Estimating High-Dimensional Directed Acyclic Graphs with the PC-Algorithm

Theresa Meier, 13.01.2023

Seminar Modeling under Dependence

How to find conditional independence (CI) relationships in a DAG?

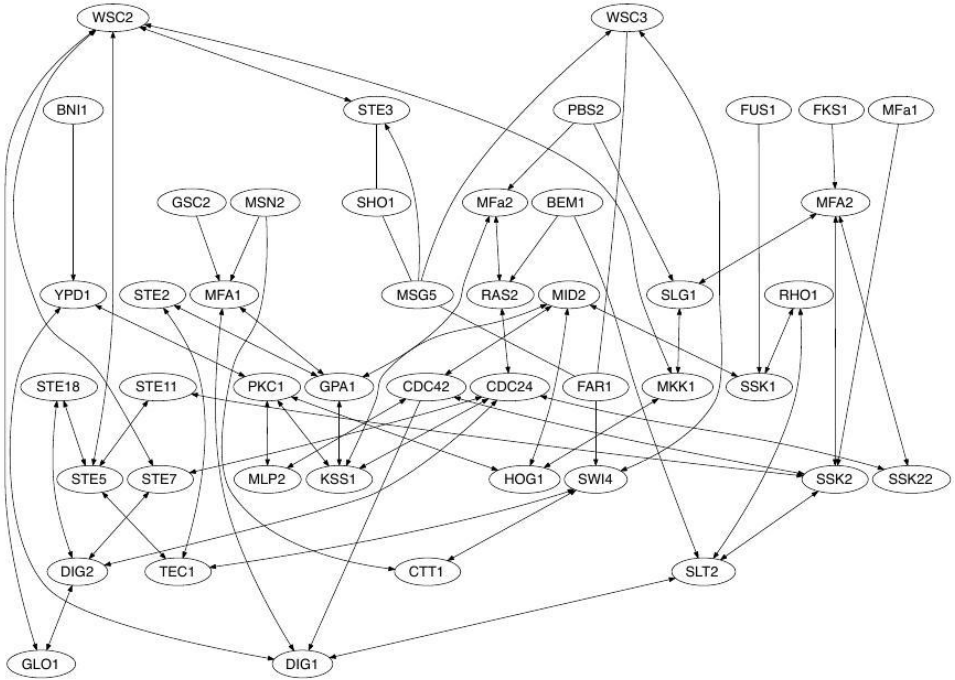


Figure 1: Output of the rank PC-algorithm for yeast gene expression data (Harris and Drten, 2013).

Agenda

1) Basic Definitions
2) Introduction to the PC-Algorithm
3) Limitations of the PC-Algorithm
4) The stable PC-Algorithm
5) The parallel PC-Algorithm
6) Simulation Study in R
7) Final Notes and Discussion

Agenda

1) **Basic Definitions**
2) Introduction to the PC-Algorithm
3) Limitations of the PC-Algorithm
4) The stable PC-Algorithm
5) The parallel PC-Algorithm
6) Simulation Study in R
7) Final Notes and Discussion

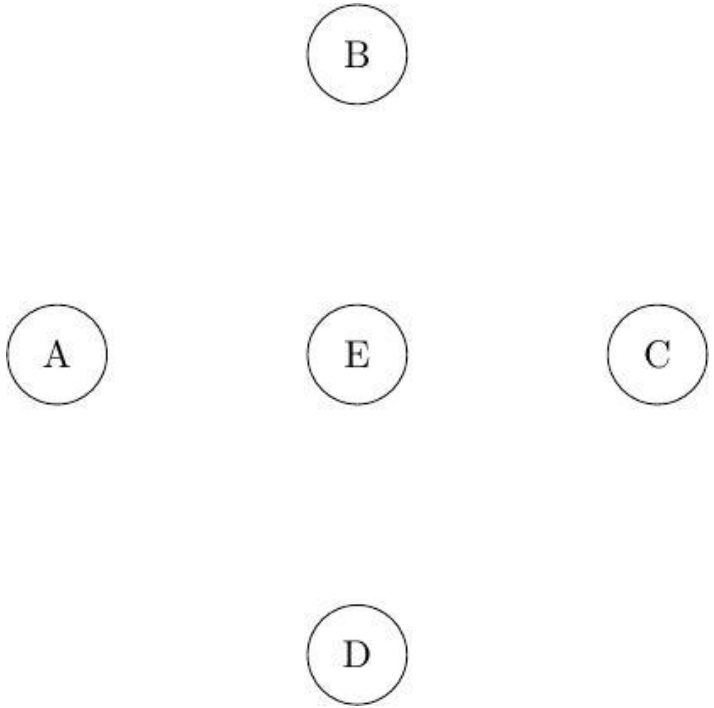
Directed Acyclic Graphs (DAGs)

All following definitions are taken from Harris and Drton (2013), Le et al. (2014) and Koller and Friedman (2009).

Directed Acyclic Graphs (DAGs)

Let $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ be a graph consisting of:

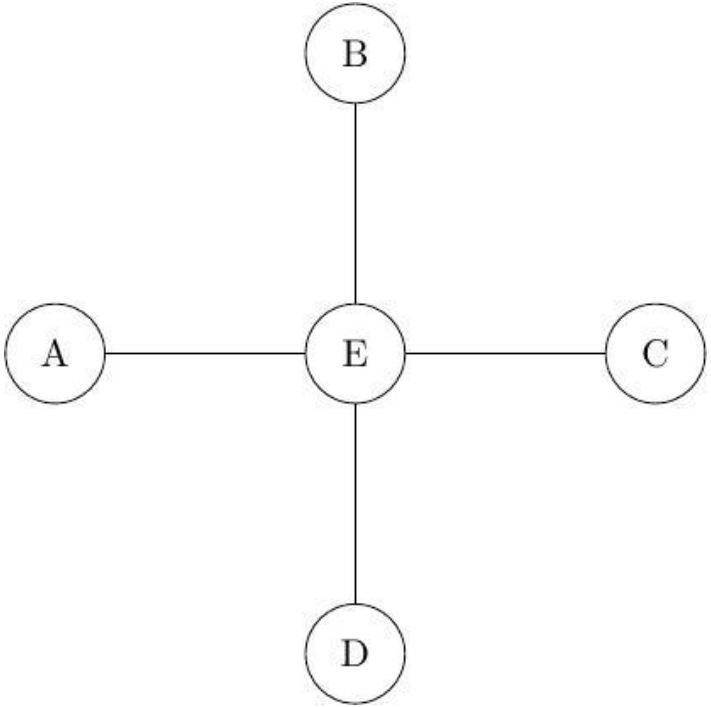
- set of nodes $\mathcal{N}$



Directed Acyclic Graphs (DAGs)

Let $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ be a graph consisting of:

- set of nodes $\mathcal{N}$
- set of edges $\mathcal{E} \subseteq \mathcal{N} \times \mathcal{N}$

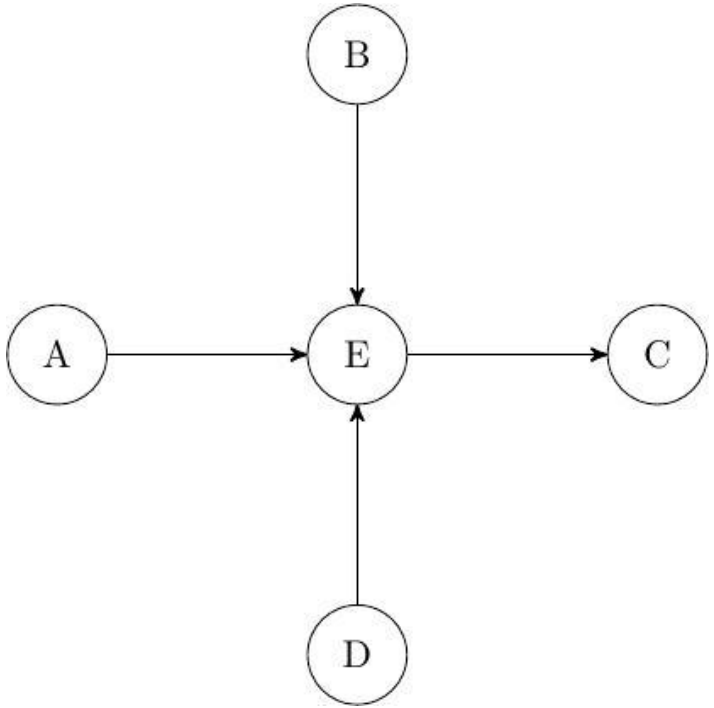


Directed Acyclic Graphs (DAGs)

Let $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ be a graph consisting of:

- set of nodes $\mathcal{N}$
- set of edges $\mathcal{E} \subseteq \mathcal{N} \times \mathcal{N}$

$\mathcal{G}$ is a **directed acyclic graph** (DAG) if $\mathcal{G}$ contains only directed edges and has no directed cycles.



Directed Acyclic Graphs (DAGs)

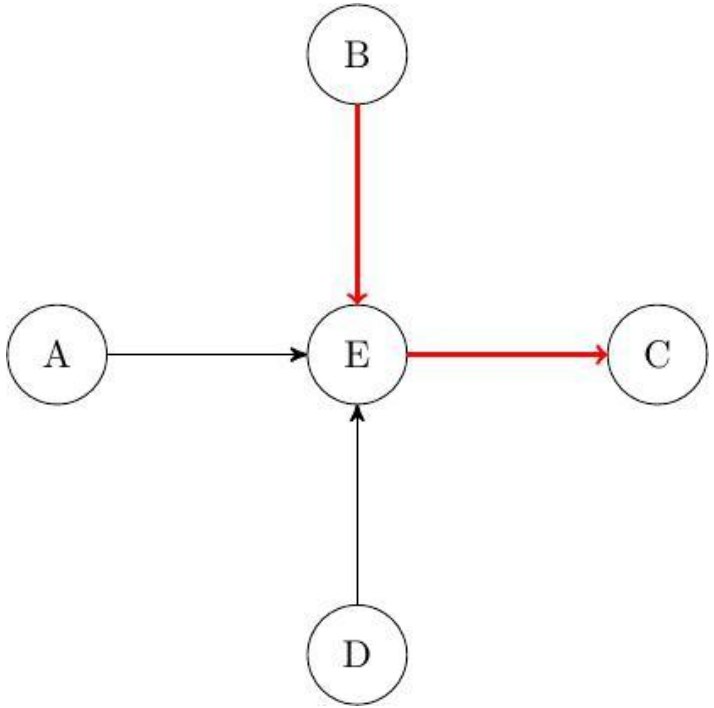
Let $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ be a graph consisting of:

- set of nodes $\mathcal{N}$
- set of edges $\mathcal{E} \subseteq \mathcal{N} \times \mathcal{N}$

$\mathcal{G}$ is a **directed acyclic graph** (DAG) if $\mathcal{G}$ contains only directed edges and has no directed cycles.

A **path** from node A_0 to node A_n in DAG $\mathcal{G}$ is a sequence of distinct nodes $(A_0, \ldots, A_n)$ such that for all $1 \leq k \leq n$ either

$$A_{k-1} \rightarrow A_k \in \mathcal{E} \text{ or } \quad A_k \rightarrow A_{k-1} \in \mathcal{E}.$$



Directed Acyclic Graphs (DAGs)

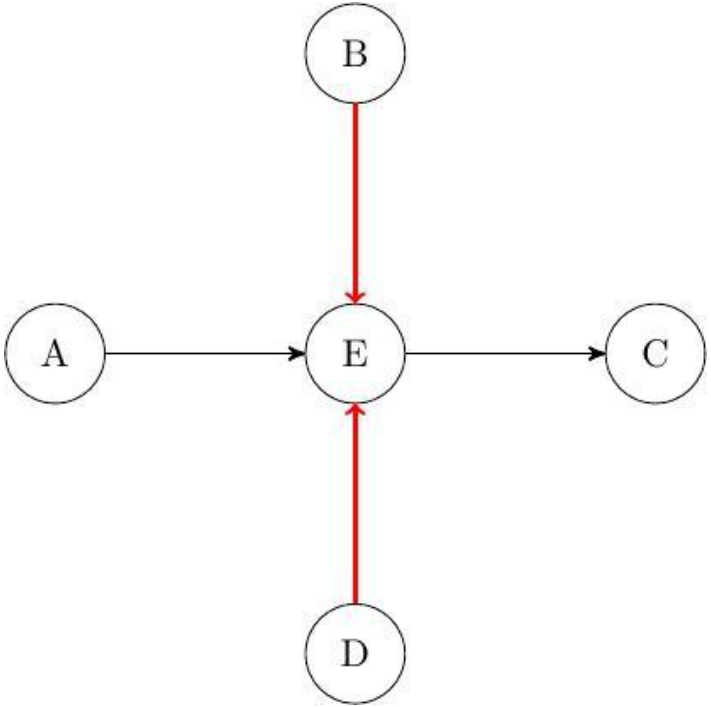
Let $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ be a graph consisting of:

- set of nodes $\mathcal{N}$
- set of edges $\mathcal{E} \subseteq \mathcal{N} \times \mathcal{N}$

$\mathcal{G}$ is a **directed acyclic graph** (DAG) if $\mathcal{G}$ contains only directed edges and has no directed cycles.

A **path** from node A_0 to node A_n in DAG $\mathcal{G}$ is a sequence of distinct nodes $(A_0, \ldots, A_n)$ such that for all $1 \leq k \leq n$ either

$$A_{k-1} \rightarrow A_k \in \mathcal{E} \text{ or } \quad A_k \rightarrow A_{k-1} \in \mathcal{E}.$$

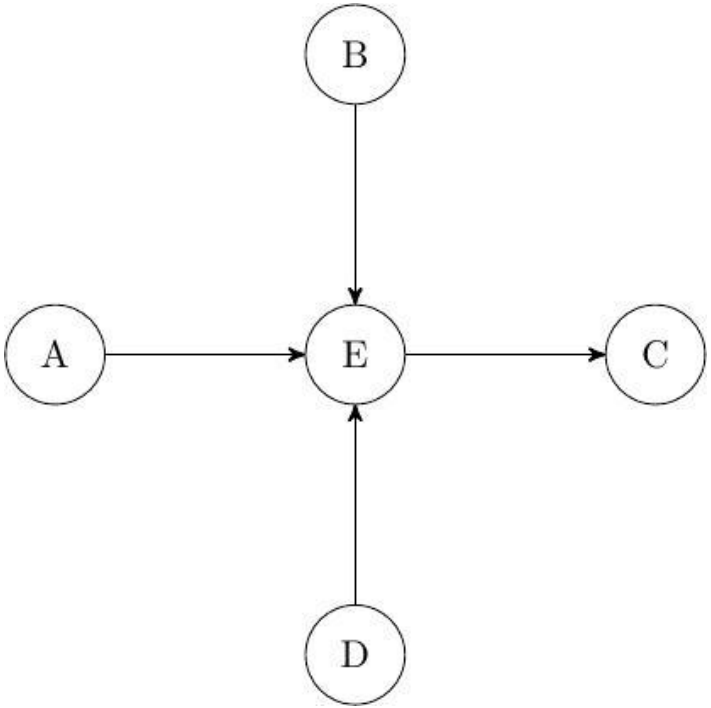


Directed Acyclic Graphs (DAGs)

A **collider** or **v-structure** is a triple of nodes $(A, B, C) \in \mathcal{M}^3$ that induces the subgraph

$$A \rightarrow B \leftarrow C$$

where A and C are not adjacent (i.e. there is no edge between A and C).

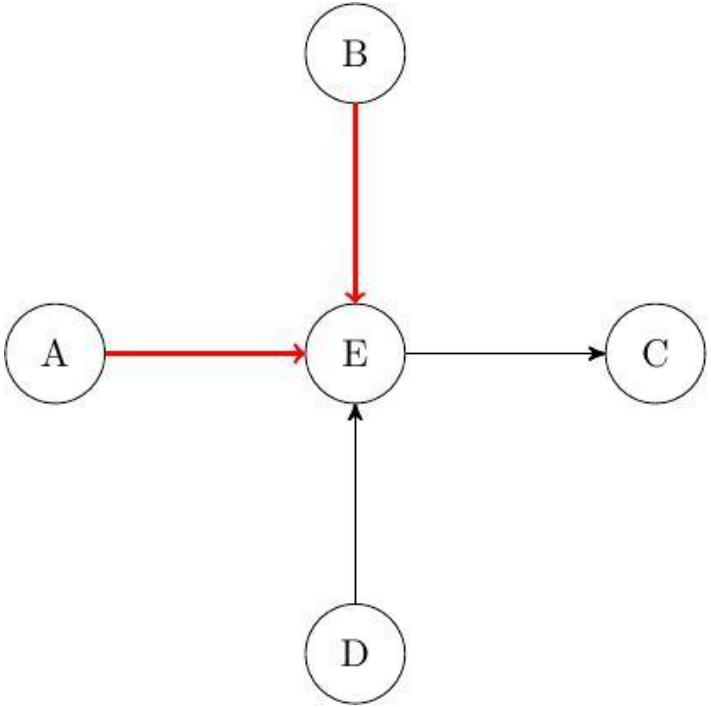


Directed Acyclic Graphs (DAGs)

A **collider** or **v-structure** is a triple of nodes $(A, B, C) \in \mathcal{N}^3$ that induces the subgraph

$$A \rightarrow B \leftarrow C$$

where A and C are not adjacent (i.e. there is no edge between A and C).



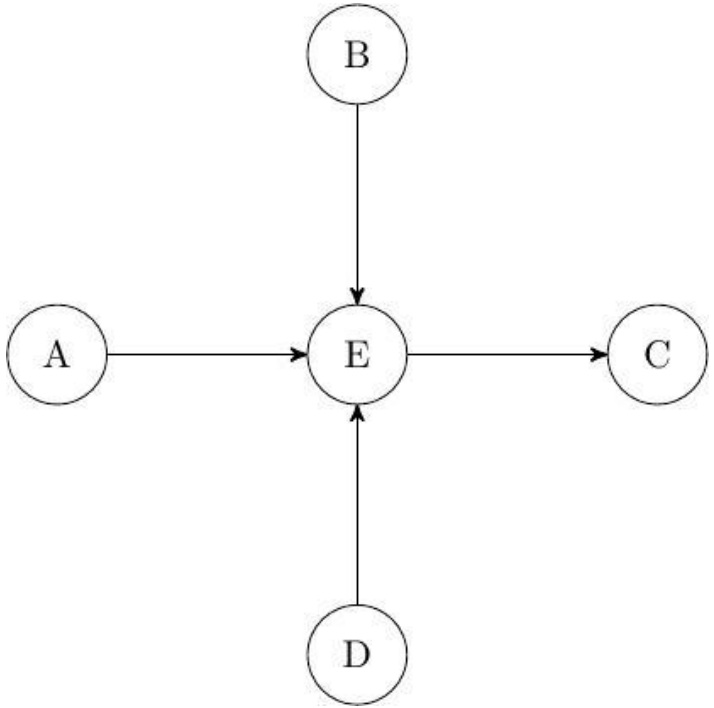
Directed Acyclic Graphs (DAGs)

A **collider** or **v-structure** is a triple of nodes $(A, B, C) \in \mathcal{N}^3$ that induces the subgraph

$$A \rightarrow B \leftarrow C$$

where A and C are not adjacent (i.e. there is no edge between A and C).

The **skeleton** of a DAG is the undirected graph obtained by converting each directed edge into an undirected edge.



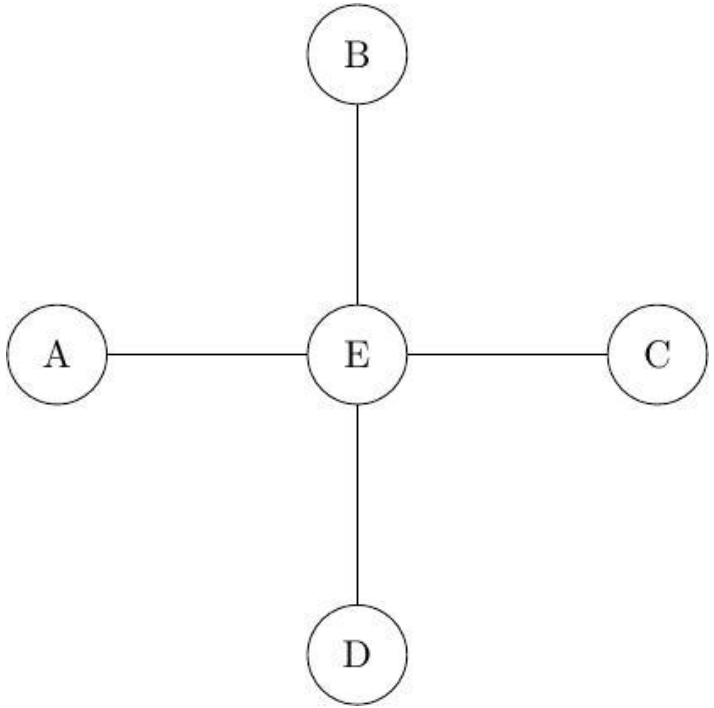
Directed Acyclic Graphs (DAGs)

A **collider** or **v-structure** is a triple of nodes $(A, B, C) \in \mathcal{N}^3$ that induces the subgraph

$$A \rightarrow B \leftarrow C$$

where A and C are not adjacent (i.e. there is no edge between A and C).

The **skeleton** of a DAG is the undirected graph obtained by converting each directed edge into an undirected edge.



d-Separation

Let $A, B \in \mathcal{N}$, $A \neq B$, and $\boldsymbol{S} \subseteq \mathcal{N}$ a set of nodes not containing A and B.

A and B are **d-separated** given $\boldsymbol{S}$ if and only if there exists no undirected path p between A and B such that

- every collider on p has a descendant in $\boldsymbol{S}$ or is in $\boldsymbol{S}$ itself, and
- no other node on p is in $\boldsymbol{S}$.

Otherwise A and B are called **d-connected**.

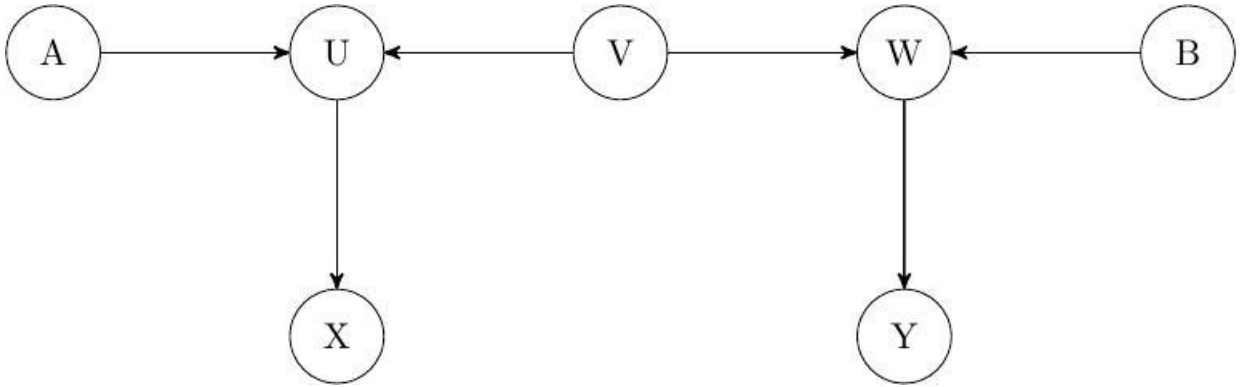
d-Separation

Let $A, B \in \mathcal{N}$, $A \neq B$, and $S \subseteq \mathcal{N}$ a set of nodes not containing A and B.

A and B are **d-separated** given S if and only if there exists no undirected path p between A and B such that

- every collider on p has a descendant in S or is in S itself, and
- no other node on p is in S.

Otherwise A and B are called **d-connected**.



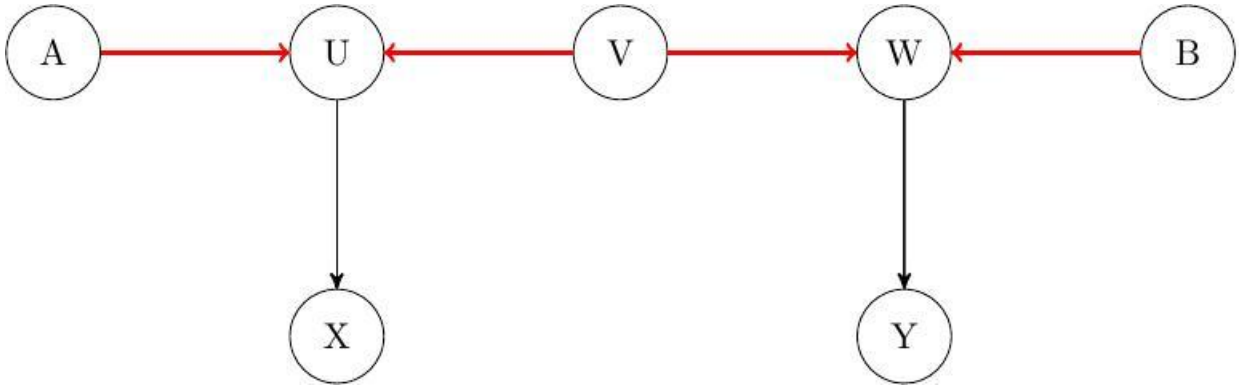
d-Separation

Let $A, B \in \mathcal{N}$, $A \neq B$, and $S \subseteq \mathcal{N}$ a set of nodes not containing A and B.

A and B are **d-separated** given S if and only if there exists no undirected path p between A and B such that

- every collider on p has a descendant in S or is in S itself, and
- no other node on p is in S.

Otherwise A and B are called **d-connected**.



