The Cost of Fewer CI Tests in PC-Style Learning

Anonymous authors
Paper under double-blind review

Abstract

PC-style structure learning is often accelerated by reducing conditional-independence (CI) tests. This objective has a hidden cost: the graph information used to skip tests must itself be computed, and aggressive pruning can force larger conditioning sets. The Path-Driven Independence Testing (PIT) family exposes this trade-off. PIT restricts separator search by reachability in the partially learned skeleton; BPIT and Opt-BPIT add a blind-blocking rule that fixes variables on long paths. Under Markovness, faithfulness, and an exact CI oracle, all three algorithms recover the correct Markov-equivalence class and use no more CI tests than PC. Simulations show broad runtime gains for PIT with competitive accuracy, while further pruning traces the cost curve.

1 Introduction

Bayesian networks represent multivariate distributions by combining a directed acyclic graph (DAG) with local conditional distributions. The graph encodes conditional-independence relations through the Markov property, and the joint distribution factorizes into local conditional probabilities (Koller and Friedman, 2009). Learning the graph from data, known as structure learning, is therefore a central problem in probabilistic graphical models. Under additional causal assumptions, the same DAG can also be interpreted as a causal graph for interventional reasoning (Pearl, 2009); the development below is stated as Bayesian-network structure learning.

Constraint-based structure learning recovers graphical structure by translating conditional independencies into edge deletions and orientations. The PC algorithm starts from a complete undirected graph, removes an edge $X - Y$ once it finds a conditioning set $\mathcal{S}$ such that $X \perp\!\!\!\perp Y \mid \mathcal{S}$, and then applies orientation rules to obtain a completed partially directed acyclic graph (CPDAG), the standard representation of the Markov-equivalence class of DAGs with the same skeleton and unshielded colliders (Spirtes and Glymour, 1991; Spirtes et al., 2000). Under Markovness, faithfulness, and an exact CI oracle, PC is exact. Its computational bottleneck is the search for separating sets: for a current edge $X - Y$, PC may test many subsets of the current neighbors of X and Y, and the number of such tests can grow combinatorially with local degree (Kalisch and Buehlmann, 2007).

A natural way to accelerate PC is to use information in the partially learned skeleton. PC already does this by searching over current adjacencies rather than all variables. A stronger path-based observation is that, when testing $X - Y$, a neighbor of X that has no path to Y after removing X does not need to be included in a separating set for the current edge. A related path-restricted rule was briefly described as PC-star in Spirtes et al. (2000), and later work on consistent separating sets also restricts candidates using reachability in the partially learned graph (Li et al., 2019). These observations suggest a simple hypothesis: reducing the number of CI tests should make PC-style learning faster.

That hypothesis is incomplete. The graph information used to avoid tests must be computed and maintained, and a rule that avoids many small conditioning sets may replace them with fewer but larger conditioning sets. Large conditioning sets are a known source of finite-sample difficulty in CI testing, motivating reduced-PC and selected-conditioning-set approaches (Sondhi and Shojaie, 2019; Kocaoglu, 2023; Lee et al., 2025). Conversely, model-based CI tests can change the relative cost of large sets (Cai et al., 2022). Thus the total cost of a

skeleton-learning run is better viewed as

$$T_{\text{total}} \approx \sum_{(X,Y,\mathcal{S})\in\mathcal{Q}} T_{\text{CI}}(X,Y,\mathcal{S}) + T_{\text{graph}} + T_{\text{reuse}}, \tag{1}$$

where $\mathcal{Q}$ is the set of CI queries actually performed, $\mathcal{S}$ is the conditioning set in the query $X \perp\!\!\!\perp Y \mid \mathcal{S}$, T_{CI} is the cost of the statistical or oracle CI decision, T_{graph} is the cost of reachability and path-derived computations in the partially learned skeleton, and T_{reuse} captures caching or reuse of previous graph and CI computations. The empirical question is where the operating point lies on the curve between test-count reduction and the incurred graph-computation and conditioning costs.

This paper studies that curve through the PIT family. PIT implements path-restricted pruning through reachability in the current graph. BPIT and Opt-BPIT add a blind-blocking step: certain long current paths can be blocked by fixing two consecutive interior nodes in the conditioning set, provided these nodes are outside conservative regions that may activate short-path colliders. All three algorithms remain exact under the same oracle assumptions as PC, and under a fixed oracle and test ordering they perform no more CI tests than PC. Their empirical behavior identifies the practical operating point of path information: cheap reachability pruning pays off broadly, while further CI-test reductions must be judged against their conditioning and graph-operation costs.

The paper develops four results. First, PIT is formulated as a reachability implementation of path-restricted PC-style separator search. Second, a blind-blocking separator construction for long paths yields BPIT and Opt-BPIT. Third, the PIT-family algorithms are proved exact under Markovness, faithfulness, and an exact CI oracle, with oracle CI-test counts bounded by those of PC under common ordering conventions. Fourth, experiments on 17 benchmark Bayesian networks show the main trade-off: PIT gives the broadest runtime improvement with competitive structural error, while more aggressive pruning can reduce the number of tests but spend the savings on graph computations and larger conditioning sets.

1.1 Related work

PC and its variants are the standard reference point for constraint-based structure learning. High-dimensional analyses show that PC can be effective in sparse problems because the search is governed by local degrees rather than the total number of variables (Kalisch and Buehlmann, 2007). PC-stable removes an important source of order dependence in the skeleton stage by freezing adjacency sets within each conditioning depth (Colombo and Maathuis, 2014), while conservative variants address ambiguity in collider orientation (Ramsey et al., 2006). Extendable PC uses the information in the current graph to update a learned Bayesian-network structure when a new variable is added, rather than relearning the full structure from scratch (Kalantari et al., 2025). These methods preserve the same basic separator-search paradigm.

Path-restricted separator search is the closest predecessor of the base algorithm. Spirtes, Glymour, and Scheines briefly describe PC-star as a version of PC that restricts candidate conditioning variables to endpoint-adjacent variables lying on undirected paths between the tested endpoints (Spirtes et al., 2000). Li et al. develop consistent separating sets, constraining separators to remain compatible with the graph learned in earlier iterations and with available partial orientations (Li et al., 2019). The present work treats this path restriction as the base point and studies both its reachability implementation and the cost of pushing the idea further.

A separate line of work restricts or reorganizes conditioning sets. The reduced PC algorithm limits the search using local graph information and is motivated partly by finite-sample unreliability of large conditioning sets (Sondhi and Shojaie, 2019). The k-PC framework and related conditioning-collection approaches study what can be learned when only small or selected conditioning sets are available (Kocaoglu, 2023; Lee et al., 2025). Complementary work asks how much structure can be learned with fewer CI tests, or whether model-based CI tests make larger sets more affordable (Shiragur et al., 2024; Cai et al., 2022). These results support the perspective that CI-test count alone is not a sufficient cost measure.

Many other structure-learning algorithms use different objectives or search spaces, including score-based methods such as GES and ARGES (Chickering, 2002; Nandy et al., 2018), permutation-based methods such

as GSP (Solus et al., 2021), hybrid methods such as MMHC (Tsamardinos et al., 2006), and recent scalable alternatives such as BOSS and XGES (Andrews et al., 2023; Nazaret and Blei, 2024). Markov-boundary and local-structure methods, including Grow-Shrink, MMPC, H2PC, and MARVEL, reduce global search by first estimating local neighborhoods or boundaries (Margaritis and Thrun, 1999; Tsamardinos et al., 2003; Gasse et al., 2014; Mokhtarian et al., 2021). These methods are useful reference points, but the main comparison in this paper isolates the PC-family trade-off among PC, PIT, BPIT, and Opt-BPIT.

2 Background

Given random variables $\mathcal{X} = \{X_1, \ldots, X_N\}$, their joint probability distribution $P(\mathcal{X})$ factorizes according to the chain rule into $\Pi_{i=1}^{N} P(X_i \mid X_1, \ldots, X_{i-1})$. Each conditional probability term may be simplified if a corresponding conditional probability independence holds. For example, $P(X_1 \mid X_2, X_3)$ becomes $P(X_1 \mid X_2)$ should $X_1 \perp X_3 \mid X_2$ hold. The problem is to find (one of) the "simplest" factorization(s) of the joint probability distribution.

A directed acyclic graph (DAG) $\mathcal{G}$ can be attributed to each factorization, where nodes are the random variables $\mathcal{X}$ and for each conditional term $P(X_i \mid X_{i_1}, \ldots, X_{i_k}), k \geq 1$, there is an incoming link from each of the conditioned variables $X_{i_1}, \ldots, X_{i_k}$ to X_i. Thus, $X_{i_1}, \ldots, X_{i_k}$ form the parents of X_i, denoted by the set $\mathbf{Pa}_{X_i}$. A sequence of nodes $\mathcal{T} = (X_1, \ldots, X_n)$, $n \geq 1$, is a *trail (path)* between X_1 and X_n if X_i and X_{i+1} are adjacent in $\mathcal{G}$ for $i = 1, \ldots, n-1$. Its *length* is $n-1$, and its *interior* is $\text{int}(\mathcal{T}) = \{X_2, \ldots, X_{n-1}\}$. It is a *directed path* from X_1 to X_n if $X_i \to X_{i+1}$ for all i. The *descendants* of node X are all nodes Y for which there exists a directed path from X to Y. In particular, X is a descendant of itself. The *reachability set* of node Y, denoted $\mathcal{R}(Y)$, is the set of nodes, including Y, that are connected to Y by some trail. A *collider* is a length-two trail (X, Z, Y) where nodes X and Y are linked to Z, i.e., $X \to Z \leftarrow Y$; Z is the *collider (center) node*. The collider is an *immorality* if there is no edge between X and Y.

The DAG implies certain conditional independencies on the variables. For example, each variable is independent of its non-descendants conditioned on its parents (Koller and Friedman, 2009). More generally, the notion of *d-separation* is defined to capture all conditional independencies imposed by the DAG (Theorem B.1). It can be shown that every d-separation in the DAG that corresponds to the factorization is also satisfied by the distribution P. Namely, $\mathcal{I}(\mathcal{G}) \subseteq \mathcal{I}(P)$, where $\mathcal{I}(\mathcal{G})$ is the set d-separations imposed by $\mathcal{G}$ and $\mathcal{I}(P)$ is the set of conditional independencies satisfied by distribution P. Thus, any DAG $\mathcal{G}$ corresponding to the desired factorization must satisfy *Markovness*, i.e., $\mathcal{I}(\mathcal{G}) \subseteq \mathcal{I}(P)$. We further assume *faithfulness*, $\mathcal{I}(P) \subseteq \mathcal{I}(\mathcal{G})$. A DAG satisfying both Markovness and faithfulness is a *P-map* for P, also referred to as the *true DAG*.

