Learning Bayesian Networks with Cops and Robbers

Topi Talvitie TOPI.TALVITIE@HELSINKI.FI
Department of Computer Science, University of Helsinki, Finland

Pekka Parviainen PEKKA.PARVIAINEN@UIB.NO
Department of Informatics, University of Bergen, Norway

Abstract

Constraint-based methods for learning structures of Bayesian networks are based on testing conditional independencies between variables and constructing a structure that expresses the same conditional independencies as indicated by the tests. We present a constraint-based algorithm that learns the structure of a Bayesian network by simulating a cops-and-a-robber game. The algorithm is designed for learning structures of low treewidth distributions and in such case it conducts conditional independence tests only with small conditioning sets.

Keywords: Bayesian networks; Structure learning; Constraint-based methods; Treewidth.

1. Introduction

Bayesian networks are representations of joint probability distributions. A Bayesian network consists of two parts: a structure and parameters. The *structure* of a Bayesian network is a *directed acyclic graph (DAG)* and it expresses conditional independencies in the distribution. The *parameters* specify the conditional distributions.

Bayesian networks are often learned from data. There are two main approaches: score-based and constraint-based. In the *score-based* approach, one assigns each DAG a score based on how well it fits to the data and then tries to find a network with maximum score. In the *constraint-based* approach, one tests conditional independencies between variables and tries to construct a DAG that expresses the same conditional independencies and dependencies that are implied by the test results.

In this paper, we take the constraint-based approach. When designing an efficient constraint-based algorithm, it is desirable that the algorithm conducts as few conditional independence tests as possible. Furthermore, the larger the conditioning set, the more unreliable the tests are. Thus, it is also desirable that the conditional independence tests use as small conditioning sets as possible. Typically, the number of conditional independence tests and the sizes of the conditioning sets depend on the structure of the underlying Bayesian network. In other words, it is possible to analyze parameterized complexity of constraint-based learning with respect to some structural parameters. For example, the number of conditional independence tests conducted by the PC algorithm Spirtes et al. (2000) and its variants is upper bounded by $n^{O(d)}$, where n is the number of nodes and d is the highest degree of the nodes in the data-generating DAG (given standard assumptions), and the sizes of conditioning sets are $O(d)$.

From a theoretical point of view, it is interesting to see to what extent it is possible to develop algorithms whose running time or the number of conditional independence tests conducted is parameterized by other structural properties. In this paper, we study parameterized complexity with respect to the treewidth of the data-generating Bayesian network. Treewidth is a property of a graph that measures how close the graph is to a tree. It is particularly interesting parameter in the context of Bayesian networks, because inference in Bayesian networks is tractable if the structure of the Bayesian network has low treewidth. Our main result (Theorem 5) shows that the structure of a Bayesian network whose treewidth is at most k can be learned in $O(n^{k+4})$ time by conducting at most $O(n^{k+3})$ conditional independence tests with conditioning sets of size at most $k+1$.

One way to define treewidth is to use a cops-and-a-robber game where cops chase a robber who moves along the edges of a graph: the lower the treewidth, the fewer cops are needed to catch the robber. Our algorithm is based on simulating this game. The challenge we face is that the graph is unknown and the cops have to conduct conditional independence tests to gain information about the graph. Intuitively, we try to find separators that partition the graph into disjoint subgraphs. Separators are powerful in the sense that once we have found one, we know that there are no arcs between the disjoint subgraphs and we can recursively solve these smaller problems.

1.1 Related Work

Constraint-based methods can be informally divided into several types. The PC algorithm (Spirtes et al., 2000) and its variants typically start with a complete network and then remove edges by conditioning on adjacent nodes. Local learning (e.g., (Aliferis et al., 2010)) is based on finding neighbors or Markov blankets of single nodes and then constructing the global DAG based on this local information. Our algorithm tries to find separators and divide nodes to smaller sets that can be solved separately. A notable example of similar algorithms is the recursive method presented by Xie and Geng (2008).

There is also work on PAC-learning graphical models with low treewidth (Chechetka and Guestrin, 2007; Narasimhan and Bilmes, 2004). The algorithm by Chechetka and Guestrin (2007) finds a tree decomposition in time $O(n^{2k+3})$[1].

It should be noted that score-based methods for finding bounded treewidth Bayesian networks (e.g., (Elidan and Gould, 2008; Korhonen and Parviainen, 2013)) have a different setting compared to our method. They find the highest-scoring network among the networks with bounded treewidth; They make no assumptions on the data-generating distribution and work also if the data-generating network has high treewidth.

2. Preliminaries

In this section, we introduce notation, definitions and known results that are used later.

2.1 Bayesian Networks

Let $G = (N, A)$ be a directed acyclic graph where N is the node set and A is the arc set. We denote the parents of the node v in G by A_v. We use the notation $G[X]$ for the subgraph of G induced by $X \subseteq N$. Furthermore, we denote $n = |N|$.