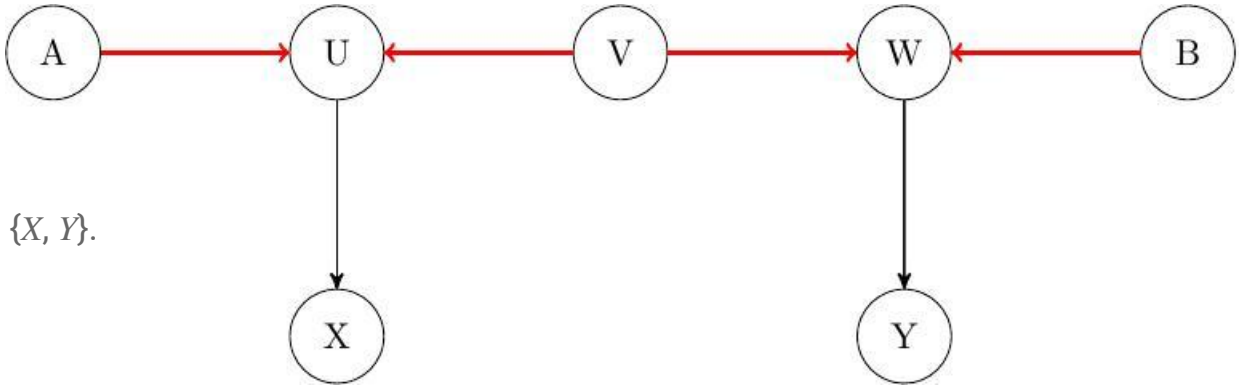
d-Separation

Let $A, B \in \mathcal{N}$, $A \neq B$, and $\boldsymbol{S} \subseteq \mathcal{N}$ a set of nodes not containing A and B.

A and B are **d-separated** given $\boldsymbol{S}$ if and only if there exists no undirected path p between A and B such that

- every collider on p has a descendant in $\boldsymbol{S}$ or is in $\boldsymbol{S}$ itself, and
- no other node on p is in $\boldsymbol{S}$.

Otherwise A and B are called **d-connected**.



A and B are **d-connected** given $\{X, Y\}$.

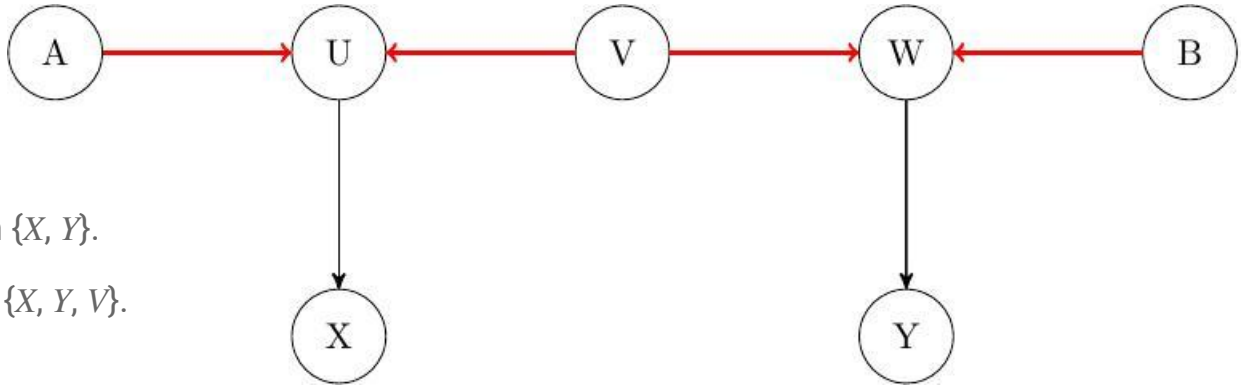
d-Separation

Let $A, B \in \mathcal{N}$, $A \neq B$, and $S \subseteq \mathcal{N}$ a set of nodes not containing A and B.

A and B are **d-separated** given S if and only if there exists no undirected path p between A and B such that

- every collider on p has a descendant in S or is in S itself, and
- no other node on p is in S.

Otherwise A and B are called **d-connected**.



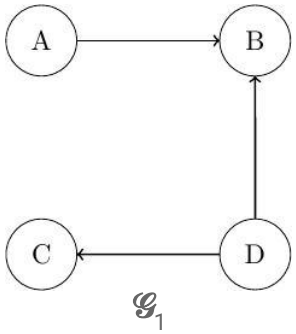
A and B are **d-connected** given $\{X, Y\}$.

A and B are **d-separated** given $\{X, Y, V\}$.

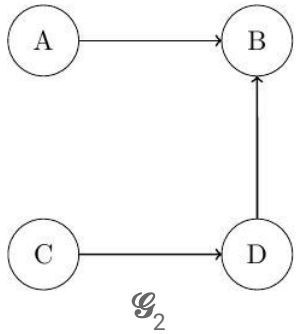
Markov equivalence class

Two DAGs with the same node set $\mathcal{N}$ are called **Markov equivalent** if and only if both share the same skeleton and the same colliders.

→ partitions space of DAGs into **equivalence classes** where all members of an equivalence class encode the same statistical model, i. e. contain the same conditional independence information and the same d-separation relations



$\mathcal{G}_1$



$\mathcal{G}_2$

Completed partially directed acyclic graph (CPDAG)

Let $[\mathcal{G}]$ be the Markov equivalence class of DAG $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ and define the edge set

$$[\mathcal{E}] = \cup_{\mathcal{H} \in [\mathcal{G}]} \mathcal{E}(\mathcal{H}),$$

where $\mathcal{E}(\mathcal{H})$ denotes the set of edges of a DAG $\mathcal{H}$. That is, $(A, B) \in [\mathcal{E}]$ if there exists a DAG $\mathcal{H} \in [\mathcal{G}]$ with the edge $A \rightarrow B$ in its edge set.

The graph $\mathcal{C}(\mathcal{G}) = (\mathcal{N}, [\mathcal{E}])$ is called the **completed partially directed acyclic graph (CPDAG)** and consists of both directed and undirected edges.

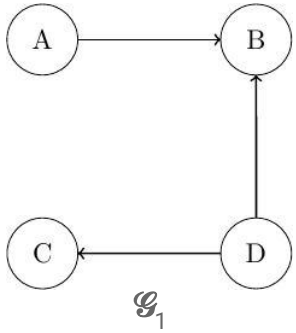
Completed partially directed acyclic graph (CPDAG)

Let $[\mathcal{G}]$ be the Markov equivalence class of DAG $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ and define the edge set

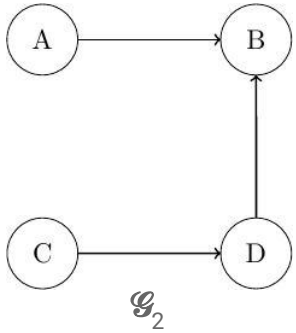
$$[\mathcal{E}] = \cup_{\mathcal{H} \in [\mathcal{G}]} \mathcal{E}(\mathcal{H}),$$

where $\mathcal{E}(\mathcal{H})$ denotes the set of edges of a DAG $\mathcal{H}$. That is, $(A, B) \in [\mathcal{E}]$ if there exists a DAG $\mathcal{H} \in [\mathcal{G}]$ with the edge $A \rightarrow B$ in its edge set.

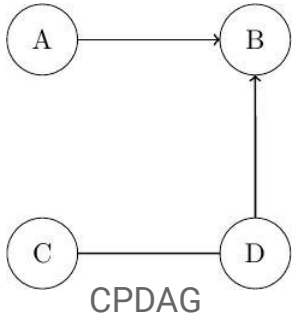
The graph $\mathcal{C}(\mathcal{G}) = (\mathcal{N}, [\mathcal{E}])$ is called the **completed partially directed acyclic graph (CPDAG)** and consists of both directed and undirected edges.



$\mathcal{G}_1$



$\mathcal{G}_2$



CPDAG

Agenda

1) Basic Definitions
2) **Introduction to the PC-Algorithm**
3) Limitations of the PC-Algorithm
4) The stable PC-Algorithm
5) The parallel PC-Algorithm
6) Simulation Study in R
7) Final Notes and Discussion

History

- Invented by **P**eter Spirtes and **C**lark Glymour in 1991 as a more efficient alternative to the SGS algorithm
- Many improvements available (stable PC, parallel PC etc.)
- Implementations in R:
  - package *pcalg* by Markus Kalisch
  - package *ParallelPC* by Thuc Duy Le, Tao Hoang, Shu Hu, and Liang Zhang

Image sources:
https://www.cmu.edu/dietrich/philosophy/people/faculty/spirtes.html (left),
https://www.cmu.edu/dietrich/philosophy/people/emeritus/glymour.html (right)

Theoretical foundation

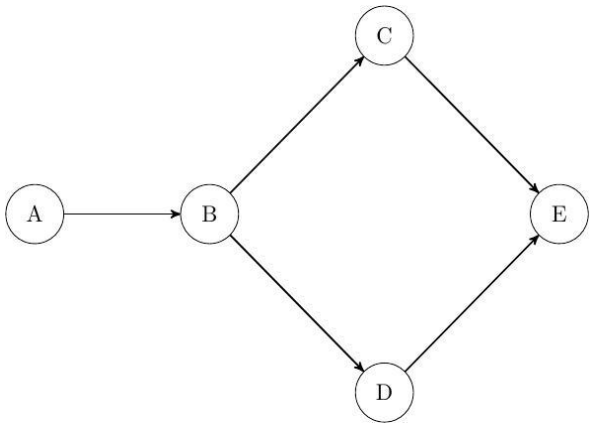
Theorem 1 (Sprites et al., 2000)

If nodes $\boldsymbol{A}$ and $\boldsymbol{B}$ are not adjacent in a DAG $\mathcal{G}$, then there is a set of nodes $\boldsymbol{S}$ which is either a subset of

$$adj(\boldsymbol{A}, \mathcal{G}) \qquad \text{or of} \qquad adj(\boldsymbol{B}, \mathcal{G})$$

such that $\boldsymbol{S}$ *d-separates* $\boldsymbol{A}$ and $\boldsymbol{B}$ in $\mathcal{G}$.

$\rightarrow \boldsymbol{S}$ disconnects $\boldsymbol{A}$ and $\boldsymbol{B}$



Proof: Sprites et al., 2000, Theorem 6.2 (page 417)

Theoretical foundation

Theorem 2 (Sprites et al., 2000)

Let the nodes $\mathcal{N}$ represent random variables $X = (X_N)_{N \in \mathcal{N}}$ which are multivariate Gaussian distributed $\mathcal{P}$ and let $\mathcal{P}$ be *faithful* to a DAG $\mathcal{G} = (\mathcal{N}, \mathcal{E})$. Assume that the perfect conditional independence information about all pairs of variables (A, B) in $\mathcal{N}$ given subsets S is given.

Then, the output of the PC-algorithm is the *CPDAG* that represents $\mathcal{G}$.

→ Even given infinite amount of data, the PC-algorithm is not able to identify the true DAG, only its equivalence class

Proof: Sprites et al., 2000, Theorem 5.1 (page 410)

Two-Step Computation

Step 1: Learning the Skeleton

Input: Complete undirected graph

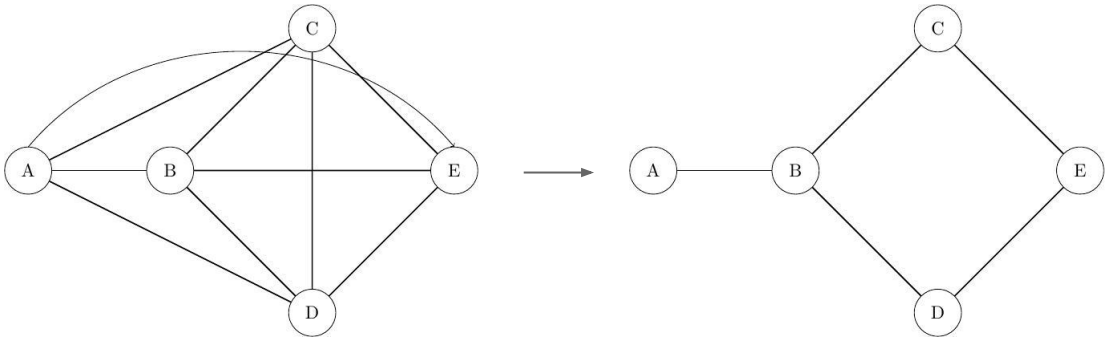
Output: Skeleton of the underlying DAG

Two-Step Computation

Step 1: Learning the Skeleton

Input: Complete undirected graph

Output: Skeleton of the underlying DAG



Example: Sprites, Glymour and Scheines, 2000

Two-Step Computation

Step 1: Learning the Skeleton

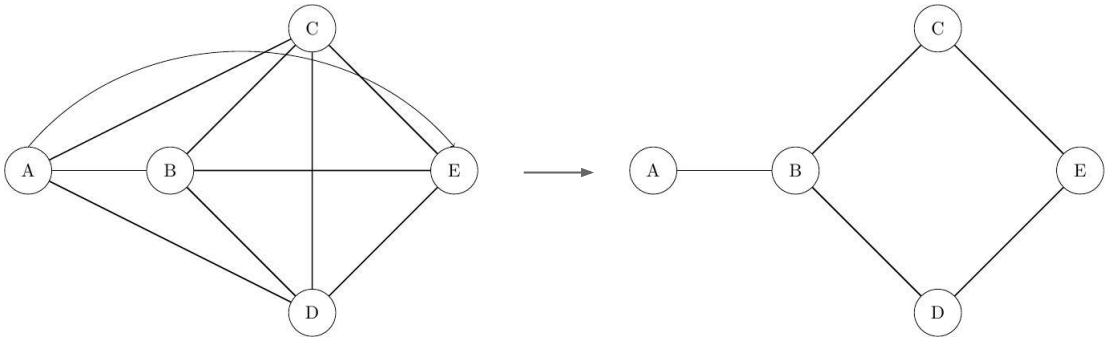
Input: Complete undirected graph

Output: Skeleton of the underlying DAG

Step 2: Learning the directions of the edges

Input: Skeleton of underlying DAG

Output: Approximated CPDAG



Example: Sprites, Glymour and Scheines, 2000

Two-Step Computation

Step 1: Learning the Skeleton

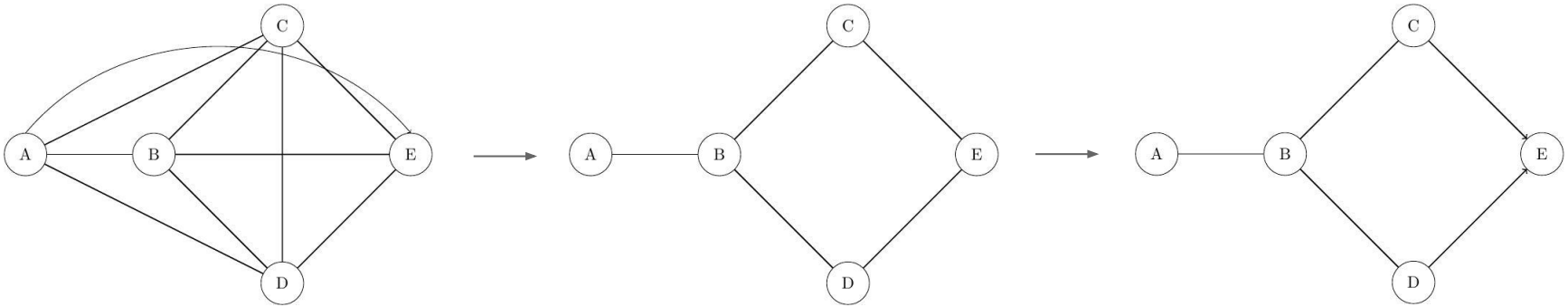
Input: Complete undirected graph

Output: Skeleton of the underlying DAG

Step 2: Learning the directions of the edges

Input: Skeleton of underlying DAG

Output: Approximated CPDAG



Example: Sprites, Glymour and Scheines, 2000

Pseudocode

Algorithm 1: Step 1: Learning the skeleton

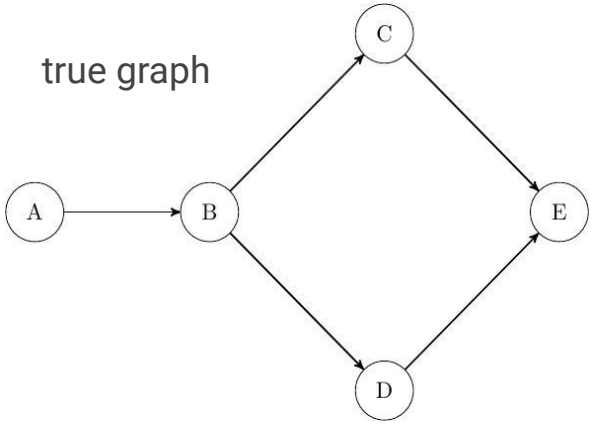
Data: Data set D with nodes $\mathcal{N}$, significance level α
Result: Skeleton $\mathcal{G}_{Skel}$, seperating sets S
$d \leftarrow 0$
repeat
  for *each ordered pair of adjacent nodes* X *and* Y **do**
    if $|adj(X, \mathcal{G}) \setminus \{Y\}| \geq d$ **then**
      for *each subset* $S \subseteq adj(X, \mathcal{G}) \setminus \{Y\}$ *and* $|S| = d$ **do**
        Test $CI(X, Y|S)$ on significance level α
        if $CI(X, Y|S)$ **then**
          Remove edge between X and Y
          Save S as separating set of (X, Y)
          Update $\mathcal{G}$ and $\mathcal{E}$
          break
        end
      end
    end
  end
  $d \leftarrow d + 1$
until $|adj(X, \mathcal{G}) \setminus \{Y\}| < d$ *for every pair of adjacent nodes*;

Kalisch and Bühlmann, 2007

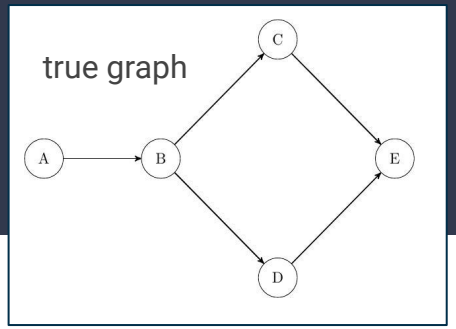
Pseudocode

Algorithm 1: Step 1: Learning the skeleton

Data: Data set D with nodes $\mathcal{N}$, significance level α
Result: Skeleton $\mathcal{G}_{Skel}$, seperating sets S
$d \leftarrow 0$
repeat
  for *each ordered pair of adjacent nodes* X *and* Y **do**
    if $|adj(X, \mathcal{G}) \setminus \{Y\}| \geq d$ **then**
      for *each subset* $S \subseteq adj(X, \mathcal{G}) \setminus \{Y\}$ *and* $|S| = d$ **do**
        Test $CI(X, Y|S)$ on significance level α
        if $CI(X, Y|S)$ **then**
          Remove edge between X and Y
          Save S as separating set of (X, Y)
          Update $\mathcal{G}$ and $\mathcal{E}$
          break
        end
      end
    end
  end
  $d \leftarrow d + 1$
until $|adj(X, \mathcal{G}) \setminus \{Y\}| < d$ *for every pair of adjacent nodes*;

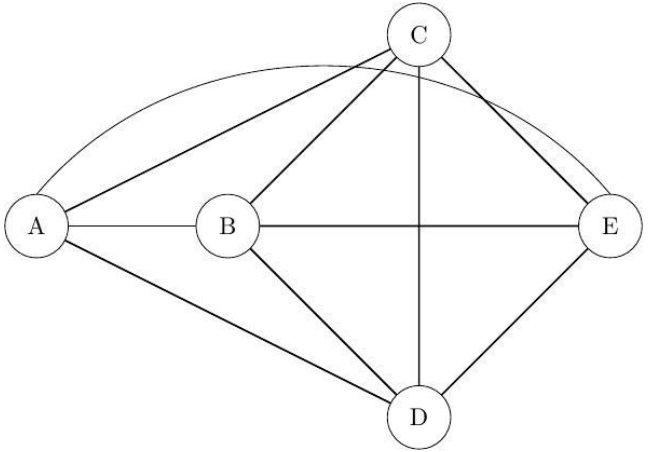


Pseudocode



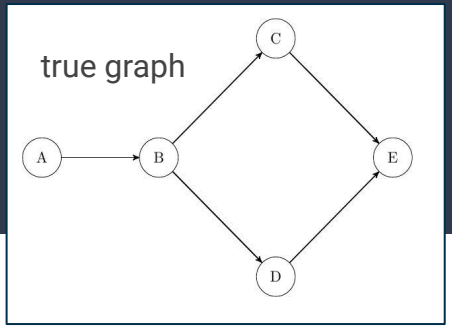
Algorithm 1: Step 1: Learning the skeleton

Data: Data set D with nodes $\mathcal{N}$, significance level α
Result: Skeleton $\mathcal{G}_{Skel}$, seperating sets S
$d \leftarrow 0$
repeat
  for *each ordered pair of adjacent nodes* X *and* Y **do**
    if $|adj(X, \mathcal{G}) \setminus \{Y\}| \geq d$ **then**
      for *each subset* $S \subseteq adj(X, \mathcal{G}) \setminus \{Y\}$ *and* $|S| = d$ **do**
        Test $CI(X, Y|S)$ on significance level α
        if $CI(X, Y|S)$ **then**
          Remove edge between X and Y
          Save S as separating set of (X, Y)
          Update $\mathcal{G}$ and $\mathcal{E}$
          break
        end
      end
    end
  end
  $d \leftarrow d + 1$
until $|adj(X, \mathcal{G}) \setminus \{Y\}| < d$ *for every pair of adjacent nodes*;



complete undirected graph

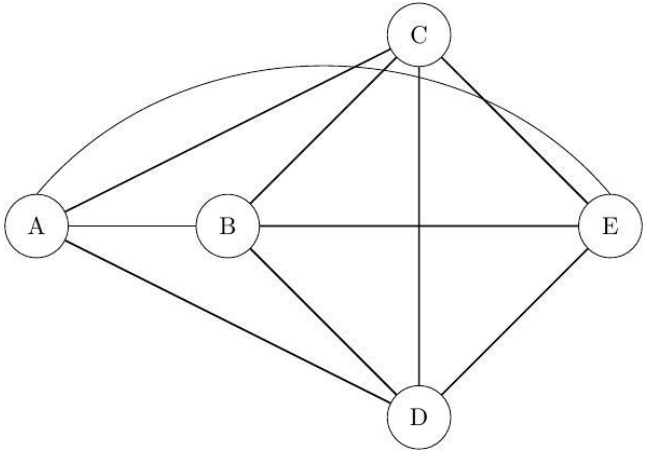
Pseudocode



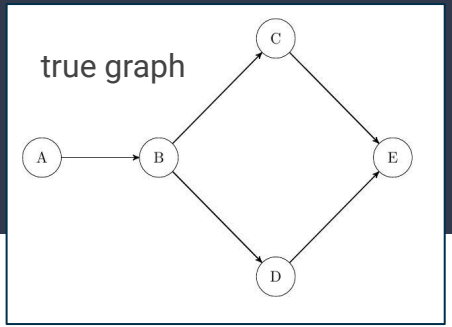
Algorithm 1: Step 1: Learning the skeleton

Data: Data set D with nodes $\mathcal{N}$, significance level α
Result: Skeleton $\mathcal{G}_{Skel}$, seperating sets S
$d \leftarrow 0$
repeat
  for *each ordered pair of adjacent nodes* X *and* Y **do**
    if $|adj(X, \mathcal{G}) \setminus \{Y\}| \geq d$ **then**
      for *each subset* $S \subseteq adj(X, \mathcal{G}) \setminus \{Y\}$ *and* $|S| = d$ **do**
        Test $CI(X, Y|S)$ on significance level α
        if $CI(X, Y|S)$ **then**
          Remove edge between X and Y
          Save S as separating set of (X, Y)
          Update $\mathcal{G}$ and $\mathcal{E}$
          break
        end
      end
    end
  end
  $d \leftarrow d + 1$
until $|adj(X, \mathcal{G}) \setminus \{Y\}| < d$ *for every pair of adjacent nodes*;

d = 0



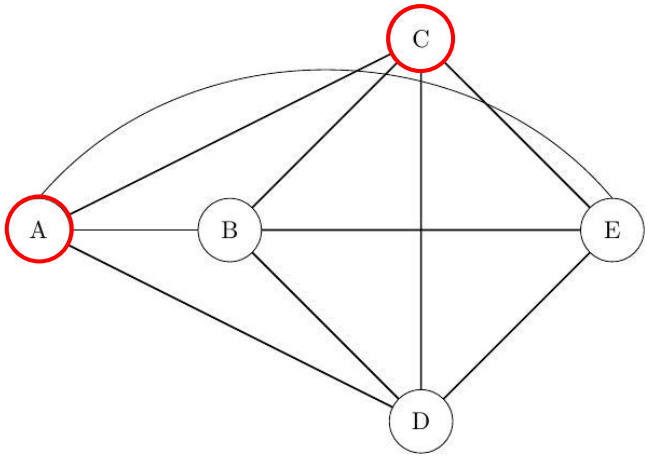
Pseudocode



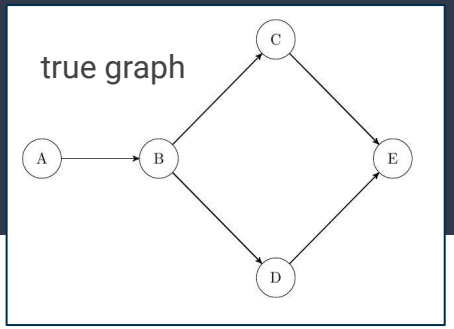
Algorithm 1: Step 1: Learning the skeleton

