Proceedings of Machine Learning Research vol TBD:1–40, 2023 2nd Conference on Causal Learning and Reasoning

Local Causal Discovery for Estimating Causal Effects

Shantanu Gupta SHANTANG@CMU.EDU
David Childers DCHILDERS@CMU.EDU
Zachary C. Lipton ZLIPTON@CMU.EDU
Carnegie Mellon University

Editors: Mihaela van der Schaar, Dominik Janzing and Cheng Zhang

Abstract

Even when the causal graph underlying our data is unknown, we can use observational data to narrow down the possible values that an average treatment effect (ATE) can take by (1) identifying the graph up to a Markov equivalence class; and (2) estimating that ATE for each graph in the class. While the PC algorithm can identify this class under strong faithfulness assumptions, it can be computationally prohibitive. Fortunately, only the local graph structure around the treatment is required to identify the set of possible ATE values, a fact exploited by local discovery algorithms to improve computational efficiency. In this paper, we introduce Local Discovery using Eager Collider Checks (LDECC), a new local causal discovery algorithm that leverages unshielded colliders to orient the treatment's parents differently from existing methods. We show that there exist graphs where LDECC exponentially outperforms existing local discovery algorithms and vice versa. Moreover, we show that LDECC and existing algorithms rely on different faithfulness assumptions, leveraging this insight to weaken the assumptions for identifying the set of possible ATE values.

Keywords: Local causal discovery, local structure learning, causal inference, causal discovery

1. Introduction

Estimating an average treatment effect (ATE) from observational data typically requires structural knowledge, which can be represented in the form of a causal graph. While a rich literature offers methods for identifying and estimating causal effects given a *known* causal graph (Tian and Pearl, 2002; Pearl, 2009; Jaber et al., 2019), many applications require that we investigate the values that an ATE could possibly take when the causal graph is unknown. In such cases, we can (i) perform causal discovery, using observational data to identify the graph up to a Markov equivalence class (MEC); and (ii) estimate the desired ATE for every graph in the MEC, thus identifying the set of possible ATE values. We denote this (unknown) set of identified ATE values by Θ^*.

Causal discovery has been investigated under a variety of assumptions (Glymour et al., 2019; Squires and Uhler, 2022). Under causal sufficiency (i.e., no unobserved variables) and faithfulness, the PC algorithm (Spirtes et al., 2000, Sec. 5.4.2) can identify the MEC of the true graph from observational data (and thus Θ^*). However, fully characterizing the MEC can be computationally expensive. Addressing this concern, Maathuis et al. (2009) proved that the local structure around the treatment node is sufficient for identifying Θ^*. Leveraging this insight, existing local causal discovery algorithms (e.g., PCD-by-PCD (Yin et al., 2008), MB-by-MB (Wang et al., 2014)) discover just enough of the graph to identify any parents and children of the treatment that PC would have discovered. These methods sequentially discover the local structure around the treatment, its neighbors, and so on, terminating whenever all neighbors of the treatment are oriented (or no remaining neighbors can be oriented).

© 2023 S. Gupta, D. Childers & Z.C. Lipton.

In this work, we introduce Local Discovery with Eager Collider Checks (LDECC), a new local causal discovery algorithm that provides an alternative method to orient the parents of a treatment X (Sec. 4). Initially, LDECC performs local discovery around X to discover its neighbors. Subsequently, LDECC chooses the same Conditional Independence (CI) tests as PC would choose given the state of the graph, with one crucial exception: whenever we find two nodes A and B such that $A \perp\!\!\!\perp B|\mathbf{S}$ for some set $\mathbf{S}$ with $X \notin \mathbf{S}$, LDECC immediately checks whether they become dependent when X is added to the conditioning set. If the test reveals dependence $A \not\perp\!\!\!\perp B|\mathbf{S} \cup \{X\}$, then X must either be a collider or a descendant of a collider that lies at the intersection of some path from A to X or from B to X. On this basis, LDECC can orient the smallest subset of X's neighbors that d-separate it from $\{A, B\}$ as parents. We prove that, under faithfulness, the identified ATE set is equal to Θ^*.

We represent the ideas underlying existing local causal discovery algorithms using a simple algorithm that we call Sequential Discovery (SD) that sequentially runs the PC algorithm locally for local structure learning. While existing algorithms differ from SD in subtle ways, they share key steps with SD allowing us to compare LDECC to this existing class of algorithms. We highlight LDECC's complementary strengths to SD in terms of computational requirements. We present classes of causal graphs where LDECC performs exponentially fewer CI tests than SD, and vice versa (Sec. 4.1). Thus, the methods can be combined profitably (by running LDECC and SD in parallel and terminating when either algorithm terminates), avoiding exponential runtimes if either algorithm's runtime is subexponential, thereby expanding the class of graphs where efficient local discovery is possible.

We also compare SD and LDECC based on their faithfulness requirements (Sec. 4.2). We show that both SD and PC require weaker assumptions to identify Θ^* than to identify the entire MEC. We also find that LDECC and SD rely on different sets of faithfulness assumptions. There are classes of faithfulness violations where one algorithm will correctly identify Θ^* while the other will not. Under the assumption that one of the algorithms' faithfulness assumptions is correct, we propose a procedure that recovers a conservative bound on the ATE set. Aiming to make this bound sharp, we prove that LDECC and SD can be combined to construct a procedure that can identify Θ^* under strictly weaker assumptions, again highlighting LDECC's complementary nature relative to existing methods. Finally, we empirically test LDECC on synthetic as well as semi-synthetic graphs (Sec. 5) and show that it performs comparably to SD (and PC) and typically runs fewer CI tests than SD [1].

2. Related Work

Several works have developed procedures for identifying the set of possible ATE values Θ^* using local information. Maathuis et al. (2009) propose IDA, an algorithm that outputs a set of ATE values by only using the local structure around the treatment X: For a given MEC, Θ^* can be identified by adjusting for all possible parent sets of X such that no new unshielded colliders get created at X. IDA has been extended to account for multiple interventions (Nandy et al., 2017), background information (Perković et al., 2017; Fang and He, 2020), and hidden variables (Malinsky and Spirtes, 2016). Hyttinen et al. (2015) combine causal discovery and inference by presenting a SAT solver-based approach to do-calculus with an unknown graph. Geffner et al. (2022) propose a deep learning based end-to-end method that learns a posterior over graphs and estimates the ATE by marginalizing over the graphs. Toth et al. (2022) propose a Bayesian active learning approach to jointly learn a posterior over causal models and some target query of interest. However, these works do not propose local discovery methods.

1. The code and data are available at https://github.com/acmi-lab/local-causal-discovery.

Other works attempt to perform causal inference under weaker assumptions than requiring the entire graph or MEC. Some works propose methods to find valid adjustment sets under the assumption that the observed covariates are pre-treatment (De Luna et al., 2011; VanderWeele and Shpitser, 2011; Entner et al., 2012, 2013; Witte and Didelez, 2019; Gultchin et al., 2020). Cheng et al. (2022a) use an anchor variable (which they call COSO) and Shah et al. (2022) use a known causal parent of the treatment to identify a valid adjustment set. Other works present data-driven methods to find valid instrumental variables (Silva and Shimizu, 2017; Cheng et al., 2022b). Watson and Silva (2022) propose a method to learn the causal order amongst variables that are descendants of some covariate set. By contrast, we do not make any partial ordering assumptions.

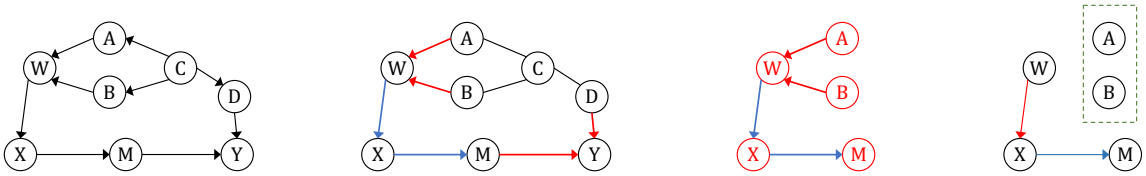
A complementary line of work focuses on only discovering the local structure around a given node. Many works attempt to find the Markov blanket (MB) or the parent-child set of a target node (Aliferis et al., 2003; Tsamardinos et al., 2006; Yu et al., 2020; Ling et al., 2021). Unlike our work, they do not distinguish between the parent and child identities of nodes inside the MB (see Aliferis et al. (2010) for a review of these methods). For determining these causal identities, existing works use sequential approaches by repeatedly finding local structures like a MB or a parent-child set starting from the target node and then propagating edge orientations within the discovered subgraph (Yin et al., 2008; Zhou et al., 2010; Wang et al., 2014; Gao and Ji, 2015; Ling et al., 2020). These methods typically differ in how they discover the local structure around each node. Cooper (1997) present an algorithm to learn pairwise causal relationships by leveraging a node that is not caused by any other nodes. Some works use independence patterns in Y-structures around a given node to find local causal relationships (Mani et al., 2012; Mooij et al., 2015; Versteeg et al., 2022).

3. Preliminaries

We assume that the causal structure of the observational data can be encoded using a DAG $\mathcal{G}^*(\mathbf{V}, \mathbf{E})$, where $\mathbf{V}$ and $\mathbf{E}$ are the set of nodes and edges. Each edge $A \to B \in \mathbf{E}$ indicates that A is a direct cause of B. We denote the treatment node by X and the outcome node by Y. For a node V, we denote its Markov blanket, neighbors, parents, children, and descendants by $\text{MB}(V)$, $\text{Ne}(V)$, $\text{Pa}(V)$, $\text{Ch}(V)$, and $\text{Desc}(V)$ (with $\text{MB}(V) = \text{Ne}(V) \cup_{N \in \text{Ch}(V)} \text{Pa}(N)$). Let $\text{Ne}^+(V) = \text{Ne}(V) \cup \{V\}$ and $\text{MB}^+(V) = \text{MB}(V) \cup \{V\}$. An unshielded collider (UC) is a triple $P \to R \leftarrow Q$ s.t. P—$Q \notin \mathbf{E}$. For a UC $\alpha = (P \to R \leftarrow Q)$, let $\text{sep}(\alpha) = \min\{|\mathbf{S}| : \mathbf{S} \subseteq \mathbf{V} \setminus \{P, Q\} \text{ and } P \perp\!\!\!\perp Q | \mathbf{S}\}$.

A DAG entails a set of CIs via *d-separation* (Pearl, 2009, Sec. 1.2.3). DAGs that entail the same set of CIs form an MEC, which can be characterized by a *completed partially directed acyclic graph* (CPDAG). For the true DAG $\mathcal{G}^*$, we denote by Θ^* the set of ATE values (of X on Y) in each DAG in the MEC corresponding to $\mathcal{G}^*$. The causal faithfulness assumption (CFA) holds iff all CIs satisfied by the observational joint distribution $\mathbb{P}(\mathbf{V})$ are entailed by $\mathcal{G}^*$. Throughout this work, we focus on causally sufficient graphs. Under the CFA, the PC algorithm recovers this MEC roughly as follows (demonstrated for the DAG in Fig. 1(*a*)): (i) estimate the skeleton by running CI tests; (ii) find UCs in this skeleton (Fig. 1(*b*) red edges); and (iii) orient additional edges using Meek's rules (Meek, 2013) (Fig. 1(*b*) blue edges) to get a CPDAG (full details on the PC algorithm are in Appendix A).

We instantiate the key ideas of existing local discovery algorithms using SD (Fig. 3). SD sequentially finds the local structure around nodes (*LocalPC* in Fig. 2) starting from X, then its neighbors, and so on (Lines 3–7). After each such local discovery step, SD orients nodes in the subgraph discovered until that point using UCs and Meek's rules like PC (Line 8; see Fig. 12 for *GetCPDAG* and Fig. 15 for *Nbrs*). If at any point all neighbors of X get oriented, SD terminates.



(*a*) True DAG $\mathcal{G}^*$ (*b*) CPDAG from PC (*c*) SD's output (*d*) LDECC

Figure 1: Demonstration of the PC, SD, and LDECC algorithms for the graph in (a).

```
def LocalPC(Skeleton 𝒰, Target V):
    s ← 0;
    while |Ne_𝒰(V)| > s do
        for B ∈ Ne_𝒰(V), S ⊆ Ne_𝒰(V) \ {B}
          s.t. |S| = s do
            if V ⫫ B|S then
                𝒰.removeEdge(V—B);
                DSep(V, B) ← S;
                break;
        end
        s ← s + 1;
    end
    return 𝒰, Ne_𝒰(V), DSep;
def isLocallyValid(X, S):
    for every A, B ∈ S do
        if isNonCollider(A—X—B) then
            return False;
    end
    return True;
```

Figure 2: Additional subroutines.

```
   Input: Treatment X.
1  Fully connected undirected graph 𝒰;
2  queue ← [X], done ← ∅;
3  while queue is not empty do
4      V ← queue.removeFirstItem();
5      𝒰, Ne_𝒰(V), DSep ← LocalPC(𝒰, V);
6      done ← done ∪ {V};
7      queue.append(Ne_𝒰(V) \ (done ∪ queue));
8      𝒢 ← GetCPDAG(𝒰[done], DSep);
9      parents, children, unoriented ← Nbrs(𝒢, X);
10     if unoriented = ∅ then break ;
11 end
12 Θ_SD ← ∅;
13 for S ⊆ (unoriented ∪ parents) do
14     if isLocallyValid(X, S) then
15         Θ_SD.add(θ_{X→Y|S});
16 end
17 return Θ_SD;
```

Figure 3: The SD algorithm.

The ATE set is estimated by applying the backdoor adjustment (Pearl, 2009, Thm. 3.3.2) with every *locally valid* parent set of X, i.e., one that does not create a new UC at X (see Lines 12–16 and *isLocallyValid* in Fig. 2). The ATE estimated using each such parent set $\mathbf{S}$ is denoted as $\theta_{X\to Y|\mathbf{S}}$ (Line 15). Consider the DAG in Fig. 1(*a*). Here, SD will use *LocalPC* to sequentially discover the neighbors of X, W, M, A, and B. Then the UC $A \to W \leftarrow B$ will be detected and, after propagating orientations via Meek's rules, both W and M will be oriented and SD will terminate (Fig. 1(*c*)) and output $\Theta_{\text{SD}} = \{\theta_{X\to Y|\{W\}}\} = \Theta^*$.

4. Local Discovery using Eager Collider Checks (LDECC)

In this section, we propose LDECC, a local causal discovery algorithm that orients the parents of X by leveraging UCs differently from SD (Prop. 4). We prove its correctness under the CFA (Thm. 5) and show its complementary nature relative to SD in terms of computational (Sec. 4.1) and

```
def UCChildren(MB(X), Ne(X), DSep):
    spousesX ← MB(X) \ Ne(X);
    children ← ∅;
    for D ∈ spousesX do
        for C ∈ Ne(X) \ DSep(D, X) do
            if C ⊥̸⊥ D|DSep(D, X) then
                children.add(C) ;
        end
    end
    return children;
def GetMNS(V ∉ Ne⁺(X)):
    s ← 0;
    while |Ne(X)| > s do
        for S ⊆ Ne(X) s.t. |S| = s do
            if V ⊥⊥ X|S then  return S ;
        end
        s ← s + 1;
    end
    return MNS not found;
def ECCParents(A, B, check=False):
    if check and {A, B} ∩ Ne(X) = ∅ then
        m_A ← GetMNS(A) ;
        m_B ← GetMNS(B);
        if m_A = m_B then return m_A;
        else return ∅;
    parents ← ∅;
    for V ∈ {A, B} do
        if V ∈ Ne(X) then parents.add(V);
        else parents.add(GetMNS(V));
    end
    return parents;
```

Figure 4: Subroutines used by LDECC.

```
Input: Treatment X.
MB(X) ← FindMarkovBlanket(X);
_, Ne(X), DSep ← LocalPC(MB⁺(X), X);
children ← UCChildren(MB(X), Ne(X), DSep);
parents ← ∅, unoriented ← ∅;
Completely connected undirected graph 𝒰;
for (A ⊥⊥ B|S) ∈ PCTest(𝒰) s.t. A, B ≠ X do
    if A, B ∈ Ne(X) and X ∉ S then
        parents.add({A, B});
        parents.add(S ∩ Ne(X));
    else if A, B ∈ Ne(X) and X ∈ S then
        MarkNonCollider(A—X—B);
        for V ∈ Ne(X) \ (S ∪ {A, B}) do
            if A ⊥̸⊥ B|{S, V} then
                children.add(V) ;
        end
    else if A ⊥̸⊥ B|{S, X} and X ∉ S then
        parents.add(ECCParents(A, B, S));
    end
    for P ∈ parents, C ∈ unoriented do
        if isNonCollider(P—X—C) then
            children.add(C) ;
    end
    unoriented ← Ne(X)\ (parents ∪ children);
    if unoriented = ∅ then  break ;
end
Θ_LDECC ← ∅;
for S ⊆ (unoriented ∪ parents) do
    if isLocallyValid(S) then
        Θ_LDECC.add(θ_{X→Y|S});
end
Output: Θ_LDECC
```

Figure 5: The LDECC algorithm.

faithfulness (Sec. 4.2) requirements. We defer the proofs to Appendix B.1. We first define a *Minimal Neighbor Separator* (MNS) which plays a key role in LDECC.

