in the bottom row), the edge expectations are sometimes improved (below the dashed diagonal line) and sometimes they are worse. It is difficult to see in these plots, but overall there are more non-edges with improved estimation than vice-versa. We quantify these accuracy gains in the next section.

From the learned edge expectations, we produce a phylogenetic graph by selecting a threshold value, t, such that if $E(e_{ij}) \geq t$ then e_{ij} is an edge in the graph. Fig. 9 shows the learned phylogenetic graphs for $t = 0.5$ with and without prior information, as well as the ground truth evolution of the Mineserver family of programs. The Mineserver family is particularly interesting because it includes branches and merges of programs. It is clear that Bayesian network discovery can recover many of the dependencies among programs. However, without expert knowledge about the ordering of the programs, many of the edges learned are incorrect (Fig. 9a). With ordering information on some pairs of programs, many more of the inferred edges are correct (Fig. 9c). Most importantly, order information that is included as a prior — rather than as a post-hoc fix as has been done previously — improves the detection of edges that are completely missed without the prior information (such as between the bottom two nodes, d908 and 3761, of the graph in Fig. 9).

6.3.2 Comparison with Competing Algorithms

We compare against other algorithms that have been used for learning malware phylogeny. The algorithms that we compare against are: MKLGC [17] which uses a graphical lasso with multiple kernel learning plus clustering of malware programs; Gupta [9], a graph pruning algorithm; and the minimum spanning tree as a naive baseline. These three algorithms each produce a single network as output, instead of an expectation on each edge as our

Bayesian network discovery algorithm produces. Although we do not generally recommend translating the results of the Bayesian discovery algorithm into a single network, for comparison to these competing algorithms, it is necessary. To generate a single network for comparison from our edge expectations, we threshold edges at $t = 0.5$ and only consider those with $E(e) \geq 0.5$ to be learned positive edges in the learned network. We calculate the precision and recall of the number of true edges identified by each algorithm. Precision is the number of true positive edges learned divided by the total number of positive edges identified. Recall is the number of true positive edges learned divided by the total number of true edges in the ground truth network. Precision and recall are good metrics to use when the true events of interest (in this case, true edges) are rare in comparison to the total number of events (in this case, all possible edges, the number of which is quadratically larger than the number of true edges) [36]. We also combine precision and recall into a single F1 score. The F1 score is the harmonic mean of precision and recall: $\text{F1} = 2 \cdot \frac{precision \cdot recall}{precision + recall}$.

Table 1 gives the results comparing precision, recall and F1 score for all of the algorithms including our Bayesian network discovery with prior information, BNPrior; and without prior information, BN. As the table shows in bold, our BNPrior algorithm outperforms the other algorithms for most metrics of the program families. BNPrior outperforms the other algorithms in overall accuracy, as represented by the F1 score. Precision and recall should be evaluated as a pair. We see that BNPrior always has either the best precision or recall, and sometimes both. Note that with BN and BNPrior, we can trade-off precision and recall by adjusting the threshold value, t, whereas Table 1 shows only the results at $t = 0.5$. For Mineserver and Koobface, BNPrior gives much better recall than any algorithm, but at a somewhat lower precision than one or two algorithms. For Mytob, BNPrior gives much better precision than any other algorithm but at a lower recall than what MKLGC gives. Mytob is a difficult learning problem for all of the algorithms. BNPrior demonstrates the most impressive performance improvement on Bagle, while the performance gain on NetworkMiner is also substantial.

Our results indicate that software, including malware, phylogeny can be inferred using Bayesian discovery and the accuracy of the ancestry relationships is improved with prior knowledge about the ordering of the programs. Bayesian networks with order information

Table 1: Phylogenetic graph reconstruction results in terms of precision of true edges, recall of true edges, and F1 score.

Dataset	Method	Precision	Recall	F1 score
Network-Miner	BNPrior	**0.5128**	**1.0000**	**0.6780**
	BN	0.2632	0.5000	0.3448
	MKLGC	0.4857	0.8500	0.6182
	Gupta	0.3810	0.4000	0.3903
	MST	0.3500	0.7000	0.4667
Mine-server	BNPrior	0.6818	**0.9375**	**0.7895**
	BN	0.0667	0.0625	0.0645
	MKLGC	0.7222	0.8125	0.7647
	Gupta	**0.8462**	0.3438	0.4889
	MST	0.0000	0.0000	0.0000
Bagle	BNPrior	**0.5263**	**0.8333**	**0.6452**
	BN	0.1026	0.1667	0.1270
	MKLGC	0.2000	0.3333	0.2500
	Gupta	0.1200	0.0120	0.1224
	MST	0.0208	0.0417	0.0278
Mytob	BNPrior	**0.2059**	0.3684	**0.2642**
	BN	0.0833	0.1579	0.1091
	MKLGC	0.1563	**0.5263**	0.2410
	Gupta	0.0500	0.0526	0.0513
	MST	0.0526	0.1053	0.0702
Koobface	BNPrior	0.4516	**0.7778**	**0.5714**
	BN	0.1923	0.2778	0.2273
	MKLGC	**0.5812**	0.5000	0.5376
	Gupta	0.3158	0.3333	0.3243
	MST	0.0278	0.0556	0.0371

