Learning Bayesian Networks with Cops and Robbers

Topi Talvitie, Pekka Parviainen (2020) Proceedings of the 10th International Conference on Probabilistic Graphical Models

Presented by: Hajar Lachheb & Clara Rivadulla Duró

Table of contents

01

Introduction

Introduction

Bayesian Networks: representations of joint probability distributions.

Structure: directed acyclic graph (DAG) which expresses conditional independencies in the distribution.

Parameters: specify conditional distributions.

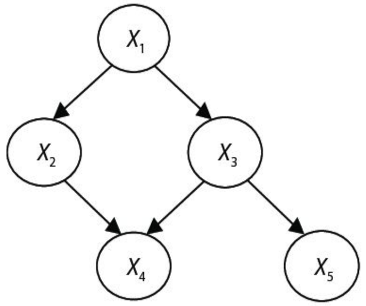
Bayesian Networks are often learned from **data**.

There are **2 main approaches**:

- Score-based
- **Constraint-based** $\longrightarrow$ Main Focus

Bayesian Networks

A Bayesian Network Structure



B Parameters

$p(X_1 = 0) = .20$
$p(X_2 = 0 \mid X_1 = 0) = .80$
$p(X_2 = 0 \mid X_1 = 1) = .80$
$p(X_3 = 0 \mid X_1 = 0) = .20$
$p(X_3 = 0 \mid X_1 = 1) = .05$
$p(X_4 = 0 \mid X_2 = 0, X_3 = 0) = .80$
$p(X_4 = 0 \mid X_2 = 1, X_3 = 0) = .80$
$p(X_4 = 0 \mid X_2 = 0, X_3 = 1) = .80$
$p(X_4 = 0 \mid X_2 = 1, X_3 = 1) = .05$
$p(X_5 = 0 \mid X_3 = 0) = .80$
$p(X_5 = 0 \mid X_3 = 1) = .40$

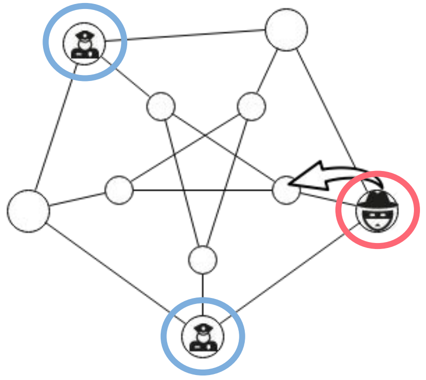
Figure 1: Example of structures and parameters in a BN.

Introduction

In this paper, parameterized complexity with respect to the **treewidth** of the data-generating Bayesian Network is studied. But what do we mean by **treewidth**?

Treewidth: property of a graph that measures how close the graph is to a tree. **Inference** in Bayesian Networks is tractable if the structure of the Bayesian Network has **low treewidth**.

- ➔ Inference in Bayesian Networks is **tractable** if the structure has **low treewidth**.
- ➔ 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.**



The challenge is that the graph is **unknown** and cops have to conduct **conditional independence tests** to gain **information** about the graph.

- ➔ We try to find **separators** that partition the graph into disjoint subgraphs. Once we have found **one**, we know that there are no **arcs between the disjoint subgraphs** and we can recursively solve these smaller problems.

02

Related Work

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.

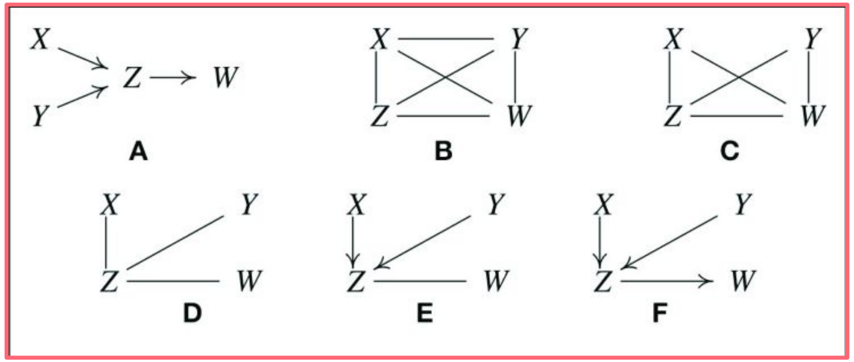


Figure 1: PC Algorithm taken from the article PC Algorithm (Spirtes et al., 2000).