Data: Data set D with nodes $\mathcal{N}$, significance level α
Result: Skeleton $\mathcal{G}_{Skel}$, seperating sets S
$d \leftarrow 0$
repeat
  for *each ordered pair of adjacent nodes* X *and* Y **do**
    if $|adj(X, \mathcal{G}) \setminus \{Y\}| \geq d$ **then**
      for *each subset* $S \subseteq adj(X, \mathcal{G}) \setminus \{Y\}$ *and* $|S| = d$ **do**
        Test $CI(X, Y|S)$ on significance level α
        if $CI(X, Y|S)$ **then**
          Remove edge between X and Y
          Save S as separating set of (X, Y)
          Update $\mathcal{G}$ and $\mathcal{E}$
          break
        end
      end
    end
  end
  $d \leftarrow d + 1$
until $|adj(X, \mathcal{G}) \setminus \{Y\}| < d$ *for every pair of adjacent nodes*;

d = 0



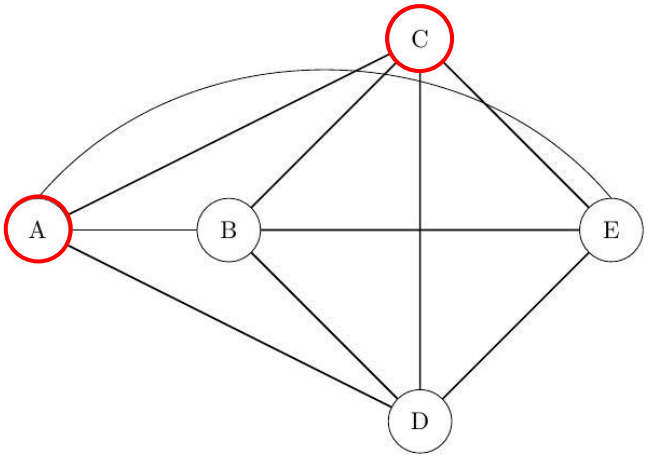
Pseudocode



Algorithm 1: Step 1: Learning the skeleton

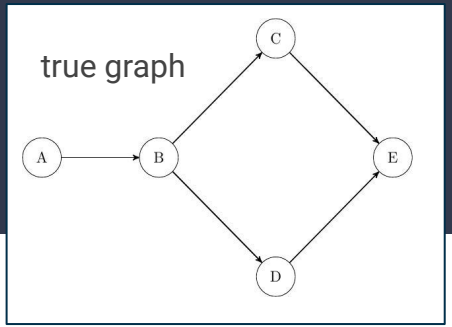
Data: Data set D with nodes $\mathcal{N}$, significance level α
Result: Skeleton $\mathcal{G}_{Skel}$, seperating sets S
$d \leftarrow 0$
repeat
  for *each ordered pair of adjacent nodes* X *and* Y **do**
    if $|adj(X, \mathcal{G}) \setminus \{Y\}| \geq d$ **then**
      for *each subset* $S \subseteq adj(X, \mathcal{G}) \setminus \{Y\}$ *and* $|S| = d$ **do**
        Test $CI(X, Y|S)$ on significance level α
        if $CI(X, Y|S)$ **then**
          Remove edge between X and Y
          Save S as separating set of (X, Y)
          Update $\mathcal{G}$ and $\mathcal{E}$
          break
        end
      end
    end
  end
  $d \leftarrow d + 1$
until $|adj(X, \mathcal{G}) \setminus \{Y\}| < d$ *for every pair of adjacent nodes*;

$d = 0$



$adj(A, \mathcal{G}) \setminus \{C\} = \{B, D, E\}$

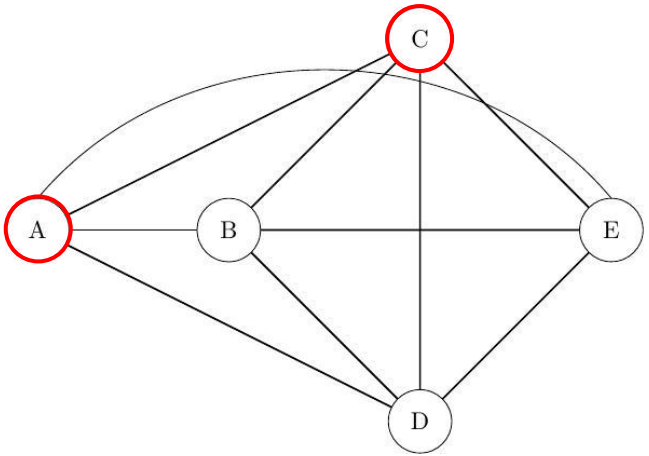
Pseudocode



Algorithm 1: Step 1: Learning the skeleton

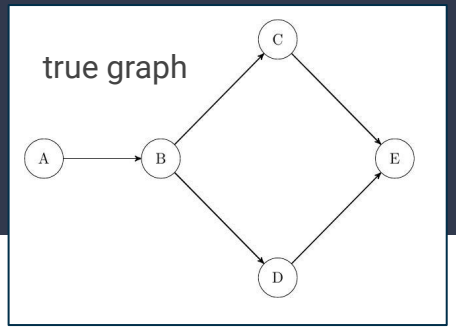
Data: Data set D with nodes $\mathcal{N}$, significance level α
Result: Skeleton $\mathcal{G}_{Skel}$, seperating sets S
$d \leftarrow 0$
repeat
  for *each ordered pair of adjacent nodes* X *and* Y **do**
    if $|adj(X, \mathcal{G}) \setminus \{Y\}| \geq d$ **then**
      for *each subset* $S \subseteq adj(X, \mathcal{G}) \setminus \{Y\}$ *and* $|S| = d$ **do**
        Test $CI(X, Y|S)$ on significance level α
        if $CI(X, Y|S)$ **then**
          Remove edge between X and Y
          Save S as separating set of (X, Y)
          Update $\mathcal{G}$ and $\mathcal{E}$
          break
        end
      end
    end
  end
  $d \leftarrow d + 1$
until $|adj(X, \mathcal{G}) \setminus \{Y\}| < d$ *for every pair of adjacent nodes*;

$d = 0$



$adj(A, \mathcal{G}) \setminus \{C\} = \{B, D, E\}$
$|S| = 0$

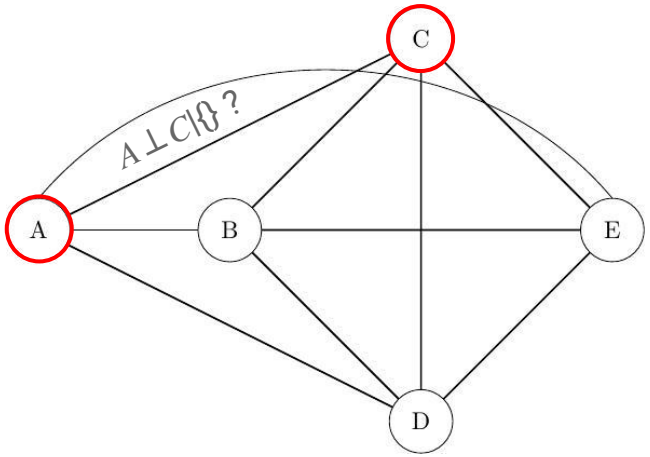
Pseudocode



Algorithm 1: Step 1: Learning the skeleton

Data: Data set D with nodes $\mathcal{N}$, significance level α
Result: Skeleton $\mathcal{G}_{Skel}$, seperating sets S
$d \leftarrow 0$
repeat
  for *each ordered pair of adjacent nodes* X *and* Y **do**
    if $|adj(X, \mathcal{G}) \setminus \{Y\}| \geq d$ **then**
      for *each subset* $S \subseteq adj(X, \mathcal{G}) \setminus \{Y\}$ *and* $|S| = d$ **do**
        Test $CI(X, Y|S)$ on significance level α
        if $CI(X, Y|S)$ **then**
          Remove edge between X and Y
          Save S as separating set of (X, Y)
          Update $\mathcal{G}$ and $\mathcal{E}$
          break
        end
      end
    end
  end
  $d \leftarrow d + 1$
until $|adj(X, \mathcal{G}) \setminus \{Y\}| < d$ *for every pair of adjacent nodes*;

d = 0



adj(**A**, G) \ {**C**} = {*B*, *D*, *E*}
|*S*| = 0

The CI test in detail

Algorithm 1: Step 1: Learning the skeleton

Data: Data set D with nodes $\mathcal{N}$, significance level α
Result: Skeleton $\mathcal{G}_{Skel}$, seperating sets S
$d \leftarrow 0$
repeat
  for *each ordered pair of adjacent nodes* X *and* Y **do**
    if $|adj(X, \mathcal{G}) \setminus \{Y\}| \geq d$ **then**
      for *each subset* $S \subseteq adj(X, \mathcal{G}) \setminus \{Y\}$ *and* $|S| = d$ **do**
        Test $CI(X, Y|S)$ on significance level α
        if $CI(X, Y|S)$ **then**
          Remove edge between X and Y
          Save S as separating set of (X, Y)
          Update $\mathcal{G}$ and $\mathcal{E}$
          break
        end
      end
    end
  end
  $d \leftarrow d + 1$
until $|adj(X, \mathcal{G}) \setminus \{Y\}| < d$ *for every pair of adjacent nodes*;

In practice: perfect CI information is unknown

→ How to estimate conditional independence from the data?

The CI test in detail

Recall: Property of the multivariate normal distribution:

For a multivariate normal distributed random vector $\boldsymbol{X} \in \mathbb{R}^p$ denote its partial correlation between components $\boldsymbol{X}^{(i)}$ and $\boldsymbol{X}^{(j)}$ $(i \neq j)$ given set $\{\boldsymbol{X}^{(r)}, r \in \boldsymbol{R} \subseteq \{1, \dots, p\} \setminus \{i, j\}\}$ by $\rho_{i,j \mid \boldsymbol{R}}$.

Then it holds:

$\rho_{i,j \mid \boldsymbol{R}} = 0$ if and only if $\boldsymbol{X}^{(i)}$ and $\boldsymbol{X}^{(j)}$ are *conditionally independent* given $\{\boldsymbol{X}^{(r)}, r \in R\}$.

→ **Idea:** Estimate partial correlation $\rho_{i,j \mid \boldsymbol{R}}$ and test whether it is close to zero

The CI test in detail

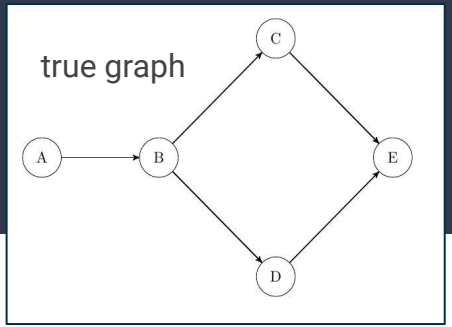
Testing of $\rho_{i,j \mid r}$ via Fisher's z-transform (Kalisch and Bühlmann, 2007):

- **Test statistic:** $Z(i,j \mid R) = 0.5 \log \left(\frac{1 + \rho_{i,j \mid \boldsymbol{R} \setminus t}}{1 - \rho_{i,j \mid \boldsymbol{R} \setminus t}} \right) \overset{H_0}{\sim} \mathcal{N}(0,1)$
- **Test:** Reject H_0: $\rho_{i,j \mid \boldsymbol{R}} = 0$ against H_1: $\rho_{i,j \mid \boldsymbol{R}} \neq 0$ if

$$(n - |R| - 3)^{0.5} \, |Z(i,j \mid R)| > \Phi^{-1}(1 - \alpha/2) \, ,$$

where Φ denotes the cumulative distribution function of a standard normal distribution

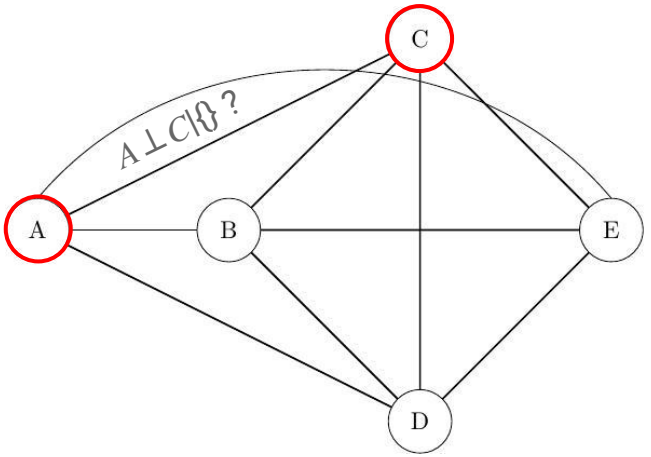
Pseudocode



Algorithm 1: Step 1: Learning the skeleton

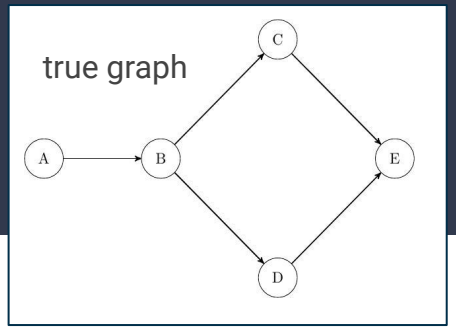
Data: Data set D with nodes $\mathcal{N}$, significance level α
Result: Skeleton $\mathcal{G}_{Skel}$, seperating sets S
$d \leftarrow 0$
repeat
  for *each ordered pair of adjacent nodes* X *and* Y **do**
    if $|adj(X, \mathcal{G}) \setminus \{Y\}| \geq d$ **then**
      for *each subset* $S \subseteq adj(X, \mathcal{G}) \setminus \{Y\}$ *and* $|S| = d$ **do**
        Test $CI(X, Y|S)$ on significance level α
        if $CI(X, Y|S)$ **then**
          Remove edge between X and Y
          Save S as separating set of (X, Y)
          Update $\mathcal{G}$ and $\mathcal{E}$
          break
        end
      end
    end
  end
  $d \leftarrow d + 1$
until $|adj(X, \mathcal{G}) \setminus \{Y\}| < d$ *for every pair of adjacent nodes*;

d = 0



adj(**A**, G) \ {**C**} = {*B*, *D*, *E*}
|*S*| = 0

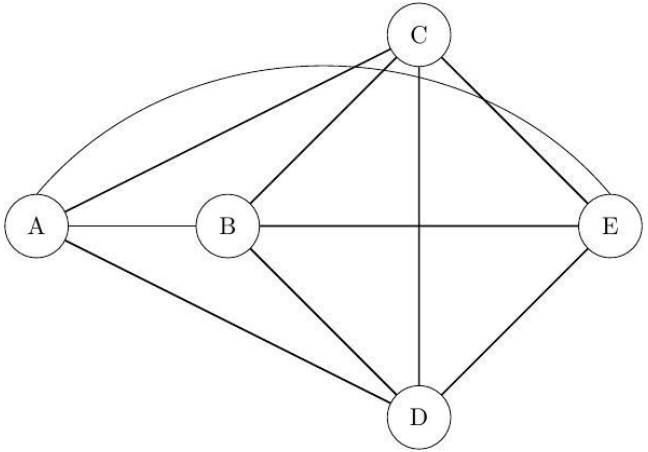
Pseudocode



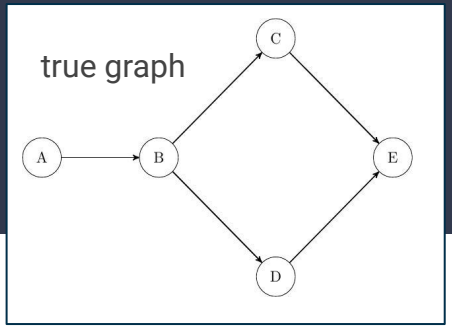
Algorithm 1: Step 1: Learning the skeleton

Data: Data set D with nodes $\mathcal{N}$, significance level α
Result: Skeleton $\mathcal{G}_{Skel}$, seperating sets S
$d \leftarrow 0$
repeat
  for *each ordered pair of adjacent nodes* X *and* Y **do**
    if $|adj(X, \mathcal{G}) \setminus \{Y\}| \geq d$ **then**
      for *each subset* $S \subseteq adj(X, \mathcal{G}) \setminus \{Y\}$ *and* $|S| = d$ **do**
        Test $CI(X, Y|S)$ on significance level α
        if $CI(X, Y|S)$ **then**
          Remove edge between X and Y
          Save S as separating set of (X, Y)
          Update $\mathcal{G}$ and $\mathcal{E}$
          break
        end
      end
    end
  end
  $d \leftarrow d + 1$
until $|adj(X, \mathcal{G}) \setminus \{Y\}| < d$ *for every pair of adjacent nodes*;

d = 1



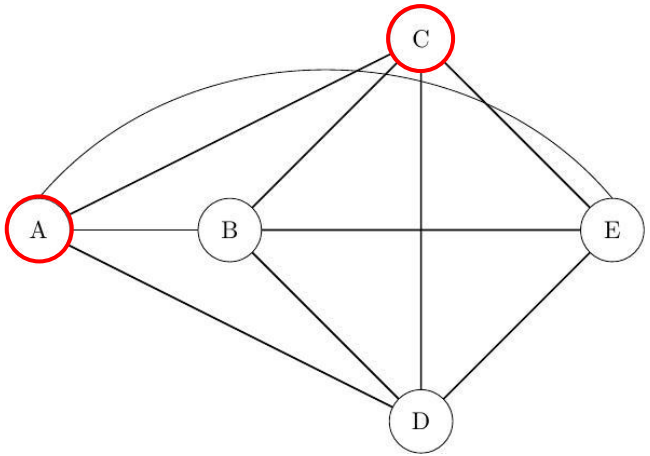
Pseudocode



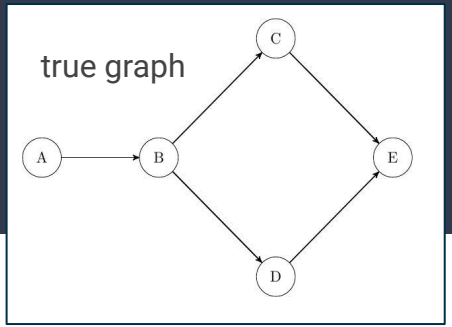
Algorithm 1: Step 1: Learning the skeleton

Data: Data set D with nodes $\mathcal{N}$, significance level α
Result: Skeleton $\mathcal{G}_{Skel}$, seperating sets S
$d \leftarrow 0$
repeat
  for *each ordered pair of adjacent nodes* X *and* Y **do**
    if $|adj(X, \mathcal{G}) \setminus \{Y\}| \geq d$ **then**
      for *each subset* $S \subseteq adj(X, \mathcal{G}) \setminus \{Y\}$ *and* $|S| = d$ **do**
        Test $CI(X, Y|S)$ on significance level α
        if $CI(X, Y|S)$ **then**
          Remove edge between X and Y
          Save S as separating set of (X, Y)
          Update $\mathcal{G}$ and $\mathcal{E}$
          break
        end
      end
    end
  end
  $d \leftarrow d + 1$
until $|adj(X, \mathcal{G}) \setminus \{Y\}| < d$ *for every pair of adjacent nodes*;

d = 1



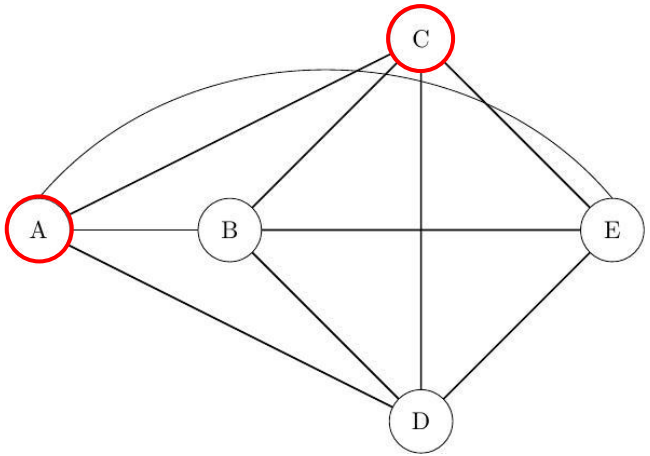
Pseudocode



Algorithm 1: Step 1: Learning the skeleton

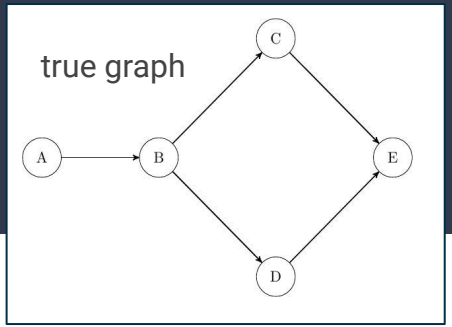
Data: Data set D with nodes $\mathcal{N}$, significance level α
Result: Skeleton $\mathcal{G}_{Skel}$, seperating sets S
$d \leftarrow 0$
repeat
  for *each ordered pair of adjacent nodes* X *and* Y **do**
    if $|adj(X, \mathcal{G}) \setminus \{Y\}| \geq d$ **then**
      for *each subset* $S \subseteq adj(X, \mathcal{G}) \setminus \{Y\}$ *and* $|S| = d$ **do**
        Test $CI(X, Y|S)$ on significance level α
        if $CI(X, Y|S)$ **then**
          Remove edge between X and Y
          Save S as separating set of (X, Y)
          Update $\mathcal{G}$ and $\mathcal{E}$
          break
        end
      end
    end
  end
  $d \leftarrow d + 1$
until $|adj(X, \mathcal{G}) \setminus \{Y\}| < d$ *for every pair of adjacent nodes*;

d = 1



$adj(A, \mathcal{G}) \setminus \{C\} = \{B, D, E\}$

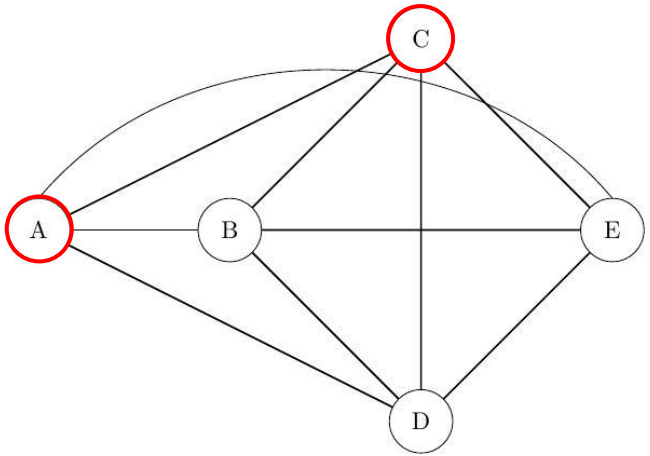
Pseudocode



Algorithm 1: Step 1: Learning the skeleton

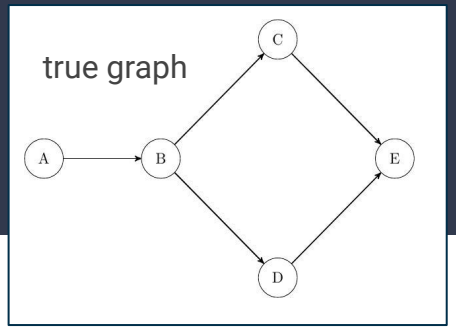
Data: Data set D with nodes $\mathcal{N}$, significance level α
Result: Skeleton $\mathcal{G}_{Skel}$, seperating sets S
$d \leftarrow 0$
repeat
  for *each ordered pair of adjacent nodes* X *and* Y **do**
    if $|adj(X, \mathcal{G}) \setminus \{Y\}| \geq d$ **then**
      for *each subset* $S \subseteq adj(X, \mathcal{G}) \setminus \{Y\}$ *and* $|S| = d$ **do**
        Test $CI(X, Y|S)$ on significance level α
        if $CI(X, Y|S)$ **then**
          Remove edge between X and Y
          Save S as separating set of (X, Y)
          Update $\mathcal{G}$ and $\mathcal{E}$
          break
        end
      end
    end
  end
  $d \leftarrow d + 1$
until $|adj(X, \mathcal{G}) \setminus \{Y\}| < d$ *for every pair of adjacent nodes*;

d = 1



$adj(A, \mathcal{G}) \setminus \{C\} = \{B, D, E\}$
$|S| = 1$

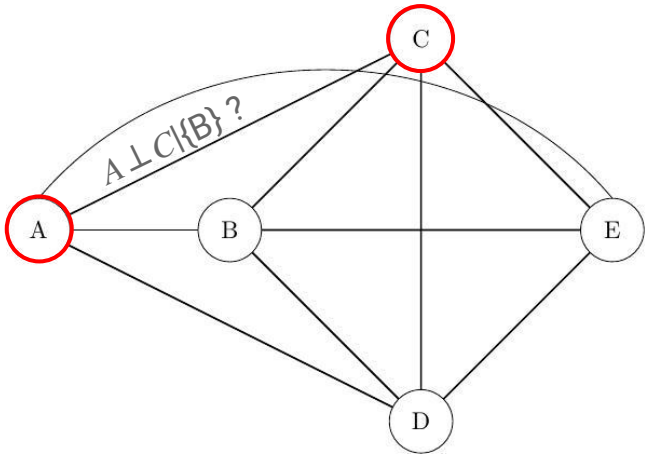
Pseudocode



Algorithm 1: Step 1: Learning the skeleton

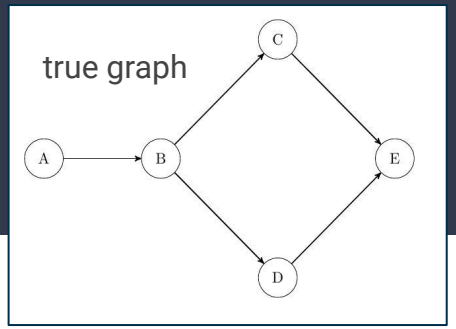
Data: Data set D with nodes $\mathcal{N}$, significance level α
Result: Skeleton $\mathcal{G}_{Skel}$, seperating sets S
$d \leftarrow 0$
repeat
  for *each ordered pair of adjacent nodes* X *and* Y **do**
    if $|adj(X, \mathcal{G}) \setminus \{Y\}| \geq d$ **then**
      for *each subset* $S \subseteq adj(X, \mathcal{G}) \setminus \{Y\}$ *and* $|S| = d$ **do**
        Test $CI(X, Y|S)$ on significance level α
        if $CI(X, Y|S)$ **then**
          Remove edge between X and Y
          Save S as separating set of (X, Y)
          Update $\mathcal{G}$ and $\mathcal{E}$
          break
        end
      end
    end
  end
  $d \leftarrow d + 1$
until $|adj(X, \mathcal{G}) \setminus \{Y\}| < d$ *for every pair of adjacent nodes*;

d = 1



$adj(A, \mathcal{G}) \setminus \{C\} = \{B, D, E\}$
$|S| = 1$

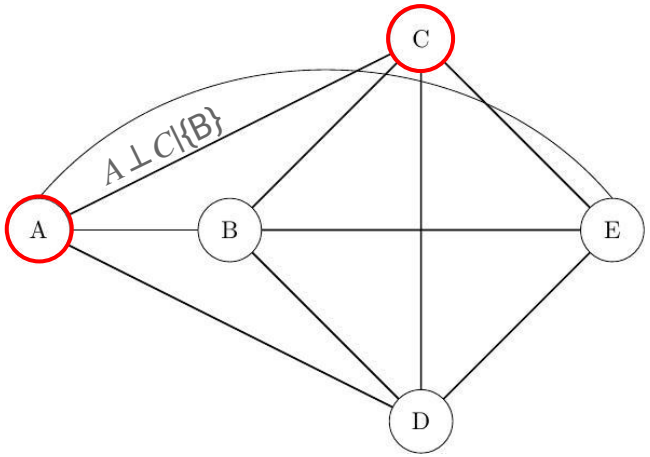
Pseudocode



Algorithm 1: Step 1: Learning the skeleton

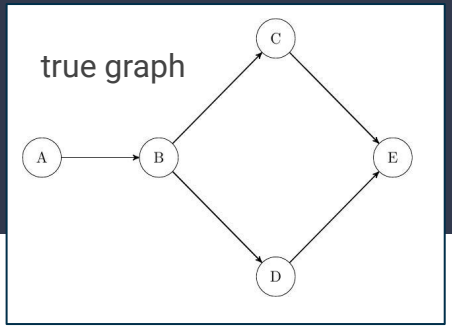
Data: Data set D with nodes $\mathcal{N}$, significance level α
Result: Skeleton $\mathcal{G}_{Skel}$, seperating sets S
$d \leftarrow 0$
repeat
  for *each ordered pair of adjacent nodes* X *and* Y **do**
    if $|adj(X, \mathcal{G}) \setminus \{Y\}| \geq d$ **then**
      for *each subset* $S \subseteq adj(X, \mathcal{G}) \setminus \{Y\}$ *and* $|S| = d$ **do**
        Test $CI(X, Y|S)$ on significance level α
        if $CI(X, Y|S)$ **then**
          Remove edge between X and Y
          Save S as separating set of (X, Y)
          Update $\mathcal{G}$ and $\mathcal{E}$
          break
        end
      end
    end
  end
  $d \leftarrow d + 1$
until $|adj(X, \mathcal{G}) \setminus \{Y\}| < d$ *for every pair of adjacent nodes*;