A distribution factorizes with respect to a DAG G if the joint probability distribution can be written in the form

$$P(N) = \prod_{v \in N} P(v \,|\, A_v).$$

1. It should be noted that Chechetka and Guestrin (2007) are interested in sample complexity and thus their set-up is somewhat different compared to ours.

If a distribution factorizes with respect to G, then it can be represented by a Bayesian network whose structure is G. Formally, a Bayesian network is a pair (G, θ), where G is a DAG and θ specifies the parameters of the local conditional distributions $P(v \,|\, A_v)$.

Random variables u and v are *conditionally independent* given a set of variables S in a distribution P if $P(u, v \,|\, S) = P(u \,|\, S)P(v \,|\, S)$. We use the notation $u \perp v \,|\, S$ to denote that u and v are conditionally independent given S.

A *collider* on a path is a node with two incoming arcs along the path. A path in a DAG is *blocked* by a set of variables S if (i) there is a collider on the path such that neither the collider or any of its descendants is in the conditioning set S or (ii) there is a non-collider on the path such that it is in the conditioning set S. Nodes u and v are *d-separated* by S if all paths between u and v are blocked by S.

It can be shown that if u and v are d-separated by S in a DAG G, then u and v are conditionally independent given S in all Bayesian networks whose structure is G. Particularly, every Bayesian network satisfies the *local Markov property*: a node is conditionally independent of its non-descendants given its parents.

In the constraint-based approach, the goal is to find a DAG G such that u and v are conditionally independent given S in the data-generating distribution if and only if u and v are d-separated by S in G. The "if" direction follows directly from the factorization property of Bayesian networks. However, to guarantee that conditional independence implies d-separation, we have to make assumptions. A key assumption is called *faithfulness*.

Definition 1 (Faithfulness) *A distribution P is faithful to a DAG G if conditional independence $v \perp u \,|\, S$ in P implies that v and u are d-separated by S in G.*

It follows that if the data-generating distribution is faithful to DAG G, all variables in G are observed and follow the same distribution then conditional independence implies d-separation.

We note that the found DAG is not necessarily unique, that is, there can be several DAGs that have the same set of d-separations (in other words, express the same set of conditional independencies). Such DAGs form a *Markov equivalence class*. It is known that two DAGs are Markov equivalent if they have the same skeleton and the same set of v-structures. The *skeleton* of a DAG is an undirected graph that is obtained from the DAG by removing the directions of the arcs. A *v-structure* is a collider structure $u \rightarrow w \leftarrow v$ such the there is no arc between u and v. A Markov equivalence class can be represented by a *completed partially directed acyclic graph (CPDAG)*[2] which has both directed and undirected edges. A directed edge in a CPDAG corresponds to an arc that points to the same direction in all DAGs in the Markov equivalence class; other edges are undirected.

2.2 Treewidth

Let $H = (N, E)$ be a undirected graph where N is the node (vertex) set and E is the edge set. Let $X = \{X_1, \ldots, X_m\}$ be a collection of subsets of N. Let T be a tree whose vertex set is X; we call the elements of X *bags*. Now, the pair (T, X) is a tree decomposition of H if

1. Every element of N is member of at least one bag, that is, $\cup_i X_i = N$,

2. Also known as an essential graph.

2. For each edge $\{u, v\} \in E$ there exists a bag X_i such that both $u \in X_i$ and $v \in X_i$
3. Running intersection property: For every node $v \in N$, the subtree of T induced by the bags containing v is connected.

The width of a tree decomposition is the size of the largest bag minus one. Treewidth of graph H is defined as the smallest width over all tree decompositions of H.

Treewidth can be equivalently defined also using the cops-and-a-robber game (Seymour and Thomas, 1993). In this game, there is one robber and $k + 1$ cops. The robber moves infinitely fast from one node to another using edges of a graph. At any given moment, a cop either occupies one node or is "in the air" with a helicopter. The goal of the robber is to evade capture and the goal of the cops is to catch the robber by landing on the node occupied by the robber. Note that the cops can move from any node to any other node but the movement of the robber is limited to the edges of the graph and it cannot move through a node occupied by a cop. When a cop moves from one node to another it spends a positive amount of time "in the air": the robber is infinitely fast so if he sees a cop landing to the node where he is currently, he has time to move away before the cop catches him.

It has been shown (Seymour and Thomas, 1993) that $k + 1$ cops have a winning strategy in this game on graph H if and only if the treewidth of the graph is at most k.

The structure of a Bayesian network is a directed graph, so the above definitions are not directly applicable to Bayesian networks. Let the *moral graph* of a DAG $G = (N, A)$ be an undirected graph $H = (N, E)$ such that an edge $\{u, v\} \in E$ if and only if $uv \in A$, $vu \in A$ or there exists $w \in N$ such that $uw \in A$ and $vw \in A$. The treewidth of a Bayesian network is defined to be the treewidth of the moral graph of its structure.