Definition 1 (Minimal Neighbor Separator) *For a DAG $\mathcal{G}(\mathbf{V}, \mathbf{E})$ and nodes X and $A \notin Ne^{+}(X)$, $mns_X(A) \subseteq Ne(X)$ is the unique set (see Prop. 3) of nodes such that (i) (d-separation) $A \perp\!\!\!\perp X | mns_X(A)$, and (ii) (minimality) for any $\mathbf{S} \subset mns_X(A)$, $A \not\perp\!\!\!\perp X|\mathbf{S}$.*

For the graph in Fig. 1(*a*), we have $\text{mns}_X(A) = \text{mns}_X(B) = \text{mns}_X(C) = \{W\}$ and $\text{mns}_X(Y) = \{W, M\}$. While an MNS need not exist for every node (see Example 6 in Appendix B), $\text{mns}_X(V)$ always exists $\forall V \notin \text{Desc}(X)$ (which is sufficient for correctness of LDECC):

Proposition 2 *For any node $V \notin (Desc(X) \cup Ne^{+}(X))$, $mns_X(V)$ exists and $mns_X(V) \subseteq Pa(X)$.*

Proposition 3 (Uniqueness of MNS) *For every node V such that $mns_X(V)$ exists, it is unique.*

Proposition 4 (Eager Collider Check) *For nodes $A, B \in \mathbf{V} \setminus Ne^{+}(X)$ and $\mathbf{S} \subseteq \mathbf{V} \setminus \{A, B, X\}$, if (i) $A \perp\!\!\!\perp B|\mathbf{S}$; and (ii) $A \not\perp\!\!\!\perp B|\mathbf{S} \cup \{X\}$; then $A, B \notin Desc(X)$ and $mns_X(A), mns_X(B) \subseteq Pa(X)$.*

Prop. 4 suggests a different strategy for orienting parents of X: if two nodes that are d-separated by $\mathbf{S}$ become d-connected by $\mathbf{S} \cup \{X\}$, the MNS of such nodes contains the parents of X (see proof in Appendix B.1). Thus, unlike PC and SD, the skeleton is not needed to propagate orientations via Meek's rules. Similar ideas have been used in prior work for causal discovery. Claassen and Heskes (2012) use minimal (in)dependencies to develop a logic-based causal discovery algorithm that also avoids graphical orientation rules. Magliacane et al. (2016) extend this and propose an algorithm that only reasons over ancestral relations rather than the space of DAGs. While these works do not study local causal discovery, their results can be used to derive an alternative proof of Prop. 4.

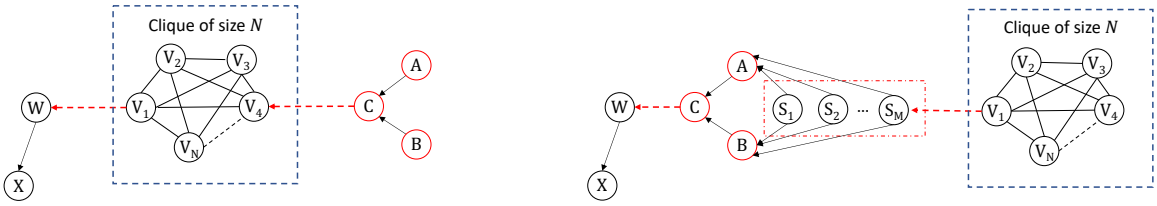
The LDECC algorithm (Fig. 5) first finds the Markov blanket $\text{MB}(X)$ (Line 1): we use the IAMB algorithm (Tsamardinos et al., 2003, Fig. 2) in our experiments (see Fig. 14 in Appendix B.1). Next, we run *LocalPC* within the subgraph containing $\text{MB}(X)$ to prune this set to obtain $\text{Ne}(X)$ (Line 2). We then use the nodes in $\text{MB}(X) \setminus \text{Ne}(X)$ to detect children C of X that are part of UCs of the form $X \rightarrow C \leftarrow D$ (Line 3 and *GetUCChildren* in Fig. 4). LDECC then starts running CI tests in the same way as PC would but excluding tests for X since we already know $\text{Ne}(X)$ (the for-loop in Line 6; see Fig. 12 for *PCTest*). Every time we detect a CI $A \perp\!\!\!\perp B|\mathbf{S}$, we check the following cases: (i) if $A, B \in \text{Ne}(X)$ and $X \notin \mathbf{S}$, then there must be a UC $A \rightarrow X \leftarrow B$ and so we mark A and B as parents (Lines 7, 8); (ii) if $A, B \in \text{Ne}(X)$ and $X \in \mathbf{S}$, then we mark A—X—B as a non-collider (Lines 10, 11); and (iii) **Eager Collider Check (ECC):** if $X \notin \mathbf{S}$ and $A \not\perp\!\!\!\perp B|\mathbf{S} \cup \{X\}$, then, by leveraging Prop. 4, we apply *GetMNS* (Fig. 4) to mark $\text{mns}_X(A), \text{mns}_X(B)$ as parents (Lines 15, 16; *ECCParents* in Fig. 4). Next, for each oriented parent, we use the non-colliders detected in Case (ii) to mark children (Lines 18, 19). LDECC terminates if there are no unoriented neighbors. The ATE set is computed similarly to SD by applying the backdoor adjustment with every *locally valid* parent set. Lines 9,12–14 are inserted to account for Meek's rules $3, 4$. The *check=True* argument in *ECCParents* aims to make it more robust to faithfulness violations (see Remark 21 in Sec. 4.2).

Consider the running example of the DAG in Fig. 1(*a*). We first find the local structure around X. Then we start running CI tests in the same way as PC. After we run $A \perp\!\!\!\perp B|C$, we do an ECC and find $A \not\perp\!\!\!\perp B|\{C, X\}$. Since $\text{mns}_X(A) = \text{mns}_X(B) = \{W\}$, we mark W as a parent of X. Next, we find that $W \perp\!\!\!\perp M|X$ and mark W—X—M as a non-collider which lets us mark M as a child (because W is a parent). LDECC terminates as all neighbors of X are oriented (Fig. 1(*d*)) and the ATE set is computed by using $\{W\}$ as the backdoor adjustment set: $\Theta_{\text{LDECC}} = \{\theta_{X \rightarrow Y|\{W\}}\} = \Theta^*$.

Theorem 5 (Correctness) *Under the CFA and with access to a CI oracle, we have $\Theta_{LDECC} \overset{set}{=} \Theta^*$.*

Remark *A DAG can have multiple valid adjustment sets. We show that local information suffices for identifying the optimal adjustment set according to the asymptotic variance criterion in Henckel et al. (2019) and we present a procedure to discover it (see Fig. 20, Prop. 44 in Appendix D.2).*

In summary, LDECC differs from SD in two key ways: (i) LDECC runs tests in a different order—SD runs *LocalPC* by recursively considering nodes starting from X whereas LDECC runs CI tests in the same order as PC; and (ii) SD uses the skeleton to orient edges from a UC via Meek's rules (like the PC algorithm) whereas LDECC uses an ECC to orient the parents of X.



(*a*) LDECC outperforms SD

(*b*) SD outperforms LDECC

Figure 6: Classes of graphs where LDECC and SD have different runtimes.

4.1. Comparison of computational requirements

In this section, we compare PC, SD, and LDECC based on the number of CI tests they perform. There are classes of DAGs where LDECC is polynomial time while SD is not, and vice versa (Props. 7,8). Using this insight, we construct an algorithm that runs in polynomial time on a larger class of DAGs (Remark 13). We defer proofs to Appendix B.2. We make the following assumption in this section:

Assumption 1 *The CFA holds and we have access to a CI oracle.*

Let the number of CI tests performed by PC, SD, and LDECC be T_{PC}, T_{SD}, and T_{LDECC}. We first show that, in the worst case, LDECC only runs a polynomial (in $|\mathbf{V}|$) number of extra CI tests compared to PC when $|\text{Ne}(X)|$ is bounded. In practice, we observe that the upper bound is quite loose: Even when $|\text{Ne}(X)|$ is large, PC also has to run a large number of tests for X.

Proposition 6 (PC vs LDECC) *We have* $T_{LDECC} \leq T_{PC} + \mathcal{O}(|\mathbf{V}|^2) + \mathcal{O}\left(|\mathbf{V}| \cdot 2^{|Ne(X)|}\right)$.

Proposition 7 (LDECC exponentially better) *There exist classes of graphs such that* $T_{LDECC} + \Theta(2^{|\mathbf{V}|}) \leq T_{PC}$; *and* $T_{LDECC} + \Theta(2^{|\mathbf{V}|}) \leq T_{SD}$.

Proof Consider the class of graphs shown in Fig. 6(*a*) where there is a clique of size N on the path from the UC $A \rightarrow C \leftarrow B$ to W. Both PC and SD will perform $\Theta(2^N)$ CI tests due to this clique. Thus, when $N \asymp |\mathbf{V}|$, SD is exponential time. By contrast, since LDECC runs tests in the same order as PC, it unshields the UC, orients W as a parent via an ECC, and terminates in $\mathcal{O}(|\mathbf{V}|^2)$ tests. ■

Qualitatively, the result shows that if there is a dense region between a UC and the parent it orients, SD might perform poorly because it has to wade through this dense region to get to the UC. By contrast, LDECC can avoid the CI tests in these dense regions as it uses an ECC to orient parents of X instead of the skeleton. However, LDECC does *not* uniformly dominate SD. There are classes of DAGs where LDECC performs exponentially more CI tests than SD:

Proposition 8 (SD exponentially better) *Let* $\mathbf{U}$ *be the set of UCs in* $\mathcal{G}^*$ *and* $M = \max_{U \in \mathbf{U}} sep(U)$. *There exist classes of graphs such that* $T_{SD} + \Theta(2^{M-1}) \leq T_{PC}$; *and* $T_{SD} + \Theta(2^{M-1}) \leq T_{LDECC}$.

Proof Consider the class of graphs shown in Fig. 6(*b*) with a UC with a separating set of size M and a clique of size N upstream of that UC. Since LDECC runs CI tests in the same order as PC, if $N \geq M$, LDECC performs $\Theta(2^{M-1})$ CI tests for nodes inside the clique (all tests of the form $V_i \perp\!\!\!\perp V_j | \mathbf{S}$ s.t. $\mathbf{S} \subseteq \{V_1, \ldots, V_N\} \setminus \{V_i, V_j\}$ and $|\mathbf{S}| < M$ will be run) before the UC is unshielded

via the test $A \perp\!\!\!\perp B | \{S_1, \ldots, S_M\}$. By contrast, SD terminates before even getting to the clique, thus avoiding these tests. So if $M \asymp |\mathbf{V}|$, LDECC will perform exponentially more tests than SD. ■

Qualitatively, this shows that if UCs have large separating sets and there are dense regions upstream of such UCs, LDECC might perform poorly whereas SD can avoid tests in these regions since it would not reach these dense regions.

Next, we present upper bounds for the computational requirements of LDECC and SD for a restricted class of graphs that we call *Locally Orientable Graphs*.

Definition 9 (Locally Orientable Graph.) *A DAG $\mathcal{G}$ is said to be* locally orientable *with respect to a node X if, $\forall V \in Ne(X)$, the edge X—V is oriented in the CPDAG for the MEC of $\mathcal{G}$. Since all nodes in $Ne(X)$ can be oriented, the ATE is point identified, i.e., $|\Theta^*| = 1$.*

Definition 10 (Parent Orienting Collider (POC)) *Consider a DAG $\mathcal{G}^*$. For each $P \in Pa(X)$, let $POC(P)$ be the set of UCs that can orient P, i.e., either (i) P is a part of that UC; or (ii) applying Meek's rules from that UC in the undirected skeleton of $\mathcal{G}^*$ will orient $P \to X$.*

In Props. 11,12, we provide sufficient graphical conditions for when both the algorithms run in polynomial time. For LDECC, we show that the time complexity of orienting the parents depends only on the size of the separating sets for the UCs, i.e., *sep* (Prop. 11). Importantly, the bound is agnostic to how far away the UCs are from the parents.

Proposition 11 (LDECC upper bound) *For a locally orientable DAG $\mathcal{G}^*$, let $D = \max_{V \in MB^+(X)} |Ne(V)|$ and $S = \max_{P \in Pa(X)} \min_{\alpha \in POC(P)} sep(\alpha)$. Then $T_{LDECC} \leq \mathcal{O}(|\mathbf{V}|^{\max\{S,D\}})$.*

For SD, we show that the time complexity of orienting a parent is upper bounded by (i) the *sep* of the closest UC that orients that parent and (ii) the complexity of discovering the neighbors of nodes on the path to that collider (Prop. 12). Importantly, this bound is agnostic to the structure of the nodes that are upstream of the closest colliders.

Proposition 12 (SD upper bound) *For a locally orientable DAG $\mathcal{G}^*$, let $\pi : \mathbf{V} \to \mathbb{N}$ be the order in which nodes are processed by SD (Line 4 of SD). For $P \in Pa(X)$, let $CUC(P) = \operatorname{argmin}_{\alpha \in POC(P)} \pi(\alpha)$ denote the closest UC to P. Let $C = \max_{P \in Pa(X)} sep(CUC(P))$, $D = \max_{V \in MB^+(X)} |Ne(V)|$, and $E = \max_{\{V : \pi(V) < \pi(CUC(P))\}} |Ne(V)|$. Then $T_{SD} \leq \mathcal{O}(|\mathbf{V}|^{\max\{C,D,E\}})$.*

Remark 13 *The following procedure demonstrates one way that LDECC and SD can be combined to broaden the class of graphs where local causal discovery can be performed efficiently: Run both SD and LDECC in parallel and terminate when either algorithm terminates. The number of CI tests performed will be $T_{combined} \leq 2 \min\{T_{SD}, T_{LDECC}\}$. This procedure will be subexponential if at least one of the algorithms is subexponential.*

4.2. Comparison of faithfulness requirements

In this section, we show that SD and LDECC rely on different faithfulness assumptions (Props. 18,20). The CFA is controversial (Uhler et al., 2013) and some works attempt to test or weaken it (Zhang and Spirtes, 2008; Ramsey et al., 2012; Zhang and Spirtes, 2016). Sharing a similar motivation, we show that LDECC and SD can be fruitfully combined to strictly weaken the faithfulness assumptions needed to identify Θ^* (Prop. 25). We defer the proofs to Appendix B.3.

Definition 14 (Adjacency Faithfulness (AF)) *Given a DAG $\mathcal{G}(\mathbf{V}, \mathbf{E})$, AF holds for node $A \in \mathbf{V}$ iff $\forall A—B \in \mathbf{E}$, we have $\forall \mathbf{S} \subseteq \mathbf{V} \setminus \{A, B\},\ A \not\perp\!\!\!\perp B|\mathbf{S}$.*

Definition 15 (Orientation Faithfulness (OF)) *Given a DAG $\mathcal{G}(\mathbf{V}, \mathbf{E})$, OF holds for an unshielded triple $A—C—B$ (where $A—B \notin E$) iff (i) when it is a UC, $A \not\perp\!\!\!\perp B|\mathbf{S}$ for any $\mathbf{S} \subseteq \mathbf{V} \setminus \{A, B\}$ s.t. $C \in \mathbf{S}$; and (ii) when it is a non-collider, $A \not\perp\!\!\!\perp B|\mathbf{S}$ for any $\mathbf{S} \subseteq V \setminus \{A, B\}$ s.t. $C \notin \mathbf{S}$.*

Definition 16 (Local Faithfulness (LF)) *Given a DAG $\mathcal{G}(\mathbf{V}, \mathbf{E})$, LF holds for node X iff (i) AF holds for X; and (ii) OF holds for every unshielded triple $A—X—B \in \mathcal{G}$.*

Proposition 17 *PC will identify the MEC of $\mathcal{G}^*$ if LF holds for all nodes.*

If the goal is to identify Θ^* (and not the entire MEC), weaker faithfulness conditions than those of Prop. 17 are sufficient for SD and PC (Prop. 18). Let $\mathcal{U}$ be the undirected graph on the nodes $\mathbf{V}$ such that an edge $A—B$ exists iff $\forall \mathbf{S} \subseteq \mathbf{V} \setminus \{A, B\},\ A \not\perp\!\!\!\perp B|\mathbf{S}$. Informally, $\mathcal{U}$ is the skeleton of the observational distribution. Let $J^* = \{(A \to C \leftarrow B) \in \mathcal{G}^*\}$ be the set of UCs in the true DAG $\mathcal{G}^*$.

Proposition 18 (Faithfulness for PC and SD) *PC and SD will identify Θ^* if (i) LF holds $\forall V \in MB^+(X)$; (ii) $\forall (A \to C \leftarrow B) \in J^*$, (a) LF holds for A, B, and C, and (b) LF holds for each node on all paths $C \to \ldots \to V \in \mathcal{G}^*$ s.t. $V \in Ne(X)$; (iii) For every edge $A—B \notin \mathcal{G}^*$, $\exists \mathbf{S} \subseteq (Ne_{\mathcal{U}}(A) \cup Ne_{\mathcal{U}}(B))$ s.t. $A \perp\!\!\!\perp B|\mathbf{S}$; and (iv) OF holds for all unshielded triples in $\mathcal{G}^*$.*

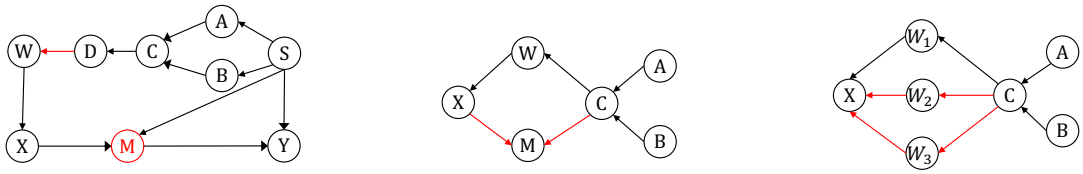
Proof Sketch By Conditions (i,ii), UCs in $\mathcal{G}^*$ are detected and orientations are propagated correctly to X. By Condition (iii), the skeleton discovered by PC and SD is a subgraph of the skeleton of $\mathcal{G}^*$ and thus there are no spurious paths. By Condition (iv), no spurious UCs are detected. Prop. 18 is weaker because AF can be violated for nodes not on paths from UCs to X. ■

We now present sufficient faithfulness conditions for LDECC. Let $\mathbf{L} = \{(A, B, \mathbf{S}) : |\{A, B\} \cap \mathrm{Ne}(X)| \leq 1,\ \mathbf{S} \subseteq \mathbf{V} \setminus \{A, B, X\}\}$ and $H^* = \{(A, B, \mathbf{S}) \in \mathbf{L} : A \perp\!\!\!\perp_{\mathcal{G}^*} B|\mathbf{S},\ A \not\perp\!\!\!\perp_{\mathcal{G}^*} B|\mathbf{S} \cup \{X\}\}$, where $\perp\!\!\!\perp_{\mathcal{G}^*}$ ($\not\perp\!\!\!\perp_{\mathcal{G}^*}$) denotes d-separation (d-connection). Informally, H^* contains the set of ECCs entailed by the true DAG $\mathcal{G}^*$. Analogously, let $H = \{(A, B, \mathbf{S}) \in \mathbf{L} : A \perp\!\!\!\perp B|\mathbf{S},\ A \not\perp\!\!\!\perp B|\mathbf{S} \cup \{X\}\}$. Informally, H contains the set of ECCs entailed by the observational joint distribution $\mathbb{P}(\mathbf{V})$.