$d = 1$



$adj(A, \mathcal{G}) \setminus \{C\} = \{B, D, E\}$
$|S| = 1 \rightarrow S = \{B\}$

Pseudocode

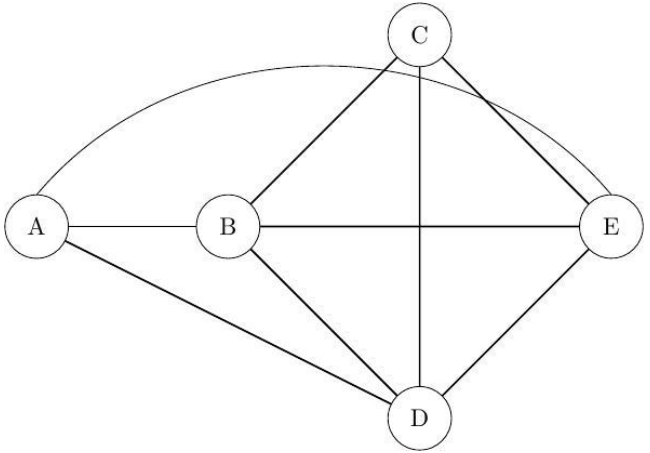


Algorithm 1: Step 1: Learning the skeleton

Data: Data set D with nodes $\mathcal{N}$, significance level α
Result: Skeleton $\mathcal{G}_{Skel}$, seperating sets S

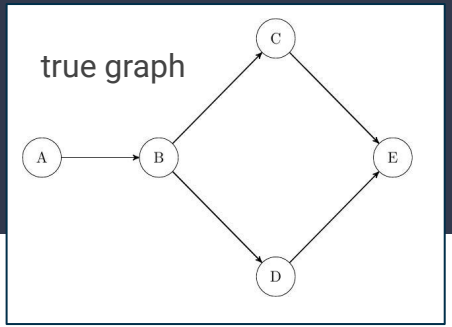
$d \leftarrow 0$
repeat
  for *each ordered pair of adjacent nodes* X *and* Y **do**
    if $|adj(X, \mathcal{G}) \setminus \{Y\}| \geq d$ **then**
      for *each subset* $S \subseteq adj(X, \mathcal{G}) \setminus \{Y\}$ *and* $|S| = d$ **do**
        Test $CI(X, Y|S)$ on significance level α
        if $CI(X, Y|S)$ **then**
          Remove edge between X and Y
          Save S as separating set of (X, Y)
          Update $\mathcal{G}$ and $\mathcal{E}$
          break
        end
      end
    end
  end
  $d \leftarrow d + 1$
until $|adj(X, \mathcal{G}) \setminus \{Y\}| < d$ *for every pair of adjacent nodes*;

$d = 1$



$adj(A, \mathcal{G}) \setminus \{C\} = \{B, D, E\}$
$|S| = 1 \rightarrow S = \{B\}$

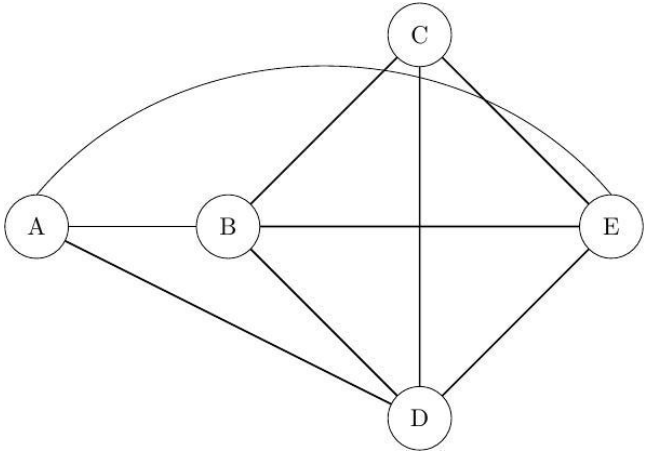
Pseudocode



Algorithm 1: Step 1: Learning the skeleton

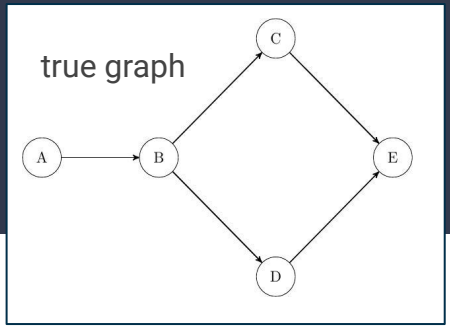
Data: Data set D with nodes $\mathcal{N}$, significance level α
Result: Skeleton $\mathcal{G}_{Skel}$, seperating sets S
$d \leftarrow 0$
repeat
  for *each ordered pair of adjacent nodes* X *and* Y **do**
    if $|adj(X, \mathcal{G}) \setminus \{Y\}| \geq d$ **then**
      for *each subset* $S \subseteq adj(X, \mathcal{G}) \setminus \{Y\}$ *and* $|S| = d$ **do**
        Test $CI(X, Y|S)$ on significance level α
        if $CI(X, Y|S)$ **then**
          Remove edge between X and Y
          Save S as separating set of (X, Y)
          Update $\mathcal{G}$ and $\mathcal{E}$
          break
        end
      end
    end
  end
  $d \leftarrow d + 1$
until $|adj(X, \mathcal{G}) \setminus \{Y\}| < d$ *for every pair of adjacent nodes*;

$d = 1$



$adj(A, \mathcal{G}) \setminus \{C\} = \{B, D, E\}$
$|S| = 1 \rightarrow S = \{B\}$

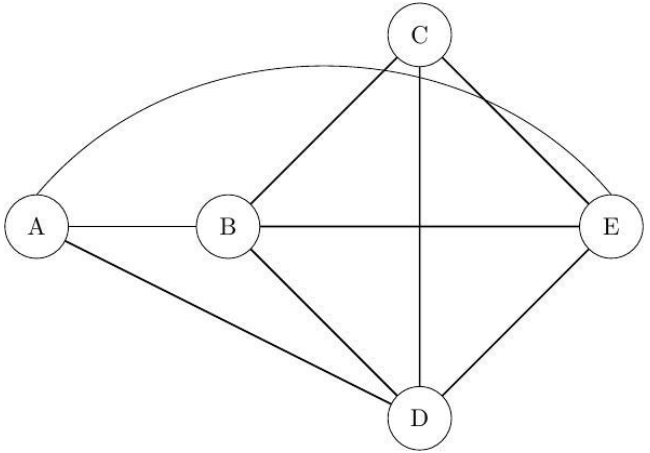
Pseudocode



Algorithm 1: Step 1: Learning the skeleton

Data: Data set D with nodes $\mathcal{N}$, significance level α
Result: Skeleton $\mathcal{G}_{Skel}$, seperating sets S
$d \leftarrow 0$
repeat
  for *each ordered pair of adjacent nodes* X *and* Y **do**
    if $|adj(X, \mathcal{G}) \setminus \{Y\}| \geq d$ **then**
      for *each subset* $S \subseteq adj(X, \mathcal{G}) \setminus \{Y\}$ *and* $|S| = d$ **do**
        Test $CI(X, Y|S)$ on significance level α
        if $CI(X, Y|S)$ **then**
          Remove edge between X and Y
          Save S as separating set of (X, Y)
          Update $\mathcal{G}$ and $\mathcal{E}$
          break
        end
      end
    end
  end
  $d \leftarrow d + 1$
until $|adj(X, \mathcal{G}) \setminus \{Y\}| < d$ *for every pair of adjacent nodes*;

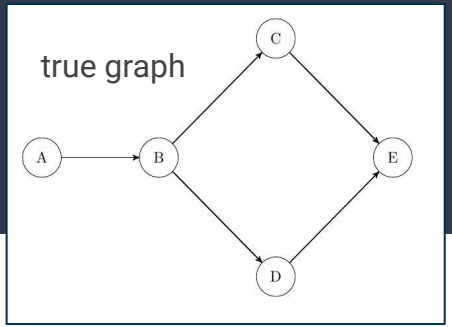
d = 1



$\mathcal{E}_{new} = \mathcal{E} \setminus \{(A, C)\}$

$\mathcal{G}_{new} = (\mathcal{N}, \mathcal{E}_{new})$

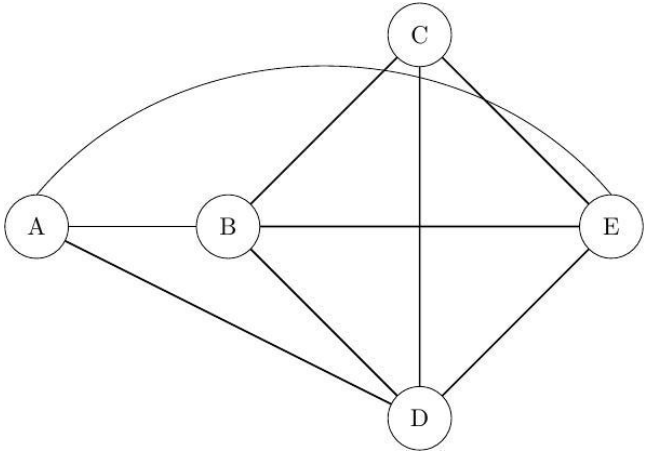
Pseudocode



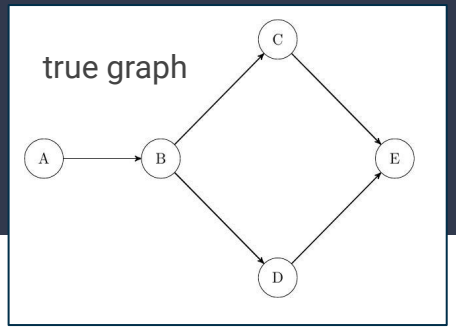
Algorithm 1: Step 1: Learning the skeleton

Data: Data set D with nodes $\mathcal{N}$, significance level α
Result: Skeleton $\mathcal{G}_{Skel}$, seperating sets S
$d \leftarrow 0$
repeat
  for *each ordered pair of adjacent nodes* X *and* Y **do**
    if $|adj(X, \mathcal{G}) \setminus \{Y\}| \geq d$ **then**
      for *each subset* $S \subseteq adj(X, \mathcal{G}) \setminus \{Y\}$ *and* $|S| = d$ **do**
        Test $CI(X, Y|S)$ on significance level α
        if $CI(X, Y|S)$ **then**
          Remove edge between X and Y
          Save S as separating set of (X, Y)
          Update $\mathcal{G}$ and $\mathcal{E}$
          break
        end
      end
    end
  end
  $d \leftarrow d + 1$
until $|adj(X, \mathcal{G}) \setminus \{Y\}| < d$ *for every pair of adjacent nodes*;

d = 1



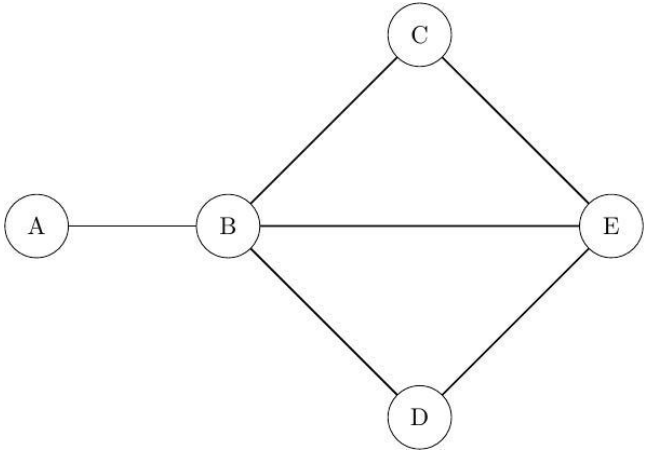
Pseudocode



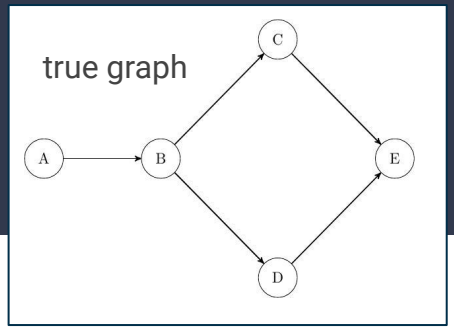
Algorithm 1: Step 1: Learning the skeleton

Data: Data set D with nodes $\mathcal{N}$, significance level α
Result: Skeleton $\mathcal{G}_{Skel}$, seperating sets S
$d \leftarrow 0$
repeat
  for *each ordered pair of adjacent nodes* X *and* Y **do**
    if $|adj(X, \mathcal{G}) \setminus \{Y\}| \geq d$ **then**
      for *each subset* $S \subseteq adj(X, \mathcal{G}) \setminus \{Y\}$ *and* $|S| = d$ **do**
        Test $CI(X, Y|S)$ on significance level α
        if $CI(X, Y|S)$ **then**
          Remove edge between X and Y
          Save S as separating set of (X, Y)
          Update $\mathcal{G}$ and $\mathcal{E}$
          break
        end
      end
    end
  end
  $d \leftarrow d + 1$
until $|adj(X, \mathcal{G}) \setminus \{Y\}| < d$ *for every pair of adjacent nodes*;

d = 2



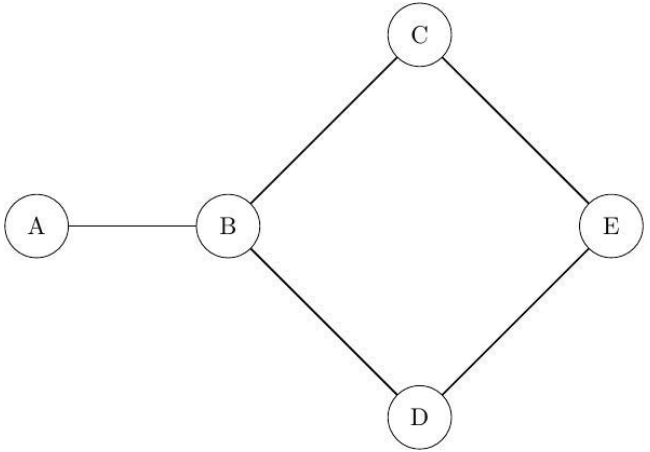
Pseudocode



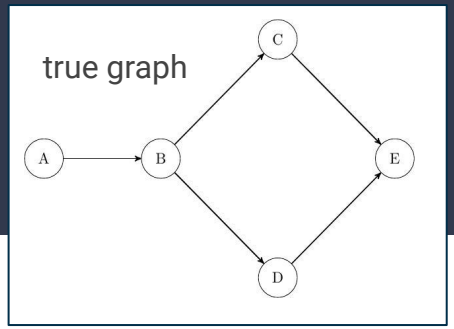
Algorithm 1: Step 1: Learning the skeleton

Data: Data set D with nodes $\mathcal{N}$, significance level α
Result: Skeleton $\mathcal{G}_{Skel}$, seperating sets S
$d \leftarrow 0$
repeat
  for *each ordered pair of adjacent nodes* X *and* Y **do**
    if $|adj(X, \mathcal{G}) \setminus \{Y\}| \geq d$ **then**
      for *each subset* $S \subseteq adj(X, \mathcal{G}) \setminus \{Y\}$ *and* $|S| = d$ **do**
        Test $CI(X, Y|S)$ on significance level α
        if $CI(X, Y|S)$ **then**
          Remove edge between X and Y
          Save S as separating set of (X, Y)
          Update $\mathcal{G}$ and $\mathcal{E}$
          break
        end
      end
    end
  end
  $d \leftarrow d + 1$
until $|adj(X, \mathcal{G}) \setminus \{Y\}| < d$ *for every pair of adjacent nodes*;

d = 3



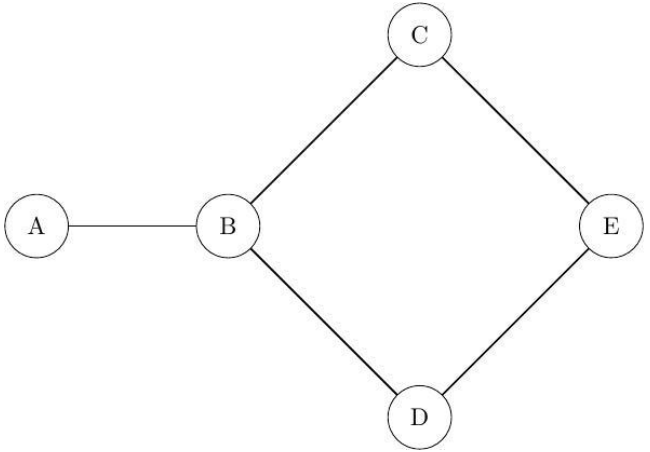
Pseudocode



Algorithm 1: Step 1: Learning the skeleton

Data: Data set D with nodes $\mathcal{N}$, significance level α
Result: Skeleton $\mathcal{G}_{Skel}$, seperating sets S
$d \leftarrow 0$
repeat
  for *each ordered pair of adjacent nodes* X *and* Y **do**
    if $|adj(X, \mathcal{G}) \setminus \{Y\}| \geq d$ **then**
      for *each subset* $S \subseteq adj(X, \mathcal{G}) \setminus \{Y\}$ *and* $|S| = d$ **do**
        Test $CI(X, Y|S)$ on significance level α
        if $CI(X, Y|S)$ **then**
          Remove edge between X and Y
          Save S as separating set of (X, Y)
          Update $\mathcal{G}$ and $\mathcal{E}$
          break
        end
      end
    end
  end
  $d \leftarrow d + 1$
until $|adj(X, \mathcal{G}) \setminus \{Y\}| < d$ *for every pair of adjacent nodes*;

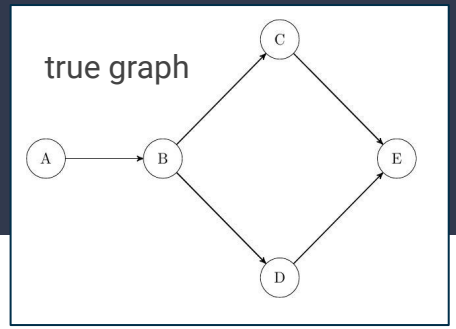
d = 3

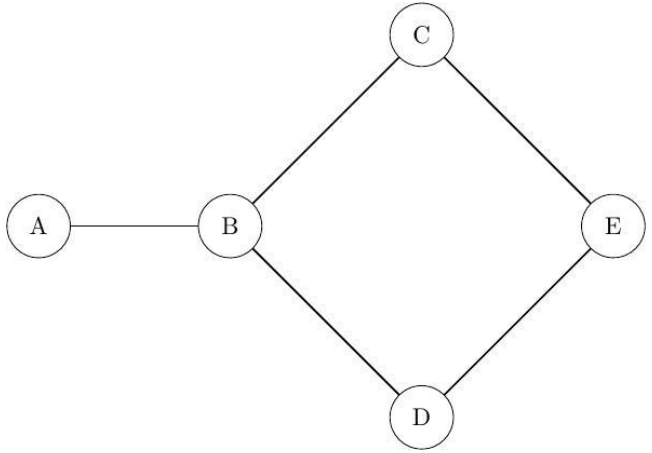


Pseudocode

Algorithm 1: Step 1: Learning the skeleton

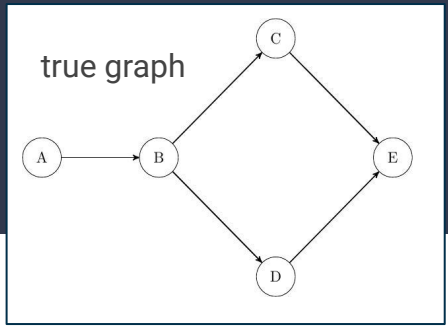
Data: Data set D with nodes $\mathcal{N}$, significance level α
Result: Skeleton $\mathcal{G}_{Skel}$, seperating sets S
$d \leftarrow 0$
repeat
  for *each ordered pair of adjacent nodes* X *and* Y **do**
    if $|adj(X, \mathcal{G}) \setminus \{Y\}| \geq d$ **then**
      for *each subset* $S \subseteq adj(X, \mathcal{G}) \setminus \{Y\}$ *and* $|S| = d$ **do**
        Test $CI(X, Y|S)$ on significance level α
        if $CI(X, Y|S)$ **then**
          Remove edge between X and Y
          Save S as separating set of (X, Y)
          Update $\mathcal{G}$ and $\mathcal{E}$
          break
        end
      end
    end
  end
  $d \leftarrow d + 1$
until $|adj(X, \mathcal{G}) \setminus \{Y\}| < d$ *for every pair of adjacent nodes*;





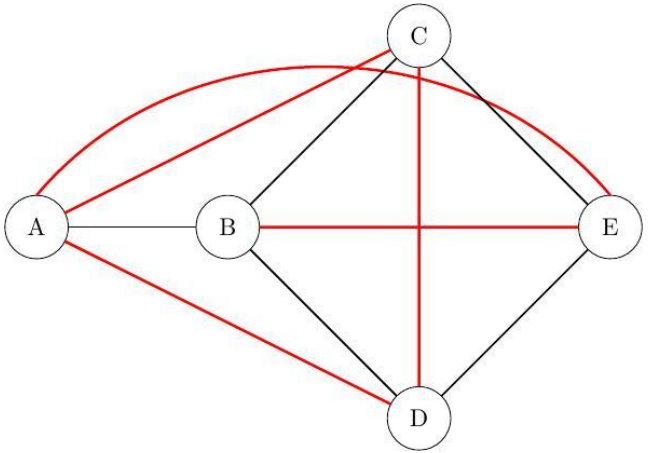
skeleton

Pseudocode



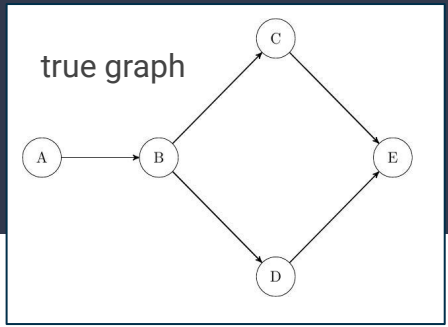
Interim results of step 1:

	Seperating set	Removed edge
$A \perp C \mid \{B\}$	{B}	$A - C$
$A \perp D \mid \{B\}$	{B}	$A - D$
$A \perp E \mid \{B\}$	{B}	$A - E$
$C \perp D \mid \{B\}$	{B}	$C - D$
$B \perp E \mid \{C, D\}$	$\{C, D\}$	$B - E$