Assumption 2.1. There exists a DAG $\mathcal{G}$ that is a P-map for distribution P, i.e., $\mathcal{I}(\mathcal{G}) = \mathcal{I}(P)$.

The factorization problem then is to find a P-map for P – a task known as *structure learning*. There is often more than one P-map for a distribution P, e.g., both DAGs $X \to Y$ and $Y \to X$ are a P-map for the distribution $\mathcal{I}(P) = \emptyset$. The set of all P-maps for a given distribution P share the same skeleton (the undirected graph obtained by removing the orientations of the edges) and immoralities (Koller and Friedman, 2009). The corresponding Markov-equivalence class is represented by a completed partially directed acyclic graph (CPDAG), a partially directed graph whose undirected edges can be oriented without creating a directed cycle or a new immorality to obtain a DAG in the class.

The PC algorithm obtains a CPDAG by first forming a completely-connected undirected graph over variables $\mathcal{X}$, then iteratively removing the edge between every two variables that are marginally independent, then those that are independent conditioned on a single other variable, then those that are independent conditioned on two other variables, and so on. Finally, the directions of the remaining edges are determined based on the orientation rules described in (Spirtes et al., 2000) (Algorithm 3). Consequently, the number of conditional independent (CI) tests performed by PC is at most $2\binom{N}{2}\sum_{i=0}^{N-2}\binom{N-2}{i}$ and is thus of order $\mathcal{O}(2^N)$.

The idea behind the algorithm is that two adjacent nodes in the P-map never become independent, regardless of what they are conditioned on, because they never become d-separated.

Lemma 2.2. *(Based on (Pearl, 2009)) Consider random variables $\mathcal{X}$ with joint distribution P that admits a P-map $\mathcal{G}$. Vertices X and Y are not adjacent in $\mathcal{G}$ if and only if $X \perp Y \mid \mathcal{U}$ for some $\mathcal{U} \subseteq \mathcal{X}$.*

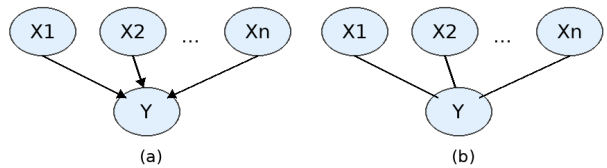

Figure 1: (a) True DAG (b) The undirected graph after testing marginal independencies by the PC algorithm.

PC is based on a sharpened version of this lemma, restricting $\mathcal{U}$ to the parents of X and Y.

Lemma 2.3. *[Lemma 3.2 in (Koller and Friedman, 2009)] Consider random variables $\mathcal{X}$ with joint distribution P that admits a P-map $\mathcal{G}$. Vertices X and Y are not adjacent in $\mathcal{G}$ if and only if $X \perp Y \mid \mathbf{Pa}_X$ or $X \perp Y \mid \mathbf{Pa}_Y$.*

Thus, in order to remove the edge between X and Y, only CI tests conditioned on the parents of X and conditioned on the parents of Y are needed. As the parents are unknown initially in the undirected graph, PC searchers over all subsets of the neighbors of X, denoted $\mathcal{N}_X$, and the neighbors of Y, denoted $\mathcal{N}_Y$, to ensure that their parents will be conditioned on during the search.

Example 1. Consider random variables $X_1, \ldots, X_n$ and Y and assume that their joint distribution admits a P-map in the form of Figure 1(a). Starting from the completely connected graph, PC algorithm performs marginal independence tests on pairs of variables. If first the links between the nodes $X_1, \ldots, X_n$ are checked, then the undirected structure in Figure 1(b) is obtained before checking the links between Y and the other nodes. This is done by marginal independence tests between X_i and X_j for all i and j, that is, $n(n-1)/2$ tests. Next, the dependence between Y and the X_i's is checked. Since these links are not spurious, i.e., exist in the P-map, PC performs all possible CI tests to ensure that they cannot be removed. For example, for the link $X_1 \to Y$, PC checks whether $X_1 \perp Y \mid \mathcal{U}$ holds for all combinations $\mathcal{U} \subseteq \{X_2, \ldots, X_n\}$ from zero to $n-1$ variables, resulting in $\mathcal{O}(2^n)$ tests.

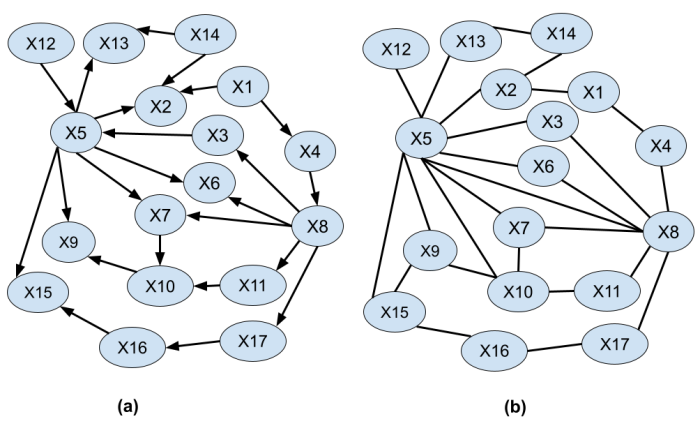

Figure 2: (a) True DAG (b) Undirected graph at some point in the PC algorithm

Example 2. Consider variables $\mathcal{X} = \{X_1, \ldots, X_{17}\}$ whose joint distribution admits the P-map $\mathcal{G}$ in Figure 2(a). Suppose that we start the PC algorithm from the undirected graph in Figure 2(b) and in particular, we are interested in determining whether the link between X_5 and X_8 also exists in the P-map. To this end, PC checks the CI's $X_5 \perp X_8 \mid \mathcal{U}$ for all subsets of the current endpoint-neighbor sets

$$\mathcal{N}_{X_5} = \{X_{12}, X_{13}, X_2, X_3, X_6, X_7, X_{10}, X_9, X_{15}\},$$
$$\mathcal{N}_{X_8} = \{X_4, X_3, X_6, X_7, X_{11}, X_{17}\}.$$

3 The Path-Driven Independence Testing (PIT) Algorithm

3.1 The idea

Is it possible to reduce the search over all subsets $\mathcal{U}$ of the neighbors $\mathcal{N}_X$ and $\mathcal{N}_Y$ in the PC algorithm? Lemma 2.3 is conservative in the sense that among the parents of X (resp. Y), some may not be even indirectly connected to Y (resp. X). When checking the link $X_1 \to Y$ in Example 1, node Y has many parents, but none are connected to X_1. Therefore, X_1 and Y are not d-separated in the true DAG conditioned on any combination of the parents of Y. Similarly, X_1 has no parent. Hence, an efficient algorithm should perform only n marginal independent tests between Y and each of $X_1, \ldots, X_n$ once the undirected structure in Figure 1(b) is reached. This observation suggests that the parents $\mathbf{Pa}_X$ (resp. $\mathbf{Pa}_Y$) in Lemma 2.3 should be limited to those that are connected to the other node Y (resp. X) by some path. We define the notion of *covering paths* accordingly.

Definition 3.1 (Covering path)**.** A *covering path/trail* for nodes X and Y is a trail between X and Y of length at least two. A covering path is *short* if the length is exactly two and *long* otherwise. The set of all covering, short-covering, and long-covering paths between X and Y are denoted by $\mathcal{T}_{XY}$, $\mathcal{T}^s_{XY}$, and $\mathcal{T}^l_{XY}$.

Rather than the whole parents $\mathbf{Pa}_X$, one may condition only on those that belong to the interior of the covering paths between X and Y, which we refer to as the *essential parents of X (with respect to Y)*, denoted by $\mathbf{Pa}_{X,Y} = \mathbf{Pa}_X \cap \{\text{int}(\mathcal{T}) : \mathcal{T} \in \mathcal{T}_{XY}\}$. This is the same as those parents of X that can reach node Y (excluding Y itself), i.e., $\mathbf{Pa}_{X,Y} = \mathbf{Pa}_X \cap \mathcal{R}_Y \setminus \{Y\}$. In Figure 1-a, $\mathbf{Pa}_{X_i,Y} = \emptyset$ for all i, and in Figure 2-a, $\mathbf{Pa}_{X_5X_8} = \{X_3\}$.

A second conservative aspect of PC is in the way Lemma 2.3 is used. As discussed earlier, since the parents are not a priori known in the undirected graph, PC may search through eventually all possible combinations of $\mathcal{N}_X$. However, to reduce the number of possible combinations, is it possible to always include some nodes in the search, namely, to always include a neighbor, say $Z \in \mathcal{N}_X$, in the CI tests as $X \perp Y | Z, \mathcal{U}$? Generally, no, because node Z may be the center node of a collider on a trail connecting X and Y. Then observing Z may activate the trail and prevent X and Y from becoming independent. An example is in Figure 2(a) in Example 2, where X_2 is not a parent of X_5 and observing it makes the trail $(X_5, X_2, X_1, X_4, X_8)$ active. Nevertheless, there are situations where observing some nodes does not make the connecting trails active. For example, if instead of just X_2, we observe all of the interior nodes of the trail, i.e., X_2, X_1, and X_4, then the trail becomes inactive. So we can include these nodes in the conditioned nodes in all of the CI tests. That is, rather than checking $X_5 \perp X_8 \mid \mathcal{U}$ for all $\mathcal{U} \subseteq \mathcal{N}_X$ and $\mathcal{U} \subseteq \mathcal{N}_Y$, we can check $X_5 \perp X_8 \mid X_2, X_1, X_4, \mathcal{U}'$ for all $\mathcal{U}' \subseteq \mathcal{N}_X \setminus \{X_2\}$ and $\mathcal{U}' \subseteq \mathcal{N}_Y \setminus \{X_4\}$, reducing the search space. The following result illustrates the nodes that can be always observed.

Lemma 3.2. *Consider a DAG $\mathcal{G}$ and let $\mathcal{T} \in \mathcal{T}^l_{XY}$ be a long-covering trail between nodes X and Y in $\mathcal{G}$. Then the followings hold:* (i) *there is an interior node of $\mathcal{T}$ that if observed, makes the trail inactive;* (ii) *if two adjacent interior nodes of $\mathcal{T}$ are observed, then the trail is inactive, regardless of whether some of the remaining nodes are observed;* (iii) *if all interior nodes of $\mathcal{T}$ are observed, the trail becomes inactive.*

Lemma 3.2 suggests that either all nodes or any pair of adjacent nodes of the interior of a long-covering path should be always observed in the CI tests. However, observing the interior of the long-covering path may activate some other trails, such as a collider between X and Y, which should be avoided. More specifically, an interior node of a long-covering path may be a collider node or a descendant in some short-covering path. For example, node X_{10} in Figure 2(a) belongs to the long-covering path $(X_5, X_9, X_{10}, X_{11}, X_8)$ but is a descendant of X_7 that forms a collider with X_5 and X_8. Observing X_{10} activates the collider and renders X_5 and X_8 dependent.