Definition 19 (MNS Faithfulness (MFF)) *Given a DAG $\mathcal{G}(\mathbf{V}, \mathbf{E})$, MFF holds for node $V \notin Ne^+(X)$ iff (a) When $mns_X(V)$ does* not *exist, $\forall \mathbf{S} \subseteq Ne(X)$, we have $V \not\perp\!\!\!\perp X|\mathbf{S}$; and (b) When $mns_X(V)$ exists, $\forall \mathbf{S} \subseteq Ne(X)$ s.t. $\mathbf{S} \neq mns_X(V)$ and $V \perp\!\!\!\perp X|\mathbf{S}$, we have $|\mathbf{S}| > |mns_X(V)|$.*

Proposition 20 (Faithfulness for LDECC) *LDECC will identify Θ^* if (i) LF holds $\forall V \in MB^+(X)$; (ii) $H \subseteq H^*$; (iii) $\forall (A, B, \mathbf{S}) \in H$, MFF holds for $\{A, B\} \setminus Ne(X)$; and (iv) $\forall (A, B, \mathbf{S}) \in H^*$ s.t. there is a UC $(A \to C \leftarrow B) \in \mathcal{G}^*$, we have (a) AF holds for A and B; and (b) $(A, B, \mathbf{S}) \in H$.*

Proof Sketch By Condition (i), the structure within $\mathrm{MB}^+(X)$ is correctly discovered. By Condition (ii), every ECC run by LDECC will be valid (since it is also part of H^*), and together with Condition (iii), every ECC will mark the correct parents. By Condition (iv), an ECC will be run for every UC that can orient parents of X, ensuring that all parents get oriented: (iv)(a) ensures that the UC gets unshielded, and (iv)(b) ensures that an ECC is run in Line 16 after the UC is unshielded. ■



(a) DAG with unfaithfulness (b) Testable violation for SD (c) MFF fails for nodes A, B

Figure 7: Different faithfulness violations.

Remark 21 *In LDECC, if we run* ECCParents *with* check=True *where we mark parents with an ECC only if GetMNS*(A) = *GetMNS*(B) *(Lines 20–24 in Fig. 4), then LDECC requires weaker assumptions than Prop. 20 (see Prop. 29 in Appendix B.3) and does better empirically (Sec. 5).*

We demonstrate that SD and LDECC rely on different faithfulness assumptions: There are faithfulness violations that affect one algorithm but not the other. This motivates a procedure that outputs a (conservative) ATE set containing the values in Θ^* under weaker assumptions (Prop. 22).

Example 1 (SD incorrect) *Consider the faithfulness violation* $W \perp\!\!\!\perp D|M$ *for the DAG in Fig. 7(a) (conditioning on the collider* M *cancels out the* $W \leftarrow D$ *edge). Thus, the edge* $W \leftarrow D$ *might be incorrectly removed and both SD and PC will not mark* W *as a parent since the orientations will not be propagated to* W*. However, LDECC can still correctly mark* W *as a parent using an ECC.*

Example 2 (LDECC incorrect) *Consider the faithfulness violations* $A \perp\!\!\!\perp X|M$ *and* $B \perp\!\!\!\perp X|M$ *for the DAG in Fig. 7(a). We can incorrectly find* $mns_X(A) = mns_X(B) = \{M\}$ *(if Line 14 in* GetMNS *tests* $A \perp\!\!\!\perp X|M$ *or* $B \perp\!\!\!\perp X|M$*) causing LDECC to orient* M *as a parent via an ECC. But these violations do not impact SD and PC since the skeleton and UCs are still discovered correctly.*

Proposition 22 (Conservative ATE set) *Consider the following procedure: (i) run both SD and LDECC to get* Θ_{SD} *and* Θ_{LDECC}*; (ii) output the set* $\Theta_{union} = \Theta_{LDECC} \cup \Theta_{SD}$*. If the faithfulness assumptions for either SD or LDECC hold, then* $\Theta^* \subseteq \Theta_{union}$*.*

Next, we show that some violations that affect SD and LDECC are detectable. We use this insight to construct a procedure (Prop. 25) that identifies Θ^* under strictly weaker faithfulness assumptions by switching to LDECC if a violation of the assumptions of SD is detected (or vice versa).

Proposition 23 (Testing faithfulness for LDECC) *Consider running the algorithm in Fig. 8 before invoking* GetMNS*(*A*) for some node* A *in LDECC. If the algorithm returns* Fail*, MFF is violated for node* A*. If the algorithm returns* Unknown*, we could not ascertain if MFF holds for node* A*.*

Proof Sketch If Line 4 returns *Fail*, there are multiple minimal sets thereby violating the uniqueness of MNS (see Example 3). If Line 7 returns *Fail*, for the detected MNS $\mathbf{S}$, there is a superset $\mathbf{S}'$ s.t. $A \perp\!\!\!\perp X|\mathbf{S}'$ but also an intermediate set $\mathbf{S}''$ s.t. $A \not\perp\!\!\!\perp X|\mathbf{S}''$ which cannot happen if MFF holds because removing nodes from $\mathbf{S}'$ should not violate independence (see Example 4). ■

Example 3 (Testing MFF) *Consider the faithfulness violations* $A \perp\!\!\!\perp X|M$ *and* $B \perp\!\!\!\perp X|M$ *for the DAG in Fig. 7(a). We might incorrectly detect* $mns_X(A) = mns_X(B) = \{M\}$ *(similar to Example 2). Line 4 in Fig. 8 will return* Fail *because for* A, B*, we have* $\mathbf{Q}_{\min} = \{\{W\}, \{M\}\}$*.*

Input: Node $A \notin \text{Ne}^+(X)$, $\text{Ne}(X)$
1 $\mathbf{Q} \leftarrow \{\mathbf{S} \subseteq \text{Ne}(X) : A \perp\!\!\!\perp X|\mathbf{S}\}$;
2 **if** $|\mathbf{Q}| = 0$ **then return** noValidMNS ;
3 $\mathbf{Q}_{\min} \leftarrow \{\mathbf{S} \in \mathbf{Q} : \forall \mathbf{S}' \subset \mathbf{S}, \mathbf{S}' \notin \mathbf{Q}\}$;
4 **if** $|\mathbf{Q}_{\min}| > 1$ **then return** Fail ;
5 $\mathbf{S} \leftarrow \text{GetElement}(\mathbf{Q}_{\min})$; // Here $|\mathbf{Q}_{\min}| = 1$.
6 **for** $\mathbf{S}' \in \mathbf{Q}$ *such that* $\mathbf{S} \subset \mathbf{S}'$ **do**
7 | **if** $\exists \mathbf{S}'' \notin \mathbf{Q}$ *s.t.* $\mathbf{S} \subset \mathbf{S}'' \subset \mathbf{S}'$ **then return** Fail ;
8 **end**
9 **return** Unknown;

Figure 8: Testing MFF.

Input: UC $\alpha = (A \to C \leftarrow B)$
1 $\mathbf{Q}(A) \leftarrow \{\mathbf{S} \subseteq \text{Ne}(X) : A \perp\!\!\!\perp X|\mathbf{S}\}$;
2 $\mathbf{Q}(B) \leftarrow \{\mathbf{S} \subseteq \text{Ne}(X) : B \perp\!\!\!\perp X|\mathbf{S}\}$;
// See Defn. 10 for *POC*.
3 $\mathbf{M} \leftarrow \{P \in \text{Pa}(X) : \alpha \in \text{POC}(P)\}$;
4 $\widetilde{M}(A) \leftarrow \{\mathbf{S} \in \mathbf{Q}(A) : \mathbf{M} \subseteq \mathbf{S}\}$;
5 $\widetilde{M}(B) \leftarrow \{\mathbf{S} \in \mathbf{Q}(B) : \mathbf{M} \subseteq \mathbf{S}\}$;
6 **if** $|\widetilde{M}(A)| = 0$ *or* $|\widetilde{M}(B)| = 0$ **then return** Fail ;
7 **return** Unknown;

Figure 9: Testing faithfulness for SD.

Example 4 (Testing MFF) *Consider the DAG in Fig. 7(c). If a faithfulness violation causes the paths $X \leftarrow W_2 \leftarrow C$ and $X \leftarrow W_3 \leftarrow C$ to cancel each other out, we will observe $A \perp\!\!\!\perp X|W_1$ and thus wrongly detect $\text{mns}_X(A) = \{W_1\}$. But Line 7 in Fig. 8 will detect this failure since $A \perp\!\!\!\perp X|\{W_1, W_2, W_3\}$ but $A \not\perp\!\!\!\perp X|\{W_1, W_2\}$ and $A \not\perp\!\!\!\perp X|\{W_1, W_3\}$.*

Proposition 24 (Testing faithfulness for SD) *Consider running the algorithm in Fig. 9 with each UC detected by SD. If the algorithm returns* Fail*, then faithfulness is violated for SD. If the algorithm returns* Unknown*, we could not ascertain if the faithfulness assumptions for SD hold.*

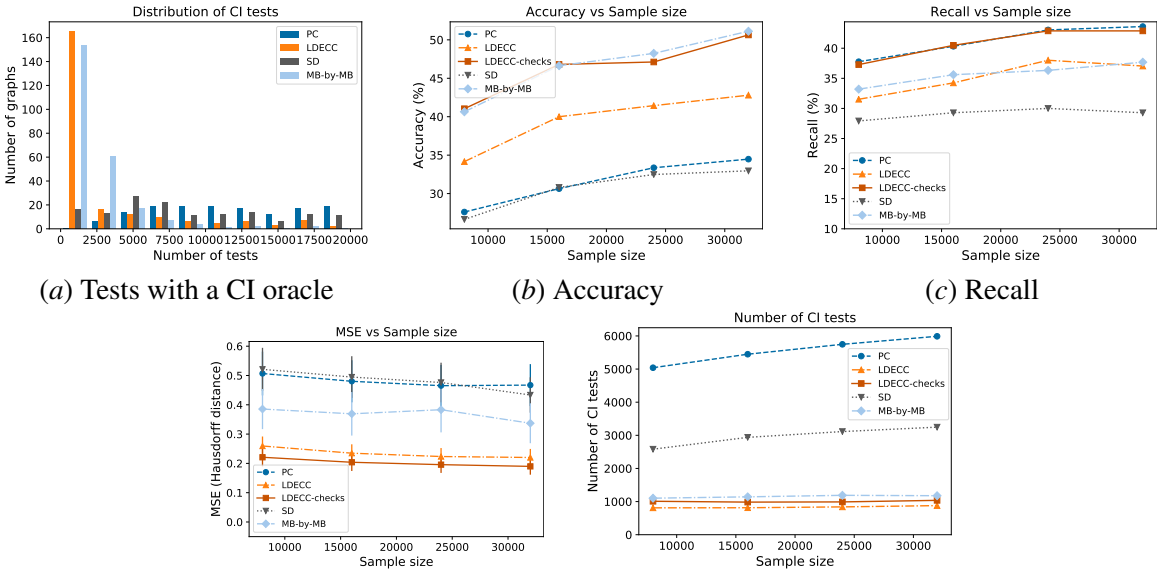
Proof Sketch In Line 3, $\mathbf{M}$ contains nodes marked as parents of X due to the UC $A \to C \leftarrow B$. In Line 6, we check that there is some superset of $\mathbf{M}$ that makes A and B independent of X: this must be true if the nodes in $\mathbf{M}$ were actually parents of X (see Example 5). ■

Example 5 (Testing faithfulness for SD) *Consider an OF violation $C \perp\!\!\!\perp X|M$ for the DAG in Fig. 7(b) that causes $X \to M \leftarrow C$ to be detected as a non-collider. SD will mark M as a parent via the UC $A \to C \leftarrow B$. But Line 6 will return* Fail *because $\mathbf{M} = \{M\}$ and $\mathbf{Q}(A) = \mathbf{Q}(B) = \{\{W\}\}$.*

Proposition 25 *Consider the following procedure: (i) run SD to get Θ_{SD}; (ii) test for faithfulness violations using Prop. 24; (iii) if a violation is found, run LDECC and output Θ_{LDECC}, else output Θ_{SD}. This procedure identifies Θ^* under strictly weaker assumptions than Prop. 18.*

Proof If the conditions of Prop. 18 hold, this procedure will output Θ_{SD} because we will not detect a faithfulness violation. However, if a violation is detected, this procedure will still identify Θ^* if the faithfulness conditions for LDECC hold. This is strictly weaker than the assumptions of Prop. 18. ■

We can construct a similar procedure for LDECC. This again shows the complementary nature of the two algorithms: if there are detectable faithfulness violations for one algorithm, we can output an alternative ATE set thereby allowing us to identify Θ^* under weaker faithfulness assumptions than would be possible with only one of the algorithms. Currently, our testing procedures can only detect some (but not all) violations. Moreover, faithfulness violations in DAGs do not occur independently of each other: an unfaithful mechanism can lead to other (entangled) faithfulness violations. As a result, the violations affecting SD and LDECC will be entangled and the given faithfulness assumptions may not be the minimal ones required for local causal discovery.



(*a*) Tests with a CI oracle (*b*) Accuracy (*c*) Recall

(*d*) MSE vs sample size (*e*) Number of CI tests

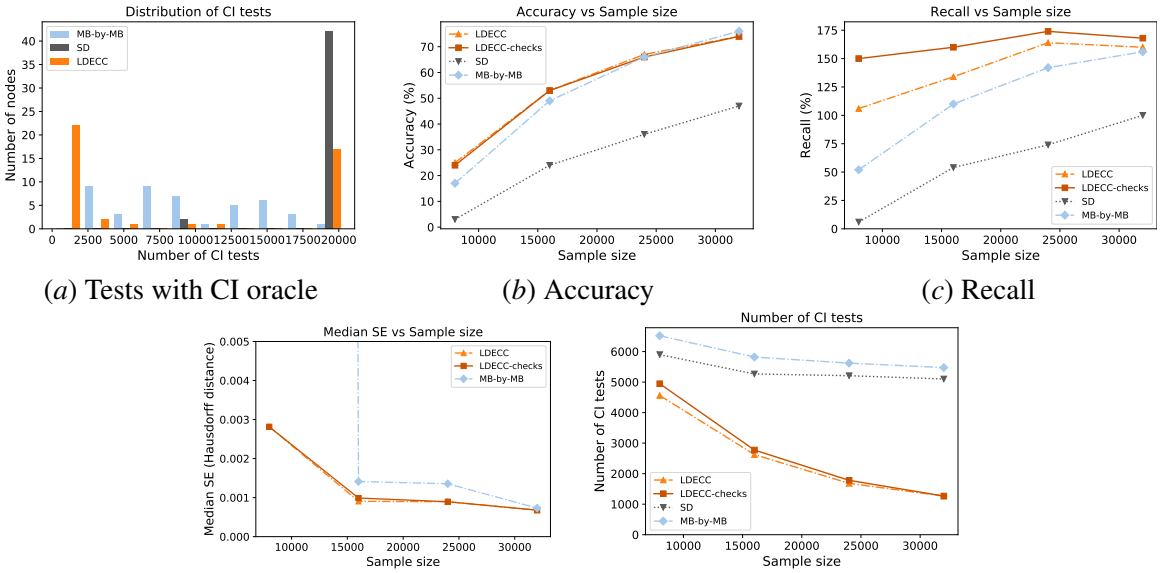
Figure 10: Results on synthetic linear Gaussian graphs.

5. Experiments

We present results on both synthetic linear graphs and the semi-synthetic *MAGIC-NIAB* graph from *bnlearn* (Scutari, 2009). We also ran the MB-by-MB approach (Wang et al., 2014), an instantiation of SD that uses IAMB (Fig. 14) and *LocalPC* for local structure learning. LDECC usually performs comparably to MB-by-MB and typically outperforms SD while running a similar number of CI tests. Results on synthetic binomial graphs and additional graphs from *bnlearn* are in Appendix C.

Results on synthetic data. We generated synthetic linear graphs with Gaussian errors, 20 covariates—non-descendants of X and Y with paths to both X and Y—and 3 mediators—nodes on some causal path from X to Y. We sampled edges between the nodes with varying probabilities and sampled the edge weights uniformly from $[-1, -0.25] \cup [0.25, 1]$ (see Appendix C.1 for the precise procedure). We compared the algorithms based on the number of CI tests performed with a CI oracle on 250 synthetic graphs (of which $\approx 77\%$ were *locally orientable*; see Defn. 9). The distribution of CI tests (Fig. 10(*a*)) shows that LDECC performs comparably to MB-by-MB and substantially outperforms SD and PC on ~ 150 graphs. Next, we evaluated our methods on 250 synthetic graphs at four sample sizes. For each graph, we generated data 5 times (so we have $N = 1250$ runs). We used the Fisher-z CI test and ordinary least squares regression for ATE estimation. We also ran LDECC with *check=True* as described in Remark 21 (denoted by *LDECC-checks*). We define *accuracy* as the fraction of times we get the ATE set that we would estimate for the true DAG $\mathcal{G}^*$, and *recall* as the fraction of times the ATE estimated with the true DAG $\mathcal{G}^*$ is contained in the estimated ATE set, i.e., whether $\widehat{\theta}_{\text{ATE}}(\mathcal{G}^*) \in \widehat{\Theta}$. We used Hausdorff distance to compute mean squared error (MSE) between the estimated $\widehat{\Theta}$ and the ground-truth set Θ^*:

$$\text{MSE}_{\text{Hausdorff}}(\{\widehat{\Theta}\}_{i=1}^N, \{\Theta^*\}_{i=1}^N) = \frac{1}{N}\sum_{i=1}^N \max\left\{\sup_{u\in\widehat{\Theta}_i}\inf_{v\in\Theta_i^*}(u-v)^2, \sup_{v\in\Theta_i^*}\inf_{u\in\widehat{\Theta}_i}(u-v)^2\right\}.$$



(*a*) Tests with CI oracle (*b*) Accuracy (*c*) Recall

(*d*) Median SE vs sample size (*e*) Average number of CI tests

Figure 11: Results on the semi-synthetic *MAGIC-NIAB* linear Gaussian graph.