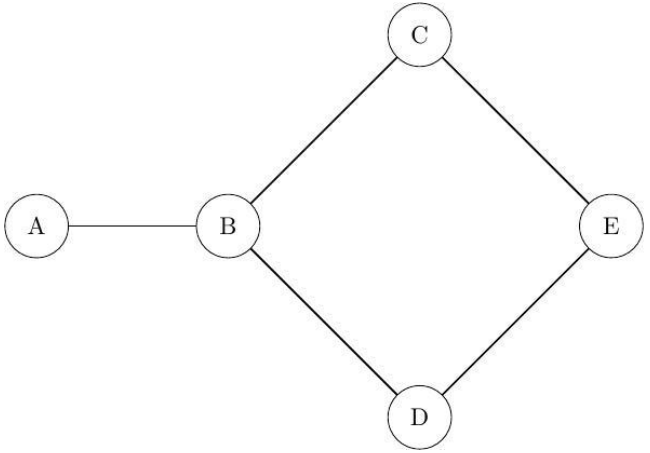
skeleton

Pseudocode

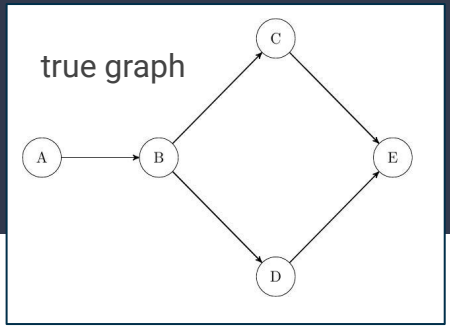


Algorithm 2: Step 2: Learning the CPDAG

Data: Skeleton $\mathcal{G}_{Skel}$, separating sets S
Result: CPDAG
for *all pairs of non-adjacent nodes* X *and* Y *with common neighbor* U **do**
  if $U \notin S(X, Y)$ **then**
    Orient $X - U - Y$ as $X \rightarrow U \leftarrow Y$
  end
end
Orient as many undirected edges as possible by applying the following rules:
repeat
  Rule 1 Orient $Y - U$ into $Y \rightarrow U$ when there is edge $X \rightarrow Y$ s.t. X and U are non-adjacent.
  Rule 2 Orient $X - Y$ into $X \rightarrow Y$ when there is $X \rightarrow U \rightarrow Y$.
  Rule 3 Orient $X - Y$ into $X \rightarrow Y$ when there is $X \rightarrow U \rightarrow Y$ and $X \rightarrow V \rightarrow Y$ s.t. U and V are non-adjacent.
  Rule 4 Orient $X - Y$ into $X \rightarrow Y$ when there is $X \rightarrow U \rightarrow V$ and $U \rightarrow V \rightarrow Y$ s.t. U and Y are non-adjacent.
until *no more edges can be oriented*;

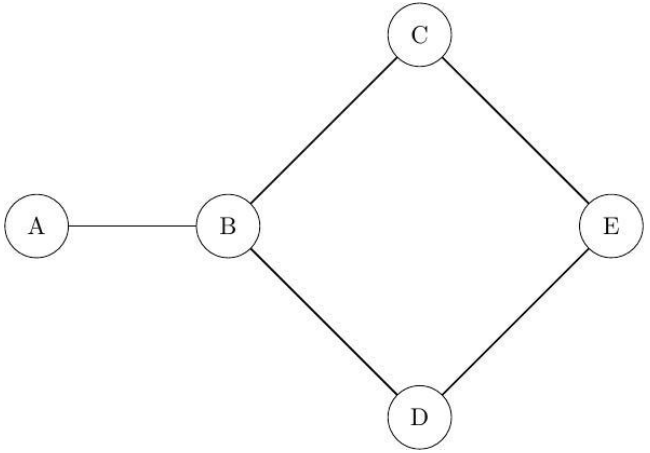


Pseudocode

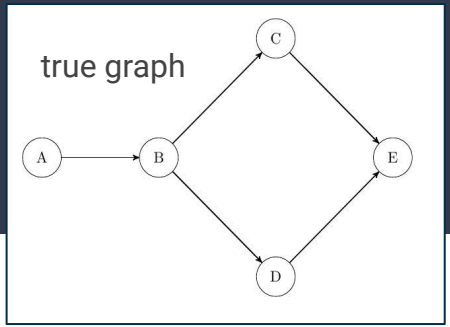


Algorithm 2: Step 2: Learning the CPDAG

Data: Skeleton $\mathcal{G}_{Skel}$, separating sets S
Result: CPDAG
for *all pairs of non-adjacent nodes* X *and* Y *with common neighbor* U **do**
  if $U \notin S(X, Y)$ **then**
    Orient $X - U - Y$ as $X \rightarrow U \leftarrow Y$
  end
end
Orient as many undirected edges as possible by applying the following rules:
repeat
  Rule 1 Orient $Y - U$ into $Y \rightarrow U$ when there is edge $X \rightarrow Y$ s.t. X and U are non-adjacent.
  Rule 2 Orient $X - Y$ into $X \rightarrow Y$ when there is $X \rightarrow U \rightarrow Y$.
  Rule 3 Orient $X - Y$ into $X \rightarrow Y$ when there is $X \rightarrow U \rightarrow Y$ and $X \rightarrow V \rightarrow Y$ s.t. U and V are non-adjacent.
  Rule 4 Orient $X - Y$ into $X \rightarrow Y$ when there is $X \rightarrow U \rightarrow V$ and $U \rightarrow V \rightarrow Y$ s.t. U and Y are non-adjacent.
until *no more edges can be oriented*;



Pseudocode



Algorithm 2: Step 2: Learning the CPDAG

Data: Skeleton $\mathcal{G}_{Skel}$, separating sets S

Result: CPDAG

for *all pairs of non-adjacent nodes* X *and* Y *with common neighbor* U **do**

  if $U \notin S(X, Y)$ **then**

    Orient $X - U - Y$ as $X \rightarrow U \leftarrow Y$

  end

end

Orient as many undirected edges as possible by applying the following rules:

repeat

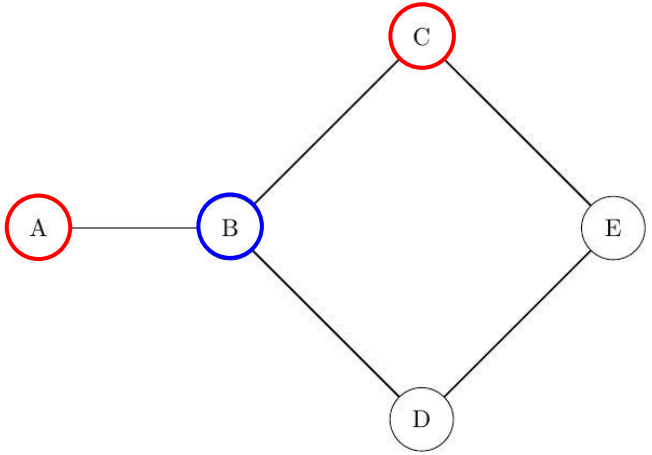
  Rule 1 Orient $Y - U$ into $Y \rightarrow U$ when there is edge $X \rightarrow Y$ s.t. X and U are non-adjacent.

  Rule 2 Orient $X - Y$ into $X \rightarrow Y$ when there is $X \rightarrow U \rightarrow Y$.

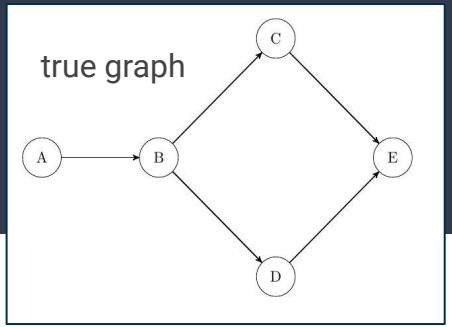
  Rule 3 Orient $X - Y$ into $X \rightarrow Y$ when there is $X \rightarrow U \rightarrow Y$ and $X \rightarrow V \rightarrow Y$ s.t. U and V are non-adjacent.

  Rule 4 Orient $X - Y$ into $X \rightarrow Y$ when there is $X \rightarrow U \rightarrow V$ and $U \rightarrow V \rightarrow Y$ s.t. U and Y are non-adjacent.

until *no more edges can be oriented*;



Pseudocode



Algorithm 2: Step 2: Learning the CPDAG

Data: Skeleton $\mathcal{G}_{Skel}$, separating sets S

Result: CPDAG

for *all pairs of non-adjacent nodes* X *and* Y *with common neighbor* U **do**

  if $U \notin S(X, Y)$ **then**

    Orient $X - U - Y$ as $X \rightarrow U \leftarrow Y$

  end

end

Orient as many undirected edges as possible by applying the following rules:

repeat

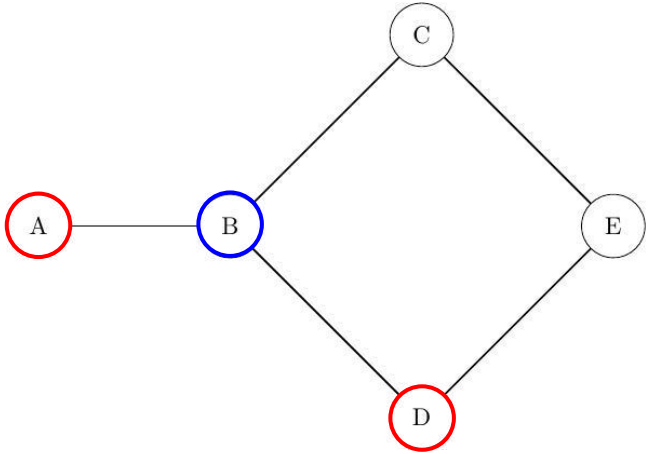
  Rule 1 Orient $Y - U$ into $Y \rightarrow U$ when there is edge $X \rightarrow Y$ s.t. X and U are non-adjacent.

  Rule 2 Orient $X - Y$ into $X \rightarrow Y$ when there is $X \rightarrow U \rightarrow Y$.

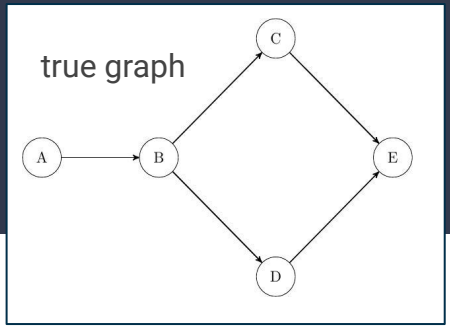
  Rule 3 Orient $X - Y$ into $X \rightarrow Y$ when there is $X \rightarrow U \rightarrow Y$ and $X \rightarrow V \rightarrow Y$ s.t. U and V are non-adjacent.

  Rule 4 Orient $X - Y$ into $X \rightarrow Y$ when there is $X \rightarrow U \rightarrow V$ and $U \rightarrow V \rightarrow Y$ s.t. U and Y are non-adjacent.

until *no more edges can be oriented*;



Pseudocode



Algorithm 2: Step 2: Learning the CPDAG

Data: Skeleton $\mathcal{G}_{Skel}$, separating sets S

Result: CPDAG

for *all pairs of non-adjacent nodes* X *and* Y *with common neighbor* U **do**

  if $U \notin S(X, Y)$ **then**

    Orient $X - U - Y$ as $X \rightarrow U \leftarrow Y$

  end

end

Orient as many undirected edges as possible by applying the following rules:

repeat

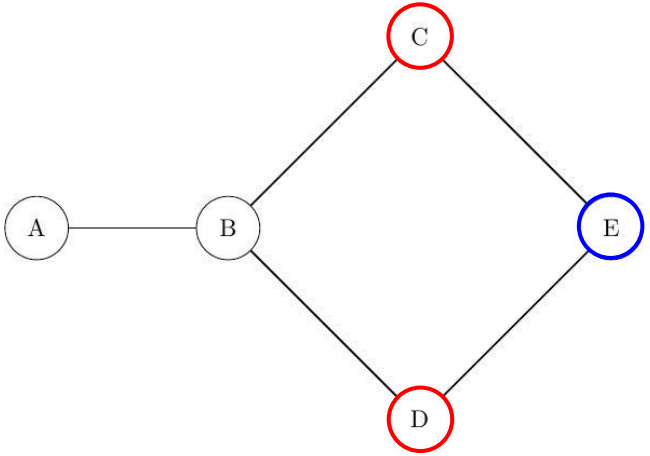
  Rule 1 Orient $Y - U$ into $Y \rightarrow U$ when there is edge $X \rightarrow Y$ s.t. X and U are non-adjacent.

  Rule 2 Orient $X - Y$ into $X \rightarrow Y$ when there is $X \rightarrow U \rightarrow Y$.

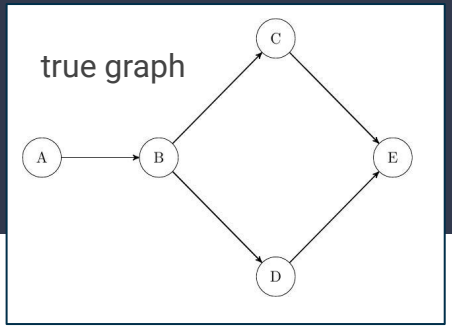
  Rule 3 Orient $X - Y$ into $X \rightarrow Y$ when there is $X \rightarrow U \rightarrow Y$ and $X \rightarrow V \rightarrow Y$ s.t. U and V are non-adjacent.

  Rule 4 Orient $X - Y$ into $X \rightarrow Y$ when there is $X \rightarrow U \rightarrow V$ and $U \rightarrow V \rightarrow Y$ s.t. U and Y are non-adjacent.

until *no more edges can be oriented*;

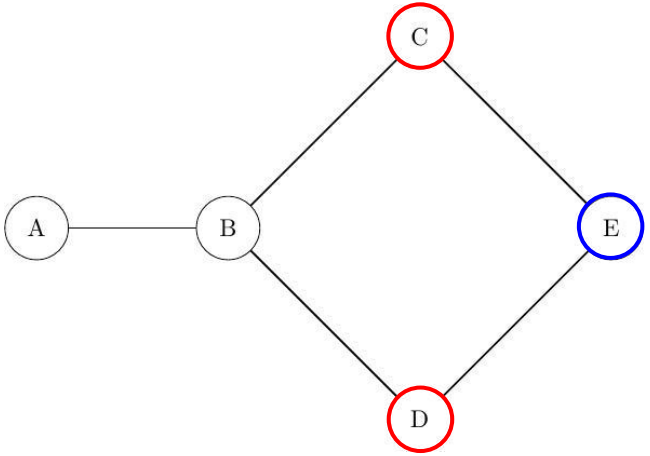


Pseudocode



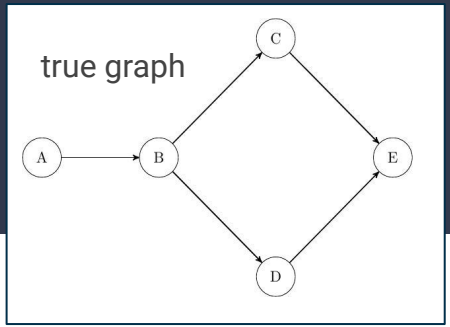
Algorithm 2: Step 2: Learning the CPDAG

Data: Skeleton $\mathcal{G}_{Skel}$, separating sets S
Result: CPDAG
for *all pairs of non-adjacent nodes* X *and* Y *with common neighbor* U **do**
  if $U \notin S(X,Y)$ **then**
    Orient $X - U - Y$ as $X \rightarrow U \leftarrow Y$
  end
end
Orient as many undirected edges as possible by applying the following rules:
repeat
  Rule 1 Orient $Y - U$ into $Y \rightarrow U$ when there is edge $X \rightarrow Y$ s.t. X and U are non-adjacent.
  Rule 2 Orient $X - Y$ into $X \rightarrow Y$ when there is $X \rightarrow U \rightarrow Y$.
  Rule 3 Orient $X - Y$ into $X \rightarrow Y$ when there is $X \rightarrow U \rightarrow Y$ and $X \rightarrow V \rightarrow Y$ s.t. U and V are non-adjacent.
  Rule 4 Orient $X - Y$ into $X \rightarrow Y$ when there is $X \rightarrow U \rightarrow V$ and $U \rightarrow V \rightarrow Y$ s.t. U and Y are non-adjacent.
until *no more edges can be oriented*;



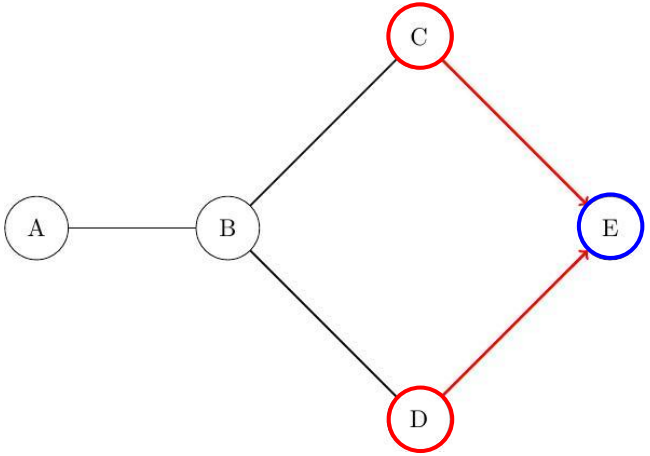
$S(C, D) = \{B\} \ni E$

Pseudocode



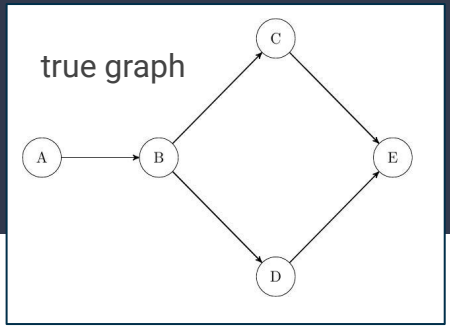
Algorithm 2: Step 2: Learning the CPDAG

Data: Skeleton $\mathcal{G}_{Skel}$, separating sets S
Result: CPDAG
for *all pairs of non-adjacent nodes* X *and* Y *with common neighbor* U **do**
  if $U \notin S(X, Y)$ **then**
    Orient $X - U - Y$ as $X \rightarrow U \leftarrow Y$
  end
end
Orient as many undirected edges as possible by applying the following rules:
repeat
  Rule 1 Orient $Y - U$ into $Y \rightarrow U$ when there is edge $X \rightarrow Y$ s.t. X and U are non-adjacent.
  Rule 2 Orient $X - Y$ into $X \rightarrow Y$ when there is $X \rightarrow U \rightarrow Y$.
  Rule 3 Orient $X - Y$ into $X \rightarrow Y$ when there is $X \rightarrow U \rightarrow Y$ and $X \rightarrow V \rightarrow Y$ s.t. U and V are non-adjacent.
  Rule 4 Orient $X - Y$ into $X \rightarrow Y$ when there is $X \rightarrow U \rightarrow V$ and $U \rightarrow V \rightarrow Y$ s.t. U and Y are non-adjacent.
until *no more edges can be oriented*;



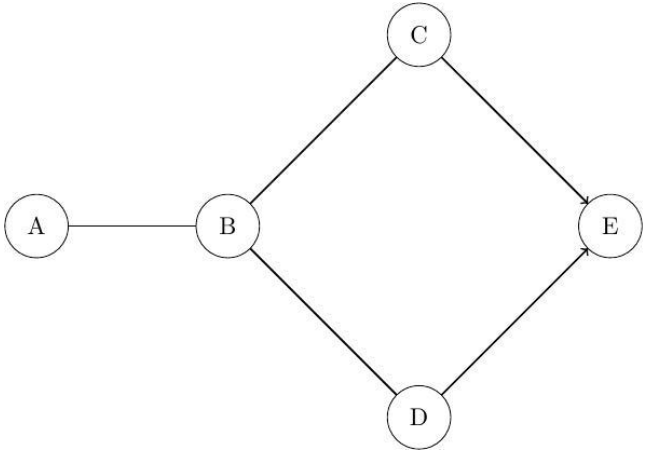
$S(C, D) = \{B\} \ni E$

Pseudocode

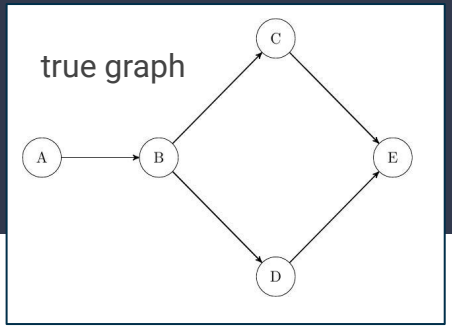


Algorithm 2: Step 2: Learning the CPDAG

Data: Skeleton $\mathcal{G}_{Skel}$, separating sets S
Result: CPDAG
for *all pairs of non-adjacent nodes* X *and* Y *with common neighbor* U **do**
  if $U \notin S(X, Y)$ **then**
    Orient $X - U - Y$ as $X \rightarrow U \leftarrow Y$
  end
end
Orient as many undirected edges as possible by applying the following rules:
repeat
  Rule 1 Orient $Y - U$ into $Y \rightarrow U$ when there is edge $X \rightarrow Y$ s.t. X and U are non-adjacent.
  Rule 2 Orient $X - Y$ into $X \rightarrow Y$ when there is $X \rightarrow U \rightarrow Y$.
  Rule 3 Orient $X - Y$ into $X \rightarrow Y$ when there is $X \rightarrow U \rightarrow Y$ and $X \rightarrow V \rightarrow Y$ s.t. U and V are non-adjacent.
  Rule 4 Orient $X - Y$ into $X \rightarrow Y$ when there is $X \rightarrow U \rightarrow V$ and $U \rightarrow V \rightarrow Y$ s.t. U and Y are non-adjacent.
until *no more edges can be oriented*;



Pseudocode



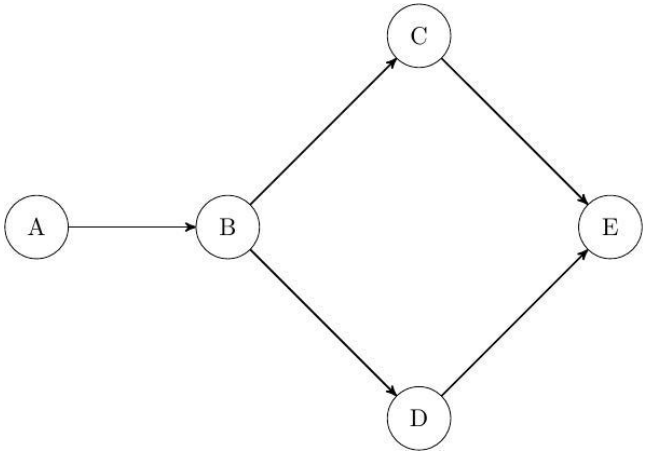
Algorithm 2: Step 2: Learning the CPDAG

Data: Skeleton $\mathcal{G}_{Skel}$, separating sets S
Result: CPDAG
for *all pairs of non-adjacent nodes* X *and* Y *with common neighbor* U **do**
  if $U \notin S(X, Y)$ **then**
    Orient $X - U - Y$ as $X \rightarrow U \leftarrow Y$
  end
end
Orient as many undirected edges as possible by applying the following rules:
repeat
  Rule 1 Orient $Y - U$ into $Y \rightarrow U$ when there is edge $X \rightarrow Y$ s.t. X and U are non-adjacent.
  Rule 2 Orient $X - Y$ into $X \rightarrow Y$ when there is $X \rightarrow U \rightarrow Y$.
  Rule 3 Orient $X - Y$ into $X \rightarrow Y$ when there is $X \rightarrow U \rightarrow Y$ and $X \rightarrow V \rightarrow Y$ s.t. U and V are non-adjacent.
  Rule 4 Orient $X - Y$ into $X \rightarrow Y$ when there is $X \rightarrow U \rightarrow V$ and $U \rightarrow V \rightarrow Y$ s.t. U and Y are non-adjacent.
until *no more edges can be oriented*;

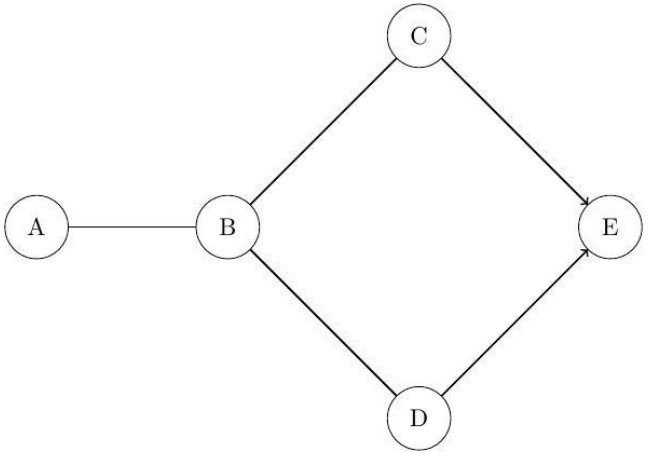
oriented based on invalid edges inducing new colliders or directed edges

Pseudocode

Final result of PC:



true graph



approximated CPDAG

Agenda

1) Basic Definitions
2) Introduction to the PC-Algorithm
3) **Limitations of the PC-Algorithm**
4) The stable PC-Algorithm
5) The parallel PC-Algorithm
6) Simulation Study in R
7) Final Notes and Discussion

Limitations of the original PC-Algorithm

Ordering

Problem:

Output graph is dependent on the order in which CI tests are performed

Solution: Stable PC-algorithm

(Colombo and Maathuis, 2012)

Runtime

Problem:

Many CI tests necessary

→ high runtime

Solution: Parallel PC-algorithm

(Le et al., 2014)

Gaussian Distribution

Problem:

Consistency only holds for Gaussian distribution

→ what if it is non-Gaussian?