Related Work

➔ **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.

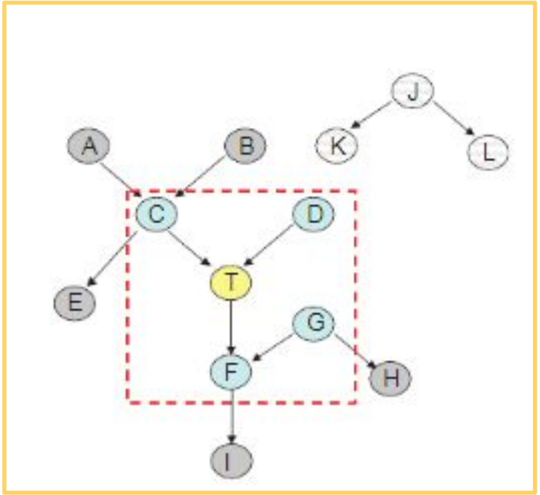


Figure 2: Example of the Local Learning Approach.

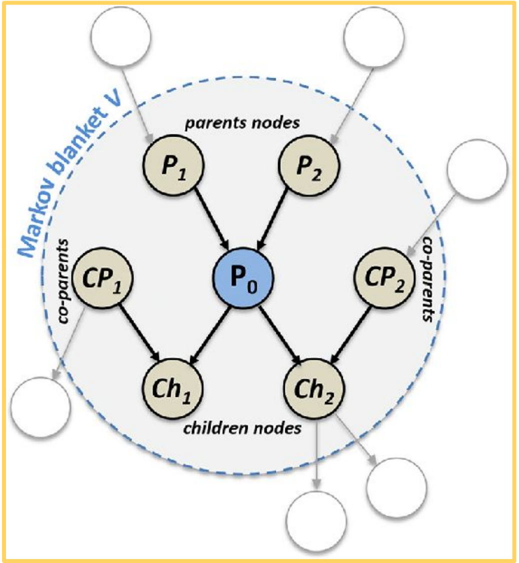
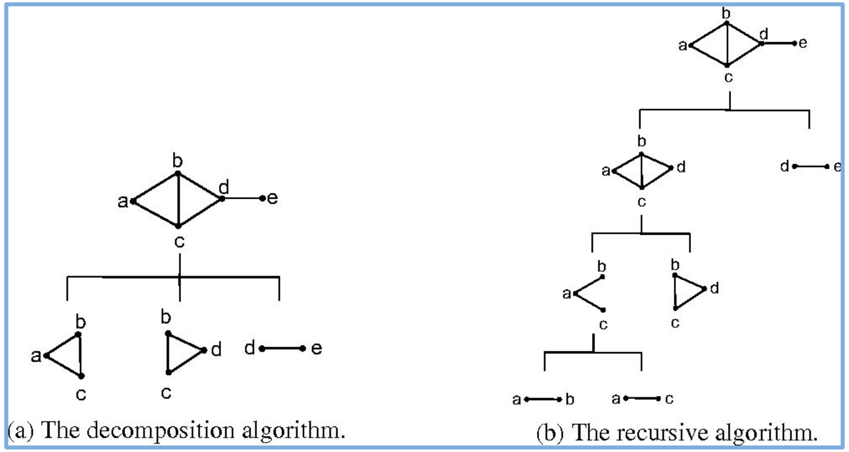


Figure 3: Example of the Markov Blankets used in the Local Learning.

Related Work

The algorithm we'll work on tries to find separators and divide nodes to smaller sets that can be solved separately.

➔ A similar algorithm to the one we'll use is the **recursive method** presented by Xie and Geng (2008).



a
b
c
d
e
(a) The decomposition algorithm.
(b) The recursive algorithm.

Figure 4: Example of the recursive algorithm.

➔ Another similar approach are **PAC-learning** graphical models with low treewidth (Chechetka and Guestrin, 2007; Narasimhan and Bilmes, 2004). The algorithm finds a tree decomposition in time **O(n2k+3).**

Related Work

Comparative Table of all related works

Method	Approach	Main Idea	Complexity	Similarity to the algorithm
PC algorithm and variants	Remove edges by conditioning	Start with a complete network and remove edges	Varies based on the implementation	No
Local learning	Find neighbors/markov blankets	Construct DAG based on local information	Varies based on the implementation	No
Recursive method	Find separators and divide	Divide nodes into smaller solvable sets	Varies based on the implementation	**Yes**
PAC-learning	Learn graphical models	Learn models with low treewidth	O(n^2k+3)	**Yes**
Score-based methods	Find highest-scoring network	Find best network within bounded treewidth	Varies based on the implementation	No

03

Definitions

Bayesian Networks

We start off by considering **G = (N, A)** as a directed acyclic graph where N is the node set and A is the arc set. Av are the parents of the node v in G.