LDECC-checks and MB-by-MB have comparable accuracy and are better than both SD and PC (Fig. 10(*b*)). In terms of recall, LDECC-checks and PC perform comparably and outperform SD and MB-by-MB (Fig. 10(*c*)). LDECC-checks has better accuracy and recall than LDECC suggesting that the check helps in practice. Both variants of LDECC have substantially lower MSE than MB-by-MB, PC, and SD (Fig. 10(*d*)) while still doing a similar number of CI tests as MB-by-MB (Fig. 10(*e*)).

Results on *MAGIC-NIAB* from bnlearn. *MAGIC-NIAB* is a linear Gaussian DAG with 44 nodes. With a CI oracle, PC performed $\approx 1.472 \times 10^6$ tests. We plot the distribution of CI tests by repeatedly setting each node as the treatment (capping the maximum number of tests per node to 20000). We see that LDECC and MB-by-MB perform better than SD with MB-by-MB performing well across all nodes (Fig. 11(*a*)). Next, we designated the nodes *G266* and *HT* as the treatment and outcome, respectively. We compared the methods by sampling the data 100 times at four sample sizes (capping the maximum number of tests run by each algorithm to 7000). LDECC has similar accuracy (Fig. 11(*b*)) and recall (Fig. 11(*c*)) as MB-by-MB and they outperform SD on both metrics. We plot *median* squared error (Fig. 11(*d*)) for MB-by-MB and LDECC (MB-by-MB does poorly on the mean due to outliers). Both are similar at large sample sizes (we omit SD because it does poorly). Additionally, LDECC performs fewer CI tests than MB-by-MB and SD (Fig. 11(*e*)).

6. Conclusion

Broadening the landscape of local causal discovery, we propose a new algorithm that uses ECCs to orient parents. We show that LDECC has different computational and faithfulness requirements compared to the existing class of sequential algorithms. This allows us to profitably combine the two methods to get polynomial runtimes on a larger class of graphs as well as identify the set of possible ATE values under weaker assumptions. In future work, we hope to further weaken the faithfulness requirements and extend our methods to handle causal graphs with latent variables.

Acknowledgments

We gratefully acknowledge Amazon AI, UPMC, Abridge, the PwC Center, the Block Center, the Center for Machine Learning and Health, and the CMU Software Engineering Institute (SEI) via Department of Defense contract FA8702-15-D-0002, for their generous support of ACMI Lab's research. In particular, we are grateful to the PwC center for supporting Shantanu Gupta as a presidential scholar during the completion of this research, and to Amazon AI who has named Shantanu Gupta a recipient of the Amazon Ph.D. fellowship for the coming year.

References

Constantin F Aliferis, Ioannis Tsamardinos, and Alexander Statnikov. Hiton: a novel markov blanket algorithm for optimal variable selection. In *AMIA annual symposium proceedings*. American Medical Informatics Association, 2003.

Constantin F Aliferis, Alexander Statnikov, Ioannis Tsamardinos, Subramani Mani, and Xenofon D Koutsoukos. Local causal and markov blanket induction for causal discovery and feature selection for classification part i: algorithms and empirical evaluation. *Journal of Machine Learning Research*, 11(1), 2010.

Debo Cheng, Jiuyong Li, Lin Liu, Kui Yu, Thuc Duy Le, and Jixue Liu. Toward unique and unbiased causal effect estimation from data with hidden variables. *IEEE Transactions on Neural Networks and Learning Systems*, 2022a.

Debo Cheng, Jiuyong Li, Lin Liu, Kui Yu, Thuc Duy Lee, and Jixue Liu. Discovering ancestral instrumental variables for causal inference from observational data. *arXiv preprint arXiv:2206.01931*, 2022b.

Tom Claassen and Tom Heskes. A logical characterization of constraint-based causal discovery. *arXiv preprint arXiv:1202.3711*, 2012.

Gregory F Cooper. A simple constraint-based algorithm for efficiently mining observational databases for causal relationships. *Data Mining and Knowledge Discovery*, 1997.

Xavier De Luna, Ingeborg Waernbaum, and Thomas S Richardson. Covariate selection for the nonparametric estimation of an average treatment effect. *Biometrika*, 2011.

Doris Entner, Patrik Hoyer, and Peter Spirtes. Statistical test for consistent estimation of causal effects in linear non-gaussian models. In *Artificial Intelligence and Statistics*, 2012.

Doris Entner, Patrik Hoyer, and Peter Spirtes. Data-driven covariate selection for nonparametric estimation of causal effects. In *Artificial Intelligence and Statistics*. PMLR, 2013.

Zhuangyan Fang and Yangbo He. Ida with background knowledge. In *Conference on Uncertainty in Artificial Intelligence*. PMLR, 2020.

Tian Gao and Qiang Ji. Local causal discovery of direct causes and effects. *Advances in Neural Information Processing Systems*, 2015.

Tomas Geffner, Javier Antoran, Adam Foster, Wenbo Gong, Chao Ma, Emre Kiciman, Amit Sharma, Angus Lamb, Martin Kukla, Nick Pawlowski, et al. Deep end-to-end causal inference. *arXiv preprint arXiv:2202.02195*, 2022.

Clark Glymour, Kun Zhang, and Peter Spirtes. Review of causal discovery methods based on graphical models. *Frontiers in genetics*, 2019.

Limor Gultchin, Matt Kusner, Varun Kanade, and Ricardo Silva. Differentiable causal backdoor discovery. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 2020.

Leonard Henckel, Emilija Perković, and Marloes H Maathuis. Graphical criteria for efficient total effect estimation via adjustment in causal linear models. *arXiv preprint arXiv:1907.02435*, 2019.

Antti Hyttinen, Frederick Eberhardt, and Matti Järvisalo. Do-calculus when the true graph is unknown. In *UAI*. Citeseer, 2015.

Amin Jaber, Jiji Zhang, and Elias Bareinboim. Causal identification under markov equivalence: Completeness results. In *International Conference on Machine Learning*. PMLR, 2019.

Zhaolong Ling, Kui Yu, Hao Wang, Lei Li, and Xindong Wu. Using feature selection for local causal structure learning. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2020.

Zhaolong Ling, Kui Yu, Yiwen Zhang, Lin Liu, and Jiuyong Li. Causal learner: A toolbox for causal structure and markov blanket learning. *arXiv preprint arXiv:2103.06544*, 2021.

Marloes H Maathuis and Diego Colombo. A generalized back-door criterion. *The Annals of Statistics*, 2015.

Marloes H Maathuis, Markus Kalisch, and Peter Bühlmann. Estimating high-dimensional intervention effects from observational data. *The Annals of Statistics*, 2009.

Sara Magliacane, Tom Claassen, and Joris M Mooij. Ancestral causal inference. *Advances in Neural Information Processing Systems*, 2016.

Daniel Malinsky and Peter Spirtes. Estimating causal effects with ancestral graph markov models. In *Conference on Probabilistic Graphical Models*. PMLR, 2016.

Subramani Mani, Peter L Spirtes, and Gregory F Cooper. A theoretical study of y structures for causal discovery. *arXiv preprint arXiv:1206.6853*, 2012.

Christopher Meek. Causal inference and causal explanation with background knowledge. *arXiv preprint arXiv:1302.4972*, 2013.

Joris M Mooij, Jerome Cremers, et al. An empirical study of one of the simplest causal prediction algorithms. In *UAI 2015 Workshop on Advances in Causal Inference*, 2015.

Preetam Nandy, Marloes H Maathuis, and Thomas S Richardson. Estimating the effect of joint interventions from observational data in sparse high-dimensional settings. *The Annals of Statistics*, 2017.

Judea Pearl. *Causality*. Cambridge university press, 2009.

Emilija Perković, Markus Kalisch, and Maloes H Maathuis. Interpreting and using cpdags with background knowledge. *arXiv preprint arXiv:1707.02171*, 2017.

Joseph Ramsey, Jiji Zhang, and Peter L Spirtes. Adjacency-faithfulness and conservative causal inference. *arXiv preprint arXiv:1206.6843*, 2012.

Andrea Rotnitzky and Ezequiel Smucler. Efficient adjustment sets for population average treatment effect estimation in non-parametric causal graphical models. *arXiv preprint arXiv:1912.00306*, 2019.

Marco Scutari. Learning bayesian networks with the bnlearn r package. *arXiv preprint arXiv:0908.3817*, 2009.

Abhin Shah, Karthikeyan Shanmugam, and Kartik Ahuja. Finding valid adjustments under non-ignorability with minimal dag knowledge. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 2022.

Ricardo Silva and Shohei Shimizu. Learning instrumental variables with structural and non-gaussianity assumptions. *Journal of Machine Learning Research*, 2017.

Peter Spirtes, Clark N Glymour, Richard Scheines, and David Heckerman. *Causation, prediction, and search*. MIT press, 2000.

Chandler Squires and Caroline Uhler. Causal structure learning: a combinatorial perspective. *arXiv preprint arXiv:2206.01152*, 2022.

Jin Tian and Judea Pearl. *A general identification condition for causal effects*. eScholarship, University of California, 2002.

Christian Toth, Lars Lorch, Christian Knoll, Andreas Krause, Franz Pernkopf, Robert Peharz, and Julius Von Kügelgen. Active bayesian causal inference. *arXiv preprint arXiv:2206.02063*, 2022.

Ioannis Tsamardinos, Constantin F Aliferis, Alexander R Statnikov, and Er Statnikov. Algorithms for large scale markov blanket discovery. In *FLAIRS conference*. St. Augustine, FL, 2003.

Ioannis Tsamardinos, Laura E Brown, and Constantin F Aliferis. The max-min hill-climbing bayesian network structure learning algorithm. *Machine learning*, 2006.

Caroline Uhler, Garvesh Raskutti, Peter Bühlmann, and Bin Yu. Geometry of the faithfulness assumption in causal inference. *The Annals of Statistics*, 2013.

Tyler J VanderWeele and Ilya Shpitser. A new criterion for confounder selection. *Biometrics*, 2011.

Philip Versteeg, Joris Mooij, and Cheng Zhang. Local constraint-based causal discovery under selection bias. In *Conference on Causal Learning and Reasoning*. PMLR, 2022.

Changzhang Wang, You Zhou, Qiang Zhao, and Zhi Geng. Discovering and orienting the edges connected to a target variable in a dag via a sequential local learning approach. *Computational Statistics & Data Analysis*, 2014.

David S Watson and Ricardo Silva. Causal discovery under a confounder blanket. In *Uncertainty in Artificial Intelligence*. PMLR, 2022.

Janine Witte and Vanessa Didelez. Covariate selection strategies for causal inference: Classification and comparison. *Biometrical Journal*, 2019.

Jianxin Yin, You Zhou, Changzhang Wang, Ping He, Cheng Zheng, and Zhi Geng. Partial orientation and local structural learning of causal networks for prediction. In *Causation and Prediction Challenge*. PMLR, 2008.

Kui Yu, Xianjie Guo, Lin Liu, Jiuyong Li, Hao Wang, Zhaolong Ling, and Xindong Wu. Causality-based feature selection: Methods and evaluations. *ACM Computing Surveys (CSUR)*, 2020.

Jiji Zhang and Peter Spirtes. Detection of unfaithfulness and robust causal inference. *Minds and Machines*, 2008.

Jiji Zhang and Peter Spirtes. The three faces of faithfulness. *Synthese*, 2016.

You Zhou, Changzhang Wang, Jianxin Yin, and Zhi Geng. Discover local causal network around a target to a given depth. In *Causality: Objectives and Assessment*. PMLR, 2010.

```
def GetCPDAG(Undirected graph 𝒰, DSep):
    for every unshielded A—C—B ∈ 𝒰 do
        if C ∉ DSep(A, B) then
            Orient A → C ← B;
    end
    CPDAG 𝒢 ← ApplyMeekRules(𝒰);
    return 𝒢;
```

```
def PCTest(Undirected graph 𝒰(V, E)):
    s ← 0;
    while ∃(A—B) ∈ E s.t. |Ne(A) \ {B}| ≥ s do
        for S ⊆ Ne(A) \ {B} s.t. |S| = s do
            if A ⊥⊥ B|S then
                𝒰.removeEdge(A—B);
                yield (A ⊥⊥ B|S);
                break;
        end
        s ← s + 1;
    end
```

```
Completely connected undirected graph 𝒰(V, E);
∀A, B ∈ V, DSep(A, B) ← null;
for (A ⊥⊥ B|S) ∈ PCTest(𝒰) do
    DSep(A, B) ← S;
end
CPDAG 𝒢 ← GetCPDAG(𝒰, DSep);
Output: 𝒢, DSep
```

Figure 12: The PC algorithm (Spirtes et al., 2000, Sec. 5.4.2).

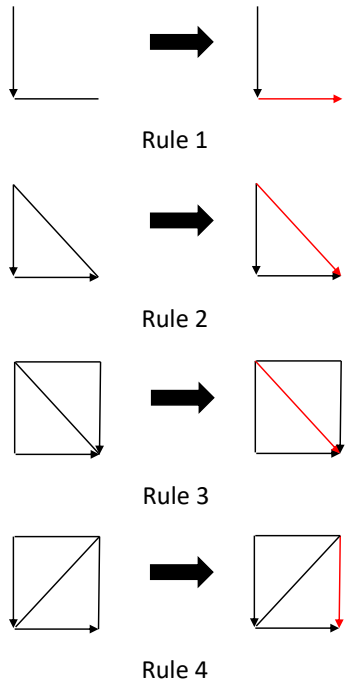


Figure 13: Meek's orientation rules.

Appendix A. Additional details on the PC algorithm

Definition 26 (CPDAG (Maathuis and Colombo, 2015, Pg. 5)) *A set of DAGs that entail the same set of CIs form an MEC. This MEC can be uniquely represented using a CPDAG. A CPDAG is a graph with the same skeleton as each DAG in the MEC and contains both directed ($\rightarrow$) and undirected (—) edges. A directed edge $A \rightarrow B$ means that the $A \rightarrow B$ is present in every DAG in the MEC. An undirected edge A—B means that there is at least one DAG in the MEC with an $A \rightarrow B$ edge and at least one DAG with the $B \rightarrow A$ edge.*

The PC algorithm (Fig. 12) starts with a fully connected skeleton and runs CI tests to remove edges. For each pair of nodes (A, B) that are adjacent in the skeleton, we run CIs of size s—starting with $s = 0$ and then increasing it by one in each subsequent iteration—until the edge is removed or the number of nodes adjacent to both A and B is less than s. Once the skeleton is found, UCs are detected and then additional edges are oriented by repeatedly applying Meek's rules (Fig. 13) until

Input: Node X.
MB$(X) \leftarrow \emptyset$;
`// Forward Pass.`
while *MB*(X) *has changed* **do**
  for $V \in \mathbf{V} \setminus (MB(X) \cup \{X\})$ **do**
    if $V \not\perp\!\!\!\perp X | MB(X)$ **then** MB(X).add(V) ;
  end
end
`// Backward Pass.`
for $V \in MB(X)$ **do**
  if $V \perp\!\!\!\perp X | (MB(X) \setminus \{X\})$ **then** MB(X).remove(V) ;
end
Output: MB(X)

Figure 14: The IAMB algorithm.

def `Nbrs(`*CPDAG* $\mathcal{G}$*, Target* X`)`**:**
  parents $\leftarrow \emptyset$;
  children $\leftarrow \emptyset$;
  unoriented $\leftarrow \emptyset$;
  for $V \in Ne_{\mathcal{G}}(X)$ **do**
    if $X \leftarrow V \in \mathcal{G}$ **then**
      parents.add(V);
    else if $X \rightarrow V \in \mathcal{G}$ **then**
      children.add(V);
    else
      unoriented.add(V);
  end
  return parents, children, unoriented;

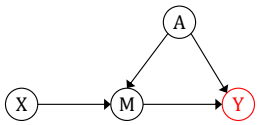
Figure 15: The *Nbrs* subroutine used by SD.

no additional edges can be oriented. Under faithfulness, with access to a CI oracle, the output of PC is a CPDAG encoding the MEC of the true DAG $\mathcal{G}^*$.

Appendix B. Additional details for Section 4

In the example below, we demonstrate that there exist causal graphs with nodes for which mns does not exist.