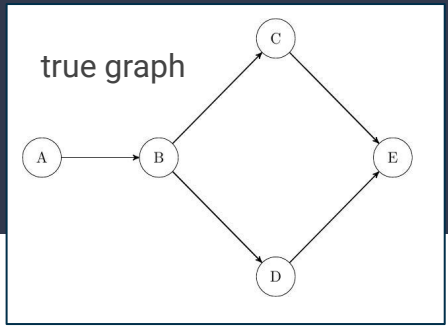
Solution: Rank PC-algorithm

(Harris and Drton, 2013)

Agenda

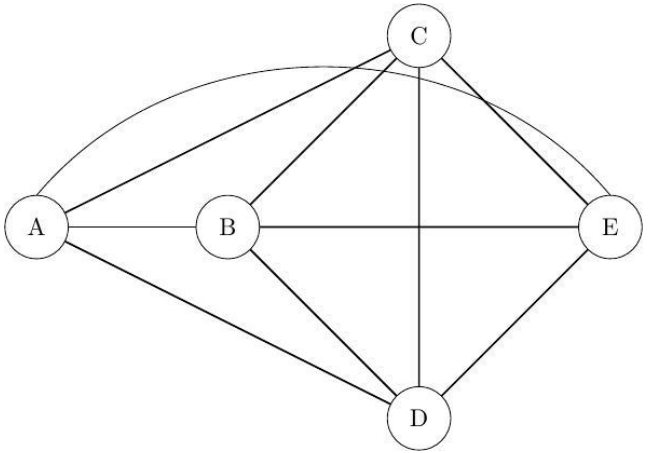
1) Basic Definitions
2) Introduction to the PC-Algorithm
3) Limitations of the PC-Algorithm
4) **The stable PC-Algorithm**
5) The parallel PC-Algorithm
6) Simulation Study in R
7) Final Notes and Discussion

The stable PC-Algorithm

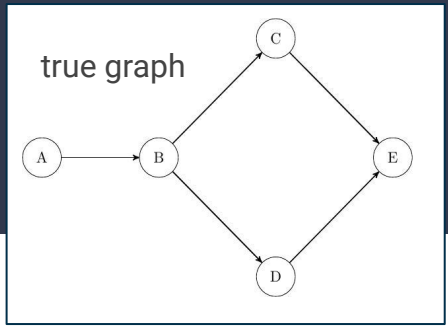


Problem:

Incorrectly removing/retaining edge results in changes of set S



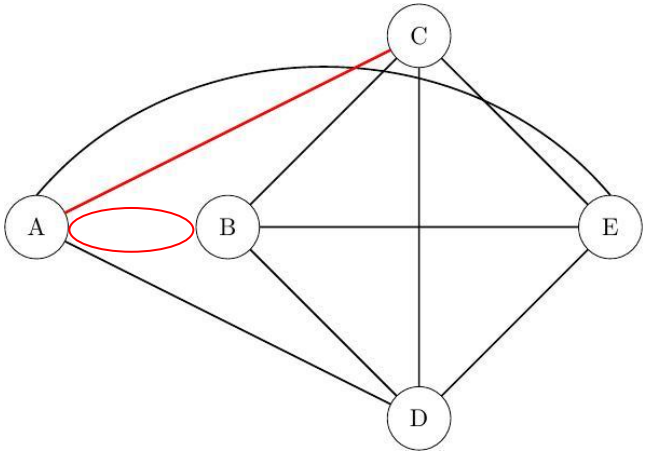
The stable PC-Algorithm



Problem:

Incorrectly removing/retaining edge results in changes of set S

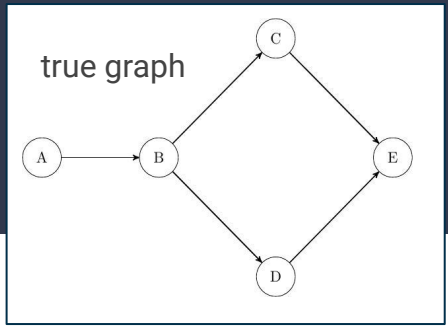
→ output graph depends on order in which CI tests are performed



$adj(A, \mathcal{G}) \setminus \{C\} = \{D, E\}$

$S \nexists B$ for all possible sets S

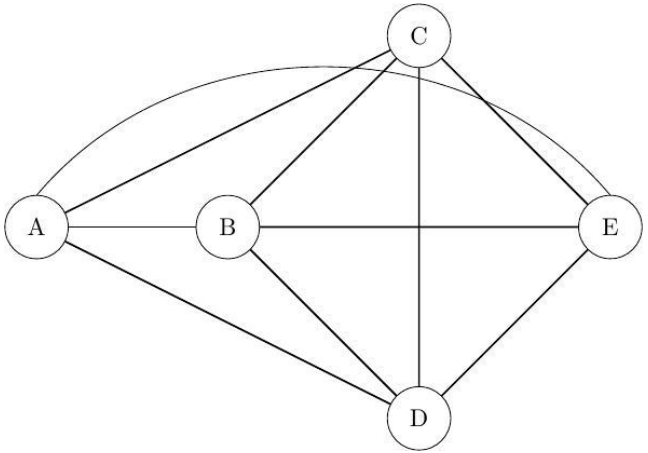
The stable PC-Algorithm



Problem:

Incorrectly removing/retaining edge results in changes of set S

→ output graph depends on order in which CI tests are performed



Idea:

Keep adjacency sets of nodes unchanged in each level d

Idea

Algorithm 1: Step 1: Learning the skeleton

Data: Data set D with nodes $\mathcal{N}$, significance level α
Result: Skeleton $\mathcal{G}_{Skel}$, seperating sets S
$d \leftarrow 0$
repeat
  for *each ordered pair of adjacent nodes* X *and* Y **do**
    if $|adj(X, \mathcal{G}) \setminus \{Y\}| \geq d$ **then**
      for *each subset* $S \subseteq adj(X, \mathcal{G}) \setminus \{Y\}$ *and* $|S| = d$ **do**
        Test $CI(X, Y|S)$ on significance level α
        if $CI(X, Y|S)$ **then**
          Remove edge between X and Y
          Save S as separating set of (X, Y)
          Update $\mathcal{G}$ and $\mathcal{E}$
          break
        end
      end
    end
  end
  $d \leftarrow d + 1$
until $|adj(X, \mathcal{G}) \setminus \{Y\}| < d$ *for every pair of adjacent nodes*;

Idea

Algorithm 1: Step 1: Learning the skeleton

Data: Data set D with nodes $\mathcal{N}$, significance level α
Result: Skeleton $\mathcal{G}_{Skel}$, seperating sets S
$d \leftarrow 0$
repeat
  for *each ordered pair of adjacent nodes* X *and* Y **do**
    if $|adj(X, \mathcal{G}) \setminus \{Y\}| \geq d$ **then**
      for *each subset* $S \subseteq adj(X, \mathcal{G}) \setminus \{Y\}$ *and* $|S| = d$ **do**
        Test $CI(X, Y|S)$ on significance level α
        if $CI(X, Y|S)$ **then**
          Remove edge between X and Y
          Save S as separating set of (X, Y)
          Update $\mathcal{G}$ and $\mathcal{E}$
          break
        end
      end
    end
  end
  $d \leftarrow d + 1$
until $|adj(X, \mathcal{G}) \setminus \{Y\}| < d$ *for every pair of adjacent nodes*;

Idea

Algorithm 1: Step 1: Learning the skeleton

Data: Data set D with nodes $\mathcal{N}$, significance level α
Result: Skeleton $\mathcal{G}_{Skel}$, seperating sets S
$d \leftarrow 0$
repeat
  for *each ordered pair of adjacent nodes* X *and* Y **do**
    if $|adj(X, \mathcal{G}) \setminus \{Y\}| \geq d$ **then**
      for *each subset* $S \subseteq adj(X, \mathcal{G}) \setminus \{Y\}$ *and* $|S| = d$ **do**
        Test $CI(X, Y|S)$ on significance level α
        if $CI(X, Y|S)$ **then**
          Remove edge between X and Y
          Save S as separating set of (X, Y)
          Update $\mathcal{G}$ and $\mathcal{E}$
          break
        end
      end
    end
  end
  $d \leftarrow d + 1$
until $|adj(X, \mathcal{G}) \setminus \{Y\}| < d$ *for every pair of adjacent nodes*;

now: update $\mathcal{G}$ and $\mathcal{E}$ at the end of level d and store the adjaceny sets of all nodes for the search of the right S

Limitation

Disadvantages:

- Adjacency sets and set S are possibly larger than in the original PC

  → more CI tests necessary

- Even more increased runtime than for the original PC

Possible Solution: Parallel PC-algorithm

Agenda

1) Basic Definitions
2) Introduction to the PC-Algorithm
3) Limitations of the PC-Algorithm
4) The stable PC-Algorithm
5) **The parallel PC-Algorithm**
6) Simulation Study in R
7) Final Notes and Discussion

Idea

Parallelism:

- Break down big task into several different smaller subtasks
- Distribute them over different cores of the computer’s CPU

Idea

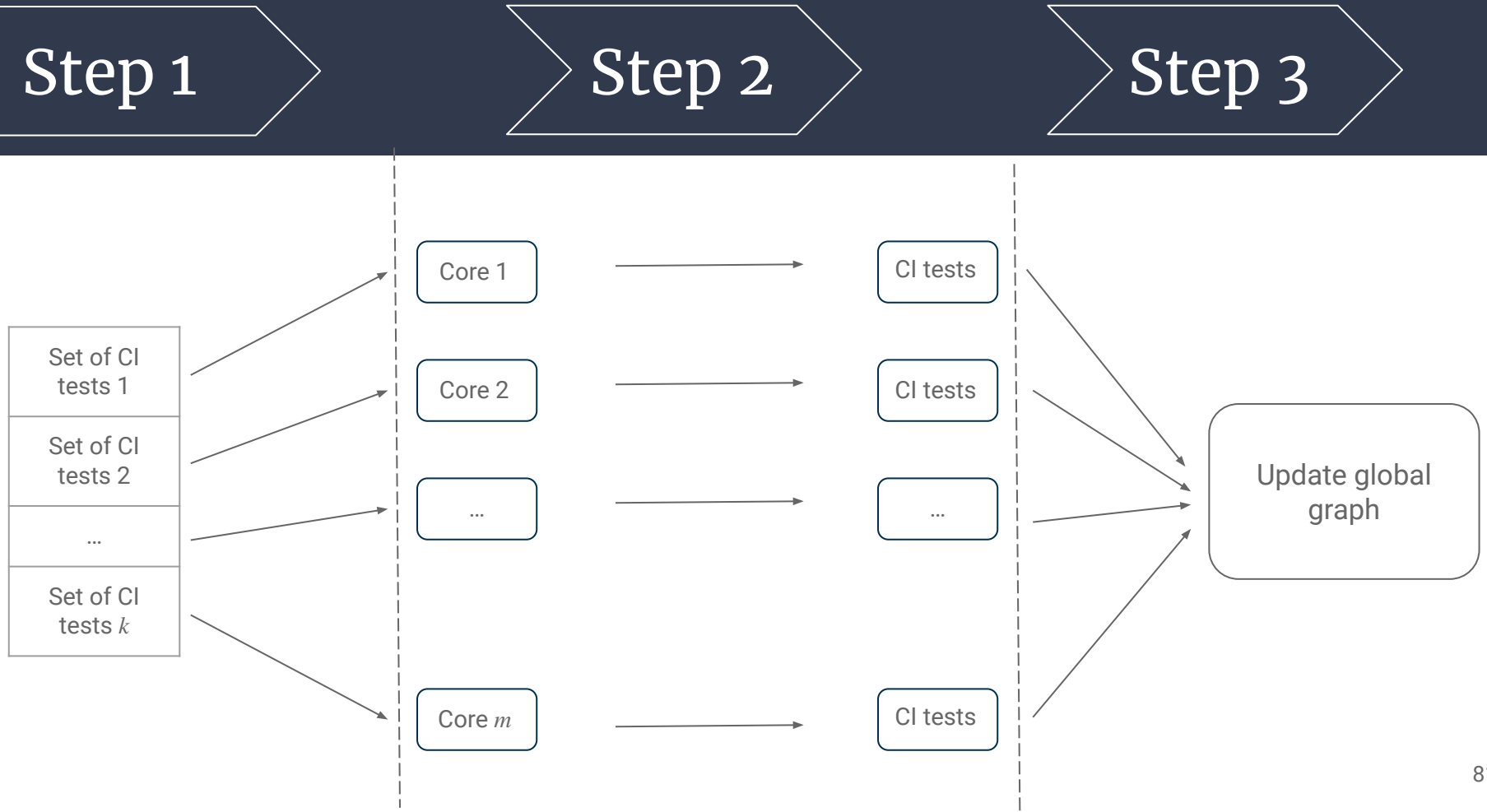
Parallelism:

- Break down big task into several different smaller subtasks
- Distribute them over different cores of the computer's CPU

Prerequisite: subtasks need to be independent

→ But: CI test results of particular level d influence CI test results of following level d + 1

→ Idea: parallelize CI tests within each level


Step 1
Step 2
Step 3
Set of CI tests 1
Set of CI tests 2
...
Set of CI tests k
Core 1
Core 2
...
Core m
CI tests
CI tests
...
CI tests
Update global graph

Agenda

1) Basic Definitions
2) Introduction to the PC-Algorithm
3) Limitations of the PC-Algorithm
4) The stable PC-Algorithm
5) The parallel PC-Algorithm
6) **Simulation Study in R**
7) Final Notes and Discussion

Comparison of all previously discussed algorithms

Data simulation → CPDAG construction → Evaluation

Repeat the following steps:

1. Simulate random DAG $\mathscr{G}$
   - dimension p
   - sparsity s
2. Draw random samples from $\mathscr{G}$
   - sample size n
   - distribution

Run original/stable/parallel PC-algorithm with different settings for α = 0.01 and 10 simulations:

- dimension p
- sample size n
- ordering

Comparison of:

- Run time in s
- Structural Hamming Distance
- True Positive Rate
- False Positive Rate
- True Discovery Rate

Reference: Kalisch et al., 2012

Evaluation measures

Structural Hamming Distance (SHD)	Number of edge insertions, deletions or flips in order to transform **estimated** CPDAG to **true** CPDAG (or vice versa)
True Positive Rate (TPR)	Number of correctly found edges in **estimated** CPDAG divided by number of true edges in **true** CPDAG
False Positive Rate (FPR)	Number of incorrectly found edges in **estimated** CPDAG divided by number of true gaps in **true** CPDAG
True Discovery Rate (TDR)	Number of correctly found edges divided by number of found edges both in **estimated** CPDAG

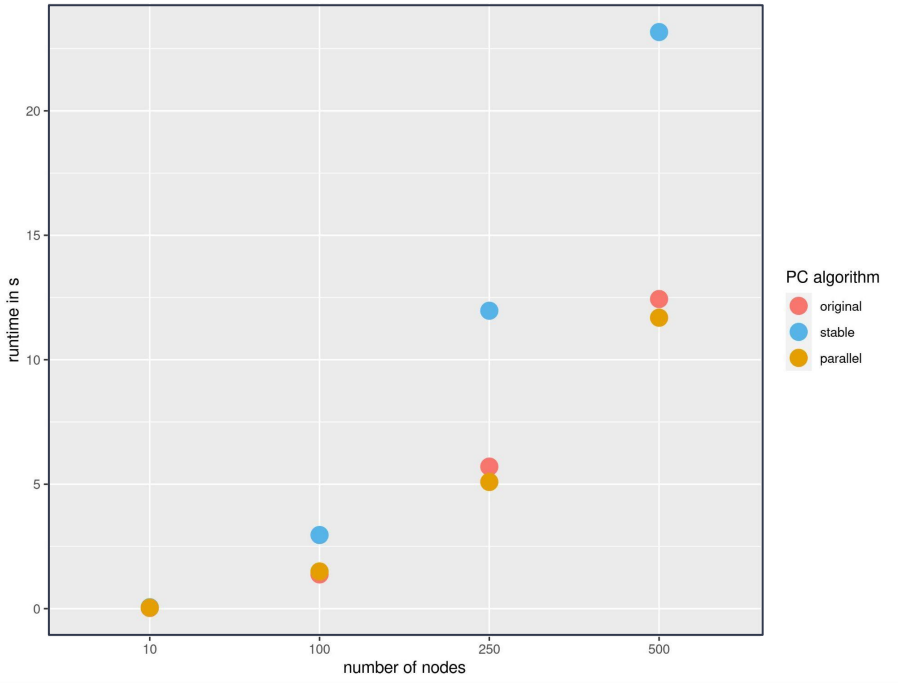
Reference: CRAN-documentation of *pcalg* package, 2022