prove to be a good model for software phylogeny. Most importantly, including order information about the variables in the network helps to learn more accurate dependencies overall.

7 Conclusions

Bayesian network learning benefits from combining expert knowledge and statistical data. These two distinct sources provide different types of information and are subject to different types of uncertainty. Our partial-order prior leverages the type of information most

accessible to humans, while maintaining the strength of statistical dependencies to be calculated directly from data. Our empirical study on benchmark data demonstrates that edge priors should be used when such information is available because edge priors are most effective at increasing the number of true positive edges identified at small training set sizes. Prior beliefs about variable order is often the only type prior information available, and our empirical results show that a prior on variable order can perform almost as well as a prior on edges. Furthermore, order priors appear to provide complementary information to observed data that allows correct edges to be identified even at large training set sizes when the data could overwhelm edge priors. The proposed pairwise order prior is efficient to implement. While the primary reason given for avoiding order-space expectation is the bias present in the calculation, we show that this bias in practice is not detrimental to identifying the edges present in the network. Furthermore, our order-space method scales to larger network sizes, while being able to leverage prior knowledge to improve the accuracy of learning for datasets with limited number of observed samples.

Bayesian network structure learning with an order prior is an effective method for reconstructing the evolution of software and malware development. We show that our approach is able to identify the correct direction of edges and also identifies edges missed by other graphical model learning methods, including an undirected model. The order prior is necessary in improving the performance of the Bayesian network learner above the performance results of competing methods.

Improving the structure of learned Bayesian networks is important in analysis domains that have both data and expert knowledge, as we discussed in this paper. As future work, we plan to look into combining human knowledge with observed data in risk analysis and decision-making systems. We envision an interactive approach of generating networks based on expert knowledge, refining the network with evidentiary data, and further modifying the network by experts to produce systems that both match the data and represent human beliefs. The efficient algorithm presented in this paper is an important first step in a knowledge-rich and data-responsive Bayesian network structure learning system that can provide beneficial information for software reverse engineers to aid their understanding of the evolution of malware and help to indicate potential future directions.

References

[1] David Heckerman, Dan Geiger, and David M Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20(3):197–243, 1995.

[2] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 2014.

[3] N. Friedman and D. Koller. Being Bayesian about network structure: A Bayesian approach to structure discovery in Bayesian networks. *Machine Learning*, 50(1):95–125, 2003.

[4] Mikko Koivisto and Kismat Sood. Exact Bayesian structure discovery in Bayesian networks. *Journal of Machine Learning Research*, 5:549–573, 2004.

[5] Diane Oyen, Blake Anderson, and Christine Anderson-Cook. Bayesian networks with prior knowledge for malware phylogenetics. In *AAAI Workshop on Artificial Intelligence and Cybersecurity*, 2016.

[6] Stephen M. Smith, Karla L. Miller, Gholamreza Salimi-Khorshidi, Matthew Webster, Christian F. Beckmann, Thomas E. Nichols, Joseph D. Ramsey, and Mark W. Woolrich. Network modelling methods for fMRI. *NeuroImage*, 54(2):875–891, 2011. ISSN 1053-8119.

[7] A.V. Werhli and D. Husmeier. Reconstructing gene regulatory networks with Bayesian networks by combining expression data with multiple sources of prior knowledge. *Statistical Applications in Genetics and Molecular Biology*, 6(1):Article15, 2007.

[8] Marco Grzegorczyk and Dirk Husmeier. Improving the structure MCMC sampler for Bayesian networks by introducing a new edge reversal move. *Machine Learning*, 71 (2-3):265–305, 2008.

[9] Archit Gupta, Pavan Kuppili, Aditya Akella, and Paul Barford. An empirical study of malware evolution. In *First International Conference on Communication Systems and Networks (COMSNETS) and Workshops*, pages 1–10, 2009.

[10] Craig Darmetko, Steven Jilcott, and John Everett. Inferring accurate histories of malware evolution from structural evidence. In *The Twenty-Sixth International FLAIRS Conference*, 2013.