In our algorithm, we will use the following property of moral graphs: If a node u is disconnected from v in $H[N \setminus X]$, then u and v are d-separated by X in G.

3. Algorithm

Our algorithm for learning the the structure G of the Bayesian network works in two phases. In the first phase, described more in detail in Section 3.1, we learn a tree decomposition of width k of the moral graph H, where k is the treewidth of H. We do this by successively increasing a parameter k starting from 1, and for each value we solve whether $k + 1$ cops can always catch one robber in the cops-and-a-robber game on H using independence queries with conditioning set size at most $k + 1$. If the answer is in the affirmative, we conclude that k is indeed the treewidth of H and obtain the tree decomposition of width k as a byproduct; otherwise, k is too small and we need to repeat the process with k incremented by one.

In the second phase, we use additional independence queries to prune edges from the supergraph induced by the tree decomposition to obtain the skeleton of G, and orient the edges to obtain the CPDAG of G. This process is described in more detail in Section 3.2.

3.1 Learning the Tree Decomposition

To check whether $k + 1$ cops can always catch the robber, we use a recursive dynamic programming algorithm that keeps track of the set C of nodes containing cops and the set R of nodes in which the robber can move. To limit the possible states we need to consider, we simplify the way the game is played without affecting the end result as follows.

- The set of cops C is always kept minimal, that is, they must be exactly the boundary ∂R of the set R, that is, the nodes in $N \setminus R$ that are adjacent in H to at least one node in R. Omitting any of these cops would expand the area accessible to the robber, and conversely, additional cops do not affect the game at all as they do not block the robber.

- The cops never retreat, that is, the cops move always by moving one cop waiting in a helicopter to a node $v \in R$, finding out which component R' of $R \setminus \{v\}$ the robber went, updating $R = R'$ and finally removing the cops that are no longer adjacent to R. The fact that this does not affect the outcome follows from the result due to Seymour and Thomas (1993) on the equivalence of searching and monotone searching in the cops-and-a-robber game.

- Initially there is only one cop on the graph in an arbitrary node $v_0 \in N$ and the robber is in one of the components of $N \setminus \{v_0\}$. This does not affect the outcome, because if the robber is hiding in v_0, the cops have to traverse that node.

To implement this algorithm, we define three functions, PRESOLVE, SOLVE and EXTRACTCOMP. The first two functions are given a pair (C, R) of sets of nodes, and they return a Boolean value on whether the cops currently occupying nodes C can win if the robber is in a set R. For SOLVE, the arguments must always be in the minimal form: $H[R]$ is a non-empty connected subgraph of H, and $C = \partial R$. The function PRESOLVE works as a preprocessor for SOLVE; in its arguments, we allow R to be empty or disconnected, and the only requirement for C is that $\partial R \subseteq C \subseteq N \setminus R$. PRESOLVE partitions $H[R]$ into components with the help of EXTRACTCOMP and calls SOLVE for each component R' and set of cops $C' = \partial R'$; the cops have to win regardless of which component the robber chooses. SOLVE considers all the ways one cop waiting in a helicopter can advance into R, calling PRESOLVE to evaluate each of them and returning true if at least one successful advancement is found. The algorithm is started by calling PRESOLVE$(\{v_0\}, N \setminus \{v_0\})$; if the return value is true, the cops win.

To obtain structural information on H, the functions use queries to a conditional independence oracle; the result of the independence query for nodes u and v with a conditioning set S, denoted by INDTEST(u, S, v), is true if $u \perp v \,|\, S$, or equivalently by the faithfulness assumption, if u and v are d-separated by S in G. The pseudocodes of the functions PRESOLVE, SOLVE, EXTRACTCOMP are given in Algorithm 1. To prove that the algorithm uses INDTEST correctly to reason about the structure of H, we need the following two lemmas.

Lemma 2 *If $R \subseteq N$ and $\partial R \subseteq C \subseteq N \setminus R$, then for all $c \in C$ it holds that $c \in \partial R$ if and only if there exists $r \in R$ such that* INDTEST$(c, C \setminus \{c\}, r)$ returns false.

Proof To prove the implication in the forward direction, assume that $c \in \partial R$. By definition, c has a neighbor $x \in R$ in H. If x and c are connected by an edge in G, then the claim holds for $r = x$ because adjacent nodes are never d-separated. Otherwise, the edge between c and x has been added to H due to moralization, which means that G contains a v-structure $c \to v \leftarrow x$ for some $v \in N \setminus \{c, x\}$. If $v \in R$, then the claim holds for $r = v$. If $v \notin R$, then because v is adjacent to $x \in R$, it holds that $v \in \partial R$ and thus $v \in C \setminus \{c\}$. Hence the claim holds for $r = x$, because the path from c to x through v in G is not blocked by $C \setminus \{c\}$.