Definition 3.3 (Potentially trigger node)**.** Consider a graph $\mathcal{G}$ with nodes $\mathcal{X}$ and let $X, Y \in \mathcal{X}$. A *potentially trigger node* of X and Y is a node that is connected to the interior node of some short-covering path of X and Y via a path that does not include X and Y. A path is *potentially trigger* if it contains a potentially trigger node. The set of all potentially trigger nodes of X and Y is denoted by $\mathcal{W}_{XY}$.

So X_{10} is a potentially trigger node of the long-covering path $(X_5, X_9, X_{10}, X_{11}, X_8)$. Thus, in Figure 2(b) when performing the CI tests over the neighbors $\mathcal{N}_X$, we cannot set X_{10}, and consequently the interior of

the covering path $(X_5, X_9, X_{10}, X_{11}, X_8)$, to be always observed. The definition is not limited to the nodes connected to colliders and covers all nodes connected to any short-covering trail. The connected nodes may or may not activate a collider, and hence, the term "potentially" trigger. The reason for this uncertainty is that in the undirected graph used in the structure learning algorithm, the edge directions and hence colliders are unknown. The remaining covering paths do not include potentially trigger nodes and are defined in the following. A *connected component* of a graph is a connected subgraph (every pair of nodes of which are connected by some trail) that itself is not a subgraph of any other connected subgraph. Given graph $\mathcal{G}$ with node set $\mathcal{X}$, the *graph* $\mathcal{G}[\mathcal{V}]$ *induced by the subset* $\mathcal{V} \subseteq \mathcal{X}$ is a graph with node set $\mathcal{V}$ where two nodes are linked if and only if they are linked in the original graph $\mathcal{G}$.

Definition 3.4 (Blindly blockable)**.** Consider graph $\mathcal{G}$ with nodes $\mathcal{X}$ and let $X, Y \in \mathcal{X}$. A *(blindly) blockable path between* X *and* Y is a long covering path without potentially trigger nodes, that is, a path $\mathcal{T} \in \mathcal{T}^l_{XY}$ such that $\mathcal{T} \cap \mathcal{W}_{XY} = \emptyset$. The set of *(blindly) blockable paths between* X *and* Y is denoted by $\mathcal{T}^b_{XY}$. The *maximal blockable set of* X *and* Y is defined by

$$\mathcal{C}_{XY} \triangleq \mathcal{X} \setminus (\mathcal{W}_{XY} \cup \{X, Y\}).$$

The *maximal blockable neighborhood of* X *relative to* Y is defined by

$$\bar{\mathcal{B}}_{XY} = \{Z_1, Z_2 \in \mathcal{T} : Z_1 \in \mathcal{N}_X, Z_2 \in \mathcal{N}_{Z_1}, \mathcal{T} \in \mathcal{T}^b_{XY}\}.$$

The *(minimal) blockable neighborhood of* X *relative to* Y, denoted $\mathcal{B}_{XY}$, is the smallest subset of $\bar{\mathcal{B}}_{XY}$ such that at least two consecutive nodes of every blockable path is included in $\mathcal{B}_{XY}$. A *connected blockable neighborhood of* X *relative to* Y is a connected component of the graph $\mathcal{G}[\mathcal{B}_{XY}]$. An *off-path blockable set of* X *and* Y is a subset of $\mathcal{C}_{XY} \setminus \mathcal{T}^b_{XY}$.

In Figure 2-a, the blockable paths between X_5 and X_8 are $(X_5, X_{13}, X_{14}, X_2, X_1, X_4, X_8)$, $(X_5, X_2, X_1, X_4, X_8)$, and $(X_5, X_{15}, X_{16}, X_{17}, X_8)$. The maximal blockable set is the complement node set of X and Y and the potentially trigger nodes in between, i.e.,

$$\mathcal{C}_{X_5X_8} = \{X_{12}, X_{13}, X_{14}, X_2, X_1, X_4, X_{15}, X_{16}, X_{17}\}.$$

In view of Lemma 3.2, this is the maximum set of nodes that can be "blindly" conditioned on without rendering X and Y independent. The maximal blockable neighborhood is the union of the neighbors and the neighbors of the neighbors of X on all non-potentially trigger covering paths between X and Y:

$$\bar{\mathcal{B}}_{X_5X_8} = \{X_{13}, X_{14}, X_2, X_1, X_{15}, X_{16}\}.$$

The idea is that according to Lemma 3.2, by observing two adjacent nodes on the blockable paths, they become inactive regardless of whether one of the two nodes is a collider node (and hence the term "blindly blockable"). The (minimal) blockable neighborhood is the same as the maximal blockable neighborhood, where some "redundant" paths that start from X and end at a neighbor of X, say Z_1, are ignored, because they do not need to be blocked if Z_1 and its neighbor Z_2 are blocked. The path $(X_5, X_{13}, X_{14}, X_2)$ is redundant, yielding

$$\mathcal{B}_{X_5X_8} = \{X_2, X_1, X_{15}, X_{16}\}.$$

The connected blockable neighborhoods are

$$\mathcal{B}^1_{X_5X_8} = \{X_2, X_1\}, \qquad \mathcal{B}^2_{X_5X_8} = \{X_{15}, X_{16}\}.$$

An off-path blockable set is a collection of blindly blockable nodes that are not on any blindly blockable path between X and Y, i.e., $\mathcal{O}_{X_5X_8} = \{X_{12}\}$.

Now, based on our discussion on the notion of covering paths (Definition 3.1), $\mathbf{Pa}_X$ in Lemma 2.3 can be restricted to the essential parents $\mathbf{Pa}_{X,Y}$. On the other hand, some essential parents belong to a blindly blockable path and may be excluded from the search space provided that the blockable neighborhood $\mathcal{B}_{XY}$ is observed. Indeed, one can show that $X \perp Y \mid \mathbf{Pa}_{X,Y} \cup \mathcal{B}_{XY}$, suggesting that the structure learning algorithm can search through $\mathcal{N}_X \setminus \mathcal{B}_{XY}$ to find $\mathcal{U} = \mathbf{Pa}_{X,Y} \setminus \mathcal{B}_{XY}$ and then the CI test $X \perp Y \mid \mathcal{U} \cup \mathcal{B}_{XY}$ can be performed.

However, in a structure learning algorithm that starts with a complete graph and iteratively deletes edges by checking CI tests, the set $\mathcal{B}_{XY}$ is not initially identified, because neither is $\mathcal{W}_{XY}$. Given graph $\mathcal{G}$ that is obtained at a specified iteration, we use superscript $\mathcal{G}$ for the sets that are defined based on $\mathcal{G}$, e.g., $\mathcal{W}^{\mathcal{G}}_{XY}$ and $\mathcal{B}^{\mathcal{G}}_{XY}$. In the beginning, the set of potentially trigger nodes $\mathcal{W}^{\mathcal{G}}_{XY}$ consists of all nodes except for X and Y and no node path is blindly blockable, resulting in $\mathcal{B}^{\mathcal{G}}_{XY} = \emptyset$. Once some edges are removed over the iterations, $\mathcal{W}^{\mathcal{G}}_{XY}$ shrinks and $\mathcal{B}^{\mathcal{G}}_{XY}$ equals the union of some connected blockable neighborhoods of X relative to Y, denoted $\mathcal{B}^{C}_{XY}$, and some superfluous blockable neighborhoods that are actually an off-path blockable set, i.e., $\mathcal{O}_{XY}$, due to some superfluous edges to be removed in future iterations. We, therefore, modify the parents $\mathbf{Pa}_X$ in Lemma 2.3 to the following.

Definition 3.5 (Separator)**.** Consider DAG $\mathcal{G}$ with nodes $\mathcal{X}$ and let $X, Y \in \mathcal{X}$. A *(path-driven) separator for X relative to Y* is defined by

$$\mathcal{M}^*_{XY} \triangleq \mathbf{Pa}_{X,Y} \cup \mathcal{B}^{C}_{XY} \cup \mathcal{O}_{XY}$$

where $\mathcal{B}^{C}_{XY}$ is the union of an arbitrary (possibly empty) collection of connected blockable neighborhoods of X relative to Y, and $\mathcal{O}_{XY}$ is an off-path blockable set of X and Y.

In Figure 1(a), for all i, $\mathcal{M}^*_{X_iY}$ can be $\emptyset$ or more generally, any subset of $\{X_1, \ldots, X_n\} \setminus \{X_i\}$ which is an off-path blockable set. In Figure 2(a), the possible values for $\mathcal{M}^*_{X_5X_8}$ are

$$\{X_3\}, \quad \{X_3, X_2, X_1\}, \quad \{X_3, X_{12}\},$$
$$\{X_3, X_2, X_1, X_{12}\}, \quad \{X_3, X_{15}, X_{16}\}, \quad \{X_3, X_{15}, X_{16}, X_{12}\}.$$

The separator is asymmetric with respect to X and Y, i.e., $\mathcal{M}^*_{XY} \neq \mathcal{M}^*_{YX}$. The following counterpart to Lemma 2.3 implies that it is necessary and sufficient for two non-adjacent variables to be independent, conditioned on their separators. The proofs are provided in the Appendix.

Lemma 3.6. *Consider random variables $\mathcal{X}$ with joint distribution P that admits a P-map $\mathcal{G}$. Nodes X and Y are not adjacent in $\mathcal{G}$ if and only if $X \perp Y \mid \mathcal{M}^*_{XY}$ for every separator $\mathcal{M}^*_{XY}$ or $X \perp Y \mid \mathcal{M}^*_{YX}$ for every separator $\mathcal{M}^*_{YX}$.*

3.2 The algorithms

The PIT rule first applies the path-restricted reachability idea. When testing an edge $X - Y$, it searches among neighbors of X that remain connected to Y in the current graph after X is removed, and analogously when the endpoint order is reversed. This is the reachability form of the PC-star/path-restricted rule, implemented without explicit path enumeration.