[11] W. Buntine. Theory refinement on Bayesian networks. In *Seventh Conference on Uncertainty in Artificial Intelligence*, pages 52–60, 1991.

[12] N. Friedman, I. Nachman, and D. Peér. Learning Bayesian network structure from massive datasets: the sparse candidate algorithm. In *Fifteenth Conference on Uncertainty in Artificial Intelligence*, pages 206–215, 1999.

[13] Thomas Verma and Judea Pearl. Equivalence and synthesis of causal models. *Proceedings of the Sixth Annual Conference on Uncertainty in Artificial Intelligence*, pages 255–270, 1991. ACM ID: 719736.

[14] David Maxwell Chickering. Learning equivalence classes of Bayesian-network structures. *The Journal of Machine Learning Research*, 2:445–498, 2002.

[15] Daniel Eaton and Kevin Murphy. Bayesian structure learning using dynamic programming and MCMC. In *Uncertainty in Artificial Intelligence (UAI)*, 2007.

[16] Gérard Wagener, Alexandre Dulaunoy, et al. Malware behaviour analysis. *Journal in computer virology*, 4(4):279–287, 2008.

[17] Blake Anderson, Terran Lane, and Curtis Hash. Malware phylogenetics based on the multiview graphical lasso. In *Advances in Intelligent Data Analysis XIII*, pages 1–12, 2014. ISBN 978-3-319-12570-1.

[18] Eugene Charniak. Bayesian networks without tears. *AI Magazine*, 12(4):50, 1991.

[19] Stuart Russel and Peter Norvig. Artificial intelligence: A modern approach. *Prentice Hall*, 2003.

[20] Thomas Dyhre Nielsen and Finn Verner Jensen. *Bayesian networks and decision graphs*. Springer Science & Business Media, 2009.

[21] Gregory F. Cooper and Edward Herskovits. A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9:309–347, 1992.

[22] Teppo Niinimaki, Pekka Parviainen, and Mikko Koivisto. Partial order MCMC for structure discovery in Bayesian networks. In *Twenty-Seventh Annual Conference on Uncertainty in Artificial Intelligence*, pages 557–564, 2011.

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

[24] I. A. Beinlich, H. J. Suermondt, R. M. Chavez, and G. F. Cooper. The ALARM monitoring system: A case study with two probabilistic inference techniques for belief networks. In *Second European Conference on Artificial Intelligence in Medicine*, volume 38, pages 247–256, 1989.

[25] Christopher Kruegel, Engin Kirda, Darren Mutz, William Robertson, and Giovanni Vigna. Polymorphic worm detection using structural information of executables. In *Recent Advances in Intrusion Detection*, pages 207–226, 2006.

[26] Blake Anderson, Daniel Quist, Joshua Neil, Curtis Storlie, and Terran Lane. Graph-based malware detection using dynamic analysis. *Journal in Computer Virology*, 7(4): 247–258, 2011.

[27] Steven A Hofmeyr, Stephanie Forrest, and Anil Somayaji. Intrusion detection using sequences of system calls. *Journal of Computer Security*, 6(3):151–180, 1998.

[28] Blake Anderson, Curtis Storlie, and Terran Lane. Improving malware classification: Bridging the static/dynamic gap. In *Proceedings of the 5th ACM Workshop on Security and Artificial Intelligence*, pages 3–14, 2012. ISBN 978-1-4503-1664-4.

[29] Clemens Kolbitsch, Paolo Milani Comparetti, Christopher Kruegel, Engin Kirda, Xiaoyong Zhou, and XiaoFeng Wang. Effective and efficient malware detection at the end host. In *USENIX Security Symposium*, pages 351–366, 2009.

[30] Symantec. Symantec Bagle security report. http://www.symantec.com/security_response/writeup.jsp?docid=2004-011815-3332-99, 2004. Accessed: September 17, 2013.

[31] Symantec. Symantec Koobface security report. http://www.symantec.com/security_response/writeup.jsp?docid=2008-080315-0217-99, 2008. Accessed: September 17, 2013.

[32] Symantec. Symantec Mytob security report. http://www.symantec.com/security_response/writeup.jsp?docid=2005-022614-4627-99, 2005. Accessed: September 17, 2013.

[33] Networkminer. http://sourceforge.net/projects/networkminer, 2013. Accessed: September 17, 2013.

[34] Mineserver. https://github.com/fador/mineserver, 2013. Accessed: September 17, 2013.

[35] Ron Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. *International Joint Conference on Artificial Intelligence*, 14(2):1137–1145, 1995.

[36] Jesse Davis and Mark Goadrich. The relationship between precision-recall and ROC curves. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 233–240, 2006.