To complete the proof of the equivalence, it suffices to observe that if $c \notin \partial R$, then $C \setminus \{c\} \supseteq \partial R$ and thus c is disconnected from R in $H[N \setminus (C \setminus \{c\})]$. This implies that for all $r \in R$, all paths

Algorithm 1

```
▷ Assumes that $R \subseteq N$, $\partial R \subseteq C \subseteq N \setminus R$ and $|C| \leq k+1$
function PRESOLVE($C, R$)
    if $R = \emptyset$
        ▷ Nowhere to go for robber $\rightarrow$ cops win
        return true
    ▷ Consider arbitrary component $R'$ of $H[R]$
    $r_0 \leftarrow$ arbitrary element of $R$
    $R' \leftarrow$ EXTRACTCOMPONENT($C, r_0$)
    ▷ Remove unnecessary cops from $C \supseteq \partial R'$ to obtain $C' = \partial R'$ (Lemma 2)
    $C' \leftarrow C$
    for $c \in C$
        if for all $r \in R'$: INDTEST$(c, C' \setminus \{c\}, r) =$ true
            $C' \leftarrow C' \setminus \{c\}$
    ▷ If all $k+1$ cops are needed, no advancements can be made $\rightarrow$ cops lose
    if $|C'| = k+1$
        return false
    ▷ Otherwise $|C'| \leq k$ and we can SOLVE the case
    if SOLVE$(C', R') =$ false
        return false
    ▷ Consider the rest of the components recursively
    return PRESOLVE$(C, R \setminus R')$

▷ Assumes that $\emptyset \neq R \subseteq N$, $H[R]$ is connected, $C = \partial R$ and $|C| \leq k$
function SOLVE($C, R$)
    ▷ Cops win if there is at least one winning advancement
    for $a \in R$
        if PRESOLVE$(C \cup \{a\}, R \setminus \{a\}) =$ true
            return true
    return false

▷ Assumes that $C \subseteq N$, $|C| \leq k+1$ and $r_0 \in N \setminus C$
function EXTRACTCOMPONENT($C, r_0$)
    ▷ Find the component $R$ of $H[N \setminus C]$ containing node $r_0$ (Lemma 3)
    $R \leftarrow \{r_0\}$, $Q \leftarrow \{r_0\}$
    while $Q \neq \emptyset$
        $r_1 \leftarrow$ arbitrary element of $Q$
        $Q \leftarrow Q \setminus \{r_1\}$
        for $r \in N \setminus (C \cup R)$
            if INDTEST$(r, C, r_1) =$ false
                $R \leftarrow R \cup \{r\}$
                $Q \leftarrow Q \cup \{r\}$
    return $R$
```

between c and r in G are blocked by $C \setminus \{c\}$, and thus INDTEST$(c, C \setminus \{c\}, r)$ is true. ■

Lemma 3 *Let $C \subseteq N$, and define graph $I_C = (N \setminus C, E_C)$ by*

$$E_C = \{\{u, v\} \, : \, u, v \in N, u \neq v \text{ and } \textsc{IndTest}(u, C, v) = \textbf{false}\}.$$

Now for each component $I_C[S]$ of I_C, $H[S]$ is a component of $H[N \setminus C]$. Consequently, we can use graph search in I_C to find components of $H[N \setminus C]$ (as we do in EXTRACTCOMPONENT *of Algorithm 1).*

Proof To prove the claim, we prove that for all pairs of adjacent nodes in I_C there exists a path between the nodes in $H[N \setminus C]$ and vice versa. The forward direction of the claim follows from the fact that if $\{u, v\} \in E_C$, then there exists a path between u and v in G not blocked by C, and thus they are connected by a path in the residual moral graph $H[N \setminus C]$.

For the other direction, assume that nodes $u, v \in N \setminus C$ are adjacent in H. If they are adjacent in G as well, then they are adjacent in I_C. Otherwise, the edge between u and v has been added to H due to moralization, and thus G contains a v-structure $u \rightarrow x \leftarrow v$ for some $x \in N \setminus \{u, v\}$. If $x \in C$, then the path between u and v through x is not blocked by C and thus $\{u, v\} \in E_C$. Otherwise $x \in N \setminus C$ and thus u and v are connected in I_C through x. ■

The cyclic recursion between PRESOLVE and SOLVE eventually reaches the base case $R = \emptyset$ in PRESOLVE, because on each call from PRESOLVE to SOLVE, the set R does not increase, and on each call from SOLVE to PRESOLVE as well as each call from PRESOLVE to itself, the set R decreases strictly. However, a naive implementation of this recursion often does a lot of recomputation, that is, the same function is called with the same arguments multiple times. To avoid this recomputation, we *memoize* the results of the functions PRESOLVE and EXTRACTCOMPONENT as well as the independence oracle INDTEST: the first time they are called with some arguments, the return value is stored into a table, and on subsequent calls with the same arguments, the value stored in the table is used instead of recomputing it.