➔ **We use the notation G[X] for the subgraph of G induced by X ⊆ 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

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

If the distribution can be **factorized** $\longrightarrow$ it can be represented by a **Bayesian network** whose structure is G

➔ 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 |Av)**.

How can we know if random variables are conditionally Independent ?

Two 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 this notation u ⊥ v | S when they are conditionally independent.**

Bayesian Networks

What do we mean by a collider?
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 :

1. There is a **collider** on the path such that neither the collider or any of its descendants is in the conditioning set S.

2. 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.

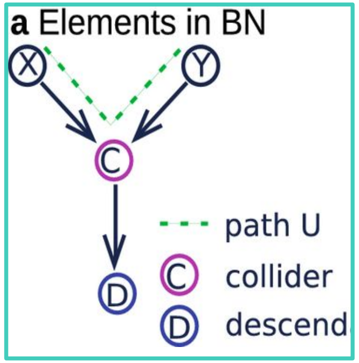


Figure 5: Collider inside of a graph.

Every Bayesian network satisfies the local Markov property: a node is conditionally independent of its non-descendants given its parents.

Bayesian Networks

Now we will understand why we needed to make sure we define everything beforehand.

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.

BUT to guarantee that conditional independence implies d-separation, we have to make **assumptions**.

One of them is called **Faithfulness**.

Faithfulness

A **distribution P** is **faithful** to a **DAG G** if **conditional independence v ⊥ u|S** in **P** implies that **v and u are d-separated** by S in G.

Two DAGs are **Markov equivalent** if they have the same **skeleton** and the **same set of v-structures.**

➔ **Skeleton :** an undirected graph that is obtained from the DAG by **removing the directions of the arcs.**

➔ **v-structures :** a collider structure **u** → **w** ← **v** such that 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.**

Treewidth

Treewidth of graph H is defined as the **smallest width** overall tree **decompositions** of H.

But what is a tree decomposition ? Can we consider a subgraph as a decomposition ?

- Let **H = (N, E)** be a **undirected graph** where **N is the node (vertex) set** and **E is the edge set**.
- Let **X = {X1, . . . , Xm}** be a **collection of subsets of N**.
- Let **T be a tree** whose **vertex set is X**; we call the elements of **X bags**.

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, **∪iXi = N**
2. For **each edge {u, v} ∈ E** there exists **a bag Xi** such that both **u ∈ Xi** and **v ∈ Xi**
3. Running intersection property: For **every node v ∈ N**, the **subtree of T** induced by the bags containing v is connected.

What do we mean by width ?

The **width** of a tree decomposition is the size of the **largest bag** minus one.

Cops and a robber game

To understand the importance of the the width, we will explain the logic behind the cops and a robber game.

In the game, we have **k+1 cops** and **1 robber**.

- ★ The robber moves fast from a node to an another using graph edges and it's **faster** than the cops.
- ★ At any moment, the cop either occupies one node or is "in the air" with a helicopter. **(In the air = Amount of time moving from a node to an another)**
- ★ The cop capture the robber if he lands in the node where the robber is.
- ★ Cops can move from a node to an another, robber moves from an edge to an another.
- ★ Robber cannot move to a node where the cop is, of course to not be captured.
- ★ Cop land slowly on a node, this give the robber time to avoid going to the node where the cop is actually.

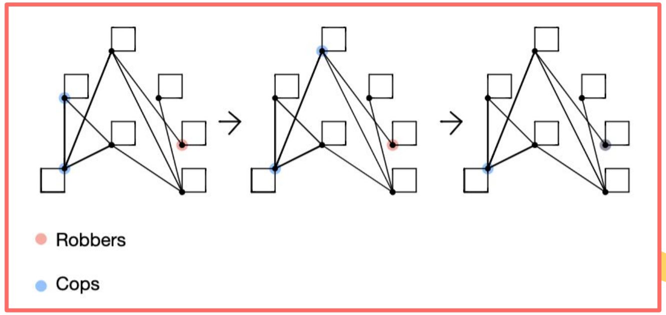


Figure 6: Robber and cops simulation.