We develop three sound and complete structure-learning algorithms. The idea is to find the essential parents $\mathbf{Pa}_{X,Y}$ (or $\mathbf{Pa}_{Y,X}$) for every non-adjacent pair of nodes X and Y. Edge directions are unknown in the undirected graph, preventing us to know the parents of X and Y. Thus, similar to PC, where $\mathbf{Pa}_X$ was extended to $\mathcal{N}^{\mathcal{G}}_X$, we extend $\mathbf{Pa}_{X,Y}$ to $\mathcal{N}^{\mathcal{G}}_{X,Y}$, i.e., neighbors of X that have a path to Y, and search through its subsets. That is to restrict the candidate conditioning set $\mathcal{U}$ in the PC algorithm to only the neighbors of X that have a path to Y when verifying the link between X and Y, such as that has been used in (Li et al., 2019). The result is the Path-Driven Independence Testing (PIT) Algorithm(PIT)–Algorithm 1.

BPIT adds the blind-blocking construction. Since the true blockable set $\mathcal{B}_{XY}$ is unknown during learning, the algorithm computes conservative current-graph quantities $\mathcal{W}^{\mathcal{G}}_{XY}$ and $\mathcal{B}^{\mathcal{G}}_{XY}$ from the undirected intermediate graph. It then fixes $\mathcal{B}^{\mathcal{G}}_{XY}$ in the conditioning set and searches only over the remaining triggering-side neighbors.

Next is Algorithm 2, which is based on Lemma 3.6. Those parents that are included in the blockable paths can be always observed and are not required to be searched through. So instead of $\mathcal{N}^{\mathcal{G}}_{X,Y}$, we only need to search through $\mathcal{N}^{\mathcal{G}}_{X,Y} \setminus \mathcal{B}_{XY}$. As discussed earlier, $\mathcal{B}_{XY}$ is unknown from the start though. Instead, at every iteration m, $\mathcal{B}^{\mathcal{G}}_{XY}$ is available, that is the blockable set in the undirected graph $\mathcal{G}$ obtained before iteration m. So we search through $\mathcal{N}^{\mathcal{G}}_{X,Y} \setminus \mathcal{B}^{\mathcal{G}}_{XY}$ which can be shown to be the same as $\mathcal{N}^{\mathcal{G}}_X \cap \mathcal{W}^{\mathcal{G}}_{XY}$. The sets $\mathcal{W}^{\mathcal{G}}_{XY}$ and $\mathcal{B}^{\mathcal{G}}_{XY}$ are obtained from Algorithm 4. The adjacency set of X in graph $\mathcal{G}$ is $\mathrm{Adj}(\mathcal{G}, X)$, and $\mathcal{R}^{\mathcal{G}}_Y$ is the reachable set of node Y in graph $\mathcal{G}$.

Algorithm 1: The Path-Driven Independence Testing (PIT) Algorithm

Input: A set of variables $\mathcal{X}$ and their joint probability distribution P
Output: A partially directed acyclic graph

Form the complete undirected graph $\mathcal{G}$ over nodes $\mathcal{X}$;
$\text{Sepset}(X, Y) = \emptyset$ for all $X, Y \in \mathcal{X}$;
$m = 0$
while *maximum node degree in* $\mathcal{G}$ *is greater than* m **do**
  for $X \in \mathcal{X}$
    for $Y \in \text{Adj}(\mathcal{G}, X)$
      for $\mathcal{U} \subseteq \mathcal{R}_Y^{\mathcal{G}\setminus\{X\}} \cap (\text{Adj}(\mathcal{G}, X) \setminus \{Y\})$ *and* $|\mathcal{U}| = m$
        if $X \perp Y \mid \mathcal{U}$
          Remove the edge $X - Y$ from $\mathcal{G}$;
          $\text{Sepset}(X, Y) \leftarrow \mathcal{U}$;
  $m = m + 1$;
Orient the edges using the orientation rules in (Spirtes et al., 2000).

Algorithm 2: The Blocked-Path Driven Independence Testing (BPIT) Algorithm

Input: A set of variables $\mathcal{X}$ and their joint probability distribution P
Output: A partially directed acyclic graph

Form the complete undirected graph $\mathcal{G}$ over nodes $\mathcal{X}$;
$\text{Sepset}(X, Y) = \emptyset$ for all $X, Y \in \mathcal{X}$;
$m = 0$
while *maximum node degree in* $\mathcal{G}$ *is greater than* m **do**
  for $X \in \mathcal{X}$
    for $Y \in \text{Adj}(\mathcal{G}, X)$
      for $\mathcal{U} \subseteq \mathcal{W}_{XY}^{\mathcal{G}} \cap \text{Adj}(\mathcal{G}, X)$ *and* $|\mathcal{U}| \in [m - |\mathcal{B}_{XY}^{\mathcal{G}} \cap \text{Adj}(\mathcal{G}, X)|, m]$
        if $X \perp Y \mid \mathcal{B}_{XY}^{\mathcal{G}} \cup \mathcal{U}$
          Remove the edge $X - Y$ from $\mathcal{G}$;
          $\text{Sepset}(X, Y) \leftarrow (\mathcal{B}_{XY}^{\mathcal{G}} \cap \text{Adj}(\mathcal{G}, X)) \cup \mathcal{U}$;
  $m = m + 1$;
Orient the edges using the orientation rules in (Spirtes et al., 2000).

Although the intuition of Algorithm 2 is based on covering paths, which are expensive to find–generally of order $\mathcal{O}(2^N)$–the algorithm does not explicitly find covering paths. When investigating the existence of a certain link $X - Y$, instead of finding every long-covering path between X and Y to see whether it is blockable by checking if any of its interior nodes is connected by some trail to the center node of a short covering path between X and Y, the following can be done (Algorithm 4): First, find all interior nodes Z of the short-covering paths. This is done by simply taking the intersection $\mathcal{N}_X^{\mathcal{G}} \cap \mathcal{N}_Y^{\mathcal{G}}$. Next, iteratively find and remove the reachable set $\mathcal{R}_Z$ of each such node $Z \in \mathcal{N}_X^{\mathcal{G}} \cap \mathcal{N}_Y^{\mathcal{G}}$ from the graph $\mathcal{G} \setminus \{X, Y\}$. Finding the reachable set of a node in a graph with N nodes can be done by for example using the adjacency matrix in $\mathcal{O}(N^2)$ (Gersting, 2006). The removed nodes form $\mathcal{W}_{XY}^{\mathcal{G}}$ and what remains is the maximal blockable set $\mathcal{C}_{XY}^{\mathcal{G}}$. For the blockable neighborhoods, we first find the reachable nodes of Y when the potentially trigger and the neighbors of X are removed from the graph, resulting in $\mathcal{R}'_Y$. Then for each neighbor of X that is in the blockable set $\mathcal{C}_{XY}^{\mathcal{G}}$, if it shares a neighbor in $\mathcal{R}'_Y$, then it has a path to Y that does not pass another neighbor of X. Hence, they can both be blocked.

Opt-BPIT uses the same separator family as BPIT but avoids re-enumerating candidate searched sets when the triggering set has not changed across conditioning depths.

A more efficient version of Algorithm 2 is Algorithm 5, where fewer number of candidate $\mathcal{U}$ sets are searched. The idea is that if the set $\mathcal{W}_{XY}^{\mathcal{G}}$ has not changed at a certain iteration m compared to the last iteration $m-1$,

then the set $\mathcal{B}^{\mathcal{G}}_{XY}$ has not changed (except that it could have become smaller to exclude some of the nodes in $\mathcal{O}^{\mathcal{G}}_{XY}$). Then testing $\mathcal{U}$'s with a cardinality smaller than m would be redundant as they were already checked in the previous iterations. In general, at iteration m, we need to search for candidate $\mathcal{U}$'s of size less than m only if new variables are included in the set $\mathcal{B}^{\mathcal{G}}_{XY}$ compared to the last iteration $m-1$. These new variables come from $\mathcal{W}^{\mathcal{G}}_{XY}$; hence, when searching through the candidates $\mathcal{U}$, we should consider the case where these new variables are already included in $\mathcal{U}$, and hence, start the search for $\mathcal{U}$ sets with size m minus the number of the new variables, resulting in Algorithm 5.

3.3 Examples

The following example illustrates how the new algorithm reduces the number of required CI tests to obtain the true DAG compared to PC. The second example illustrates the usage of the blockable paths.

Example 1 (revisited)**.** By using Algorithm 2 to obtain the true DAG in Figure 1(a), first, a complete undirected graph is constructed, and then similar to PC, the graph in Figure 2(b) is obtained after performing the iterations for $m = 0$. However, then for checking each of the links $Y - X_i$, $i \in \{1, \dots, n\}$, No covering path exists for each pair of variables Y and X_i, i.e.,

$$\mathcal{T}^{s}_{X_iY} = \mathcal{T}^{l}_{X_iY} = \mathcal{T}^{b}_{X_iY} = \mathcal{B}_{X_iY} = \mathcal{W}_{X_iY} = \emptyset.$$

Consequently, the edge deletion part terminates. The resulting number of CI tests is $\mathcal{O}(n^3)$ while for the PC algorithm, it was $\mathcal{O}(2^{n-1})$.

Example 2 (revisited)**.** Regarding $X_5 - X_8$ in Figure 2(b), we have

$$\mathcal{W}_{X_5X_8} = \{X_3, X_6, X_7, X_9, X_{10}, X_{11}, X_{15}, X_{16}, X_{17}\},$$
$$\mathcal{B}_{X_5X_8} = \{X_2, X_1\}.$$

Thus,

$$\mathcal{W}_{X_5X_8} \cap \mathrm{Adj}(\mathcal{G}, X_5) = \{X_3, X_6, X_7, X_9, X_{15}\},$$
$$\mathcal{B}_{X_5X_8} \cap \mathrm{Adj}(\mathcal{G}, X_5) = \{X_2\}.$$

So by checking all subsets $\mathcal{U} \in \{X_3, X_6, X_7, X_9, X_{15}\}$ where the size of $\mathcal{U}$ belongs to the interval $[m-1, m]$, a separator is guaranteed to be found for $m = 1$ or 2, e.g., $\mathcal{M}^{*}_{X_5X_8} = \{X_3\}$ or $\{X_3, X_2\}$.

3.4 Soundness, completeness, and Complexity

Theorem 3.7. *Consider random variables $\mathcal{X}$ with joint distribution P that admits a P-map. The output graph $\hat{\mathcal{G}}$ of PIT, BPIT, and Opt-BPIT is a CPDAG representing the P-map class for P.*

The complexity of PIT, BPIT, Opt-BPIT, and PC is of order 2^N, as in a fully connected graph, all edges must be tested using all possible CI tests. However, the number of CI tests in the PIT-family algorithms is always less than or equal to that of the standard PC algorithm (Propositions 3.8 and B.2). Table 1 compares these algorithms across various graph structures, including naive Bayes (NB) (where one node is connected to all others), star (the same structure but with reversed edges), and a forest skeleton (FS) (a DAG with the skeleton without cycles). Here, d represents the maximum number of neighbors, while s denotes the maximum number of neighbors that have a path to a common node, where $s \leq d$.