For time complexity estimations, we assume that we can do elementary set operations on subsets of N in constant time. This assumption is reasonable, as in the typical usable range of this algorithm, n is not vastly larger than the machine word size. We compute the time complexity of the algorithm separately for each memoized function as the product of the number of possible arguments and the time complexity of the function under the assumption that calls to memoized functions run in $O(1)$ time (as we can use a hash table to implement the memoization tables).

For PRESOLVE, there are $O(n^{k+2})$ possible arguments (C, R), as there are $O(n^{k+1})$ ways to choose a set $C \subseteq N$ with $|C| \leq k+1$, and R must be one of the $O(n)$ components in $H[N \setminus C]$. As the time complexity of a single call (excluding the calls to the memoized functions) is $O(nk)$, the total time complexity for PRESOLVE is $O(n^{k+3}k)$. For EXTRACTCOMPONENT, there are $O(n^{k+2})$ possible arguments (C, r_0), and as the time complexity of a single call is $O(n^2)$, the total time complexity is $O(n^{k+4})$, which also subsumes the total time complexity of PRESOLVE.

The independence oracle INDTEST is only called for arguments (u, S, v) with $|S| \leq k + 1$, and thus the number of independence queries is bounded by $O(n^{k+3})$. The time complexity of each independence query depends on the way the oracle is implemented. However, time complexity does not depend on n and it is upper bounded by the number of samples and thus increasing k increases time complexity only for small values of k. Thus, asymptotically we can treat time complexity of a conditional independence test as a constant.

In the form presented in Algorithm 1, the algorithm only answers the question on whether $k+1$ cops can always win the cops-and-a-robber game, or equivalently, whether H has treewidth at most k. The algorithm can, however, be extended to produce a tree decomposition of width k as a byproduct in the affirmative case. The tree decomposition is obtained from the part of the recursion tree of the algorithm that causes the true return value; more exactly, from all SOLVE calls, we need to remove all but the first PRESOLVE call that returns true. This recursion tree is converted into a tree decomposition by converting each PRESOLVE(C, R) call into a bag containing nodes C.

The tree decomposition constructed this way covers all the nodes, because in the root PRESOLVE call we have $C = \{v_0\}$ and $R = N \setminus \{v_0\}$, and in the PRESOLVE calls under it, R decreases only by partitioning or adding a removed node to the set of cops C. The tree decomposition satisfies the running intersection property, because for each PRESOLVE(C, R) call corresponding to a bag with nodes C, the bags in the two subtrees under it contain only nodes from C and one of the partitions R' and $R \setminus R'$ of R, and the nodes in R cannot appear in bags outside these subtrees. Because $|C| \leq k+1$ in all the PRESOLVE calls, the width of the tree decomposition is k.

This extension of constructing the tree decomposition may be implemented either by making the functions in Algorithm 1 return a pointer to the constructed tree decomposition subtree, or by a separate postprocessing step which uses the memoization tables to trace the part of the recursion tree that causes the true return value. Either way, the extension does not affect the time complexity, and thus we obtain the following result.

Theorem 4 *A tree decomposition of optimal width k can be found for the moral graph H in $O(n^{k+4})$ time and $O(n^{k+3})$ calls to the independence oracle.*

3.2 Postprocessing

From the algorithm of Section 3.1, we obtain a tree decomposition of width k for H consisting of bags $X = \{X_1, \ldots, X_m\}$. By creating a graph in N where each bag X_i is a clique, we obtain a supergraph of H. We extract the skeleton of G from the supergraph by removing the extra edges based on the following observation: If nodes $u, v \in N$ are not connected by an edge in G, then u is not a parent or a descendant of v or vice versa (with u and v swapped). This means that by the local Markov property, u and v are d-separated by one of the parent sets A_u and A_v in G. For any $x \in N$, the set $A_x \cup \{x\}$ forms a clique in the moral graph H, and thus for some bag X_i it holds that $A_x \cup \{x\} \subseteq X_i$. Therefore we can check whether we should remove the edge between u and v by running INDTEST(u, S, v) for all $X_i \in X$ containing u or v and all subsets $S \subseteq X_i \setminus \{a, b\}$; the edge should be removed if at least one query returns true.

After recovering the skeleton of G, we still need to orient some of its edges to construct the CPDAG. We do this similarly to the PC algorithm (Spirtes et al., 2000). First, to orient the v-structures of the graph, we consider each edge $\{u, v\}$ removed in the previous phase. Let $S \subseteq N$ be the conditioning set that caused the removal, that is, INDTEST(u, S, v) returned true. For all nodes x that are adjacent to both u and v in the skeleton but are not in S, we orient the edges to the configuration $u \rightarrow x \leftarrow v$. After orienting the v-structures, we complete the orientation by using the Meek rules (Meek, 1995).