Treewidth of a Bayesian Network

The previous definition is applied to undirected graphs, the structure of the Bayesian Network is a directed graph. Do not stress, we can still apply it with slightly new changes.

The **treewidth** of a Bayesian network is defined to be the **treewidth** of the **moral graph** of its **structure**.

But what's a moral graph then?

Let the **moral graph** of a **DAG G = (N, A)** be an undirected graph **H = (N, E)** such that **an edge {u, v} ∈ E** if and only if **uv ∈ A**, **vu ∈ A** or there exists **w ∈ N** such that **uw ∈ A** and **vw ∈ A**.

What are we going to use in our algorithm ?

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

After setting all the notions and definitions, it's time to define and explain the algorithm used in our paper.

04

Algorithm

Algorithm

The algorithm for learning the **structure *G*** of the Bayesian network consists of **two phases**:

➔ *Phase 1*: **Learning the tree decomposition**

This phase consists of learning a tree decomposition of treewidth k of the moral graph H. We do this by

- Successively increase k starting from 1, and solve whether $k + 1$ cops can always catch one robber on H using independence queries with conditioning set size at most $k + 1$.
- If they can, k is indeed the treewidth of H and we obtain the tree decomposition of width k as a byproduct.
- Otherwise, k is too small and we need to repeat the process with $k + 1$.

➔ *Phase 2*: **Postprocessing**

- Prune edges from the supergraph induced by the tree decomposition to obtain the skeleton of G.
- Orient edges to obtain the CPDAG of G.

Phase 1

How do we know if *k + 1* cops can always catch the robber?

- We use a recursive dynamic programming algorithm that keeps track of the set of nodes containing cops, ***C***, and the set nodes in which the robber can move, ***R***.
- To limit the possible states we need to consider, we simplify the game and assume that:
  - ★ C is exactly the boundary ∂R of the set R (nodes in $N \setminus R$ that are adjacent in H to at least one node in R).
  - ★ Cops never retreat, which means that they always move a 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.
  - ★ Initially, there's only one cop on the graph in a randomly selected node $v_0 \in N$ and the robber is somewhere in $N \setminus \{v_0\}$.
- None of these changes alter the outcome.

Phase 1

To implement the algorithm, they define **three functions**. The first two functions are given a pair *(C, R)* of sets of nodes, and they return a Boolean value on whether the cops occupying nodes *C* can win.

➔ ***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 $C' = \partial R'$.

➔ ***Solve***

- The arguments must always be in the minimal from: *H[R]* is a non-empty connected subgraph of *H*, and $C = \partial R$.
- It 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.

➔ ***ExtractComponent***

The algorithm is started by calling *PreSolve({v0}, N \ {v0})*; if the return value is true, the cops win. The recursion between *PreSolve* and *Solve* eventually reaches the base case $R = \varnothing$ in *PreSolve*. To avoid recomputation, results from the first two functions are kept in a table along with the independence oracle *IndTest*.