Example 6 (MNS does not exist) *In the following graph, for node $Y \in Desc(X)$, mns_X does not exist because there is no subset of $Ne(X)$ that d-separates Y from X.*



B.1. Omitted Proofs for Section 4

Proposition 2 *For any node $V \notin (Desc(X) \cup Ne^+(X))$, $mns_X(V)$ exists and $mns_X(V) \subseteq Pa(X)$.*

Proof Let $\mathbf{Q} = \text{Desc}(X) \cup \text{Ne}^+(X)$. Since $\text{Pa}(X)$ blocks all backdoor paths from X, for every $V \notin \mathbf{Q}$, we have $V \perp\!\!\!\perp X | \text{Pa}(X)$. Therefore, for every $V \notin \mathbf{Q}$, there exists some subset $\mathbf{S} \subseteq \text{Pa}(X)$ such that $V \perp\!\!\!\perp X | \mathbf{S}$. ■

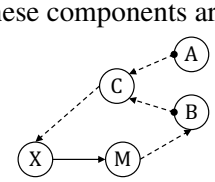
Proposition 3 *[Uniqueness of MNS] For nodes V s.t. $mns_X(V)$ exists, it is unique.*

Proof We will prove this by contradiction. Consider a node V with two MNSs: $\mathbf{S}_1 \subseteq \text{Ne}(X)$ and $\mathbf{S}_2 \subseteq \text{Ne}(X)$ with $\mathbf{S}_1 \neq \mathbf{S}_2$. If $\mathbf{S}_1 \subset \mathbf{S}_2$ or $\mathbf{S}_2 \subset \mathbf{S}_1$, then minimality is violated. Hence, going

forward we will only consider the case where $\mathbf{S}_1 \setminus \mathbf{S}_2 \neq \emptyset$ and $\mathbf{S}_2 \setminus \mathbf{S}_1 \neq \emptyset$. Consider any node $A \in \mathbf{S}_1 \setminus \mathbf{S}_2$. For $\mathbf{S}_2$ to be a valid MNS, some nodes in $\mathbf{S}_2 \setminus \mathbf{S}_1$ must block all paths from V to X that contain A (this is because if this path were to be only be blocked by some nodes in $\mathbf{S}_1$, then minimality of $\mathbf{S}_1$ will be violated as $\mathbf{S}_1 \setminus \{A\}$ would also have been a valid MNS). This means that there is a path from V to X through some nodes in $\mathbf{S}_2 \setminus \mathbf{S}_1$ that cannot be blocked by $\mathbf{S}_1$ (else these nodes in $\mathbf{S}_1$ would have blocked the paths from V to A violating minimality of $\mathbf{S}_1$). This contradicts the fact that $\mathbf{S}_1$ is a valid MNS. Therefore, we must have $\mathbf{S}_1 = \mathbf{S}_2$. ■

Proposition 4 *[Eager Collider Check] For nodes $A, B \in \mathbf{V} \setminus Ne^+(X)$, any $\mathbf{S} \subset \mathbf{V} \setminus \{A, B, X\}$, if (i) $A \perp\!\!\!\perp B|\mathbf{S}$; and (ii) $A \not\perp\!\!\!\perp B|\mathbf{S} \cup \{X\}$; then $A, B \notin Desc(X)$ and $mns_X(A), mns_X(B) \subseteq Pa(X)$.*

Proof We prove this by contradiction. Let's say there is a child M of X such that $M \in \text{mns}_X(B)$ or $M \in \text{mns}_X(A)$. First, note that if Conditions (i, ii) hold, then there is a path of the form $A\bullet \rightarrow C$ and $B\bullet \rightarrow C$ and $C \rightarrow \ldots X$, where $\bullet$ means that there can be either an arrowhead or tail (i.e., there can be either a directed path $A \rightarrow \ldots \rightarrow C$ or a backdoor path $A \leftarrow \ldots \rightarrow C$ and likewise for B) with $C \notin \mathbf{S}$. W.l.o.g., let's say that $M \in \text{mns}_X(B)$ (the argument for node A follows similarly). Then there is a directed path from X to B through M (i.e., $X \rightarrow M \rightarrow \ldots \rightarrow B$). There cannot be a path $B \rightarrow \ldots \rightarrow M$ because then M will be a collider and therefore we will have $M \notin \text{mns}_X(B)$. These components are illustrated in the figure below:



We now show that $B \notin \text{Desc}(X)$ (the argument for node A is the same). There cannot be a directed path $B \rightarrow \ldots \rightarrow C$ because otherwise a cycle $B \rightarrow \ldots \rightarrow C \rightarrow \ldots \rightarrow X \rightarrow M \rightarrow \ldots \rightarrow B$ gets created. Thus the path from B to C must be of the form $B \leftarrow \ldots \rightarrow C$. Note that there is an active path between A and B through X ($A\bullet \rightarrow \ldots \rightarrow C \rightarrow \ldots \rightarrow X \rightarrow M \rightarrow \ldots \rightarrow B$). Since $A \perp\!\!\!\perp B|\mathbf{S}$, there are two possibilities: (i) $\mathbf{S}$ contains X to block this path which contradicts the definition of $\mathbf{S}$ (where $X \notin \mathbf{S}$); or (ii) $\mathbf{S}$ blocks all paths between A and X or between B and X in which case A and B cannot become dependent when additionally conditioned on X thereby violating Condition (ii). Therefore, we have that $A, B \notin \text{Desc}(X)$ and by Prop. 2, $\text{mns}_X(A)$ and $\text{mns}_X(B)$ will be valid and only contain parents of X. ■

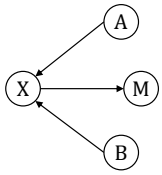
Claassen and Heskes (2012) use minimal (in)dependencies to construct three logical rules which are sound and complete for performing causal discovery. While their algorithm cannot directly be used for local causal discovery, we show below that Lemma 3 in their paper can be used to simplify the proof of Eager Collider Check:

Proof [Alternative proof of ECC] Since we have $A \perp\!\!\!\perp B|\mathbf{S}$ and $A \not\perp\!\!\!\perp B|\mathbf{S} \cup \{X\}$, by Claassen and Heskes (2012, Lemma 3), we have $A, B \notin \text{Desc}(X)$. Therefore, by Prop. 2, $\text{mns}_X(A)$ and $\text{mns}_X(B)$ will be valid and only contain parents of X. ■

B.1.1. PROOF OF CORRECTNESS OF LDECC UNDER THE CFA

Lemma 27 *Consider a DAG $\mathcal{G}(\mathbf{V}, \mathbf{E})$ and a node $X \in \mathbf{V}$. Let $A, B \in Pa(X)$ be two parents of X such that A—$B \notin \mathbf{E}$. Then, for any $\mathbf{S} \subseteq \mathbf{V} \setminus \{A, B, X\}$ such that $A \perp\!\!\!\perp B|\mathbf{S}$, we have $Ch(X) \cap \mathbf{S} = \emptyset$.*

Proof We prove this by contradiction. Let's say that there is child $M \in \mathbf{S}$ (the relevant component of the graph is shown in the figure below).



Since M is a child of X, conditioning on M opens up the path $A \to X \leftarrow B$ rendering A and B dependent conditioned on $\mathbf{S}$. Therefore, we must have $M \notin \mathbf{S}$. ■

Theorem 5 *[Correctness] Under the CFA and with access to a CI oracle, we have $\Theta_{LDECC} \overset{set}{=} \Theta^*$.*

Proof We will prove the correctness of LDECC by showing that (i) every orientable neighbor of the treatment X will get oriented correctly by LDECC; and (ii) every unorientable neighbor of the treatment will remain unoriented.

We assume that the function *FindMarkovBlanket* finds the Markov blanket correctly under the CFA. The IAMB algorithm, which we use in our experiments, has this property. Additionally, the function *RunLocalPC* will also return the correct $\text{Ne}(X)$ under the CFA and with a CI oracle.

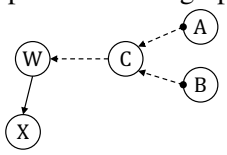
Parents are oriented correctly. In PC, edges get oriented using UCs and then additional orientations are propagated via the application of Meek's rules (Figure 13).

The simplest case is where two parents form a UC at X. Consider parents W_1 and W_2 that get oriented because they form a UC $W_1 \to X \leftarrow W_2$. Lines 7,8 will mark W_1 and W_2 as parents.

We will now consider parents that get oriented due to each of the four Meek rules and show that LDECC orients parents for each of the four cases.

Meek Rule 1*:*

Consider a parent W that gets oriented due to the application of Meek's rule 1. This can only happen due to some UC $A \to C \leftarrow B$ from which these orientations have been propagated (relevant components of the graph are illustrated in the figure below).

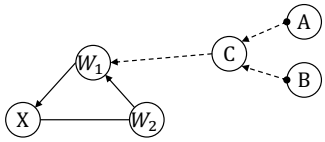


Thus there is a directed path $C \to \ldots \to W \to X$. This would mean that $W \in \text{mns}_X(A)$ and $W \in \text{mns}_X(B)$. Thus Line 16 will mark W as a parent.

Meek Rule 2*:*

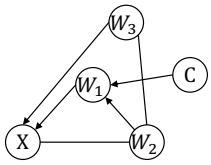
Consider a parent W_2 that gets oriented due to the application of Meek's rule 2. In this case, we have an oriented path $W_2 \to W_1 \to X$ but the edge W_2—X is unoriented (and Meek Rule 2 must be applied to orient it).

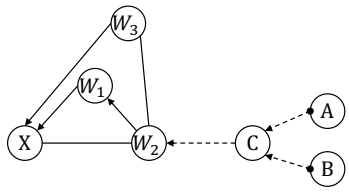
The first possibility is that the $X \leftarrow W_1$ was oriented due to some UC $A \rightarrow C \leftarrow B$ with a path $C \rightarrow \ldots \rightarrow W_1$ (relevant components of the graph are illustrated in the figure below):



In this case, there would be a collider at W_1: $C \rightarrow \ldots \rightarrow W_1 \leftarrow W_2$. Thus if $W_1 \in \text{mns}_X(A)$, then $W_2 \in \text{mns}_X(A)$ and thus LDECC will mark W_2 as a parent in Line 16.

The other possibility is that $X \leftarrow W_1$ was oriented due to a UC like $W_3 \rightarrow X \leftarrow W_1$ but there is an edge W_3—W_2 which causes the collider $W_2 \rightarrow X \leftarrow W_3$ to be shielded and due to this, the W_2—X remained unoriented. However, by definition of the Meek rule, the edge $W_2 \rightarrow W_1$ is oriented. Thus, (i) either there is a UC of the form $W_2 \rightarrow W_1 \leftarrow C$; or (ii) there is a UC from which the $W_2 \rightarrow W_1$ orientation was propagated. The relevant components of the graph for these two cases are illustrated in the figures below.

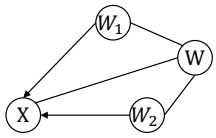




For Case (i), $W_2 \in \text{mns}_X(C)$ and for Case (ii), $W_2 \in \text{mns}_X(A)$ and $W_2 \in \text{mns}_X(B)$. In both cases, LDECC will mark W_2 as a parent in Lines 16.

Meek Rule 3*:*

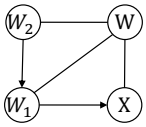
Consider a parent W that gets oriented due to the application of Meek's rule 3 (relevant component of the graph is shown in the figure below).



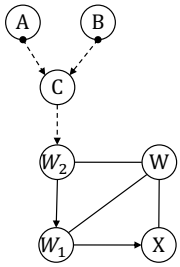
By definition of the Meek rule, W_1—W—W_2 is a non-collider (because if it were a collider, the edges would have been oriented since this triple is unshielded) and therefore for any $\mathbf{S} \subseteq \mathbf{V} \setminus \{W_1, W_2\}$ such that $W_1 \perp\!\!\!\perp W_2 | \mathbf{S}$, we have $W \in \mathbf{S}$. Thus Line 9 will mark W as a parent.

Meek Rule 4*:*

Consider a parent W that gets oriented due to the application of Meek's rule 4. The relevant component of the graph is shown in the figure below.

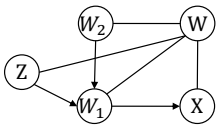


The first possibility is that the orientations $W_2 \rightarrow W_1 \rightarrow X$ were propagated from a UC $A \rightarrow C \leftarrow B$ with a path $C \rightarrow \ldots \rightarrow W_2$ (the relevant components of the graph are shown in the figure below).



In this case, due to the non-collider W_2—W—X (because if it were a collider, the edges would have been oriented since this triple is unshielded), we have $W \in \text{mns}_X(A)$ and thus W will be marked as a parent in Line 16.

The second possibility (similar to the Meek rule 2 case) is that $W_2 \to W_1$ was oriented due to a UC like $Z \to W_1 \leftarrow W_2$ but there is an edge Z—W which shields the $Z \to W_1$—W causing the W_1—W edge to remain unoriented (the relevant components of the graph are shown in the figure below).



In this case, we would have $W \in \text{mns}_X(Z)$ and thus W gets marked as a parent in Line 16.

Children are oriented correctly. We now similarly show that children of X get oriented correctly.

The simplest case is when there is a UC of the form $X \to M \leftarrow V$. Since $\text{MB}(X)$ and $\text{Ne}(X)$ are correct, the function *GetUCChildren* (Fig. 4) will mark M as a child.

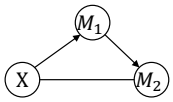
Now, we consider each Meek rule one at a time and show that LDECC will orient children for each rule.

Meek Rule 1*:*

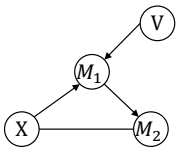
Consider a child M that gets oriented due to the application of Meek's rule 1. This can only happen if there is some parent W that gets oriented and W—X—M forms an unshielded non-collider. In this case, Line 19 will mark M as a child.

Meek Rule 2*:*

Consider a child M_2 that gets oriented due to the application of Meek's rule 2: there is an oriented path $X \to M_1 \to M_2$ but the X—M_2 edge is still unoriented (the relevant component of the graph is shown in the figure below).



One possibility is that there is UC of the form $V \to M_1 \to X$ which orients the $X \to M_1$ edge and V—M_1—M_2 is a non-collider which orients the $M_1 \to M_2$ edge (the relevant components of the graph are shown in the figure below).



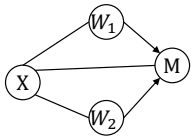
In this case, since V is a spouse (i.e., parent of child) of X, the function *GetUCChildren* (Fig. 4) will mark M_2 as a child.

Note that if the $X \to M_1$ was oriented due to a UC upstream of X via the application of Meek rule 1, this would also cause the $X \to M_2$ edge to be oriented (and thus Meek rule 2 would not apply).

The other possibility is that there might be a UC of the form $M_1 \to M_2 \leftarrow Z$ that can orient $M_1 \to M_2$. However, for the $X \to M_1$ edge to remain unoriented, there must be an edge Z—X to shield the X—M_2—Z collider. If this happens, Meek rule 3 would apply (which we handle separately as shown next).

Meek Rule 3*:*

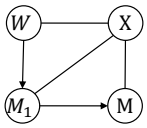
Consider a child M that gets oriented due to the application of Meek's rule 3 (the relevant component of the graph is shown in the figure below).



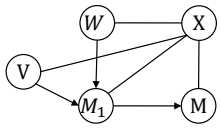
By definition of the Meek rule, W_1—X—W_2 is a non-collider (because if it were a collider, the edges would have been oriented since this triple is unshielded) and since $W_1 \to M \leftarrow W_2$ forms a collider, we have $W_1 \not\perp\!\!\!\perp W_2 | \mathbf{S} \cup \{M\}$ for any $\mathbf{S}$ s.t. $W_1 \perp\!\!\!\perp W_2 | \mathbf{S}$. Thus Line 13 will mark M as a child.

Meek Rule 4*:*

Consider a child M that gets oriented due to the application of Meek's rule 4 (the relevant component of the graph is shown in the figure below).

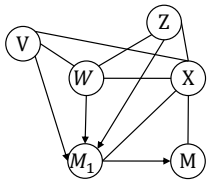


One possibility such that the $W \to M_1$ gets oriented leaving the edges W—X, X—M, and X—M_1 unoriented is if there is a UC of the form $V \to M_1 \leftarrow W$ where there is an edge V—X to shield the X—M_1 edge (the relevant component of the graph is shown in the figure below).



Here the triple W—X—V must be a non-collider to keep the X—M edge unoriented (otherwise applying Meek rule 1 from the UC $W \to X \leftarrow V$ would orient $X \to M$). So for any $\mathbf{S}$ such that $V \perp\!\!\!\perp W | \mathbf{S}$, we must have $X \in \mathbf{S}$ and $V \not\perp\!\!\!\perp W | \mathbf{S} \cup \{M\}$. Therefore, Line 13 will mark M as a child.

The other possibility is that the $W \to M_1$ gets oriented due to Meek rule 3 (the relevant component of the graph is shown in the figure below).



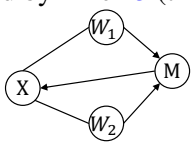
Here the Z—X and V—X must be present to keep the X—M_1 edge unoriented (because otherwise an unshielded collider would be created). Furthermore, the triple Z—X—V must be a non-collider in order to keep the X—M edge unoriented (otherwise applying Meek rule 1 from the UC $Z \to X \leftarrow V$ would orient $X \to M$). So for any $\mathbf{S}$ such that $V \perp\!\!\!\perp Z|\mathbf{S}$, we must have $X \in \mathbf{S}$ and $V \not\perp\!\!\!\perp Z|\mathbf{S} \cup \{M\}$. Therefore, Line 13 will mark M as a child.

No spurious orientations. Now we prove that nodes are never oriented the wrong way by LDECC.

We show that *GetUCChildren* (Fig. 4) will never orient a parent as a child. We prove this by contradiction. Let's say there is a parent W that is oriented as a child by *GetUCChildren*. This will happen if there is a node $D \in \text{MB}(X) \setminus \text{Ne}(X)$ s.t. $D \not\perp\!\!\!\perp W|\text{DSep}(D, X)$ with $W \notin \text{DSep}(D, X)$. Thus, there would a path D—W. Since W is a parent, there would be a path D—$W \to X$ leading to $W \in \text{DSep}(D, X)$ which is a contradiction.

Line 8 can never mark a child as a parent since otherwise the CFA would be violated.

Line 13 will not mark a parent as a child. Consider a parent M that incorrectly gets marked as a child by Line 13 (the relevant component of the graph is shown in the figure below).



For Line 13 to be reached, the if-condition in Line 10 must be *True*. This will happen if $W1$—X—$W2$ is a non-collider. Thus at least one of W_1 or W_2 is a child. W.l.o.g., let's assume that W_1 is a child. If that happens, a cycle gets created: $X \to W_1 \to M \to X$. Therefore M can never be oriented by Line 13.

Line 16 cannot mark a child as a parent because of the correctness of the ECC (Prop. 4).

Line 9 will not mark a child as a parent. Both A and B from Line 7 are parents of X. By Lemma 27, the set $\mathbf{S}$ in Line 9 cannot contain a child.

Line 19 will not add a child as a parent because otherwise the CFA would be violated. ■

B.2. Omitted proofs for computational requirements (Sec. 4.1)

Proposition 6 *[PC vs LDECC] We have* $T_{LDECC} \leq T_{PC} + \mathcal{O}(|\mathbf{V}|^2) + \mathcal{O}\left(|\mathbf{V}| \cdot 2^{|Ne(X)|}\right)$.