A graph whose treewidth is k has less than nk edges. Thus, the supergraph has $O(nk)$ edges, and for each of them, we make $O(n2^k)$ independence queries because $|X| = O(n)$ and $|X_i| \leq k+1$ for all $X_i \in X$. The total time complexity of extracting the skeleton of G is thus $O(n^2 2^k k)$. The time complexity of the orientation phase can be loosely bounded by $O(n^5)$. In the whole postprocessing phase, we only perform independence queries $\textsc{IndTest}(u, S, v)$ with $|S| \leq k$, and there are only $O(n^{k+2})$ of them. Thus the time and independence query complexities of the postprocessing phase are dominated by those of tree decomposition finding, and hence by Theorem 4 we obtain the following result.

Theorem 5 *The CPDAG of G can be learned in $O(n^{k+4})$ time and $O(n^{k+3})$ queries to the independence oracle, where k is the treewidth of the moral graph H.*

4. Experiments

While the main focus of this paper is in the theoretical results, we also assessed the practical performance of the algorithm experimentally to validate the correctness of the algorithm and offer pointers for further research. We made a C++ implementation[3] of our algorithm, along with the PC algorithm (Spirtes et al., 2000) that was used as a baseline for comparisons. We used the discrete Bayesian networks in the *bnlearn* (Scutari, 2010) repository[4] as benchmark instances.

Prior to running the other experiments, we verified that both the algorithm and the implementation are correct by checking that the implementation gives the correct result in a large number of randomly generated Bayesian network structure DAGs of sizes at most 32 and varying densities when the independence oracle is exact (given by d-separation in the DAG). We also checked that the treewidth of the moral graph found by the algorithm matched that given by an external treewidth solver (Tamaki, 2019).

4.1 Number of Independence Tests

As the sizes of the conditioning sets in the conditional independence queries increase, more data is required to answer the queries accurately. For the performance of the structure learning algorithm, the number of independence queries is also an important factor. Thus, we perform an experiment where we measure the number of distinct independence tests of each conditioning set size our algorithm and the PC algorithm uses on the benchmark instances. In this experiment, we use the exact oracle, which means that both algorithms find the correct CPDAG (we also verified this). Neither of the algorithms was given any external information such as the maximum size of the conditioning sets.

Figure 1 shows the results of the experiment. In the larger instances, the number of queries is in billions, and to keep the computational load reasonable, we limited the experiment to benchmark networks with at most 128 nodes. Our algorithm typically performs more independence queries than the PC algorithm, but in some instances (such as child, hailfinder and hepar2), the PC algorithm needs much larger conditional independence tests than our algorithm as the treewidth is larger than the maximum degree, while in some instances (such as water), the opposite happens.

3. https://github.com/ttalvitie/learning-bns-with-cops-and-robbers
4. https://www.bnlearn.com/bnrepository/