Proposition 3.8. *Consider random variables $\mathcal{X} = \{X_1, \dots, X_N\}$ with joint probability distribution P that admits a P-map. Assume that for each m, subsets $\mathcal{U}$ in PIT, BPIT, Opt-BPIT, and PC are tested according to the lexicographical order whenever the same candidate set is available. Then the total number of CI tests performed by PIT, BPIT, and Opt-BPIT is less than or equal to that in PC.*

Both statements are oracle statements. In finite samples, early CI errors can change the intermediate graph and therefore change which later tests are considered. For this reason, the empirical section reports runtime, CI-test counts, maximum conditioning-set sizes, and structural error separately rather than treating CI-test count as a direct proxy for speed or accuracy.

The worst-case complexity remains exponential, because in a complete graph all candidate edges and conditioning subsets may still have to be considered. The advantage appears when reachability or blockability removes endpoint-neighbor candidates. If d is the maximum endpoint degree and $s \leq d$ is the maximum number of endpoint neighbors that remain reachable from the opposite endpoint, then the path-restricted dependence replaces d by s in bounded-degree regimes.

Table 1: Number of CI tests for special structures.

Method	Bounded degree	Star	NB	FS
PC	N^{d+2}	2^{N-1}	2^{N-1}	N^{d+2}
PIT	N^{s+2}	N^2	N^3	N^3

4 Experiments

The experiments focus on the trade-off between reducing CI-test counts and incurring graph-computation and conditioning costs. The main controlled comparison uses PC, PIT, BPIT, and Opt-BPIT on discrete bnlearn networks. Additional representative baselines are reported on a paired subset of the same tasks.

The experiments use the standard bnlearn network repository (Scutari, 2022). The 17 networks range from 5 to 724 nodes; their sizes are listed in Appendix Table 3. For each network, datasets of sizes $n \in \{100, 1000, 10000\}$ were generated from the corresponding Bayesian network. Each algorithm was run on the same generated data. The reported metrics are runtime, CI-test count, maximum conditioning-set size, and structural Hamming distance (SHD) divided by the number of true arcs and reported as a percentage. Larger networks occasionally produced timeouts or memory failures, denoted by missing entries in the detailed tables.

Table 2 reports medians over the paired networks for which all four controlled methods completed. There are 17 networks in total, but only 12 paired networks in this table: the larger networks are retained in the detailed tables, but at least one of PC, PIT, BPIT, or Opt-BPIT timed out or ran out of memory on them in a given sample regime. The table separates runtime, CI-test count, maximum conditioning-set size, and structural error because these quantities do not move together. PIT is the fastest method in all three sample regimes. At $n = 10000$, PIT has median runtime about 17 seconds, compared with about 28 for PC, 29 for BPIT, and 26 for Opt-BPIT. Median SHD at $n = 10000$ is also lowest for PIT on these paired networks.

Figures 3 and 4 show runtime and structural error for the $n = 10000$ paired networks, ordered by number of nodes. The runtime plot shows the main speed pattern: PIT is usually the fastest PC-family method, while Opt-BPIT often improves over BPIT but remains slower than PIT because the blind-blocking search still uses larger conditioning sets. The SHD plot shows that the runtime gain of PIT is not obtained by a systematic loss of accuracy; PIT has the best median SHD among the four controlled methods on these paired networks. BPIT and Opt-BPIT sometimes improve a network-level SHD value, but their larger conditioning sets make the accuracy benefit non-uniform.

The detailed $n = 10000$ runtime, CI-count, and SHD results are reported in Appendix Tables 4 to 6; the $n = 100$ and $n = 1000$ tables are also in the appendix. The tables support the same pattern as the summaries. PIT reduces runtime broadly relative to PC. BPIT and Opt-BPIT reduce CI-test counts on several networks, especially when the graph contains blockable long paths, but they also increase the largest conditioning sets used by the remaining tests. Opt-BPIT reduces some redundant enumeration relative to BPIT; it does not remove the main statistical and computational effect of blind blocking, because the fixed blocking variables remain in the conditioning sets.

Appendix Table 7 summarizes representative baselines on the same $n = 10000$ tasks. To keep the comparison paired, the medians are computed only over the nine networks on which every method in the table completed: Cancer, Earthquake, Survey, Asia, Sachs, Child, Insurance, Water, and Mildew. The selected baselines are PC-Stable (Colombo and Maathuis, 2014), ConservativePC (Ramsey et al., 2006), ConsistentPC (Li et al., 2019), GES (Chickering, 2002), BOSS (Andrews et al., 2023), MMHC (Tsamardinos et al., 2006), MMPC (Tsamardinos et al., 2003), and HillClimbing (Koller and Friedman, 2009). Bold marks the best

Table 2: Median performance over paired complete cases. Lower runtime and SHD are better. CI-test counts and maximum conditioning sizes are reported separately because blind blocking can reduce the former while increasing the latter.

Sample size	Method	Paired networks	Median time	Median CI tests	Median max $\lvert\mathcal{S}\rvert$	Median SHD %
100	PC	12/17	1.96	**487.5**	2	**121.1**
	PIT	12/17	**0.813**	897	**1**	138.1
	BPIT	12/17	1.21	873	17	147.6
	Opt-BPIT	12/17	1.21	872	17	147.5
1000	PC	12/17	12.5	**959**	2.5	**114.5**
	PIT	12/17	**5.48**	1,520	**2**	121.3
	BPIT	12/17	8.38	1,366	15.5	119.7
	Opt-BPIT	12/17	8.21	1,361	15.5	125.4
10000	PC	12/17	27.87	**2,025.5**	4	74.65
	PIT	12/17	**16.55**	2,803	**3**	**68.07**
	BPIT	12/17	29.21	2,380	15.5	87.87
	Opt-BPIT	12/17	25.6	2,407	15.5	96.21

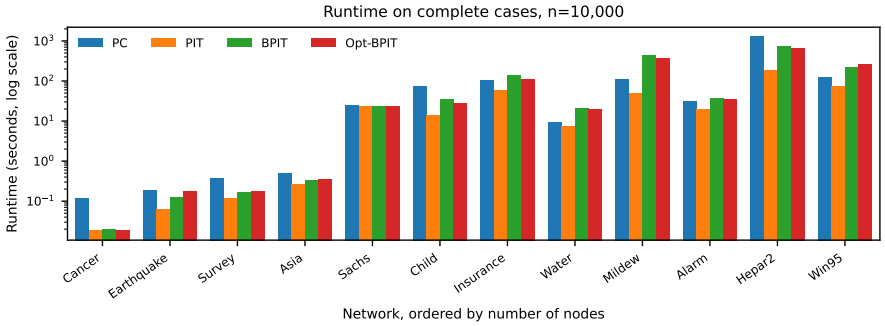

Figure 3: Runtime by network size for the $n = 10000$ paired networks. The y-axis is logarithmic. Networks are ordered by number of nodes.

median and underline marks the second-best median in each metric. On this paired subset, PIT has the best median runtime; its median SHD is tied for second with PC-Stable and behind ConservativePC, so the broader baseline result supports the runtime claim and a competitive-accuracy claim rather than uniform SHD dominance.

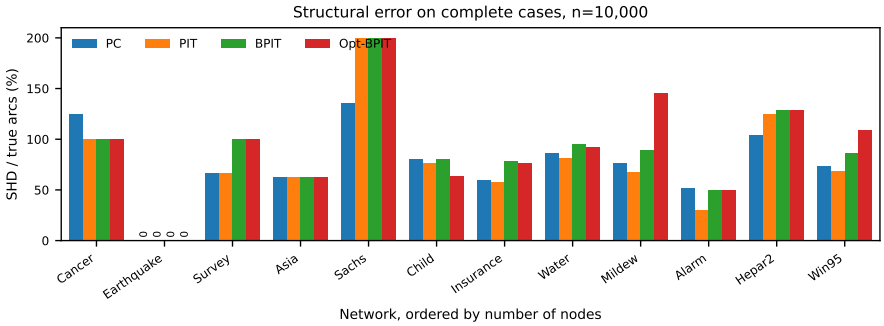

Figure 4: Structural error by network size for the $n = 10000$ paired networks. Lower values are better. The plot separates the runtime conclusion from the accuracy conclusion: PIT is not only faster in the median, but also competitive in structural error.

5 Discussion and conclusion

The results clarify the role of path information in PC-style structure learning. PIT occupies the useful first operating point: it removes endpoint-neighbor candidates that cannot reach the opposite endpoint in the partially learned skeleton. This reachability computation is inexpensive, does not inflate conditioning sets, and gives broad runtime gains in the reported simulations. The paired baseline comparison in Appendix Table 7 shows the same runtime pattern beyond the PIT-family comparison: among PC-Stable, ConservativePC, ConsistentPC, GES, BOSS, MMHC, MMPC, and HillClimbing on the common completed $n = 10000$ subset, PIT has the best median runtime.

BPIT and Opt-BPIT show what happens when the same path idea is pushed further. Their blind-blocking rule is exact under the oracle assumptions: long paths outside the triggering region can be blocked by fixing consecutive interior nodes, and the resulting algorithms recover the same Markov-equivalence class as PC. Empirically, however, the extra pruning changes the cost profile. BPIT often lowers the number of CI tests, but the remaining tests use larger conditioning sets and require additional graph computations. Opt-BPIT reduces redundant enumeration and can therefore be faster than BPIT, but it inherits the same large fixed conditioning sets; this explains why its runtime can improve without producing a corresponding accuracy improvement.

The main conclusion is that path information has an operating point. Cheap reachability pruning pays off broadly, while further CI-test reductions must be judged against their conditioning and graph-operation costs. This is not a limitation of the oracle theory: all PIT-family algorithms are exact under Markovness, faithfulness, and exact CI decisions. It is a practical statement about finite-sample structure learning, where reducing the number of tests is useful only when the saved tests cost more than the graph information and enlarged conditioning sets used to avoid them. In PC-style learning, fewer CI tests help only when they are saved at the right computational price.

References

Bryan Andrews, Joseph Ramsey, Ruben Sanchez-Romero, Jazmin Camchong, and Erich Kummerfeld. Fast scalable and accurate discovery of DAGs using the best order score search and grow-shrink trees. In *Advances in Neural Information Processing Systems*, volume 36, pages 63945–63956, 2023.