Proof LDECC performs $\mathcal{O}\left(|\mathbf{V}| \cdot 2^{|\text{Ne}(X)|}\right)$ CI tests to find $\text{Ne}(X)$. After that LDECC runs CI tests like PC. The *GetMNS* function requires $\mathcal{O}(2^{|\text{Ne}(X)|})$ CI tests. The $\mathcal{O}(|\mathbf{V}|^2)$ term accounts for the extra CI tests of the form $A \not\perp\!\!\!\perp B|\mathbf{S} \cup \{X\}$ we might run for ECCs. ■

Proposition 11 *[LDECC upper bound] For a locally orientable DAG* $\mathcal{G}^*$*, let* $D = \max_{V \in MB^+(X)} |Ne(V)|$ *and* $S = \max_{P \in Pa(X)} \min_{\alpha \in POC(P)} sep(\alpha)$*. Then* $T_{LDECC} \leq \mathcal{O}(|\mathbf{V}|^{\max\{S,D\}})$.

Proof Since the graph is locally orientable, all neighbors of X will get oriented. The complexity of discovering the neighbors of X is upper bounded by $\mathcal{O}(|\mathbf{V}|^{|\text{MB}^+(X)|})$; that of discovering any non-colliders of the form A—X—B is $\mathcal{O}(|\mathbf{V}|^D)$; and in order to unshield the colliders that orient the parents, LDECC runs $\mathcal{O}(|\mathbf{V}|^S)$ tests. Thus the total number of CI tests is $\mathcal{O}(|\mathbf{V}|^{\max\{S,D\}})$. ■

Proposition 12 *[SD upper bound] For a locally orientable DAG $\mathcal{G}^*$, let $\pi : \mathbf{V} \to \mathbb{N}$ be the order in which nodes are processed by SD (Line 4 of SD). For $P \in Pa(X)$, let $CUC(P) = \operatorname{argmin}_{\alpha \in POC(P)} \pi(\alpha)$ denote the closest UC to P. Let $C = \max_{P \in Pa(X)} sep(CUC(P))$, $D = \max_{V \in MB^+(X)} |Ne(V)|$, and $E = \max_{\{V : \pi(V) < \pi(CUC(P))\}} |Ne(V)|$. Then $T_{SD} \leq \mathcal{O}(|\mathbf{V}|^{\max\{C,D,E\}})$.*

Proof Like LDECC, the complexity of discovering the neighbors of X and non-colliders of the form A—X—B is at most $\mathcal{O}(|\mathbf{V}|^D)$ tests. Then, in order to sequentially reach the closest UCs, SD runs $\mathcal{O}(|\mathbf{V}|^E)$ tests. Once the collider is reached, SD runs $\mathcal{O}(|\mathbf{V}|^C)$ CI tests to unshield them. ■

B.3. Omitted proofs for faithfulness requirements (Sec. 4.2)

Proposition 17 *PC will identify the MEC of $\mathcal{G}^*$ if LF holds for all nodes.*

Proof It is known that the PC algorithm correctly identifies the MEC of $\mathcal{G}^*$ if AF and OF hold for all nodes (see e.g., Zhang and Spirtes (2008)). AF for all nodes ensures that the skeleton is recovered correctly. OF for all unshielded triples ensures that UCs are detected correctly and that the orientations propagated via Meek's rules are correct. ■

Proposition 18 *[Faithfulness for PC and SD] PC and SD will identify Θ^* if (i) LF holds $\forall V \in MB^+(X)$; (ii) $\forall (A \to C \leftarrow B) \in J^*$, (a) LF holds for A, B, and C, and (b) LF holds for each node on all paths $C \to \ldots \to V \in \mathcal{G}^*$ s.t. $V \in Ne(X)$; (iii) For every edge A—$B \notin \mathcal{G}^*$, $\exists \mathbf{S} \subseteq (Ne_{\mathcal{U}}(A) \cup Ne_{\mathcal{U}}(B))$ s.t. $A \perp\!\!\!\perp B | \mathbf{S}$; and (iv) OF holds for all unshielded triples in $\mathcal{G}^*$.*

Proof Similar to the proof of Thm. 5, we will prove this by showing that all neighbors of X get oriented correctly and the unorientable neighbors remain unoriented.

The key ideas of the proof are as follows: (1) Condition (i) guarantees that the structure inside $\text{MB}^+(X)$ is discovered correctly which further ensures that Meek rules 2-4 work correctly (since, as shown in the proof of Thm. 5, they are only applied inside $\text{MB}^+(X)$); (2) Condition (ii)(a) guarantees that each UC in $\mathcal{G}^*$ is detected and unshielded; (3) Condition (ii)(b) guarantees that orientations from each UC in $\mathcal{G}^*$ are propagated correctly to X; (4) Condition (iii) guarantees that the undirected skeleton discovered by PC and SD is a subgraph of the skeleton of $\mathcal{G}^*$: This is because PC and SD remove an edge A—B by running CI tests by conditioning on neighbors of A and B in the current undirected skeleton; and (5) Condition (iv) guarantees that in the skeleton recovered by PC and SD, there are no incorrectly detected UCs.

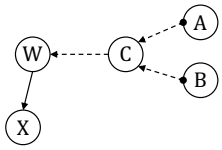
Parents are oriented correctly. In PC and SD, edges get oriented using UCs and then additional orientations are propagated via the application of Meek's rules (Figure 13).

The simplest case is where two parents form a UC at X. Consider parents W_1 and W_2 that get oriented because they form a UC $W_1 \to X \leftarrow W_2$. Condition (i) ensures they are marked as parents.

We will now consider parents that get oriented due to each of the four Meek rules and show that SD and PC orient parents for each of the four cases.

Meek Rule 1*:*

Consider a parent W that gets oriented due to the application of Meek's rule 1. This can only happen due to some UC $A \to C \leftarrow B$ from which these orientations have been propagated (relevant components of the graph are illustrated in the figure below).



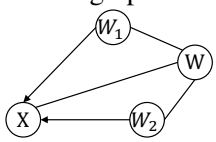
Thus there is a directed path $C \to \ldots \to W \to X$. By Condition (ii)(a), the UC $A \to C \leftarrow B$ will get detected correctly, and by Condition (ii)(b), the orientations will be propagated correctly along this path.

Meek Rule 2*:*

Consider a parent W_2 that gets oriented due to the application of Meek's rule 2. In this case, we have an oriented path $W_2 \to W_1 \to X$ but the edge W_2—X is unoriented (and Meek Rule 2 must be applied to orient it). Condition (i) ensures that this relevant component of the graph is discovered correctly.

Meek Rule 3*:*

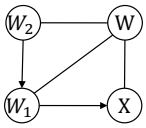
Consider a parent W that gets oriented due to the application of Meek's rule 3 (relevant component of the graph is shown in the figure below).



Condition (i) ensures that this relevant component of the graph is discovered correctly.

Meek Rule 4*:*

Consider a parent W that gets oriented due to the application of Meek's rule 4. The relevant component of the graph is shown in the figure below.



Condition (i) ensures that this relevant component of the graph is discovered correctly.

Children are oriented correctly. We now similarly show that children of X get oriented correctly.

The simplest case is when there is a UC of the form $X \to M \leftarrow V$. Condition (i) ensures that this UC is discovered correctly.

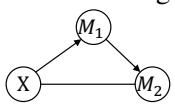
Now, we consider each Meek rule one at a time and show that SD and PC will orient children for each rule.

Meek Rule 1*:*

Consider a child M that gets oriented due to the application of Meek's rule 1. This can only happen if there is some parent W that gets oriented and W—X—M forms an unshielded non-collider. Condition (i) ensures that this unshielded non-collider is discovered correctly.

Meek Rule 2*:*

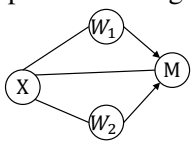
Consider a child M_2 that gets oriented due to the application of Meek's rule 2: there is an oriented path $X \to M_1 \to M_2$ but the X—M_2 edge is still unoriented (the relevant component of the graph is shown in the figure below).



Condition (i) ensures that this relevant component of the graph is discovered correctly.

Meek Rule 3*:*

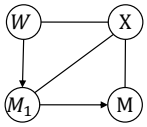
Consider a child M that gets oriented due to the application of Meek's rule 3 (the relevant component of the graph is shown in the figure below).



Condition (i) ensures that this relevant component of the graph is discovered correctly.

Meek Rule 4*:*

Consider a child M that gets oriented due to the application of Meek's rule 4 (the relevant component of the graph is shown in the figure below).



Condition (i) ensures that this relevant component of the graph is discovered correctly.

No spurious orientations. As argued in the preamble of the proof, Conditions (iii, iv) ensure that there are no incorrectly detected UCs. Condition (ii) ensures that no incorrect orientations are propagated from the detected UCs. ■

Proposition 20 *[Faithfulness for LDECC] LDECC will identify Θ^* if (i) LF holds $\forall V \in MB^+(X)$; (ii) $H \subseteq H^*$; (iii) $\forall (A, B, \mathbf{S}) \in H$, MFF holds for $\{A, B\} \setminus Ne(X)$; and (iv) $\forall (A, B, \mathbf{S}) \in H^*$ s.t. there is a UC $(A \to C \leftarrow B) \in \mathcal{G}^*$, we have (a) AF holds for A and B; and (b) $(A, B, \mathbf{S}) \in H$.*

Proof The proof is very similar to that of Theorem 5. The high-level idea is as follows. For any nodes in $\text{Ne}(X)$ that are oriented *without* using ECCs, Condition (i) will ensure they get oriented correctly as they only depend on the structure inside $\text{MB}(X)$. For nodes that get oriented via ECCs, Condition (iv)(a) ensures that each UC eventually gets unshielded. For any UC $A \to C \leftarrow B$ in Condition (iv), since AF holds for both A and B, $\text{Ne}(A)$ and $\text{Ne}(B)$ are detected correctly. Since $\exists \mathbf{S} \subseteq (\text{Ne}(A) \cup \text{Ne}(B))$ s.t. $A \perp\!\!\!\perp B | \mathbf{S}$, we will eventually remove the A—B thereby unshielding this collider. Condition (iv)(b) ensures that we run an ECC for this UC, i.e., the if-block in Line 15 is entered. Next, by Condition (iii), since MFF holds for A and B, the *GetMNS* function will correctly return the parents of X that this UC orients.

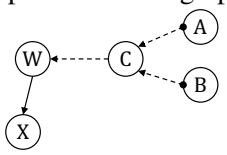
We assume that the function *FindMarkovBlanket* identifies the Markov blanket correctly under Condition (i). The IAMB algorithm, which we use in our experiments, has this property. Additionally, the function *RunLocalPC* will also identify the correct $\text{Ne}(X)$ under Condition (i).

Parents are oriented correctly. The simplest case is where two parents form a UC at X. Consider parents W_1 and W_2 that get oriented because they form a UC $W_1 \to X \leftarrow W_2$. By Condition (i), Lines 7,8 will mark W_1 and W_2 as parents.

We will now consider parents that get oriented due to each of the four Meek rules and show that LDECC orients parents for each of the four cases.

Meek Rule 1*:*

Consider a parent W that gets oriented due to the application of Meek's rule 1. This can only happen due to some UC $A \to C \leftarrow B$ from which these orientations have been propagated (relevant components of the graph are illustrated in the figure below).

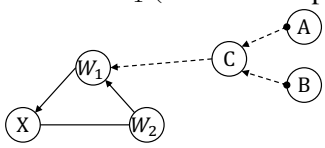


As explained in the preamble, Conditions (iii, iv) ensure that ECCs mark parent correctly and thus Line 16 will mark W as a parent.

Meek Rule 2*:*

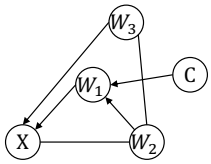
Consider a parent W_2 that gets oriented due to the application of Meek's rule 2. In this case, we have an oriented path $W_2 \to W_1 \to X$ but the edge W_2—X is unoriented (and Meek Rule 2 must be applied to orient it).

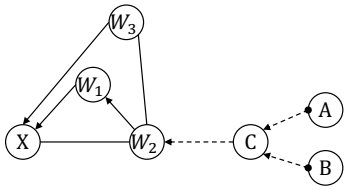
The first possibility is that the $X \leftarrow W_1$ was oriented due to some UC $A \to C \leftarrow B$ with a path $C \to \ldots \to W_1$ (relevant components of the graph are illustrated in the figure below):



By Conditions (iii, iv), LDECC will mark W_2 as a parent in Line 16.

The other possibility is that $X \leftarrow W_1$ was oriented due to a UC like $W_3 \to X \leftarrow W_1$ but there is an edge W_3—W_2 which causes the collider $W_2 \to X \leftarrow W_3$ to be shielded and due to this, the W_2—X remained unoriented. However, by definition of the Meek rule, the edge $W_2 \to W_1$ is oriented. Thus, (i) either there is a UC of the form $W_2 \to W_1 \leftarrow C$; or (ii) there is a UC from which the $W_2 \to W_1$ orientation was propagated. The relevant components of the graph for these two cases are illustrated in the figures below.

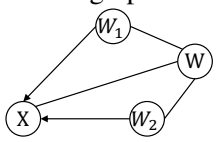




In both cases, by Conditions (iii, iv), LDECC will mark W_2 as a parent in Lines 16.

Meek Rule 3*:*

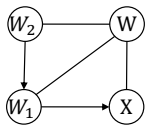
Consider a parent W that gets oriented due to the application of Meek's rule 3 (relevant component of the graph is shown in the figure below).



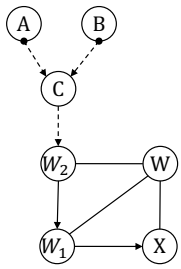
By definition of the Meek rule, W_1—W—W_2 is a non-collider (because if it were a collider, the edges would have been oriented since this triple is unshielded) and therefore for any $\mathbf{S} \subseteq \mathbf{V} \setminus \{W_1, W_2\}$ such that $W_1 \perp\!\!\!\perp W_2 | \mathbf{S}$, by Condition (i), we have $W \in \mathbf{S}$. Thus Line 9 will mark W as a parent.

Meek Rule 4*:*

Consider a parent W that gets oriented due to the application of Meek's rule 4. The relevant component of the graph is shown in the figure below.

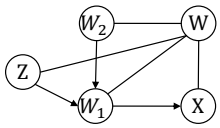


The first possibility is that the orientations $W_2 \to W_1 \to X$ were propagated from a UC $A \to C \leftarrow B$ with a path $C \to \ldots \to W_2$ (the relevant components of the graph are shown in the figure below).



In this case, due to the non-collider W_2—W—X (because if it were a collider, the edges would have been oriented since this triple is unshielded), we have $W \in \text{mns}_X(A)$ and thus by Conditions (iii, iv), W will be marked as a parent in Line 16.

The second possibility (similar to the Meek rule 2 case) is that $W_2 \to W_1$ was oriented due to a UC like $Z \to W_1 \leftarrow W_2$ but there is an edge Z—W which shields the $Z \to W_1$—W causing the W_1—W edge to remain unoriented (the relevant components of the graph are shown in the figure below).



In this case, we would have $W \in \text{mns}_X(Z)$ and thus by Conditions (iii, iv), W gets marked as a parent in Line 16.

Children are oriented correctly. We now similarly show that children of X get oriented correctly.

The simplest case is when there is a UC of the form $X \to M \leftarrow V$. By Condition (i), since MB(X) and Ne(X) are correct, the function *GetUCChildren* (Fig. 4) will mark M as a child.

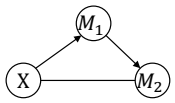
Now, we consider each Meek rule one at a time and show that LDECC will orient children for each rule.

Meek Rule 1*:*

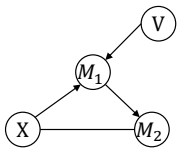
Consider a child M that gets oriented due to the application of Meek's rule 1. This can only happen if there is some parent W that gets oriented and W—X—M forms an unshielded non-collider. By Condition (i), Line 19 will mark M as a child.

Meek Rule 2*:*

Consider a child M_2 that gets oriented due to the application of Meek's rule 2: there is an oriented path $X \to M_1 \to M_2$ but the X—M_2 edge is still unoriented (the relevant component of the graph is shown in the figure below).



One possibility is that there is UC of the form $V \to M_1 \to X$ which orients the $X \to M_1$ edge and V—M_1—M_2 is a non-collider which orients the $M_1 \to M_2$ edge (the relevant components of the graph are shown in the figure below).



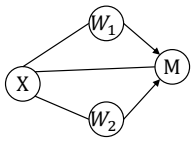
In this case, since V is a spouse (i.e., parent of child) of X, by Condition (i), the function *GetUCChildren* (Fig. 4) will mark M_2 as a child.

Note that if the $X \to M_1$ was oriented due to a UC upstream of X via the application of Meek rule 1, this would also cause the $X \to M_2$ edge to be oriented (and thus Meek rule 2 would not apply).

The other possibility is that there might be a UC of the form $M_1 \to M_2 \leftarrow Z$ that can orient $M_1 \to M_2$. However, for the $X \to M_1$ edge to remain unoriented, there must be an edge Z—X to shield the X—M_2—Z collider. If this happens, Meek rule 3 would apply (which we handle separately as shown next).

Meek Rule 3*:*

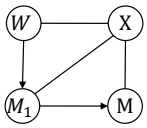
Consider a child M that gets oriented due to the application of Meek's rule 3 (the relevant component of the graph is shown in the figure below).



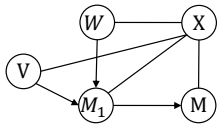
By definition of the Meek rule, W_1—X—W_2 is a non-collider (because if it were a collider, the edges would have been oriented since this triple is unshielded) and since $W_1 \to M \leftarrow W_2$ forms a collider, we have $W_1 \not\perp\!\!\!\perp W_2 | \mathbf{S} \cup \{M\}$ for any $\mathbf{S}$ s.t. $W_1 \perp\!\!\!\perp W_2 | \mathbf{S}$. Thus, by Condition (i), Line 13 will mark M as a child.

Meek Rule 4*:*

Consider a child M that gets oriented due to the application of Meek's rule 4 (the relevant component of the graph is shown in the figure below).

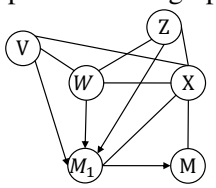


One possibility such that the $W \to M_1$ gets oriented leaving the edges W—X, X—M, and X—M_1 unoriented is if there is a UC of the form $V \to M_1 \leftarrow W$ where there is an edge V—X to shield the X—M_1 edge (the relevant component of the graph is shown in the figure below).