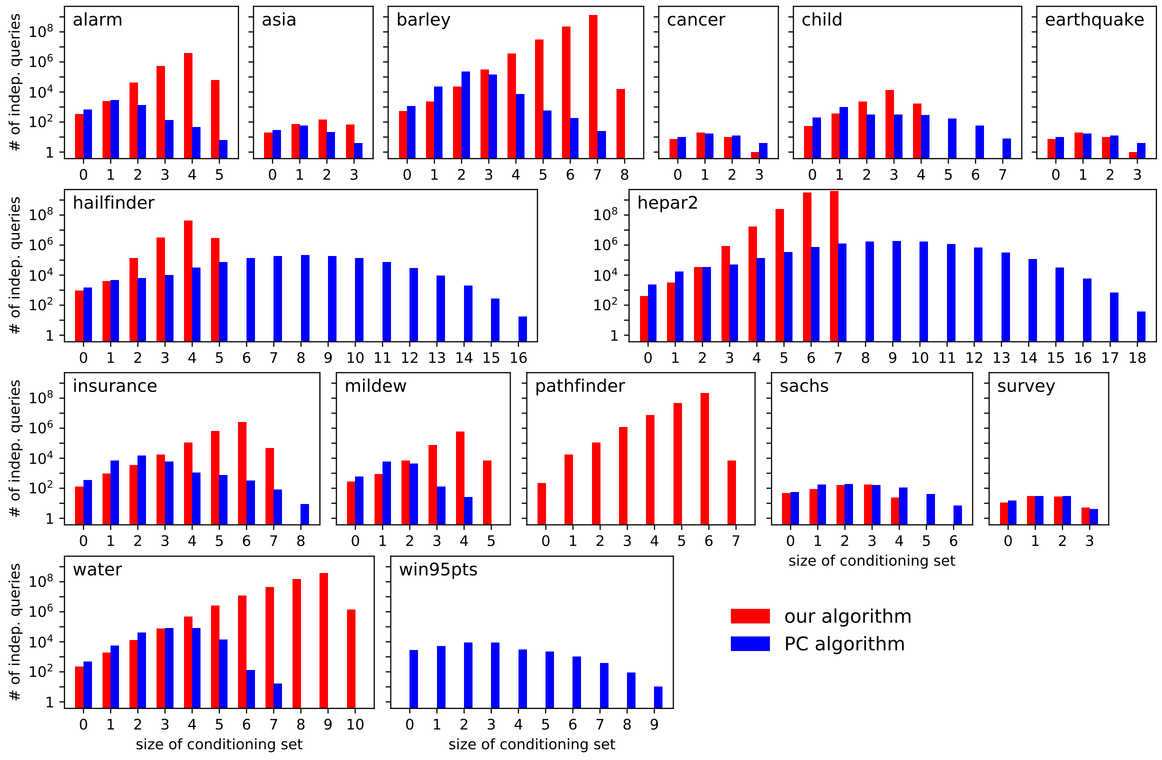

Figure 1: The number of independence tests of each conditioning set size our algorithm and the PC algorithm uses in learning the benchmark networks with the exact oracle. Note that the number of independence tests is shown on logarithmic scale. The results for one of the algorithms is missing for pathfinder and win95pts; this caused by the algorithm exceeding the 12 hour time limit. The last nonzero bar is always at treewidth plus one for our algorithm and maximum degree minus one for the PC algorithm. Note that because the counts are shown on log-scale and the differences are large, the height of the highest bar gives a reasonable estimate for the order-of-magnitude of the total number of tests.

4.2 Structure Learning Accuracy on Generated Data

In real life, we do not have access to a perfect independence oracle. Thus, the previous experiment does not show the behavior of the algorithm in the more realistic case that the independence tests give sometimes incorrect results due to limited data. Therefore, for our second experiment, we generate data from the benchmark network and use it to learn back the Bayesian network using statistical independence tests (Pearson's χ^2-test with a significance level 0.05) for the independence queries. Then we measure how close the learned network is from the generating network using the structural Hamming distance (SHD).

To make sure that our algorithm terminates without cyclic recursion even when the independence tests give erroneous results, we need to enforce the semi-invariant that the set of robbers R decreases on recursive calls. We do this by setting $R' \leftarrow R' \cap R$ after obtaining the component R' from EXTRACTCOMPONENT in the implementation of the PRESOLVE function in Algorithm 1.