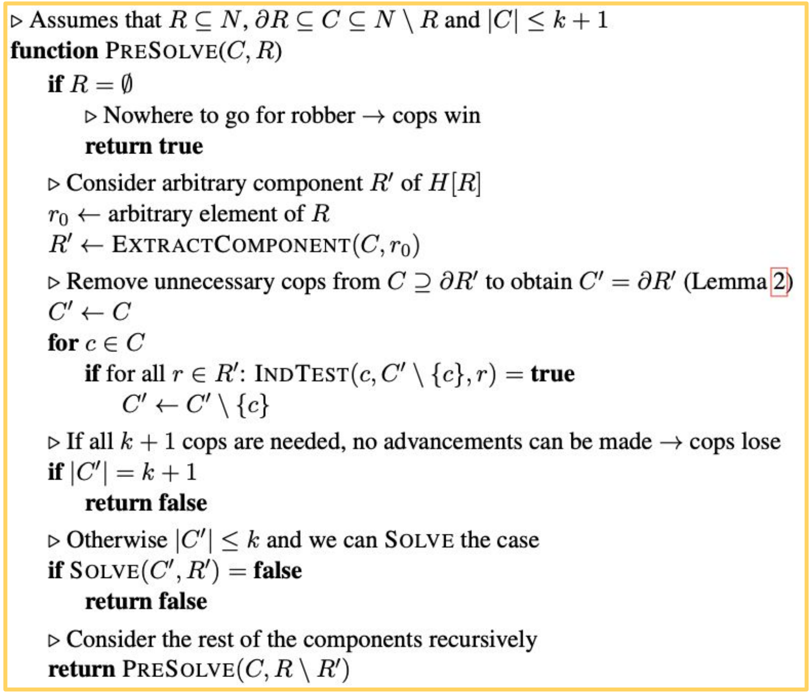
Phase 1

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 $u \perp v \mid S$, or if *u* and *v* are d-separated by *S* in *G*. To prove that *IndTest* works correctly, we need **two lemmas**:

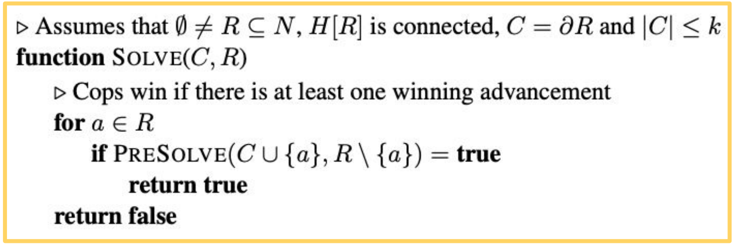
If $R \subseteq N$ and $\partial R \subseteq C \subseteq N \setminus R$, then for all $c \in C$ it holds that $c \in \partial R$ if and only if there exists $r \in R$ such that INDTEST$(c, C \setminus \{c\}, r)$ returns false.

Let $C \subseteq N$, and define graph $IC = (N \setminus C, EC)$ by $EC = \{\{u,v\} : u,v \in N, u \neq v \text{ and } INDTEST(u,C,v) = false\}$.

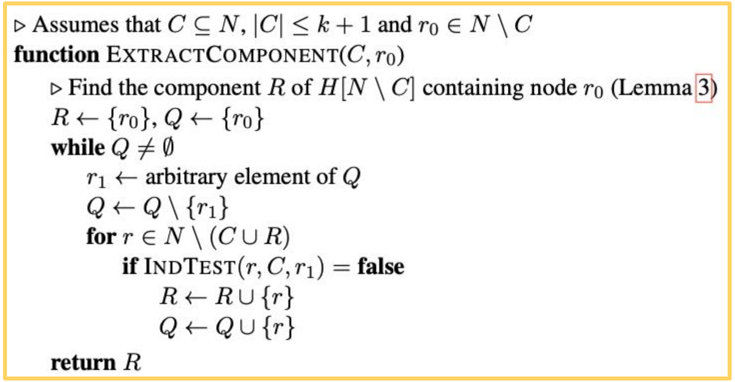
Phase 1



```
▷ 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 $R$
```

Figure 7: Pseudocode of the algorithm.

Phase 1

The algorithm explained until now only tells us whether *k + 1* cops can always win the game, or whether *H* has a treewidth of at most *k*. Now, how do we obtain a **tree decomposition** of width *k*?

➔ The tree decomposition is obtained from the part of the recursion tree of the algorithm that returns true.

➔ From all the *Solve* calls, remove all but the first call to *PreSolve* that returns true.

➔ This recursion tree is converted into a tree decomposition by converting each *PreSolve(C,R)* call into a bag with *C* nodes.

Phase 2

- From the previous algorithm, we get a tree decomposition of width k for H consisting of bags $X = \{X_1,...,X_m\}$. 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 extra edges**: 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. 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 so for some bag X_i, $A_x \cup \{x\} \subseteq X_i$. We can check if we can 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 $S \subseteq X_i \setminus \{a, b\}$. If at least one query returns true, the edge must be removed.
- However, we still need to **orient some edges** of the skeleton of G to **construct the CPDAG** similarly to the PC algorithm.
    1. 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. 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 $u \rightarrow x \leftarrow v$.
    2. After orienting the v-structures, we complete the orientation by using the **Meek rules** (Meek, 1995).

The CPDAG of G can be learned in O(nk+4) time and O(nk+3) queries to the independence oracle, where k is the treewidth of the moral graph H.

05 Experiments

Experiments

Researchers of this paper made a **C++ implementation** to validate the correctness of the algorithm. They also used the **PC algorithm** as a baseline for comparison, along with discrete BNs in the *bnlearn* repository as benchmark instances. They previously checked that the treewidth of the moral graph found by the algorithm matched one given by an external treewidth solver (*Tamaki, 2019*).