Here the triple W—X—V must be a non-collider to keep the X—M edge unoriented (otherwise applying Meek rule 1 from the UC $W \to X \leftarrow V$ would orient $X \to M$). So for any $\mathbf{S}$ such that $V \perp\!\!\!\perp W|\mathbf{S}$, by Condition (i), we must have $X \in \mathbf{S}$ and $V \not\perp\!\!\!\perp W|\mathbf{S} \cup \{M\}$. Therefore, Line 13 will mark M as a child.

The other possibility is that the $W \to M_1$ gets oriented due to Meek rule 3 (the relevant component of the graph is shown in the figure below).



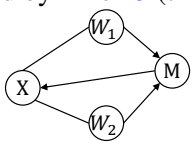
Here the Z—X and V—X must be present to keep the X—M_1 edge unoriented (because otherwise an unshielded collider would be created). Furthermore, the triple Z—X—V must be a non-collider in order to keep the X—M edge unoriented (otherwise applying Meek rule 1 from the UC $Z \to X \leftarrow V$ would orient $X \to M$). So for any $\mathbf{S}$ such that $V \perp\!\!\!\perp Z|\mathbf{S}$, by Condition (i), we must have $X \in \mathbf{S}$ and $V \not\perp\!\!\!\perp Z|\mathbf{S} \cup \{M\}$. Therefore, Line 13 will mark M as a child.

No spurious orientations. Now we prove that nodes are never oriented the wrong way by LDECC.

By Condition (i), *GetUCChildren* (Fig. 4) will never orient a parent as a child.

Line 8 can never mark a child as a parent since otherwise Condition (i) would be violated.

Line 13 will not mark a parent as a child. Consider a parent M that incorrectly gets marked as a child by Line 13 (the relevant component of the graph is shown in the figure below).



For Line 13 to be reached, the if-condition in Line 10 must be *True*. This will happen if $W1$—X—$W2$ is a non-collider. Thus at least one of W_1 or W_2 is a child. W.l.o.g., let's assume that W_1 is a child. If that happens, a cycle gets created: $X \to W_1 \to M \to X$. Therefore, Condition (i) ensures that M can never be oriented by Line 13.

By Condition (ii), every ECC that is run is valid, and thus Line 16 cannot mark a child as a parent because of the correctness of the ECC (Prop. 4).

Similarly, by Condition (i), Line 9 will not mark a child as a parent. Both A and B from Line 7 are parents of X. By Lemma 27, a child cannot d-separate two non-descendants nodes and thus the set $\mathbf{S}$ in Line 9 cannot contain a child.

Line 19 will not add a child as a parent because otherwise Condition (i) would be violated. ■

We now provide sufficient faithfulness conditions for LDECC when *ECCParents*$(A, B, \mathbf{S})$ is run with *check*=*True*.

For a node $V \notin \text{Ne}^+(X)$, let $\mathbf{Q}(V) = \{\mathbf{S} \subseteq \text{Ne}(X) : V \perp\!\!\!\perp X|\mathbf{S}\}$ and $\mathbf{Q}_{\min}(V) = \{\mathbf{S} \in \mathbf{Q}(V) : |\mathbf{S}| = \min_{\mathbf{S}' \in \mathbf{Q}(V)} |\mathbf{S}'|\}$. With H^* and H as defined in Sec. 4.2, let $H^{(\text{check})} = \{(A, B, \mathbf{S}) \in H : \{A, B\} \cap \text{Ne}(X) = \emptyset$; and $\mathbf{Q}_{\min}(A) \cap \mathbf{Q}_{\min}(B) \neq \emptyset\}$; let $H^{(\text{single})} = \{(A, B, \mathbf{S}) \in H : |\{A, B\} \cap \text{Ne}(X)| = 1\}$; and let $H^{(\text{total})} = H^{(\text{check})} \cup H^{(\text{single})}$.

Lemma 28 *For a true graph $\mathcal{G}^*$, let $(A \to C \leftarrow B)$ be a UC such that $(A, B, \mathbf{S}) \in H^*$ for some subset $\mathbf{S}$. Then, we have that $mns_X(A) = mns_X(B)$.*

Proof Since $(A, B, \mathbf{S}) \in H^*$, by Prop. 4, this UC can be used to orient parents via an ECC. We begin by considering node A. By definition of MNS, we have $A \perp\!\!\!\perp X | \text{mns}_X(A)$. Since the nodes in $\text{mns}_X(A)$ are on the path from $X \leftarrow \ldots \leftarrow C \leftarrow A$, conditioning on $\text{mns}_X(A)$ opens up the $A \rightarrow C \leftarrow B$ path (because the nodes in $\text{mns}_X(A)$ are descendants of C). Therefore, we have $\text{mns}_X(A) \subseteq \text{mns}_X(B)$. We can make a similar argument for node B to show that $\text{mns}_X(B) \subseteq \text{mns}_X(A)$. Combining the two statements, we get $\text{mns}_X(A) = \text{mns}_X(B)$. ■

Proposition 29 (Weaker faithfulness for LDECC) *If* ECCParents*($A, B, \mathbf{S}$) is run with* check=True*, then LDECC will identify Θ^* if: (i) LF holds for every $V \in Ne^+(X)$; (ii) $H^{(total)} \subseteq H^*$; (iii) $\forall (A, B, \mathbf{S}) \in H^{(total)}$, MFF holds for $\{A, B\} \setminus Ne(X)$; and (iv) $\forall (A, B, \mathbf{S}) \in H^*$ s.t. there is a UC $(A \rightarrow C \leftarrow B) \in \mathcal{G}^*$, we have (a) AF holds for A and B; and (b) $(A, B, \mathbf{S}) \in H^{(total)}$. Futhermore, Conditions (i)-(iv) of this proposition are implied by the conditions of Prop. 20 (i.e., this is a weaker sufficient faithfulness condition for LDECC).*

Proof The can be proved in the same way as Prop. 20 with the crucial difference that the set of ECCs that LDECC now runs is restricted to the set $H^{(\text{total})}$. Condition (ii) now ensures that every ECC that is run by LDECC is a valid ECC. We just have to show that even by restricting the ECCs to $H^{(\text{total})}$, we still run an ECC for the UCs. For this, we leverage Lemma 28 which states that the MNS of nodes A and B for the UC $A \rightarrow C \leftarrow B$ will be the same. Since, by Condition (iii), MFF holds for such nodes A and B, the check *GetMNS*(A) = *GetMNS*(B) will succeed and thus, the ECCs for these UCs will still be run.

Now, we show that the conditions of this proposition are implied by those of Prop. 20. Conditions (i),(iv)(a) of both propositions are the same. Observe that $H^{(\text{total})} \subseteq H$. Therefore, Conditions (ii),(iii) of Prop. 20 imply Conditions (ii),(iii) of this proposition. Finally, we show that Condition (iv)(b) of this proposition is also implied by the conditions of Prop. 20. Consider any $(A, B, \mathbf{S}) \in H^*$ such that the UC $A \rightarrow C \leftarrow B \in \mathcal{G}^*$ and $\{A, B\} \cap \text{Ne}(X) = \emptyset$. If $|\{A, B\} \cap \text{Ne}(X)| = 1$, then $(A, B, \mathbf{S}) \in H^{(\text{single})} \subseteq H^{(\text{total})}$. If $\{A, B\} \cap \text{Ne}(X) = \emptyset$, then by Condition (iii) of Prop. 20, MFF holds for A and B are therefore $Q_{\min}(A) \cap Q_{\min}(B) = \text{mns}_X(A) = \text{mns}_X(B)$. Thus, $(A, B, \mathbf{S}) \in H^{(\text{check})} \subseteq H^{(\text{total})}$. ■

Proposition 23 *[Testing faithfulness for LDECC] Consider running the algorithm in Fig. 8 before invoking* GetMNS*(A) for some node A in LDECC. If the algorithm returns* Fail*, MFF is violated for node A. If the algorithm returns* Unknown*, we could not ascertain if MFF holds for node A.*

Proof The set $\mathbf{Q}$ contains all subsets $\mathbf{S} \subseteq \text{Ne}(X)$ s.t. $A \perp\!\!\!\perp X | \mathbf{S}$ (Lines 1–4). The set $\mathbf{Q}_{\min}$ contains those sets from $\mathbf{Q}$ that are minimal, i.e., for every $\mathbf{S}' \in \mathbf{Q}_{\min}$, there is no subset of $\mathbf{S}'$ in $\mathbf{Q}$. If $|\mathbf{Q}_{\min}| > 1$, then there are multiple possible $\text{mns}_X(A)$ violating the uniqueness of MNS (Prop. 3). Thus we return *Fail* (Example 3 demonstrates this failure case).

Next, if Line 5 is reached, we know that $|\mathbf{Q}_{\min}| = 1$ and $\mathbf{S}$ is the single element from $\mathbf{Q}_{\min}$. Line 7 returns *Fail* if there is a set $\mathbf{S}'$ such that $\mathbf{S} \subset \mathbf{S}'$ and $\mathbf{S}' \in \mathbf{Q}$, and an intermediate set $\mathbf{S}''$ such that $\mathbf{S} \subset \mathbf{S}'' \subset \mathbf{S}'$ and $\mathbf{S}'' \notin \mathbf{Q}$ (Example 4 demonstrates this failure case). Here MFF fails because if $\mathbf{S}$ was the correct $\text{mns}_X(A)$, then we must have $\mathbf{S}'' \in \mathbf{Q}$. This is because, since $\mathbf{S}' \in \mathbf{Q}$, there cannot be an active path from A to X through nodes in $\mathbf{S}' \setminus \mathbf{S}$ (otherwise, we would not have $A \perp\!\!\!\perp X | \mathbf{S}$). Therefore, adding nodes in $\mathbf{S}'' \setminus \mathbf{S}$ to the conditioning set should not violate the independence. But

since we have $\mathbf{S}'' \notin \mathbf{Q}$, there must be a faithfulness violation and $\mathbf{S}$ is not guaranteed to be equal to $\text{mns}_X(A)$.

If Line 9 is reached, we have not been able to detect an MFF violation. However, the algorithm is not *complete*, i.e., a failure to detect a violation does not mean a violation does not exist. So we return *Unknown* which signifies that we were unable to ascertain if MFF was violated for A.

The additional tests are performed in Line 1. Since these tests are performed for every subset $\mathbf{S} \subseteq \text{Ne}(X)$, the number of extra CI tests is $\mathcal{O}(2^{|\text{Ne}(X)|})$.

■

Proposition 24 *[Testing faithfulness for SD] Consider running the algorithm in Fig. 9 with each UC detected by SD. If the algorithm returns* Fail*, then faithfulness is violated for SD. If the algorithm returns* Unknown*, we could not ascertain if the faithfulness assumptions for SD hold.*

Proof In Line 3, $\mathbf{M}$ contains every neighbor of X that gets oriented as a parent due to the input UC $A \to C \leftarrow B$ by SD. If the faithfulness assumptions for SD did hold, then all nodes in $\mathbf{M}$ would actually be parents of X. Thus there should be at least one subset $\mathbf{S} \subseteq \text{Ne}(X)$ such that $A \perp\!\!\!\perp X|\mathbf{S}$ and $\mathbf{M} \subseteq \mathbf{S}$, and likewise for B. If such a subset is not found, this means that one of the nodes that was marked as a parent was actually a child. In this case, Line 6 would return *Fail*. Similarly to the LDECC case, if we are unable to detect a faithfulness violation, we return *Unknown* to indicate that we could not determine if the faithfulness assumptions for SD hold. Since CI tests are performed for every subset $\mathbf{S} \subseteq \text{Ne}(X)$, the number of extra CI tests is $\mathcal{O}(2^{|\text{Ne}(X)|})$. ■

Appendix C. Experiments

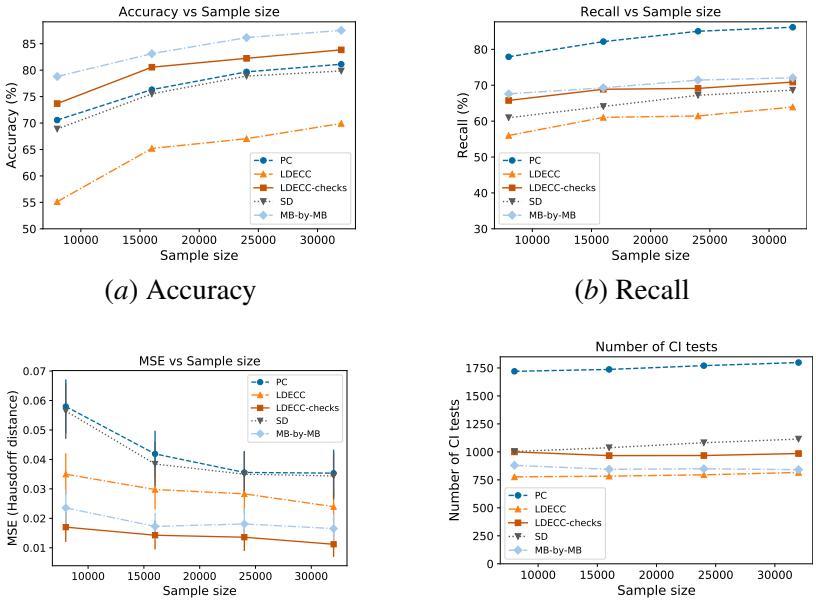
C.1. More details for the synthetic linear graph experiments.

We generate synthetic linear graphs with Gaussian errors, $N_c = 20$ covariates—non-descendants of X and Y with paths to both X and Y—and $N_m = 3$ mediators—nodes on some causal paths from X to Y. We generate edges between the different types of nodes with varying probabilities: (i) We connect the covariates to the treatment with probability p_{cx}; (ii) We connect one covariate to another with probability p_{cc}; (iii) We connect the covariates to the outcome with probability p_{cy}; (iv) We connect the treatment to the mediators with probability p_{mx}; (v) We connect one mediator to another with probability p_{mm}; (vi) We connect the mediators to the outcome with probability p_{my}; (vi) We connect a mediators to a covariate with probability p_{cm}. For our experiments, we have used $p_{cx} = p_{cc} = p_{cy} = p_{mx} = p_{mm} = p_{my} = 0.1$ and $p_{cm} = 0.05$.

For each node V, we generate data using the following structural equation:

$$v := b_V^\top \text{pa}(v) + \epsilon_V, \;\; \epsilon_V \sim \mathcal{N}(0, \sigma_V^2),$$

where v and $\text{pa}(v)$ are the realized values of node V and its parents, respectively; $b_V^\top \in \mathbb{R}^{|\text{Pa}(V)|}$ is the vector denoting the edge coefficients; and ϵ_V is an independently sampled noise term. Each element of b_V is sampled uniformly from the interval $[-1, -0.25] \cup [0.25, 1]$ and σ_V^2 is sampled independently from a uniform distribution $U(0.1, 0.2)$.



(*a*) Accuracy (*b*) Recall

(*c*) MSE vs sample size (*d*) Average number of CI tests

Figure 16: Results on synthetic linear Erdos-Renyi graphs.

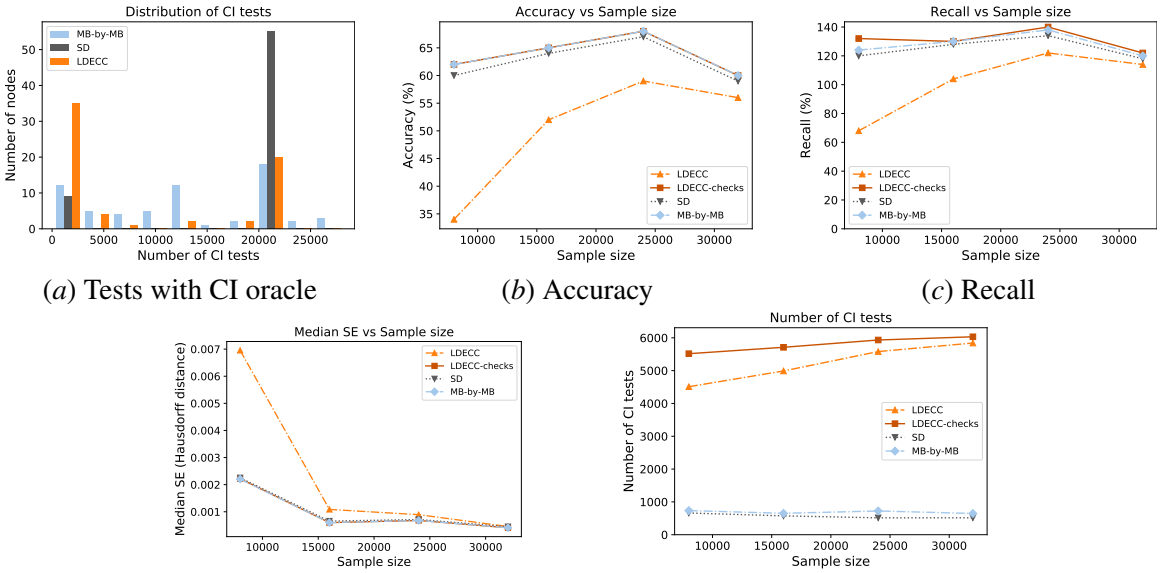
C.2. Results on synthetic linear binomial (Erdos-Renyi) graphs.

We also test our methods on binomial graphs of size $|\mathbf{V}| = 30$. In binomial graphs, an edge between two nodes is generated with some probability p_e. For our experiments, we use $p_e = 0.8$. We generate the data using linear Gaussian models where the model parameters are sampled in the same way as Sec. C.1. We compare PC, SD, MB-by-MB, and LDECC based on accuracy (Fig. 16(*a*)), recall (Fig. 16(*b*)), MSE (Fig. 16(*c*)), and number of CI tests (Fig. 16(*d*)) across four different sample sizes.

In terms of accuracy, MB-by-MB performs the best with LDECC-checks performing slightly better than SD and PC. In terms of recall, PC performs the best with LDECC-checks, MB-by-MB, and SD doing comparably. We also observe that LDECC-checks has higher accuracy and recall than LDECC. LDECC-checks and MB-by-MB have the lower MSE than PC and SD Both variants of LDECC have lower MSE than PC and SD and all four local causal discovery algorithms perform a comparable number of CI tests (and fewer tests than PC).