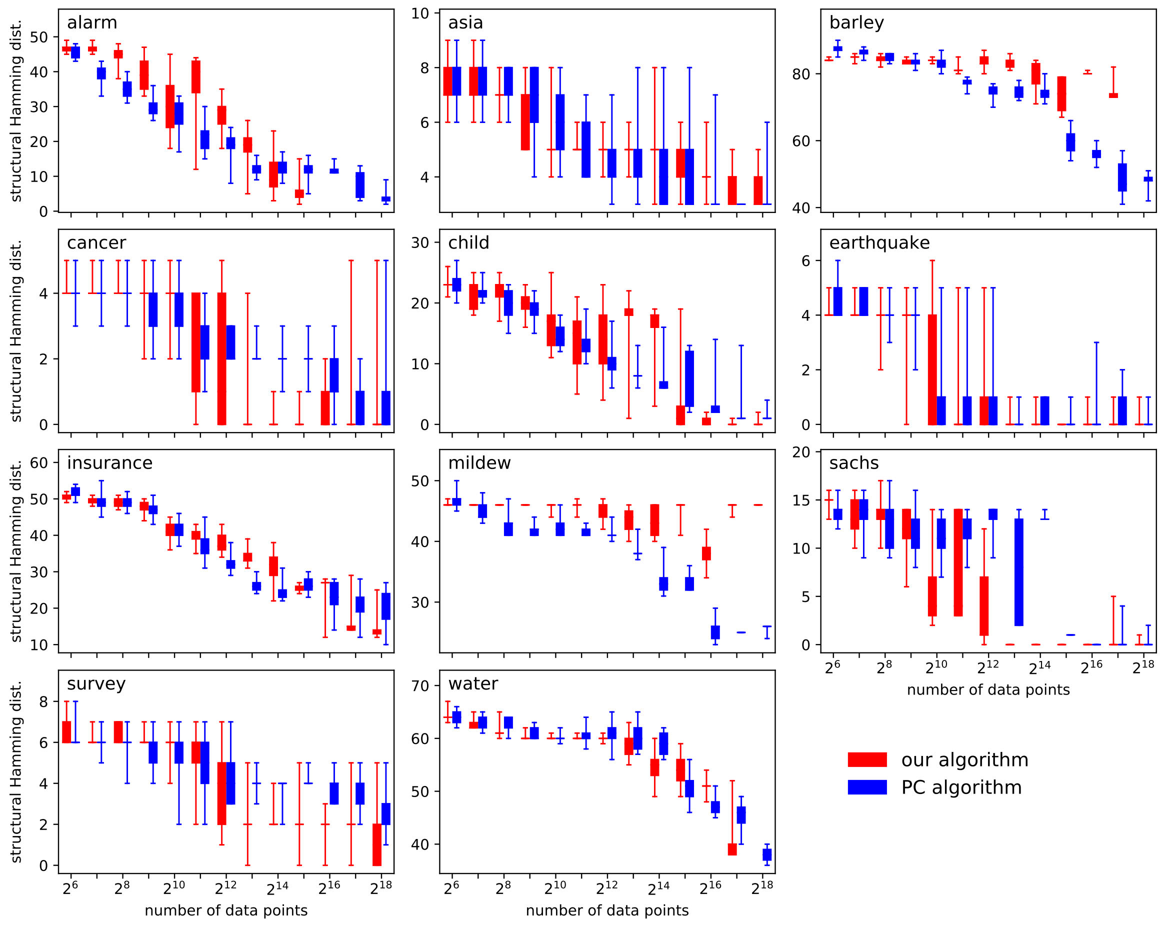

Figure 2: The structural Hamming distance of the learned CPDAG compared to the CPDAG of the generating network as a function of the number of data points. For each network and number of data points, 25 datasets were generated. In the box plot, the box shows the area between the lower and upper quartiles of the resulting SHD values, and the whiskers show the minimum and maximum over all the 25 repetitions. Some boxes are missing, because the algorithm exceeded the 6 hour time limit.

Figure 2 shows the SHD between the learned network and the correct network as a function of the number of data points. As the independence tests are computationally intensive for large datasets, we limited the experiment to networks with at most 50 nodes. For the question on which algorithm learns the network more accurately, the results of this experiment are mixed. In some cases, the high number of independence queries used by our algorithm makes it susceptible to learning too sparse networks, as we get more false positives. For instance, in mildew and barley, the algorithm learns almost empty networks even with hundreds of thousands (i.e., 2^{17} and 2^{18}) of data points. In alarm, as we increase the number of data points from 1024 to 2048, the maximum conditioning set size typically increases from 3 to 4, which in turn causes the accuracy of the learned netwoks to deteriorate because of the higher inaccuracy in the independence tests.

5. Discussion

To our knowledge, our algorithm is the most efficient Bayesian network structure learning algorithm with respect to the treewidth of the data-generating distribution. While our algorithm is interesting from the theoretical point of view, it is not a great choice for a general purpose learning algorithm in practice. While it may be competitive when the underlying Bayesian network has very low treewidth, the algorithm becomes quickly inefficient when treewidth increases. As our experiments showed, this happens often with typical Bayesian networks. The main source of inefficiency is that in high treewidth distributions the algorithm has to try lots of conditioning sets unsuccessfully before it finds separators.

References

C. F. Aliferis, A. Statnikov, I. Tsamardinos, S. Mani, and X. D. Koutsoukos. Local causal and Markov blanket induction for causal discovery and feature selection for classification part I: Algorithms and empirical evaluation. *Journal of Machine Learning Research*, 11:171–234, 2010.

A. Chechetka and C. Guestrin. Efficient principled learning of thin junction trees. In *Advances in Neural Information Processing Systems (NIPS)*, 2007.

G. Elidan and S. Gould. Learning bounded treewidth Bayesian networks. *Journal of Machine Learning Research*, 9:2699–2731, 2008.

J. Korhonen and P. Parviainen. Exact learning of bounded tree-width Bayesian networks. In *Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics*, volume 31 of *Proceedings of Machine Learning Research*, pages 370–378, 2013.

C. Meek. Causal inference and causal explanation with background knowledge. In *UAI'95: Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 403–410, 1995.

M. Narasimhan and J. Bilmes. PAC-learning bounded tree-width graphical models. In *UAI'04: Proceedings of the Twentieth Conference on Uncertainty in Artificial Intelligence*, pages 410–417, 2004.

M. Scutari. Learning Bayesian networks with the bnlearn R package. *Journal of Statistical Software*, 35(3):1–22, 2010.

P. D. Seymour and R. Thomas. Graph searching and a min-max theorem for tree-width. *Journal of Combinatorial Theory, Series B*, 58(1):22–33, 1993.

P. Spirtes, C. N. Glymour, and R. Scheines. *Causation, Prediction, and Search.* MIT Press, 2000.

H. Tamaki. Positive-instance driven dynamic programming for treewidth. *Journal of Combinatorial Optimization*, 37(4):1283–1311, 2019.

X. Xie and Z. Geng. A recursive method for structural learning of directed acyclic graphs. *Journal of Machine Learning Research*, 9:459–483, 2008.