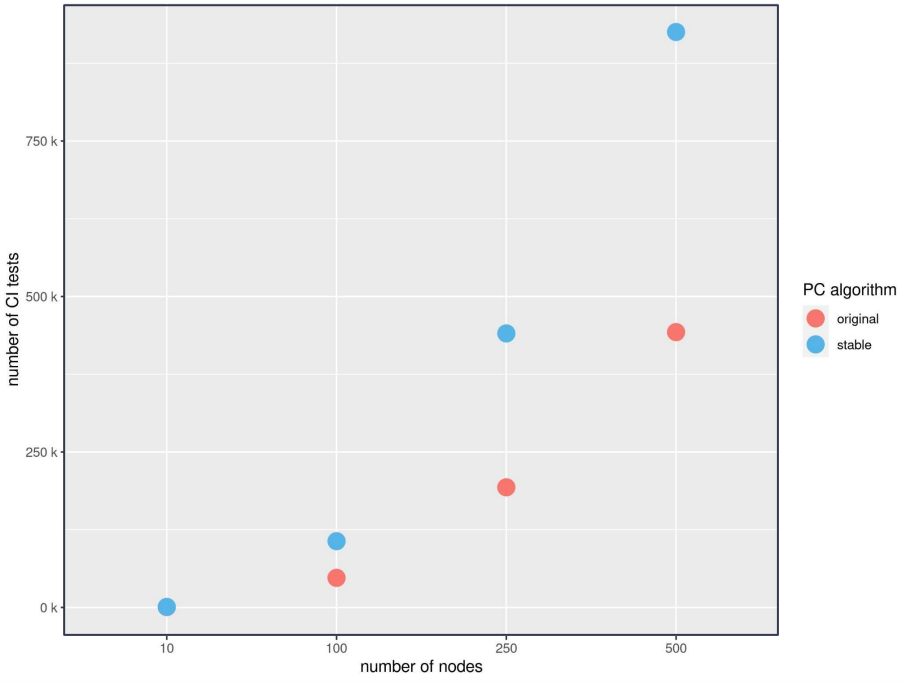
Setting 1: different dimensions

n = 1000

$p \in \{10, 100, 250, 500\}$

expected neighborhood size: 5



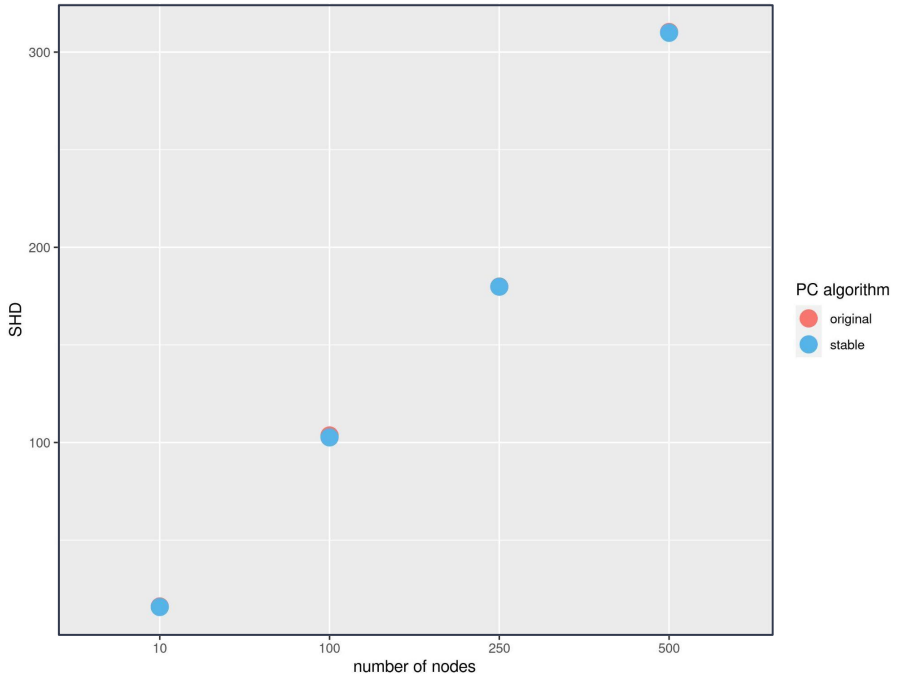


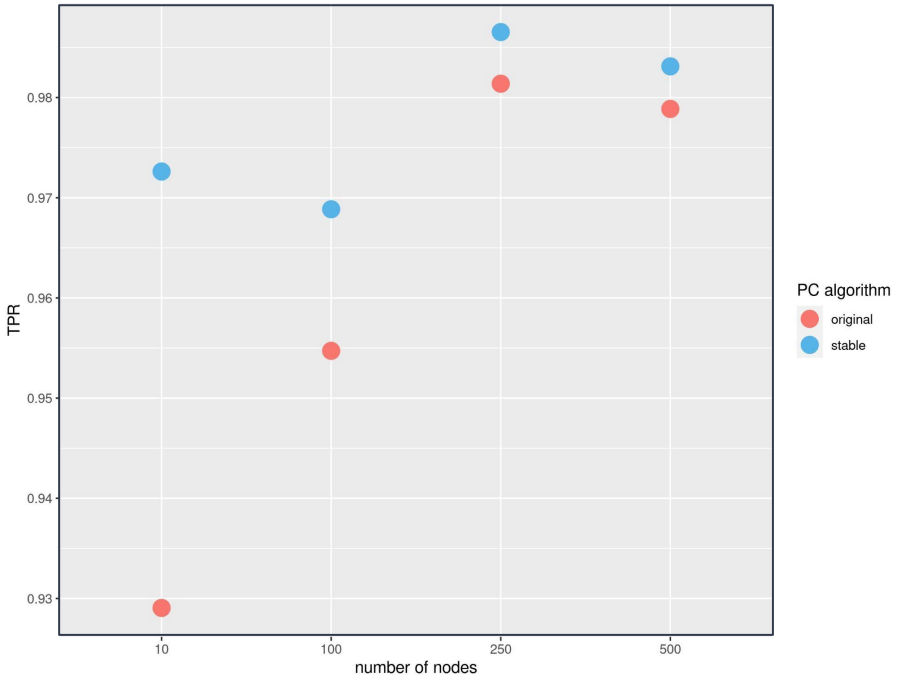
Setting 1: different dimensions

n = 1000

$p \in \{10, 100, 250, 500\}$

expected neighborhood size: 5



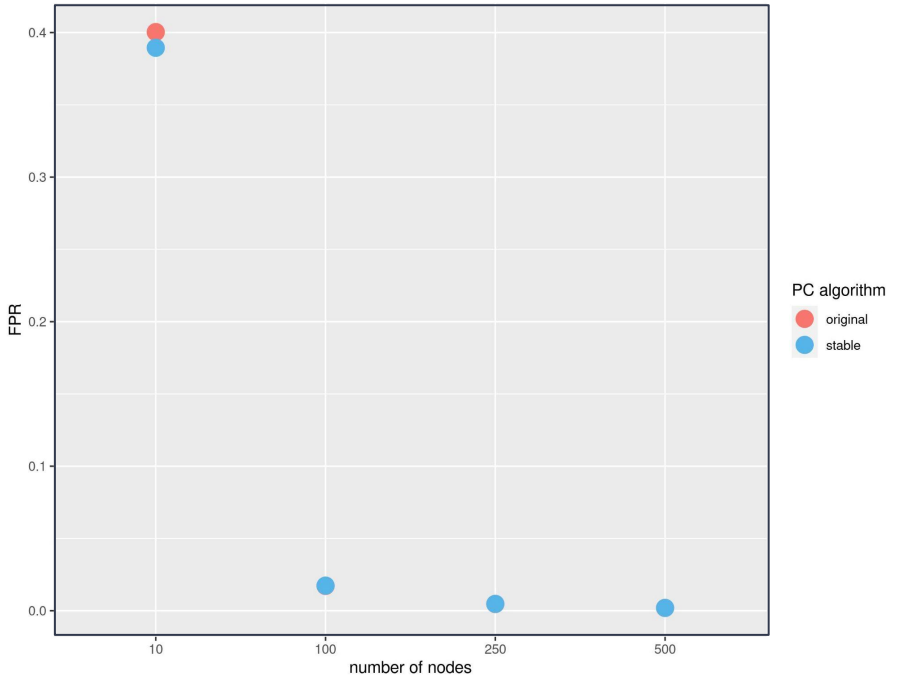


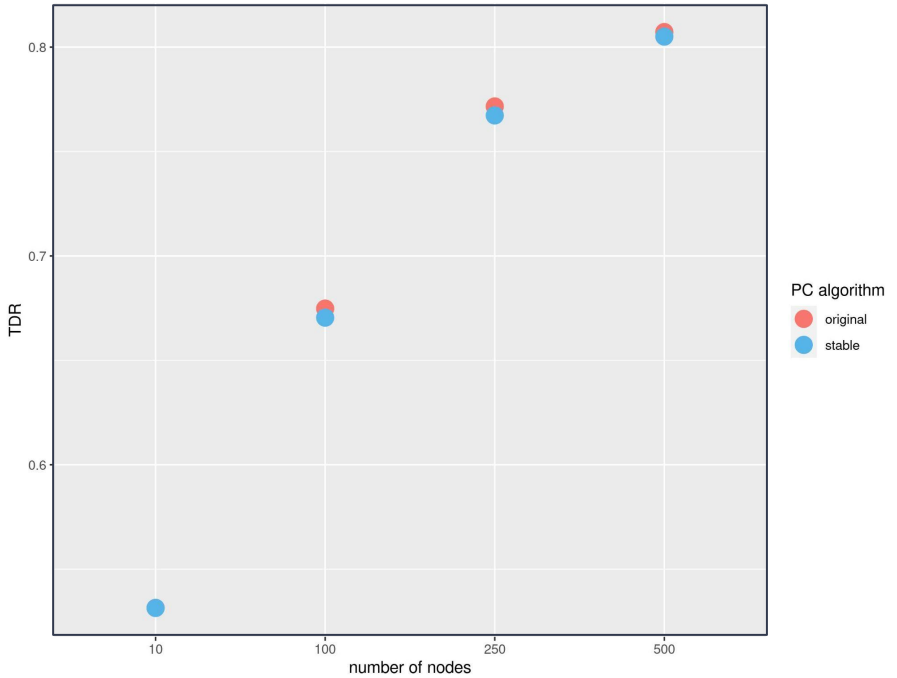
Setting 1: different dimensions

n = 1000

$p \in \{10, 100, 250, 500\}$

expected neighborhood size: 5



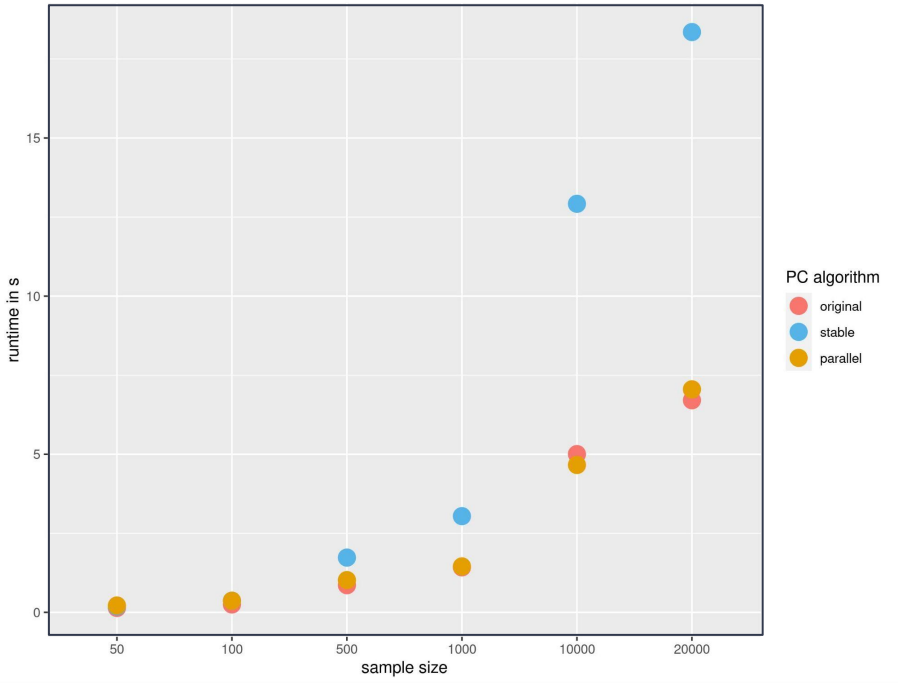


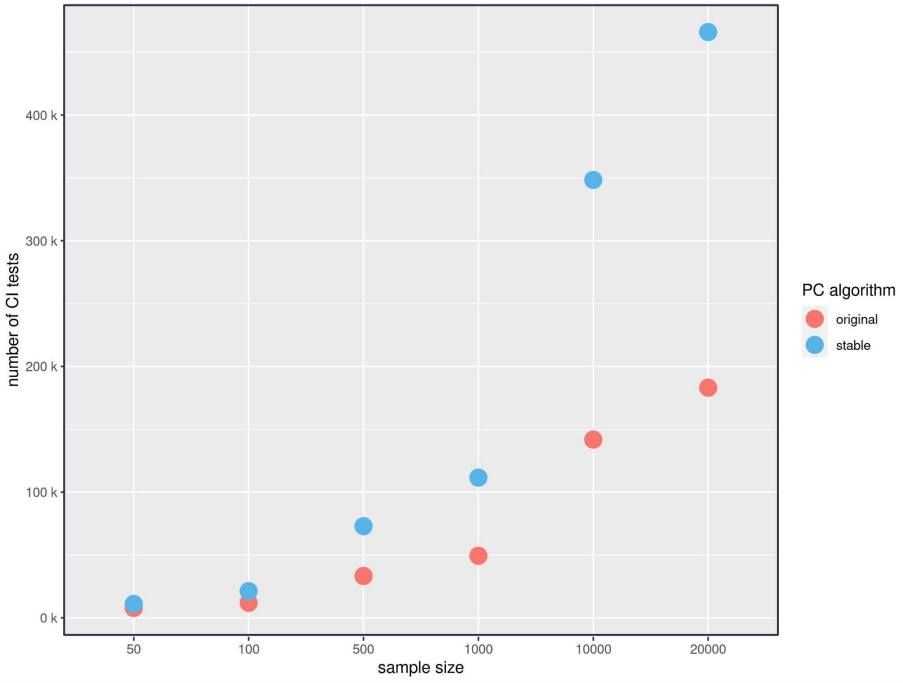
Setting 2: different sample sizes

$n \in \{50, 100, 500, 1k, 10k, 20k\}$

$p = 100$

expected neighborhood size: 5



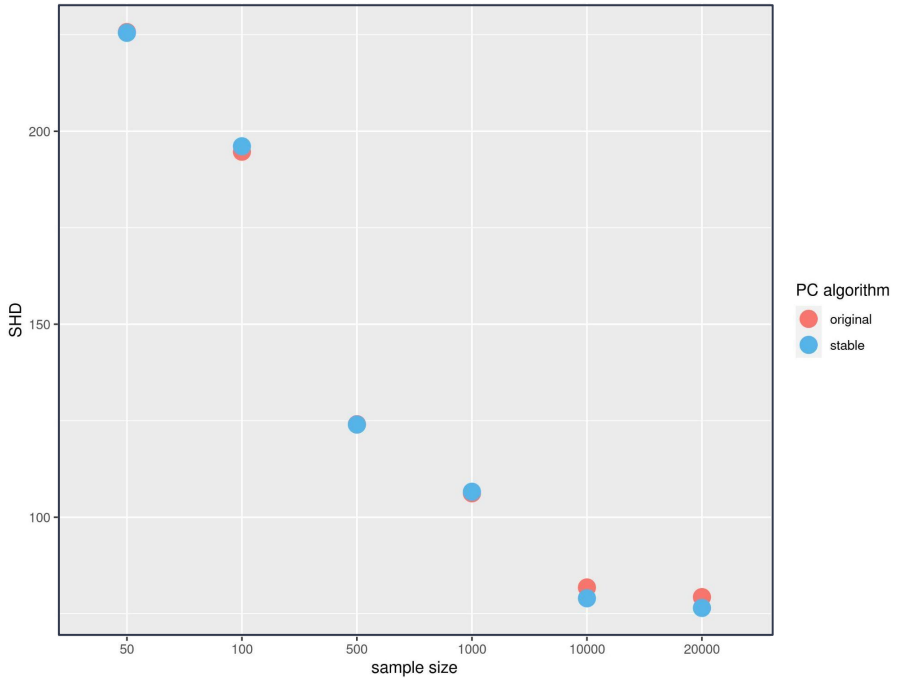


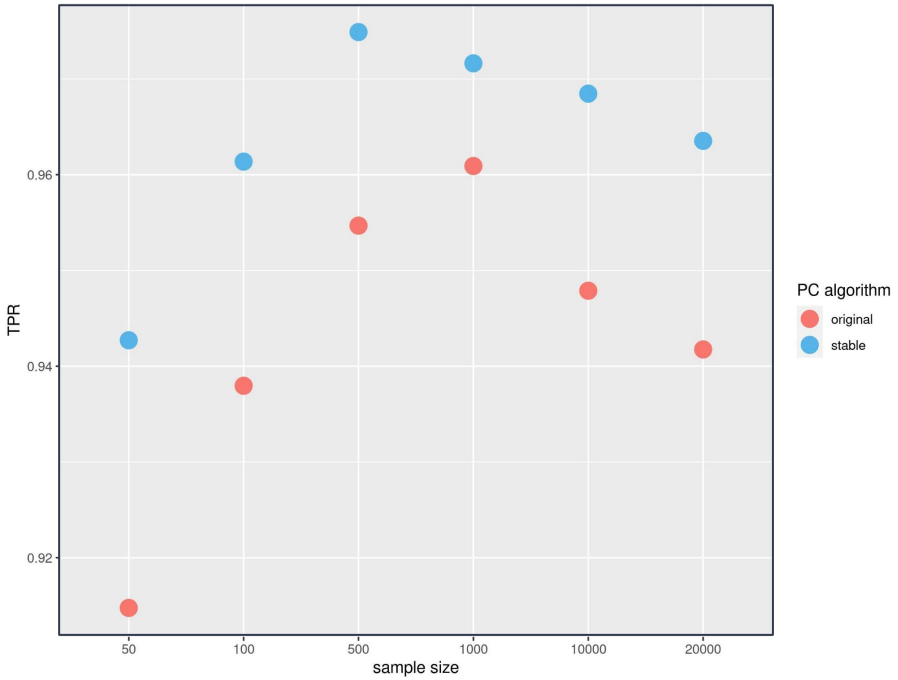
Setting 2: different sample sizes

$n \in \{50, 100, 500, 1k, 10k, 20k\}$

$p = 100$

expected neighborhood size: 5



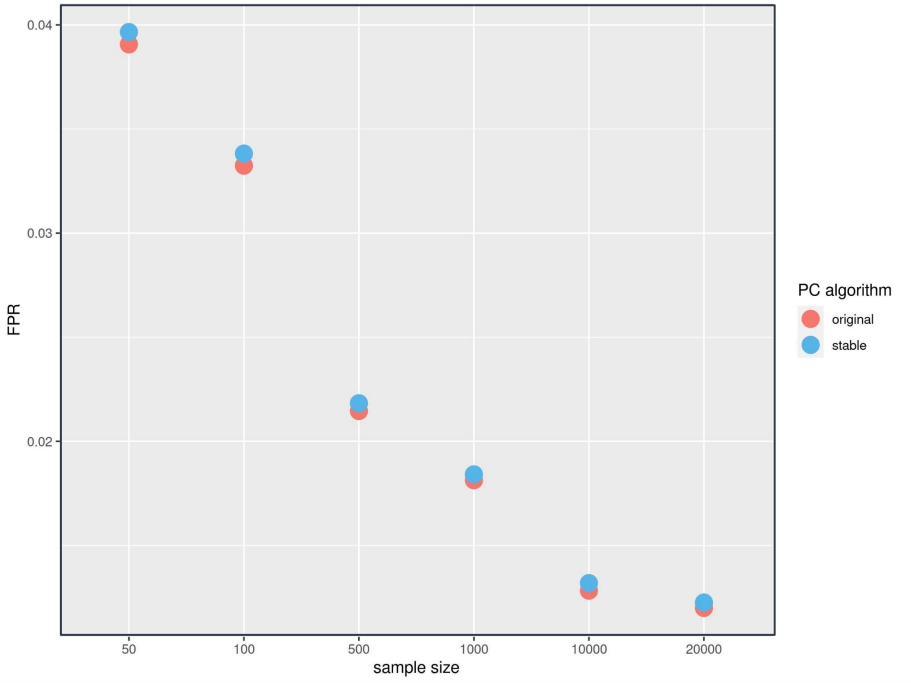


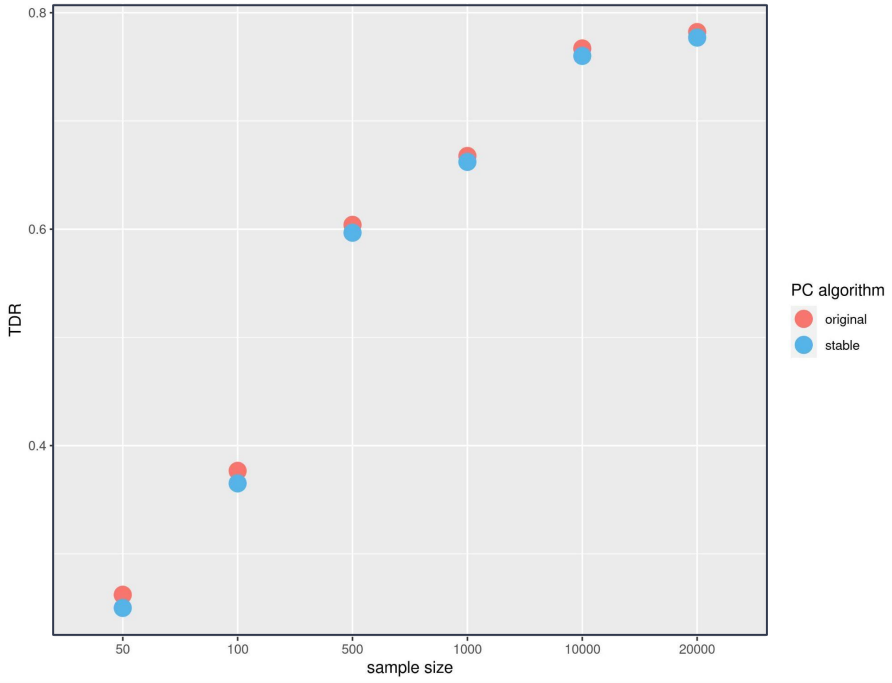
Setting 2: different sample sizes

$n \in \{50, 100, 500, 1k, 10k, 20k\}$

$p = 100$

expected neighborhood size: 5



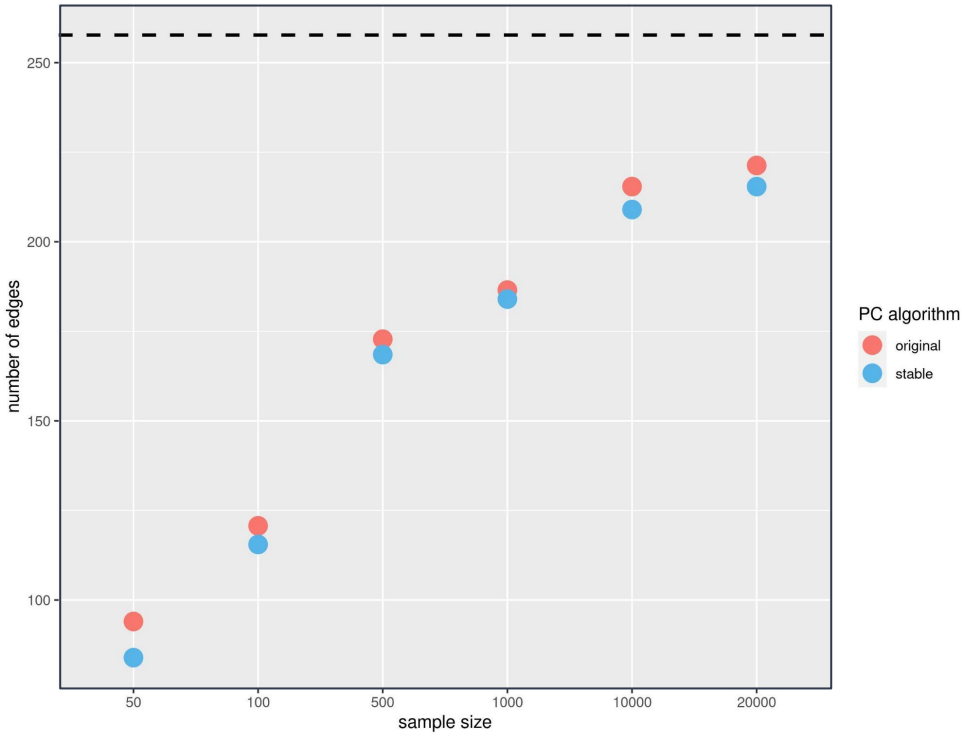


Setting 2: different sample sizes

$n \in \{50, 100, 500, 1k, 10k, 20k\}$

$p = 100$

number of simulations: 10

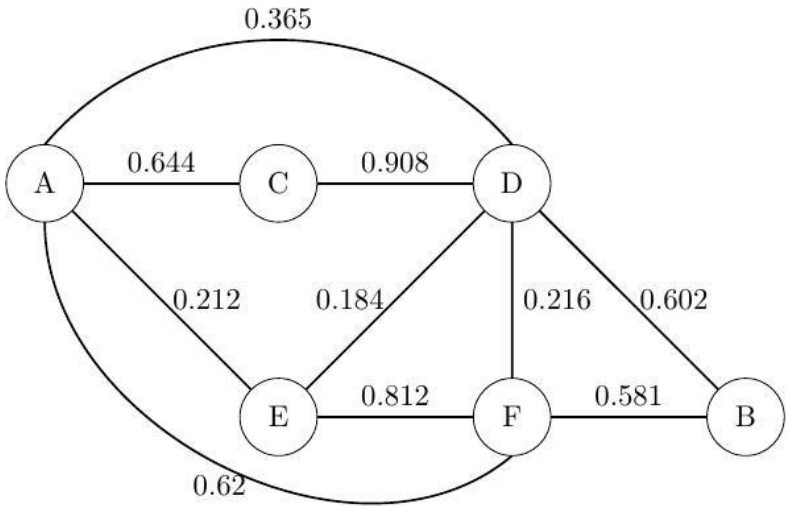


Setting 3: different ordering

Parameter settings:

- sample size: n = 20
- number of nodes: p = 6
- expected neighborhood size: 3
- significance level: α = 0.05
- 4 different orderings of variables

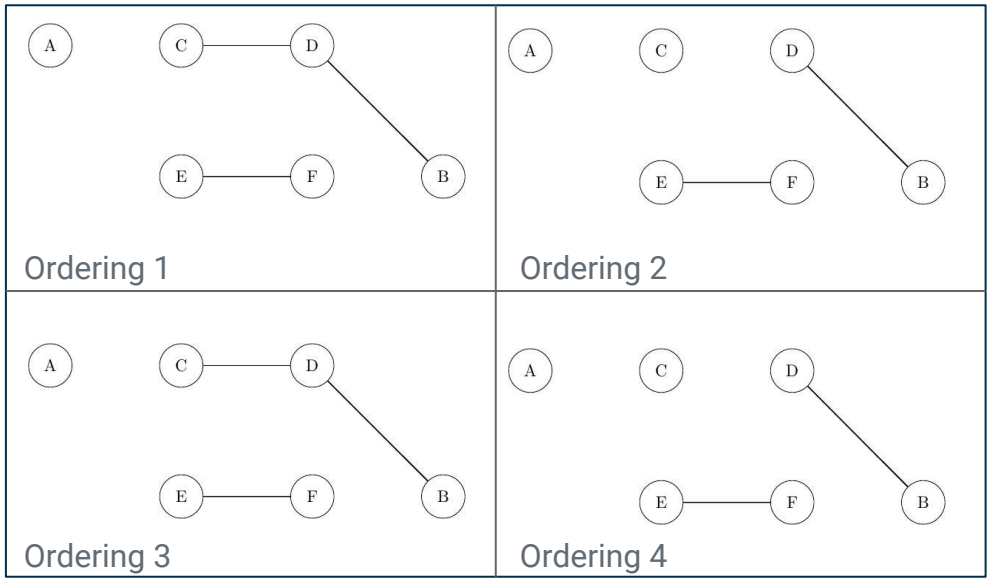
→ Compare different outcomes of original and stable PC-algorithm

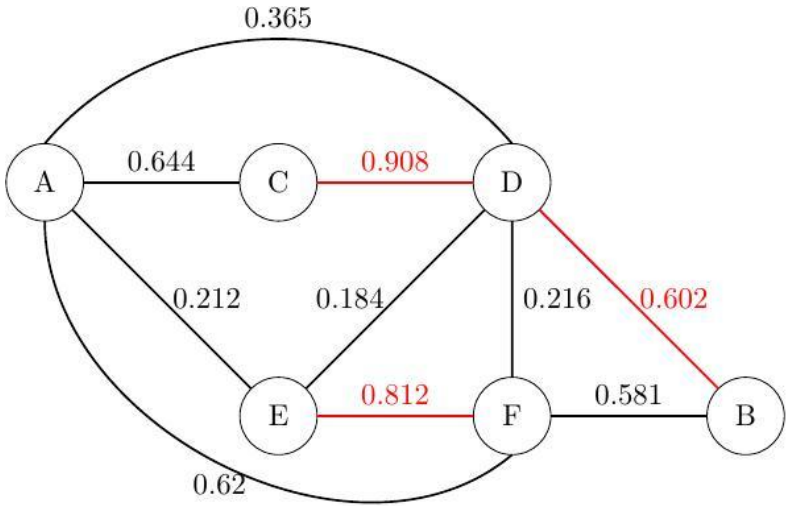


true weighted skeleton

Setting 3: different ordering

Results of original PC-algorithm

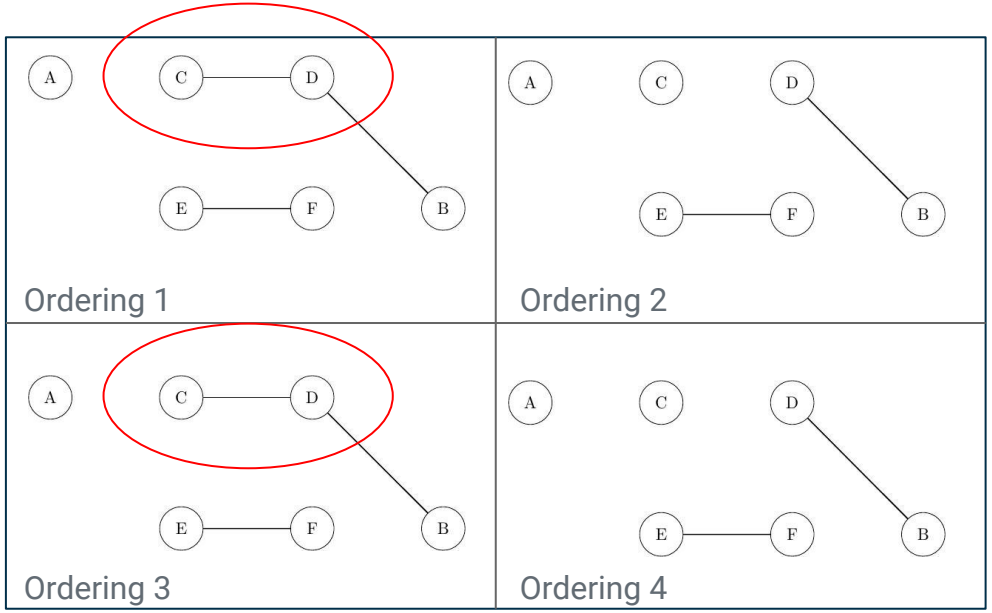


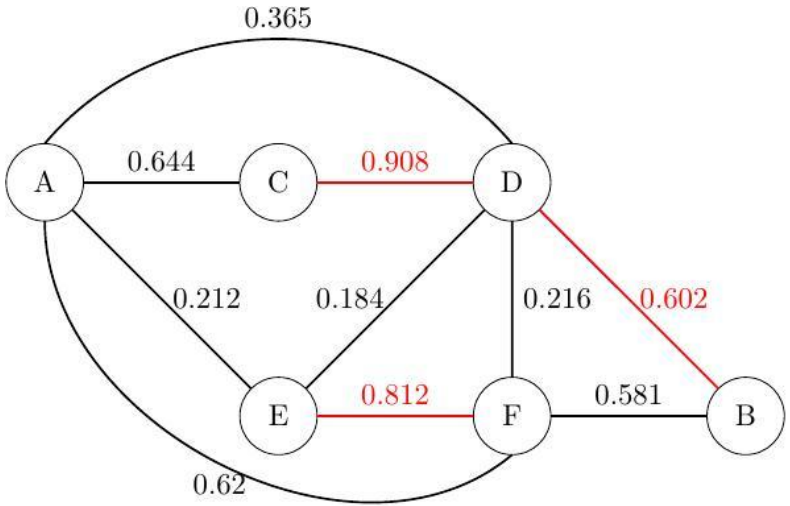


true weighted skeleton

Setting 3: different ordering

Results of original PC-algorithm

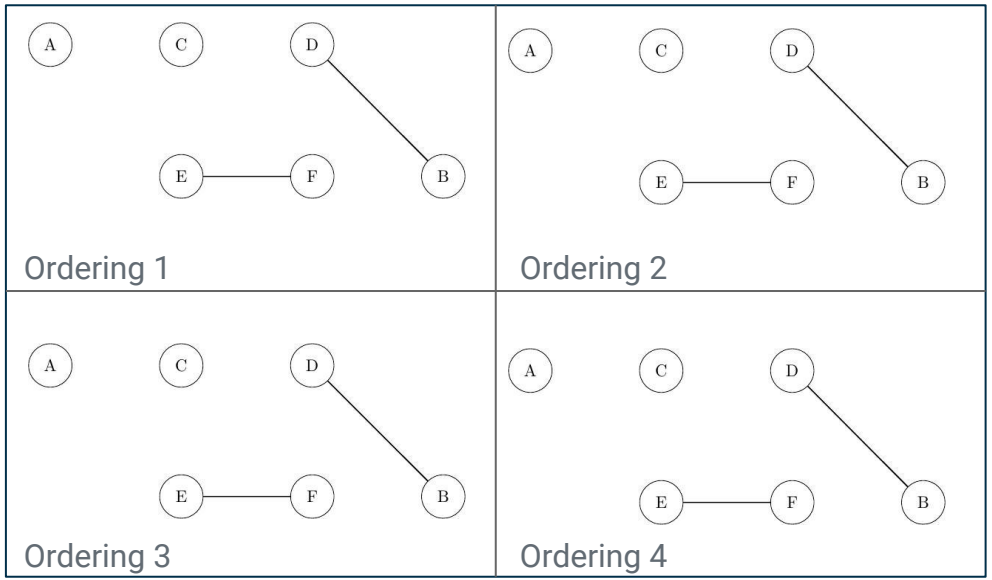


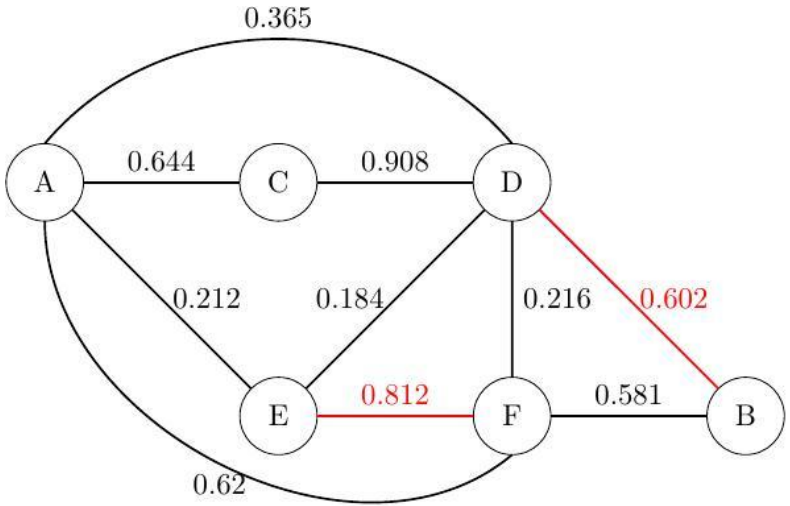


true weighted skeleton

Setting 3: different ordering

Results of stable PC-algorithm





true weighted skeleton

Setting 3: different ordering

Results for higher dimensions:

- sample size: n = 100
- number of nodes: p = 50
- significance level: α = 0.05
- sparsity: s = 0.06
- expected neighborhood size: 3
- 10 different orderings

ordering	1	2	3	4	5	6	7	8	9	10
original	106	108	110	110	110	108	110	110	110	106
stable	102	102	102	102	102	102	102	102	102	102

Table 1: Number of edges in the estimated skeleton for 10 different orderings estimated by original and stable PC-algorithm.