C.3. Additional results on semi-synthetic graphs.

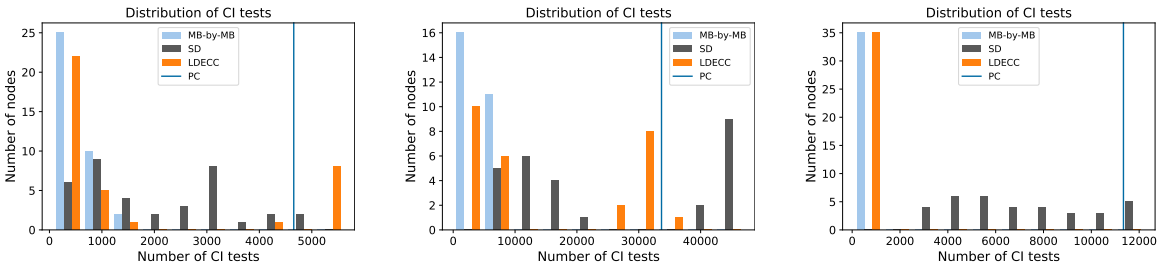
We also present results on the linear Gaussian *MAGIC-IRRI* graph from *bnlearn* (Fig. 17). We plot the distribution of CI tests with a CI oracle by repeatedly setting each node as the treatment (capping the maximum number of tests per node to 20000). We see that LDECC and MB-by-MB perform differently across different nodes and outperform SD on most nodes (Fig. 17(*a*)). Next, we designated the nodes *G6003* and *BROWN* as the treatment and outcome, respectively. At four sample sizes, we sample data from the graph 100 times (capping the maximum number of tests run by each algorithm to 7000). In terms of both accuracy (Fig. 17(*b*)) and recall (Fig. 17(*c*)), LDECC-checks, SD, and MB-by-MB perform comparably while LDECC does worse. In terms of Median SE, LDECC-checks,



(*a*) Tests with CI oracle (*b*) Accuracy (*c*) Recall

(*d*) Median SE vs sample size (*e*) Average number of CI tests

Figure 17: Results on the semi-synthetic *MAGIC-IRRI* graph.



(*a*) *Alarm* graph. (*b*) *Insurance* graph. (*c*) *Mildew* graph.

Figure 18: Comparison of PC, SD, MB-by-MB, and LDECC based on the number of CI tests (with a CI oracle) on three discrete graphs from *bnlearn*.

SD, and MB-by-MB perform comparably (Fig. 17(*d*)) but LDECC performs substantially more CI tests (on average) than SD and MB-by-MB (Fig. 17(*e*)).

Finally, we compare PC, SD, MB-by-MB, and LDECC based on the number of CI tests (with access to a CI oracle) on three discrete graphs from *bnlearn*: *Alarm* (Fig. 18(*a*)), *Insurance* (Fig. 18(*b*)), and *Mildew* (Fig. 18(*c*)). We plot the distribution of tests for SD, MB-by-MB, and LDECC by setting each node in the graph as the treatment. We see that for most nodes on all three graphs, MB-by-MB and LDECC outperform SD. MB-by-MB has the best performance for all three graphs and performs well across all nodes.

Input: Treatment X, Outcome Y, $\mathbf{Z}$.
$\mathbf{M} \leftarrow \text{Ch}(X) \cup \text{Uo}(X)$;
$\mathbf{I} \leftarrow \{V : \text{GetMNS}(V) = \text{invalid}\}$;
$\text{PossDesc}(X) \leftarrow \mathbf{M} \cup \mathbf{I} \cup \{V \in \mathbf{V} \setminus \mathbf{I} : \mathbf{M} \cap \text{GetMNS}(V) \neq \emptyset\}$;
if $\mathbf{Z} \cap \mathit{PossDesc}(X) \neq \emptyset$ **then return** False ;
for $Q \in (\mathit{Pa}(X) \cup \mathit{Uo}(X)) \setminus \mathbf{Z}$ **do**
  if $Q \not\perp\!\!\!\perp Y | \{X\} \cup \mathbf{Z}$ **then return** False ;
end
return True;

Figure 19: Checking the Generalized Backdoor Criterion.

Appendix D. Covariate adjustment using local information

D.1. Checking the backdoor criterion.

The *backdoor criterion* (Pearl, 2009, Defn. 3.3.1) is a sufficient (but not necessary) condition for a given subset of nodes $\mathbf{Z}$ to be a valid adjustment set with respect to a treatment X and an outcome Y. Maathuis and Colombo (2015) extended this criterion to be applicable to various MECs including CPDAGs. We begin by briefly introducing the existing results in Maathuis and Colombo (2015) and then we prove that it is possible to check the backdoor criterion using only local information around X and $\mathcal{O}(|\mathbf{V}| \cdot 2^{|\text{Ne}(X)|})$ additional CI tests.

For introducing existing results, let the true DAG be $\mathcal{G}^*$ and let the corresponding CPDAG that represents the MEC of $\mathcal{G}^*$ be $\mathcal{C}^*$ (see Defn. 26).

Definition 30 (Possibly causal path) *A path $A — \ldots — B$ is said to be possibly causal if there is at least one DAG in $\mathcal{C}^*$ with a directed path from A to B: $A \to \ldots \to B$.*

Definition 31 (Visible and invisible edges (Maathuis and Colombo, 2015, Defn. 3.1)) *All directed edges in a CDPAG are said to be visible. All undirected edges in a CPDAG are said to be invisible.*

Definition 32 (Backdoor path (Maathuis and Colombo, 2015, Defn. 3.2)) *We say that path between X and Y is a backdoor path if this path does not have a visible edge out of X.*

Definition 33 (Definite non-collider (Maathuis and Colombo, 2015, Defn. 3.3)) *A nonendpoint vertex V_j on a path $< \ldots, V_i, V_j, V_k, \ldots >$ in a CPDAG is a definite non-collider if the triple $< V_i, V_j, V_k >$ is unshielded and the edges V_i—V_j and V_j—V_k are undirected.*

Definition 34 (Definite status path (Maathuis and Colombo, 2015, Defn. 3.4)) *A nonendpoint vertex B on a path p in a CPDAG is said to be of a definite status if it is either a collider or a definite non-collider on p. The path p is said to be of a definite status if all nonendpoint vertices on the path are of a definite status.*

Definition 35 (Possible Descendant) *A node A is a possible descendant of a node V iff there is a possibly causal path from V to A.*

Definition 36 (Generalized backdoor criterion (GBC) (Maathuis and Colombo, 2015, Defn. 3.7)) *A set of variables* $\mathbf{Z}$ *satisfies the backdoor criterion relative to* (X, Y) *in a CPDAG if the following two conditions: (1)* $\mathbf{Z}$ *does not contain any possible descendants of* X*; and (2)* $\mathbf{Z}$ *blocks every definite status backdoor path from* X *to* Y*.*

Intuitively, the GBC checks whether $\mathbf{Z}$ satisfies the backdoor criterion for every DAG in an MEC. Having introduced the GBC, we extend this result and prove that it is possible to check this criterion using only local information. Our procedures are related to existing methods in prior work used for covariate selection (VanderWeele and Shpitser, 2011; Entner et al., 2013) which also use very similar testing strategies. However, these works make slightly stronger assumptions on the known partial orderings (e.g., that all covariates are pre-treatment) whereas our goal is to demonstrate that a similar testing strategy along with the output of a local causal discovery algorithm is also sufficient to determine if a given subset is a valid adjustment set. These prior works also accommodate latent pre-treatment variables whereas we make the assumption of causal sufficiency throughout our work.

Definition 37 (Unoriented nodes) *We define unoriented nodes, denoted by* $Uo(X)$*, as the set of nodes* $V \in Ne(X)$ *such that the edge* X—V *is invisible in* $\mathcal{C}^*$*.*

Importantly, both local discovery procedures, SD and LDECC, find $\text{Uo}(X)$. These nodes are stored in the variable *unoriented* in the algorithms (See Figs. 3,5).

Lemma 38 *Let* $\mathbf{M} = Ch(X) \cup Uo(X)$ *and* $\mathbf{I} = \{V : mns_X(V) = invalid\}$*. The possible descendants of a node* X *are* $PossDesc(X) = \mathbf{M} \cup \mathbf{I} \cup \{V \in \mathbf{V} \setminus \mathbf{I} : \mathbf{M} \cap mns_X(V) \neq \emptyset\}$*.*

Proof The nodes in $\mathbf{M}$ are possible descendants of X. By Prop. 2, nodes in $\mathbf{I}$ are also possible descendants. For any possible descendant $V \notin \text{Ne}^+(X) \cup \mathbf{I}$ of X, there must be a path $X \to M \to \ldots \to V$, where $M \in \mathbf{M}$, in at least one DAG. Therefore, it must be the case that for every such node V, we have $\mathbf{M} \cap \text{mns}_X(V) \neq \emptyset$. ■

Proposition 39 (Checking the GBC) *Let* $\mathbf{M} = Ch(X) \cup Uo(X)$*, where* $Uo(X)$ *is defined in Defn. 37. Let* $PossDesc(X) = \mathbf{M} \cup \{V : \mathbf{M} \cap mns_X(V) \neq \emptyset\}$*. Consider a subset of nodes* $\mathbf{Z}$*. Let* $\mathbf{Q} = (Pa(X) \cup Uo(X)) \setminus \mathbf{Z}$*. Then* $\mathbf{Z}$ *satisfies the backdoor criterion for every DAG in the MEC iff: (i)* $\mathbf{Z} \cap PossDesc(X) = \emptyset$*, and (ii)* $\forall Q \in \mathbf{Q}, Q \perp\!\!\!\perp Y | \{X, \mathbf{Z}\}$*. The algorithm for checking the GBC is given in Fig. 19 and it performs* $\mathcal{O}(|\mathbf{V}| \cdot 2^{|Ne(X)|})$ *additional CI tests.*

Proof By Lemma 38, $\text{PossDesc}(X)$ contains the possible descendants of X. Condition (i) is therefore necessary since the descendants of X cannot satisfy the backdoor criterion. Thus, for the rest of the proof, we assume that $\mathbf{Z} \cap \text{PossDesc}(X) = \emptyset$.

We first prove the forward direction: If $\mathbf{Z}$ is a valid adjustment set then $\forall Q \in \mathbf{Q}, Q \perp\!\!\!\perp Y | \{X, \mathbf{Z}\}$. Since $\mathbf{Z}$ is a valid adjustment set, $\mathbf{Z}$ blocks all possibly backdoor paths in every DAG in the MEC. Therefore, we have $\forall Q \in \mathbf{Q}$, $Q \perp\!\!\!\perp Y | \{X, \mathbf{Z}\}$ because otherwise there will at least one DAG where the path $X \leftarrow Q$—$\ldots \to Y$ will be open for some $Q \in \mathbf{Q}$.

Next, we prove the backward direction: if $\forall Q \in \mathbf{Q}, Q \perp\!\!\!\perp Y | \{X, \mathbf{Z}\}$, then $\mathbf{Z}$ is a valid adjustment set. Firstly, for all $P \in \text{Pa}(X) \cap \mathbf{Z}$ (i.e., $P \notin \mathbf{Q}$), all backdoor paths of the form $X \leftarrow P$—$\ldots$—Y are blocked because $P \in \mathbf{Z}$. Since for all $Q \in \mathbf{Q}$, we have $Q \perp\!\!\!\perp Y | \{X, \mathbf{Z}\}$, all possible backdoor

paths $X \leftarrow Q — \ldots \rightarrow Y$ are blocked by $\mathbf{Z}$ and therefore it is a valid adjustment set in every DAG of the MEC.

Finally, we perform $\mathcal{O}(|\mathbf{V}| \cdot 2^{|\text{Ne}(X)|})$ additional CI tests in Lines 2,3 to find $\text{PossDesc}(X)$ (because for every node, we can find its MNS in $\mathcal{O}(2^{|\text{Ne}(X)|})$ tests). Next, in Line 6, since we only run tests for $N \in (\text{Pa}(X) \cup \text{Uo}(X)) \setminus \mathbf{Z}$, we perform $\mathcal{O}(|\text{Ne}(X)|)$ extra CI tests. ■

D.2. Finding the optimal adjustment set.

A given DAG can have multiple valid adjustment sets. Henckel et al. (2019)[Sec. 3.4] introduce a graphical criterion for linear models for determining the optimal adjustment set, i.e., the set with the lowest asymptotic variance. This criterion was later shown to hold non-parametrically (Rotnitzky and Smucler, 2019). We begin by introducing the existing results and then prove that we can find the optimal adjustment set using only local information and $\mathcal{O}(|\mathbf{V}|)$ additional CI tests (see (Fig. 20)).

Like the previous section, let the true DAG be $\mathcal{G}^*$ and let the corresponding CPDAG that represents the MEC of $\mathcal{G}^*$ be $\mathcal{C}^*$ (see Defn. 26).

Definition 40 (Possible causal nodes (Henckel et al., 2019, Sec. 3.4, Pg. 29)) *The causal nodes relative to (X, Y), denoted by posscn(X, Y), are all nodes on possibly causal paths from X to Y, excluding X.*

Definition 41 (Forbidden nodes (Henckel et al., 2019, Sec. 3.4, Pg. 29)) *The forbidden nodes relative to (X, Y), denoted by forb(X, Y), are defined as*

$$\textit{forb}(X, Y) = \textit{PossDesc}(\textit{posscn}(X, Y)) \cup \{X\}.$$

Definition 42 (Optimal adjustment set (Henckel et al., 2019, Defn. 3.12)) *The optimal adjustment set relative to X, Y is defined as*

$$\mathbf{O}(X, Y, \mathcal{C}^*) = \textit{Pa}(\textit{posscn}(X, Y)) \setminus \textit{forb}(X, Y).$$

We now show that it is possible to find the optimal adjustment set using only local information.

Lemma 43 *Given a CPDAG $\mathcal{C}$, the optimal adjustment set does not contain PossDesc(X).*

Proof As stated in Defn. 42, the optimal adjustment set is $\mathbf{O}(X, Y, \mathcal{C}^*) = \text{Pa}(\text{posscn}(X, Y)) \setminus \text{forb}(X, Y)$. Therefore, we only need to show that possible descendants of X that are *not* on a causal path from X to Y cannot be in $\mathbf{O}$ (otherwise they will be in $\text{forb}(X, Y)$). Consider a node $V \in \text{Desc}(X)$ not on a causal path from X to Y. This node cannot be a parent of any node in $\text{cn}(X, Y)$. This is because if that happens, then V must also belong to $\text{cn}(X, Y)$ which leads to a contradiction. ■

Corollary 1 *The optimal adjustment set satisfies the GBC.*

Proof By Lemma 43, the optimal adjustment set does not contain any possible descendants of X. Furthermore, by definition, it is a valid adjustment set and therefore blocks all backdoor paths from X to Y. Therefore, it satisfies the GBC. ■

Input: Treatment X, Outcome Y, $\text{Pa}(X)$, $\text{Ch}(X)$, unoriented $\text{Uo}(X)$.

1 $\mathbf{M} \leftarrow \text{Ch}(X) \cup \text{Uo}(X)$;
2 $\mathbf{I} \leftarrow \{V : \text{GetMNS}(V) = \text{invalid}\}$;
3 $\text{PossDesc}(X) \leftarrow \mathbf{M} \cup \mathbf{I} \cup \{V \in \mathbf{V} \setminus \mathbf{I} : \mathbf{M} \cap \text{GetMNS}(V) \neq \emptyset\}$;
4 $\mathbf{Z} \leftarrow (\mathbf{V} \setminus \text{PossDesc}(X))$;
5 $\mathbf{O} \leftarrow \text{HenckelPrune}(X, Y, \mathbf{Z})$ (Henckel et al., 2019, Alg. 1) (Fig. 21) ;
6 **for** $M \in (Pa(X) \cup Uo(X)) \setminus \mathbf{O}$ **do**
7 | **if** $M \not\perp\!\!\!\perp Y | \{X, \mathbf{O}\}$ **then return** *noValidAdj*;
8 **end**

Output: $\mathbf{O}$

Figure 20: Finding the optimal adjustment set.

1 **def** `HenckelPrune`(*Treatment* X, *Outcome* Y, *Subset* $\mathbf{Z}$) **:**
2 | $\mathbf{Z}' \leftarrow \mathbf{Z}$;
3 | **for** $Z \in \mathbf{Z}$ **do**
4 | | **if** $Y \perp\!\!\!\perp Z | \{X\} \cup (\mathbf{Z}' \setminus \{Z\})$ **then** $\mathbf{Z}' \leftarrow \mathbf{Z}' \setminus \{Z\}$;
5 | **end**
6 | **return** $\mathbf{Z}'$;

Figure 21: The *HenckelPrune* function.

Proposition 44 (Optimal adjustment set) *Consider the algorithm in Fig. 20. It performs* $\mathcal{O}(|\mathbf{V}| \cdot 2^{|Ne(X)|})$ *CI tests and (i) returns* `noValidAdj` *if there is no valid adjustment that applies to all DAGs in the MEC; (ii) else, returns the optimal adjustment set that is valid for all DAGs in the MEC (denoted by* $\mathbf{O}$*).*

Proof In Line 4, $\mathbf{Z} = \mathbf{V} \setminus \text{PossDesc}(X)$ represents the largest possible set that could satisfy the GBC, if any such set exists. Therefore, this set $\mathbf{Z}$ would be a superset of the optimal adjustment set, if it exists (by Cor. 1). In Line 5, we invoke the pruning procedure in Henckel et al. (2019, Algorithm 1) which outputs the optimal adjustment when starting from a superset (see Fig. 21). In Line 6, we verify that the pruned set $\mathbf{O}$ is a valid adjustment set (see Prop 39). Henckel et al. (2019, Theorem 3.13(i)) also prove that an optimal adjustment set exists iff there is some valid adjustment set. Thus, if Line 7 is reached, it means that there is no valid adjustment set that applies to every DAG in the MEC.

We perform $\mathcal{O}(|\mathbf{V}| \cdot 2^{|\text{Ne}(X)|})$ CI tests to find $\text{PossDesc}(X)$. The *HenckelPrune* function and Line 6 perform $\mathcal{O}(|\mathbf{V}|)$ additional CI tests. ■