They performed **two experiments**:

➔ *Experiment 1*

- Measure the **number of distinct independence tests** of each conditioning set size the algorithm and the PC algorithms use on the benchmark instances.
- Use the **exact oracle**, which means that both algorithms find the correct CPDAG.
- Neither of the algorithms were given any external information.

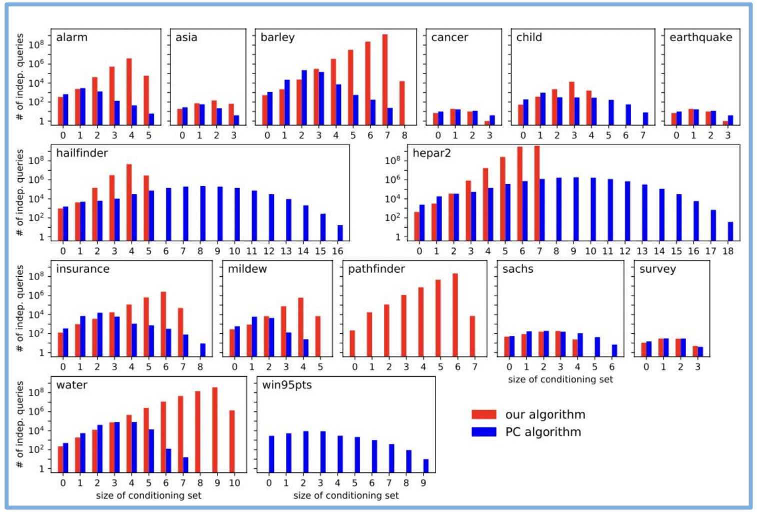


Figure 8: Number of independence tests of each conditioning set size of the algorithm and the PC algorithm.

Experiments

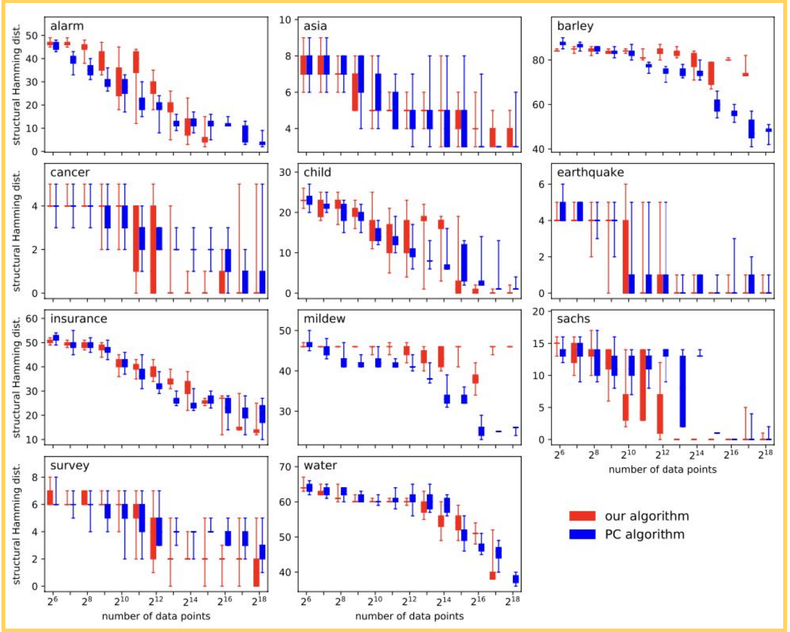


Figure 9: Structural Hamming distance of the learned CPDAG compared to the CPDAG of the generating network as a function of the number of data points.

➔ *Experiment 2*

- **Generate data from the benchmark network** and use it to **learn back the Bayesian network** using statistical independence tests (Pearson's χ2-test with a significance level 0.05) for the independence queries.
- Measure how close the learned network is from the generating network using the **structural Hamming distance** (SHD).

06

Conclusions

Conclusions

- This algorithm is the most efficient BN structure learning algorithm with respect to the treewidth of the data-generating distribution.
- However, it’s not a good choice for a general purpose learning algorithm in practice.
- The algorithm is inefficient when the treewidth increases, because in high treewidth distribution it has to try lots of conditioning sets unsuccessfully before it finds separators.

THE END

Thank you so much for your attention!