Erica Cai, Andrew McGregor, and David Jensen. Improving the efficiency of the PC algorithm by using model-based conditional independence tests. *arXiv preprint arXiv:2211.06536*, 2022.

David Maxwell Chickering. Optimal structure identification with greedy search. In *Journal of Machine Learning Research*, volume 3, pages 507–554, 2002.

Diego Colombo and Marloes H. Maathuis. Order-independent constraint-based causal structure learning. *Journal of Machine Learning Research*, 15(1):3741–3782, 2014.

Maxime Gasse, Alex Aussem, and Haytham Elghazel. Hybrid HPC: A scalable hybrid bayesian network structure learning algorithm. *Expert Systems with Applications*, 41(15):6755–6772, 2014.

Judith L. Gersting. *Mathematical Structures for Computer Science*. W. H. Freeman, 2006.

Hamid Kalantari, Russell Greiner, and Pouria Ramazi. Extendable and iterative structure learning strategy for bayesian networks. In *The Thirteenth International Conference on Learning Representations*, 2025.

Markus Kalisch and Peter Buehlmann. Estimating high-dimensional directed acyclic graphs with the PC-algorithm. *Journal of Machine Learning Research*, 8:613–636, 2007.

Murat Kocaoglu. Characterization and learning of causal graphs with small conditioning sets. *arXiv preprint arXiv:2301.09028*, 2023.

Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.

Kenneth Lee, Bruno Ribeiro, and Murat Kocaoglu. Constraint-based causal discovery from a collection of conditioning sets. In *Proceedings of the Forty-First Conference on Uncertainty in Artificial Intelligence*, volume 286 of *Proceedings of Machine Learning Research*, pages 2486–2516. PMLR, 2025.

Honghao Li, Vincent Cabeli, Nadir Sella, and Herve Isambert. Constraint-based causal structure learning with consistent separating sets. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Dimitris Margaritis and Sebastian Thrun. Bayesian network induction via local neighborhoods. In *Advances in Neural Information Processing Systems*, volume 12, 1999.

Elaheh Mokhtarian, Mahdi Akbari, AmirEmad Ghassami, and Negar Kiyavash. A recursive markov boundary-based approach to causal structure learning. In *Proceedings of the KDD Workshop on Causal Discovery*, Proceedings of Machine Learning Research. PMLR, 2021.

Preksha Nandy, Alain Hauser, and Marloes H. Maathuis. High-dimensional consistency in score-based and hybrid structure learning. *The Annals of Statistics*, 46(6A):3151–3183, 2018.

Achille Nazaret and David M. Blei. Extremely greedy equivalence search. In *Proceedings of the Fortieth Conference on Uncertainty in Artificial Intelligence*, Proceedings of Machine Learning Research. PMLR, 2024.

Judea Pearl. *Causality: Models, Reasoning, and Inference*. Cambridge University Press, 2 edition, 2009.

Joseph Ramsey, Jiji Zhang, and Peter Spirtes. Adjacency-faithfulness and conservative causal inference. In *Proceedings of the Twenty-Second Conference on Uncertainty in Artificial Intelligence*, pages 401–408, 2006.

Marco Scutari. bnlearn: Bayesian network repository. *Online repository*, 2022. URL https://www.bnlearn.com/bnrepository/.

Kirankumar Shiragur, Jiaqi Zhang, and Caroline Uhler. Causal discovery with fewer conditional independence tests. In *Proceedings of the Forty-First International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 45060–45078. PMLR, 2024.

Liam Solus, Yuhao Wang, and Caroline Uhler. Consistency guarantees for greedy permutation-based causal inference algorithms. *Biometrika*, 108(4):795–814, 2021.

Arjun Sondhi and Ali Shojaie. The reduced PC-algorithm: Improved causal structure learning in large random networks. *Journal of Machine Learning Research*, 20(164):1–31, 2019.

Peter Spirtes and Clark Glymour. An algorithm for fast recovery of sparse causal graphs. *Social Science Computer Review*, 9(1):62–72, 1991.

Peter Spirtes, Clark Glymour, and Richard Scheines. *Causation, Prediction, and Search*. MIT Press, 2 edition, 2000.

Ioannis Tsamardinos, Constantin F. Aliferis, and Alexander Statnikov. Time and sample efficient discovery of markov blankets and direct causal relations. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 673–678, 2003.

Ioannis Tsamardinos, Laura E. Brown, and Constantin F. Aliferis. The max-min hill-climbing Bayesian network structure learning algorithm. *Machine Learning*, 65(1):31–78, 2006.

A Additional experimental tables

Table 3: The bnlearn benchmark networks used in the main comparison. Average degree is $2|E|/|V|$ for the skeleton.

Network	Nodes	Arcs	Avg. deg.	Network	Nodes	Arcs	Avg. deg.
Cancer	5	4	1.60	Alarm	37	46	2.49
Earthquake	5	4	1.60	Hepar2	70	123	3.51
Survey	6	6	2.00	Win95	76	112	2.95
Asia	8	8	2.00	Pathfinder	109	195	3.58
Sachs	11	17	3.09	Andes	223	338	3.03
Child	20	25	2.50	Diabetes	413	602	2.92
Insurance	27	52	3.85	Pigs	441	592	2.68
Water	32	66	4.12	Link	724	1125	3.11
Mildew	35	46	2.63				

Table 4: Runtime in seconds, $n = 10000$.

Network	PC	PIT	BPIT	Opt-BPIT
Cancer	0.12	0.0186	0.0194	0.019
Earthquake	0.1843	0.0624	0.1273	0.1743
Survey	0.3614	0.1176	0.1691	0.1707
Asia	0.4875	0.2626	0.331	0.34
Sachs	24.27	23.9	23.04	23.43
Child	73.03	13.86	35.39	27.77
Insurance	101.7	58.53	135.8	112.3
Water	9.297	7.211	20.38	19.68
Mildew	110	48.57	430	358.4
Alarm	31.46	19.25	36.48	34.85
Hepar2	1,264	180.3	710.5	634.8
Win95	124.1	73.15	222.7	267.2
Pathfinder	–	86,400	–	–
Andes	1,553	1,799	–	86,400
Diabetes	–	86,400	–	–
Pigs	86,400	86,400	86,400	–
Link	86,400	86,400	–	–

Table 5: CI-test count with maximum conditioning-set size in parentheses, $n = 10000$.

Network	PC	PIT	BPIT	Opt-BPIT
Cancer	32(2)	20(0)	20(0)	20(0)
Earthquake	46(3)	54(1)	54(1)	54(1)
Survey	58(3)	70(2)	36(4)	36(4)
Asia	103(3)	154(2)	166(5)	166(5)
Sachs	1,538(6)	1,544(6)	1,478(6)	1,478(6)
Child	2,513(6)	2,870(5)	2,190(14)	2,248(14)
Insurance	5,745(6)	8,956(6)	8,178(18)	8,228(18)
Water	1,493(3)	2,736(3)	2,570(17)	2,566(17)
Mildew	5,165(4)	4,778(3)	4,116(30)	4,126(30)
Alarm	3,472(4)	5,566(3)	5,216(25)	5,210(25)
Hepar2	31,695(8)	23,022(5)	15,788(65)	16,192(65)
Win95	13,846(6)	21,924(5)	20,270(73)	24,214(68)
Pathfinder	–	–	–	–
Andes	176,423(9)	166,348(9)	137,946(218)	–
Diabetes	–	–	–	–
Pigs	–	–	–	–
Link	–	–	–	–

Table 6: SHD divided by the number of true arcs, in percent, $n = 10000$. Lower is better.

Network	PC	PIT	BPIT	Opt-BPIT
Cancer	125	100	100	100
Earthquake	0	0	0	0
Survey	66.67	66.67	100	100
Asia	62.5	62.5	62.5	62.5
Sachs	135.3	200	200	200
Child	80	76	80	64
Insurance	59.62	57.69	78.85	76.92
Water	86.36	81.82	95.45	92.42
Mildew	76.09	67.39	89.13	145.7
Alarm	52.17	30.43	50	50
Hepar2	104.1	124.4	128.4	128.4
Win95	73.21	68.75	86.61	108.9
Pathfinder	–	–	–	–
Andes	69.82	71.6	216.6	–
Diabetes	–	–	–	–
Pigs	–	–	–	–
Link	–	–	–	–

Table 7: Representative baseline summary at $n = 10000$. Medians are paired over the same nine completed networks.

Method	Common networks	Median time	Median SHD %
PC	9	<u>9.30</u>	76.09
PC-Stable	9	13.02	<u>67.39</u>
ConservativePC	9	31.24	**62.50**
ConsistentPC	9	14.26	143.93
PIT	9	**7.21**	<u>67.39</u>
BPIT	9	20.38	89.13
Opt-BPIT	9	19.68	92.42
GES	9	5,937.10	91.30
BOSS	9	332.69	96.00
MMHC	9	3,175.30	162.50
MMPC	9	304.37	144.00
HillClimbing	9	6,836.40	212.50

B Proofs and algorithms

Definition B.1 (d-separation)**.** (Koller and Friedman, 2009) Consider DAG $\mathcal{G}$ with node set $\mathcal{X}$. A trail $\mathcal{T}$ between two nodes $X, Y \in \mathcal{X}$ is *active* relative to (or given) a set of nodes $\mathcal{Z} \subseteq \mathcal{X}$ if *(i)* for each collider on $\mathcal{T}$, at least one of the descendants of the collider node is in $\mathcal{Z}$, and *(ii)* no other node on $\mathcal{T}$ is in $\mathcal{Z}$. The node subsets $\mathcal{X}_1, \mathcal{X}_2 \subseteq \mathcal{X}$ are *d-separated* given $\mathcal{Z}$, denoted $d-sep_{\mathcal{G}}(\mathcal{X}_1, \mathcal{X}_2 \mid \mathcal{Z})$, if there is no active trail between any node $X_1 \in \mathcal{X}_1$ and any node $X_2 \in \mathcal{X}_2$ given $\mathcal{Z}$. The set of all d-separations in $\mathcal{G}$ is denoted by $\mathcal{I}(\mathcal{G})$.