Results of the simulation study

- **Runtime:**
  - increasing the dimensionality and increasing the sample size increases the runtime
  - parallel PC is a lot faster than stable PC (especially with high number of cores)
  - original PC: similar to parallel because of less CI tests
- **Dimension:** increasing the dimensionality improves TPR, FPR and TDR, but SHD increases, too
- **Sample size:** increasing the sample size increases the goodness of fit except for TPR

Agenda

1) Basic Definitions
2) Introduction to the PC-Algorithm
3) Limitations of the PC-Algorithm
4) The stable PC-Algorithm
5) The parallel-PC-Algorithm
6) Simulation Study in R
7) **Final Notes and Discussion**

Conclusion

Goal of PC-algorithm: detecting conditional independence relationships between nodes in a DAG whose underlying data-generating distribution is faithful and multivariate normal and outputs the corresponding CPDAG

Ordering

Problem:

Output graph is dependent on the order in which CI tests are performed

Solution: Stable PC-algorithm

Runtime

Problem:

Many CI tests necessary

→ high runtime

Solution: Parallel PC-algorithm

Gaussian Distribution

Problem:

Consistency only holds for Gaussian distribution

→ what if it is non-Gaussian?

Solution: Rank PC-algorithm

References

Harris N., Drton M. (2013). *PC Algorithm for Nonparanormal Graphical Models,* Journal of Machine Learning Research, 14, p. 3365-3383.

Le T.D., Hoang T., Li J., Liu L., Liu H., Hu S. (2014). *A fast PC algorithm for high dimensional causal discovery with multi-core PCs,* Journal of LaTeX class files, Vol. 13, No. 9.

Koller D., Friedman N. (2009). *Probabilistic Graphical Models: Principles and Techniques,* MIT Press, Cambridge.

Sprites P., Glymour C., Scheines R. (2000). *Causation, Prediction, and Search,* 2nd edition, MIT Press, Cambridge.

Kalisch M., Bühlmann P. (2007). *Estimating High-Dimensional Directed Acyclic Graphs with the PC-Algorithm,* Journal of Machine Learning Research, 8, p. 613-636.

Colombo D., Maathuis, M.H. (2012). *A modification of the PC algorithm yielding order-independant skeletons,* ArXiv e-prints.

Kalisch M., Mächler M., Colombo D., Maathuis, M.H., Bühlmann P. (2012). *Causal Inference Using Graphical Models with the R package pcalg,* Journal of Statistical Software, Vol. 47, Issue 11.

Backup

Faithfulness

Let the nodes $\mathcal{N}$ represent random variables $\boldsymbol{X} = (X_N)_{N \in \mathcal{N}}$.

The joint distribution $\mathcal{P}$ of $\boldsymbol{X}$ is **faithful** to DAG $\mathcal{G}$ if for any triple of pairwise disjoint subsets $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{S} \subseteq \mathcal{N}$, we have that $\boldsymbol{S}$ d-separates $\boldsymbol{A}$ and $\boldsymbol{B}$ if and only if $X_{\boldsymbol{A}}$ and $X_{\boldsymbol{B}}$ are conditionally independent given $X_{\boldsymbol{S}}$, i.e. $X_{\boldsymbol{A}} \perp X_{\boldsymbol{B}} \mid X_{\boldsymbol{S}}$.

→ not a strict assumption in practice as the majority of distributions is faithful

→ non-faithful distributions of the multivariate Gaussian family form a Lebesgue null-set in the space of distributions associated with a DAG

→ under faithfulness, statistical tests of conditional independence can be used to determine d-separation relations in a DAG

Assumptions

- Data-generating distribution is *faithful*

  → conditional independence relations can be read off the graph

- All nodes $\mathcal{N}$ correspond to multivariate *gaussian* distributed random variables

  → conditional independence can be inferred from partial correlations:

For a multivariate normal distributed random vector $\boldsymbol{X} \in \mathbb{R}^p$ denote its partial correlation between components $\boldsymbol{X}^{(i)}$ and $\boldsymbol{X}^{(j)}$ $(i \neq j)$ given set $\{\boldsymbol{X}^{(r)}, r \in \boldsymbol{R} \subseteq \{1, \dots, p\} \setminus \{i, j\}\}$ by $\rho_{i,j \mid \boldsymbol{R}}$.

Then it holds:

$\rho_{i,j \mid \boldsymbol{R}} = 0$ if and only if $\boldsymbol{X}^{(i)}$ and $\boldsymbol{X}^{(j)}$ are *conditionally independent* given $\{\boldsymbol{X}^{(r)}, r \in R\}$.

The CI test in detail

Recursive estimation of $\rho_{i,j \mid \boldsymbol{r}}$ (Kalisch and Bühlmann, 2007):

$\rho_{i,j \mid \varnothing}$: regular correlation coefficient

For some $t \in R$:

$$\rho_{i,j \mid \boldsymbol{R} \setminus t} = \frac{\rho_{i,j \mid \boldsymbol{R} \setminus t} - \rho_{i,t \mid \boldsymbol{R} \setminus t}\, \rho_{j,t \mid \boldsymbol{R} \setminus t}}{\sqrt{(1 - \rho^2_{i,t \mid \boldsymbol{R} \setminus t})\,(1 - \rho^2_{j,t \mid \boldsymbol{R} \setminus t})}}$$

(iteratively increase set R)

Idea of the rank PC-algorithm (RPC)

Replace the standard Pearson-type correlation by **rank-based** measures of correlation in the CI tests:

1) Estimate Spearman's ρ:

$$\hat{\rho}^S = \frac{\frac{1}{n}\sum_{i=1}^{n}\left(\frac{\mathrm{rank}(X_i)}{n+1}-\frac{1}{2}\right)\left(\frac{\mathrm{rank}(Y_i)}{n+1}-\frac{1}{2}\right)}{\sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\frac{\mathrm{rank}(X_i)}{n+1}-\frac{1}{2}\right)^2}\sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\frac{\mathrm{rank}(Y_i)}{n+1}-\frac{1}{2}\right)^2}} = 1-\frac{6}{n\left(n^2-1\right)}\sum_{i=1}^{n}\left(\mathrm{rank}(X_i)-\mathrm{rank}(Y_i)\right)^2$$

2) Estimate the corresponding correlation matrix
3) Determine partial correlations
4) Test for conditional independence

} same procedure as in original PC-algorithm

Consistency for high-dimensional data

Theorem 3 (Kalisch and Bühlmann, 2007) :

Under the assumption that the data are realisations from independent *multivariate* Gaussian random vectors s.t. their distribution is *faithful* to a suitably *sparse* DAG $\mathcal{G}$ and an additional regularity assumption, both step 1 and step 2 of the PC-algorithm are consistent for high-dimensional data, i.e. there exists $\alpha_n \to 0$ $(n \to \infty)$ and $\beta_n \to 0$ $(n \to \infty)$ such that

$$\mathbb{P}(\mathcal{G}_{\text{skel},n,\text{ PC}}(\alpha_n) = \mathcal{G}_{\text{skel},n,\text{ true}}) = 1 - \mathcal{O}(\exp(-Cn^{1-2d})) \to 1 \quad (n \to \infty) \text{ for some } 0 < C < \infty \text{ and } 0 < d < 0.5$$

and

$$\mathbb{P}(\mathcal{G}_{\text{CPDAG, PC}}(\alpha_n) = \mathcal{G}_{\text{CPDAG, true}}) = 1 - \mathcal{O}(\exp(-Cn^{1-2d})) \to 1 \quad (n \to \infty) \text{ for some } 0 < C < \infty \text{ and } 0 < d < 0.5$$

This result holds for all three presented PC-algorithms.

In Theory: the “perfect” significance level α

Kalisch and Bühlmann, 2007, showed in their proof of the consistency of the PC-algorithm that in theory

$$\alpha = \alpha_n = 2\,(1 - \Phi(n^{0.5}\, c_n / 2))$$

which depends on the unknown lower bound c_n of partial correlations

$$\inf\{|\rho_{i,j\,|\,\boldsymbol{R}}|\colon i, j, R \text{ with } \rho_{i,j\,|\,\boldsymbol{R}} \neq 0\} \geq c_n.$$

That means that increasing the sample size we should ideally decrease the significance level.

Runtime of the different PC-algorithms

- original PC (Kalisch and Bühlmann, 2007):
  - worst case: exponential in number of nodes
  - but: if true underlying DAG is sparse, polynomial in number of nodes (often reasonable assumption in practice)
- stable PC:
  - especially for higher dimensions number of CI tests increases exponentially (see simulation study)

    → even higher runtime
- parallel PC:
  - simulation study shows that parallel PC has best runtime of all three for higher dimensions

Sketch: Proof of Theorem 1

Suppose that A and B are not adjacent, and A is not an ancestor of B. Let the total order *Ord* on the variables in $\mathcal{G}$ be such that all ancestors of A and all ancestors of B except for A are prior to A, and all other nodes are after A.

Then $S(\mathcal{G}, A)$ for order *Ord* is a subset of D-SEP(A, B). Hence one can show that if B is not in D-SEP(A, B) then D-SEP(A, B) d-separates A from B in G. B is in D-SEP(A, B) if and only if there is a path from A to B in which each node except the endpoints is a collider on the path, and each node on the path is an ancestor of A or B.

But then there is an inducing path between A and B, and one can show that A and B are adjacent, contrary to the assumption.

Reference: Sprites et al., 2000

Sketch: Proof of Theorem 2

- Faithfulness assumption → conditional independence in joint distribution of the nodes is equivalent to the d-separation relations
- Determination of the skeleton: two variables in a DAG are d-separated by a subset of the remaining variables if and only if they are d-separated by their parent nodes
- PC-algorithm is guaranteed to check these conditional interdependencies: at any point in the algorithm the current graph is a supergraph of the true CPDAG and the algorithm checks conditional independencies given all subsets of the adjacency sets (which include the parent sets)

Reference: Colombo and Maathuis, 2012

The parallel PC-Algorithm

Three-stage process within each level d :

Step 1: Distribute CI tests evenly among the cores by grouping CI tests of same edge together.

Step 2: Each core performs its set of CI tests in parallel with other cores.

Step 3: Integrate CI test results from all cores into global graph.

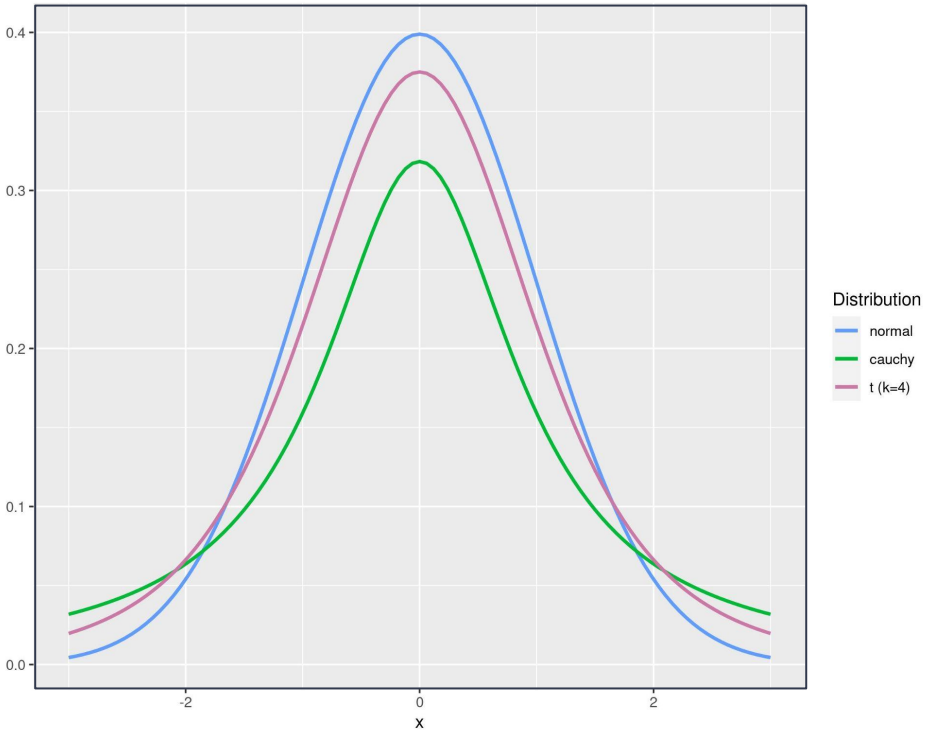
Possible extension for big data sets: additional memory-efficient indicator to detect free memory

Setting: different distributions

$n \in \{100, 1000\}$

$p \in \{10, 100\}$

expected neighborhood size: 3



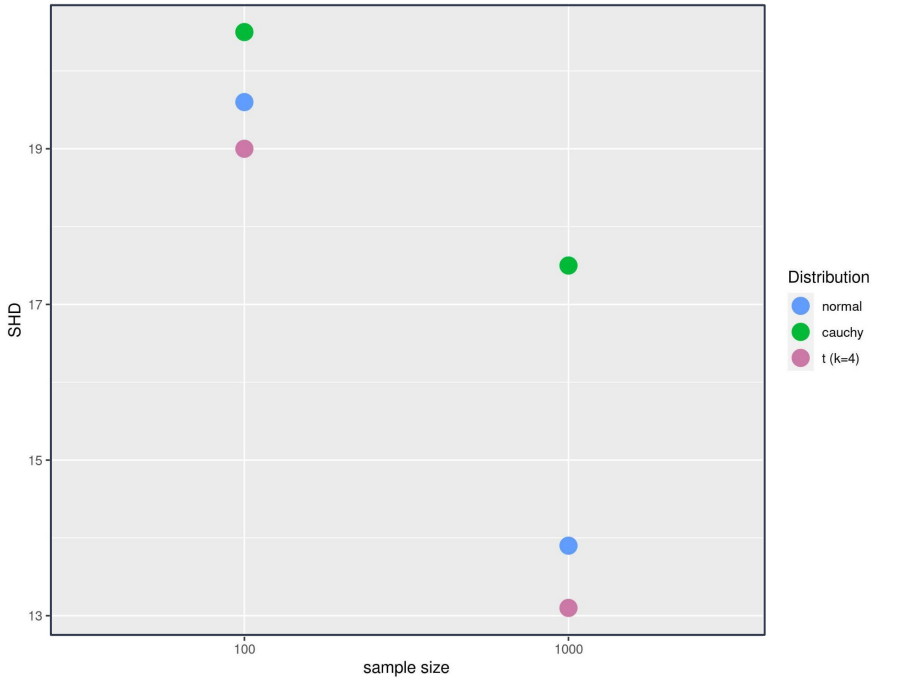
Setting: different distributions

$n \in \{100, 1000\}$

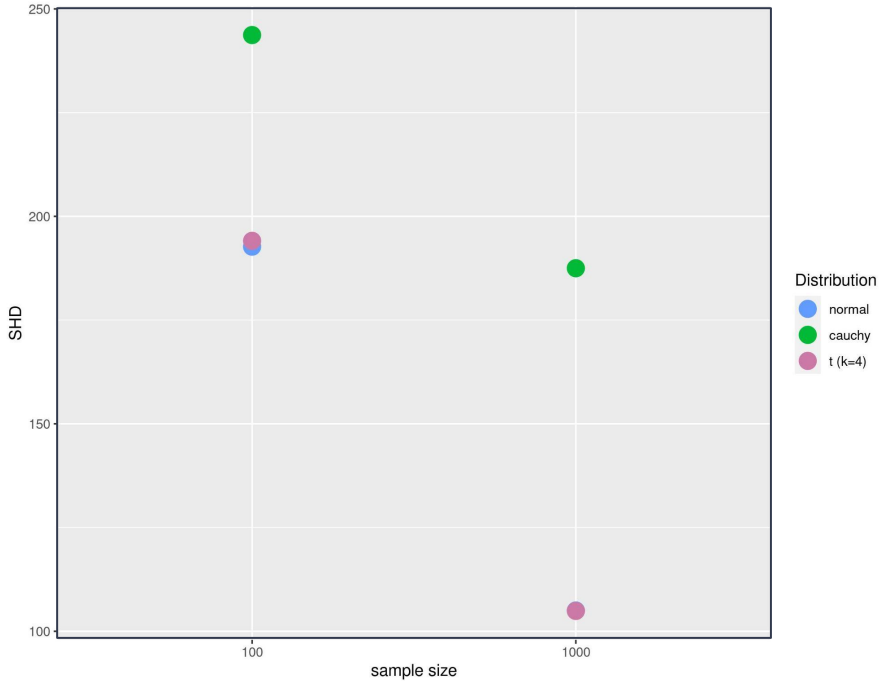
$p \in \{10, 100\}$

expected neighborhood size: 3

p = 10



p = 100

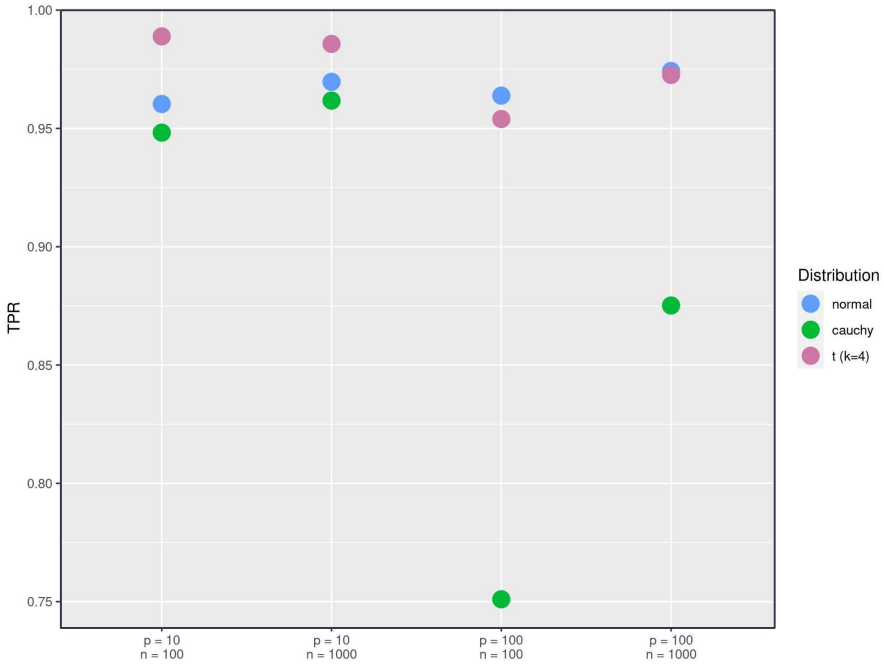


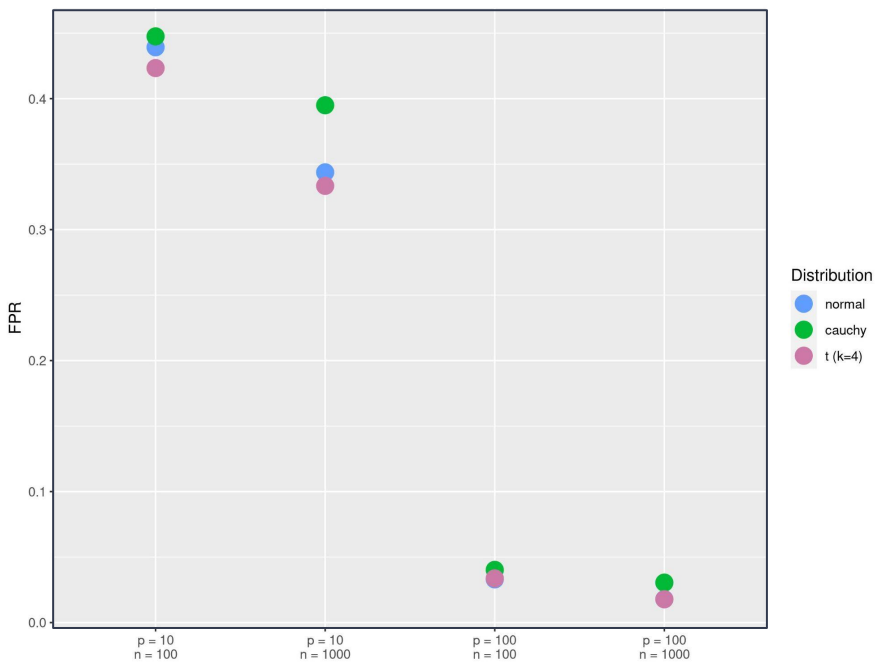
Setting: different distributions

$n \in \{100, 1000\}$

$p \in \{10, 100\}$

expected neighborhood size: 3



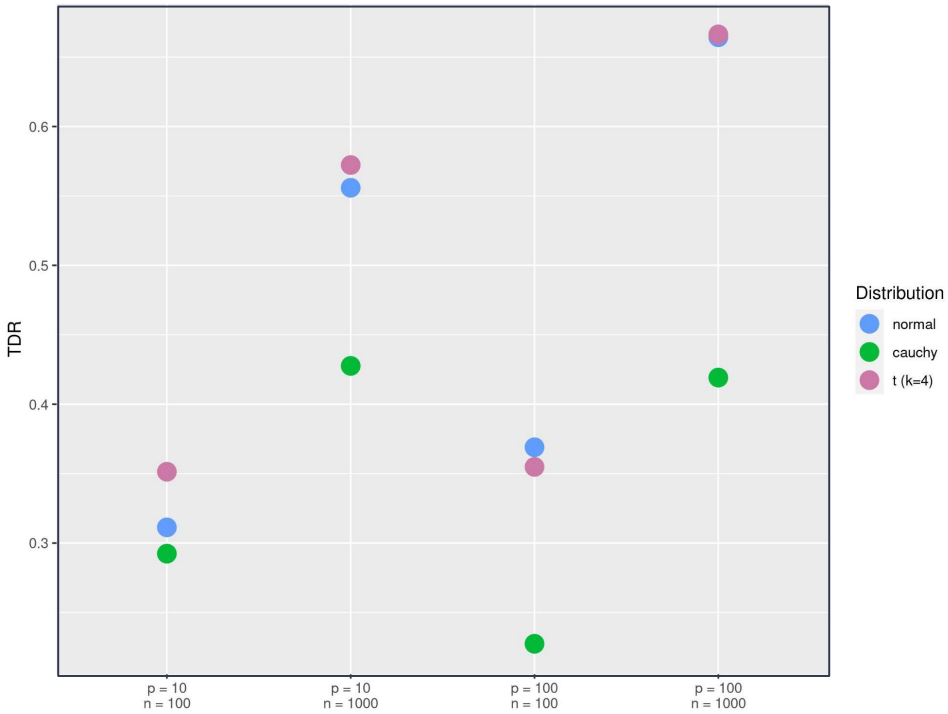


Setting: different distributions

$n \in \{100, 1000\}$

$p \in \{10, 100\}$

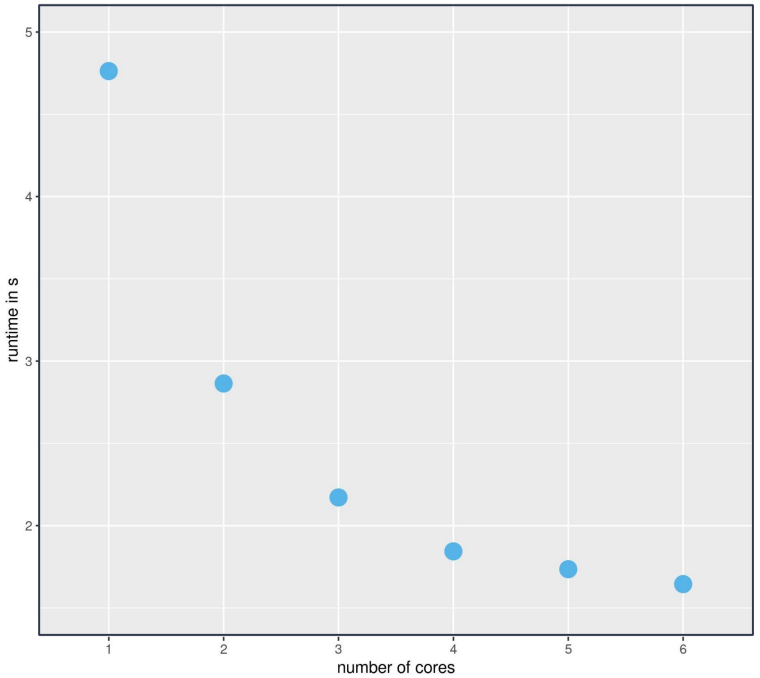
expected neighborhood size: 3



Setting: different number of cores

Parameter settings for parallel PC-algorithm:

- sample size: n = 1000
- number of nodes: p = 100
- expected neighborhood size: 30
- number of cores: 1 to 6



Conclusion

- The PC-algorithm detects conditional independence relationships between nodes in a DAG whose underlying data-generating distribution is faithful and multivariate normal.
- The output is the CPDAG, the representant of the Markov equivalence class of the true graph, if the conditional independence relationships are known.
- Two-Step-Computation via learning the skeleton and orienting edges
- Limitations of the original PC:
    - ordering → stable PC: update edge set and graph in the end of each level, not in between
    - runtime → parallel PC: run stable PC on multiple cores
    - normal distribution → rank PC: adapt CI test to be more robust

Possible extensions of the PC-algorithm

- **Conservative PC-algorithm** (CPC) by Ramsey et al. (1995): weaker form of faithfulness
- **Fast Causal Inference algorithm** (FCI) by Spirtes et al. (1999): Learn a Markov equivalence class of DAGs with latent and selection variables
- **Cyclic causal discovery** (CCD) by Richardson (1996): Learn Markov equivalence classes of directed (not necessarily acyclic) graphs under the assumption of causal sufficiency
- **Score-based** and **hybrid** methods

Reference: Causal Inference - Theory and Applications in Enterprise Computing by Dr. Matthias Uflacker, Johannes Huegle, Christopher Schmidt, 2019