Proof of Lemma 3.2. Since $|\mathcal{T}| \geq 2$, there is at least one non-collider triple in the trail. So the middle variable in this triple blocks the trail, which proves the first item. For the second, note that both of the adjacent nodes say U and V, cannot be collider nodes. So U and its two neighbors on $\mathcal{T}$ or V and its two neighbors on $\mathcal{T}$ form a non-collider triple, which again becomes inactive once U and V are observed. Consequently, the trail is blocked. The third item immediately follows either of the first two. □

Proof of Lemma 3.6 (sufficiency) We prove the result for $\mathcal{M}^*_{YX} = \mathbf{Pa}_{X,Y} \cup \mathcal{B}^C_{XY} \cup \mathcal{O}_{XY}$, where $\mathcal{B}^C_{XY}$ and $\mathcal{O}_{XY}$ are the arbitrary sets in Definition 3.5. In view of Lemma 2.3, X and Y are independent conditioned on $\mathbf{Pa}_X$ or $\mathbf{Pa}_Y$. Without loss of generality assume that $X \perp Y \mid \mathbf{Pa}_X$. It follows that $X \perp Y \mid \mathbf{Pa}_{X,Y}$, because the parents of X that are not reachable by Y in $\mathcal{G} \setminus \{X\}$ do no activate a trial between X and Y. Now consider a covering path $\mathcal{T}$ between X and Y. If $\mathcal{T}$ is not connected to a node in $\mathcal{B}^C_{XY} \cup \mathcal{O}_{XY}$ by some trail in $\mathcal{G} \setminus \{X, Y\}$, then neither $\mathcal{B}_{XY}$ nor $\mathcal{O}_{XY}$ may trigger a collider on path $\mathcal{T}$. Hence, $\mathcal{T}$ is inactive once conditioned on $\mathbf{Pa}_{X,Y} \cup \mathcal{B}^C_{XY} \cup \mathcal{O}_{XY}$ as it was already inactive when conditioned on $\mathbf{Pa}_{X,Y}$. So consider the case where $\mathcal{T}$ is connected to a node in $\mathcal{B}^C_{XY} \cup \mathcal{O}_{XY}$ by some trail in $\mathcal{G} \setminus \{X, Y\}$. Then $\mathcal{T}$ is connected to $\mathcal{B}^C_{XY}$ as $\mathcal{O}_{XY}$ is off-path blockable. It follows that $\mathcal{T}$ is connected to some connected blockable neighborhood $\mathcal{B}^c_{XY}$ that is the union of $\mathcal{B}_{XY}$ and a connected component of $\mathcal{C}_{XY}$, denoted by $\mathcal{C}^c_{XY}$. Any node that is reachable by $\mathcal{B}^c_{XY}$ in $\mathcal{G} \setminus \{X, Y\}$ also belongs to the connected component $\mathcal{C}^c_{XY}$. Thus, the interior of $\mathcal{T}$ belongs to the connected component $\mathcal{C}^c_{XY}$, i.e., $\text{int}(\mathcal{T}) \subseteq \mathcal{C}^c_{XY}$. So $\mathcal{T}$ is blockable, and is blocked by observing $\mathcal{B}_{XY} \cap \mathcal{T}$ in view of Lemma 3.2. On the other hand, $\mathcal{B}_{XY} \cap \mathcal{T} = \mathcal{B}^c_{XY} \cap \mathcal{T}$. Moreover, Lemma 3.2 also implies that observing additional nodes does not make the path active. So $\mathcal{T}$ is inactive if $\mathbf{Pa}_{X,Y} \cup \mathcal{B}^C_{XY} \cup \mathcal{O}_{XY}$ is observed. Therefore, every covering path between X and Y is inactive, implying that X and Y are d-separated, completing the proof as $\mathcal{G}$ is a P-map. (necessity) Should X and Y be independent conditioned on any set including $\mathcal{M}^*_{XY}$, they are d-separated in the P-map $\mathcal{G}$ according to Lemma 2.2. Thus, they cannot be linked. □

Proof of Theorem 3.7. We only prove the theorem for Algorithm 2; the others can be proven similarly. Consider nodes $X, Y \in \mathcal{X}$. Should there be an edge between them in the P-map, they would not become independent conditioned on any subset of the remaining nodes according to Lemma 2.2. So none of the CI tests in either of the algorithms will be positive, preserving the edge.

Now consider the case where X and Y are not adjacent in the P-map. In view of Lemma 3.6, it suffices to show that at least one separator $\mathcal{M}^*_{XY}$ and at least one separator $\mathcal{M}^*_{YX}$ is found by the algorithm. We only prove the first one. Consider an arbitrary iteration m of the 2 and let $\mathcal{G}$ be the graph before performing the search over the possible sets $\mathcal{U}$. Let $\mathcal{W}_{XY}$, $\mathcal{B}_{XY}$, and $\mathcal{C}_{XY}$ be the set of potentially trigger nodes, blockable neighbors, and maximal blockable set of the P-map, and $\mathcal{W}^{\mathcal{G}}_{XY}$, $\mathcal{B}^{\mathcal{G}}_{XY}$, and $\mathcal{C}^{\mathcal{G}}_{XY}$, the corresponding sets in $\mathcal{G}$. Since $\mathcal{G}$ is a subgraph of the P-map, $\mathcal{W}_{XY} \subseteq \mathcal{W}^{\mathcal{G}}_{XY}$ and in turn $\mathcal{C}^{\mathcal{G}}_{XY} \subseteq \mathcal{C}_{XY}$. Hence, every blockable covering path $\mathcal{T}^{\mathcal{G}}$ of X and Y in $\mathcal{G}$ satisfies $\text{int}(\mathcal{T}^{\mathcal{G}}) \subseteq \mathcal{C}_{XY}$. Thus, if $\mathcal{T}^{\mathcal{G}}$ also exists in the P-map, then it is a blockable path, otherwise, all interior nodes of $\mathcal{T}^{\mathcal{G}}$ are off-path blockable. In the first case, $\text{int}(\mathcal{T}^{\mathcal{G}})$ belongs to a connected component of $\mathcal{C}_{XY}$, implying $\mathcal{T}^{\mathcal{G}} \cap \mathcal{B}^{\mathcal{G}}_{XY} = \mathcal{T}^{\mathcal{G}} \cap \mathcal{B}^c_{XY}$ for some connected blockable neighborhood of X relative to Y, $\mathcal{B}^c_{XY}$. So $\mathcal{B}^{\mathcal{G}}_{XY} = \mathcal{B}^C_{XY} \cup \mathcal{O}_{XY}$ for a union $\mathcal{B}^C_{XY}$ of some connected blockable neighborhoods of X relative to Y and for some off-path blockable set $\mathcal{O}_{XY} \subseteq \mathcal{C}_{XY}$. Therefore, $\mathbf{Pa}_{X,Y} \cup \mathcal{B}^{\mathcal{G}}_{XY}$ is a separator.

So it suffices for the set $\mathcal{U}$ to equal $\mathbf{Pa}_{X,Y} \setminus \mathcal{B}^{\mathcal{G}}_{XY}$ at some point in the algorithm. On the other hand, $\mathbf{Pa}_{X,Y} \setminus \mathcal{B}^{\mathcal{G}}_{XY}$ is the intersection of $\mathbf{Pa}_X \setminus \mathcal{B}^{\mathcal{G}}_{XY}$ with the reachable set of Y in $\mathcal{G} \setminus \{X\}$. Therefore, $\mathbf{Pa}_{X,Y} \setminus \mathcal{B}^{\mathcal{G}}_{XY} = \mathbf{Pa}_X \setminus \mathcal{C}^{\mathcal{G}}_{XY}$, implying that $\mathbf{Pa}_{X,Y} \setminus \mathcal{B}^{\mathcal{G}}_{XY} = \mathbf{Pa}_X \cap \mathcal{W}^{\mathcal{G}}_{XY}$.

Let $|\mathbf{Pa}_{X,Y}| = k$. Consider the iteration with $m = k$ in the algorithm, and let $\mathcal{G}_k$ be the corresponding graph. It holds that $\mathbf{Pa}_{X,Y} \setminus \mathcal{B}^{\mathcal{G}}_{XY} = \mathbf{Pa}_X \cap \mathcal{W}^{\mathcal{G}_k}_{XY} \subseteq \mathcal{N}^{\mathcal{G}_k}_X \cap \mathcal{W}^{\mathcal{G}_k}_{XY}$. On the other hand, $|\mathbf{Pa}_{X,Y} \setminus \mathcal{B}^{\mathcal{G}}_{XY}| = k - |\mathbf{Pa}_{X,Y} \cap \mathcal{B}^{\mathcal{G}_k}_{XY}|$.

Moreover, $|\mathbf{Pa}_{X,Y} \cap \mathcal{B}^{\mathcal{G}_k}_{XY}| \leq |\mathcal{N}^{\mathcal{G}_k}_X \cup \mathcal{B}^{\mathcal{G}_k}_{XY}|$. Hence, $k - |\mathcal{N}^{\mathcal{G}_k}_X \cap \mathcal{B}^{\mathcal{G}_k}_{XY}| \leq |\mathbf{Pa}_{X,Y} \setminus \mathcal{B}^{\mathcal{G}_k}_{XY}| \leq k$. Therefore, by searching through all subsets $\mathcal{U}$ of $\mathcal{N}^{\mathcal{G}_k}_X \cup \mathcal{B}^{\mathcal{G}_k}_{XY}$, where $k - |\mathcal{N}^{\mathcal{G}_k}_X \cap \mathcal{B}^{\mathcal{G}_k}_{XY}| \leq |\mathcal{U}| \leq k$, the desired $\mathcal{U}$ is guaranteed to be found, such that $X \perp Y \mid \mathcal{B}^{\mathcal{G}_k}_{XY} \cup \mathcal{U}$. By going through all values of m the value k is also found. So the edge between X and Y will be deleted. Therefore, the skeletons of $\hat{\mathcal{G}}$ and the P-map matches.

The orientation step of the edges in the stated algorithms is the same as the one in PC, with the difference that now $\text{Sepset}(X, Y)$ may include additional nodes, i.e., those in $\mathcal{B}_{XY}$, that are not a common neighbor of X and Y. However, this will not affect the orientations as the purpose of having the set $\text{Sepset}(X, Y)$ is to check if a common neighbor belongs to it, whereas those in $\mathcal{B}_{XY}$ are not a common neighbor of X and Y. □

Proof of Proposition 3.8. We prove by contradiction. Assume on the contrary that the proposition is violated for the first time for nodes X and Y and iteration m. Consider the case where the obtained graph in both algorithms is the same so far. Also, consider the case where there is an edge between X and Y in the true graph. In PC, all possible $\mathcal{U}$'s of size m from the set $\text{Adj}(\mathcal{G}, X)$ are checked, which equals $\binom{|\text{Adj}(\mathcal{G},X)|}{m}$ many candidates. In Algorithm 2, $\text{Adj}(\mathcal{G}, X)$ is partitioned into $\mathcal{W}^{\mathcal{G}}_{XY}$ and $\text{Adj}(\mathcal{G}, X) \setminus \mathcal{W}^{\mathcal{G}}_{XY}$. Without any optimization, the number of candidates would equal that in PC, i.e., to choose k variables from $\mathcal{W}^{\mathcal{G}}_{XY}$ and $m - k$ from $\text{Adj}(\mathcal{G}, X) \setminus \mathcal{W}^{\mathcal{G}}_{XY}$, i.e.,

$$\sum_k \binom{|\mathcal{W}^{\mathcal{G}}_{XY}|}{k} \binom{|\text{Adj}(\mathcal{G}, X) \setminus \mathcal{W}^{\mathcal{G}}_{XY}|}{m-k} = \binom{|\text{Adj}(\mathcal{G}, X)|}{m}.$$

However, the choice of $m - k$ variables from the set $\text{Adj}(\mathcal{G}, X) \setminus \mathcal{W}^{\mathcal{G}}_{XY}$ does not happen in Algorithm 2; instead, the whole subset $\mathcal{B}^{\mathcal{G}}_{XY} \subseteq \text{Adj}(\mathcal{G}, X) \setminus \mathcal{W}^{\mathcal{G}}_{XY}$ is observed. This reduces the number of candidate $\mathcal{U}$'s to $\sum_k \binom{|\mathcal{W}^{\mathcal{G}}_{XY}|}{k}$ which is less than or equal to $\binom{|\text{Adj}(\mathcal{G},X)|}{m}$. Thus, for every iteration m, the number of candidate $\mathcal{U}$'s that are searched in Algorithm 2 is no more than that in PC. Now, if the graph obtained by Algorithm 2 had fewer links compared to that obtained by Algorithm 2, then $\mathcal{W}^{\mathcal{G}}_{XY}$ might have been smaller, but that does not make the total number of tests to exceed $\binom{|\text{Adj}(\mathcal{G},X)|}{m}$.

Now consider the case where there is no edge between X and Y in the true graph. According to the proof of Theorem 3.7, if a certain $\mathcal{U}$ at iteration m renders X and Y independent in PC, so does the $\mathcal{B}^{\mathcal{G}}_{XY} \cup \mathcal{U}$ in Algorithm 2. So if the edge $X - Y$ is removed by PC at iteration m, it will also be removed by Algorithm 2 by at most iteration m. On the other hand, the same lexicographical ordering is supposed for both algorithms. This means that Algorithm 2 searches through the candidates for the set $\mathcal{U}$ in the same order that PC does, just that some candidates may not be searched in Algorithm 2, and that some candidates appear in a smaller cardinality as some members belong to the set $\mathcal{B}^{\mathcal{G}}_{XY}$ which are not searched through. This means that the Algorithm 2 takes at most the same number of iterations as that in PC to find the proper $\mathcal{U}$, a contradiction. □

Proposition B.2. *Consider random variables $\mathcal{X}$ with joint distribution P that admits P-map $\mathcal{G}$. If $\mathcal{G}$ contains no cycle, then the total number of CI tests performed by PIT, BPIT, and Opt-BPIT is $\mathcal{O}(N^3)$.*

Proof of Proposition B.2. First, we prove that if between two nonadjacent nodes X and Y at most, there is one path in the skeleton graph, these nodes are separated by observing at most one node. If a collider node exists in the path between X and Y, then X and Y are d-separated given the empty set, and if there is no collider node in the path, by observing each node Z on the path X and Y are d-separated given Z. So, by checking the marginal independence test between X and Y or the CI test between X and Y given Z the spurious edge between them is removed. Since there is no cycle in the skeleton, there is at most one path between any two nodes in the true DAG. Therefore, by observing at most one variable on the path between any two nodes, all spurious edges can be removed. Verifying the correctness of the remaining edges does not require additional CI tests, as there are no alternative paths with mediator nodes between any two adjacent nodes. Thus, the number of CI tests is $\mathcal{O}(N^3)$ in the worst case. □

If d represents the maximum number of neighbors in the true DAG, and s denotes the maximum number of neighbors reachable from another node in the true DAG, the number of CI tests is bounded by $\mathcal{O}(N^s)$ in BPIT, while the PC algorithm requires $\mathcal{O}(N^d)$ CI tests. Since s is always less than or equal to d, it follows that $\mathcal{O}(N^s) \leq \mathcal{O}(N^d)$. Table 1 indicates the number of CI tests for some special structures.

Algorithm 3: The PC Algorithm

Input: A set of variables $\mathcal{X}$ and their joint probability distribution P
Output: A partially directed acyclic graph

Form the complete undirected graph $\mathcal{G}$ over nodes $\mathcal{X}$;
$\text{Sepset}(X, Y) = \emptyset$ for all $X, Y \in \mathcal{X}$;
$m = 0$
while *maximum node degree in* $\mathcal{G}$ *is greater than* m **do**
  for $X \in \mathcal{X}$
    for $Y \in \text{Adj}(\mathcal{G}, X)$
      for $\mathcal{U} \subseteq \text{Adj}(\mathcal{G}, X) \setminus \{Y\}$ *and* $|\mathcal{U}| = m$
        if $X \perp Y \mid \mathcal{U}$
          Remove the edge $X - Y$ from $\mathcal{G}$;
          $\text{Sepset}(X, Y) \leftarrow \mathcal{U}$;
  $m = m + 1$;
Orient the edges using the orientation rules in (Spirtes et al., 2000).

Algorithm 4: The $\mathcal{W}$, $\mathcal{B}$ and $\mathcal{C}$-finding Algorithm

Input: Undirected graph $\mathcal{G}$; nodes X and Y
Output: $\mathcal{W}^{\mathcal{G}}_{XY}, \mathcal{B}^{\mathcal{G}}_{XY}, \mathcal{C}^{\mathcal{G}}_{XY}$

$\mathcal{W}^{\mathcal{G}}_{XY} = \mathcal{B}^{\mathcal{G}}_{XY} = \emptyset$;
for $Z \in \text{Adj}(\mathcal{G}, X) \cap \text{Adj}(\mathcal{G}, Y)$
  $\mathcal{W}^{\mathcal{G}}_{XY} \leftarrow \mathcal{W}^{\mathcal{G}}_{XY} \cup \mathcal{R}^{\mathcal{G} \setminus \{X,Y\}}_{Z}$; // Reachable nodes from common neighbors of X and Y
$\mathcal{C}^{\mathcal{G}}_{XY} \leftarrow \mathcal{X} \setminus (\{X, Y\} \cup \mathcal{W}^{\mathcal{G}}_{XY})$; // The maximal blockable set
$\mathcal{R}'_{Y} \leftarrow \mathcal{R}^{\mathcal{G} \setminus (\mathcal{W}^{\mathcal{G}}_{XY} \cup (\text{Adj}(\mathcal{G}, X) \setminus \{Y\}))}_{Y}$; // Reachable nodes from Y in the absence of trigger nodes and X's neighbors
for $Z \in \text{Adj}(\mathcal{G}, X) \cap \mathcal{C}^{\mathcal{G}}_{XY}$ // blockable neighbors of X
  if $\text{Adj}(\mathcal{G}[\mathcal{C}^{\mathcal{G}}_{XY}], Z) \cap \mathcal{R}'_{Y} \neq \emptyset$ // that can reach Y via a blockable path
    $\mathcal{B}^{\mathcal{G}}_{XY} \leftarrow \mathcal{B}^{\mathcal{G}}_{XY} \cup \{Z, \text{Adj}(\mathcal{G}[\mathcal{C}^{\mathcal{G}}_{XY}], Z)\}$; // The blockable neighborhood

Algorithm 5: The optimized Path-Driven Independence Testing Algorithm(Optimized PDIT)

Input: A set of variables $\mathcal{X}$ and their joint probability distribution P
Output: A partially directed acyclic graph

Form the complete undirected graph $\mathcal{G}$ over nodes $\mathcal{X}$;
$\text{Sepset}(X, Y) = \emptyset$ for all $X, Y \in \mathcal{X}$; $\bar{\mathcal{B}}_{XY} = \emptyset$;$\bar{\mathcal{W}}_{XY} = \emptyset$;
$m = 0$
while *maximum node degree in* $\mathcal{G}$ *is greater than* m **do**
  for $X \in \mathcal{X}$
    for $Y \in \text{Adj}(\mathcal{G}, X)$
      for $\{\mathcal{U} \subseteq \mathcal{W}^{\mathcal{G}}_{XY} \cap \text{Adj}(\mathcal{G}, X)$ *and* $|\mathcal{U}| \in [m - |\mathcal{B}^{\mathcal{G}}_{XY} \cap \bar{\mathcal{W}}_{XY} \cap \text{Adj}(\mathcal{G}, X)|, m]\}$
        if $X \perp Y \mid \mathcal{B}^{\mathcal{G}}_{XY} \cup \mathcal{U}$
          Remove the edge $X - Y$ from $\mathcal{G}$;
          $\text{Sepset}(X, Y) \leftarrow (\mathcal{B}^{\mathcal{G}}_{XY} \cap \text{Adj}(\mathcal{G}, X)) \cup \mathcal{U}$;
      $\bar{\mathcal{W}}_{XY} = \mathcal{W}^{\mathcal{G}}_{XY}$;
  $m = m + 1$;
Orient the edges using the orientation rules in (Spirtes et al., 2000).

C Detailed bnlearn results

C.1 Sample size 100

Table 8: Runtime in seconds, $n = 100$.

Network	PC	PIT	BPIT	Opt-BPIT
Cancer	8.221	0.0136	0.0145	0.0191
Earthquake	0.3839	0.0134	0.0252	0.0249
Survey	0.0213	0.0209	0.0212	0.0214
Asia	0.2363	0.0618	0.3342	0.297
Sachs	172.9	0.2549	0.4733	0.2753
Child	1.298	0.972	1.137	1.111
Insurance	1.742	1.451	2.078	2.126
Water	1.839	0.6543	1.29	1.305
Mildew	2.087	1.626	8.199	5.282
Alarm	2.982	2.26	4.239	4.412
Hepar2	16.64	3.458	20	19.4
Win95	10.86	4.948	26.58	26.55
Pathfinder	34.51	27.49	–	–
Andes	38.94	47.23	–	–
Diabetes	285.6	480	–	–
Pigs	222.9	575	–	–
Link	–	3,600	–	–

Table 9: CI-test count with maximum conditioning-set size in parentheses, $n = 100$.

Network	PC	PIT	BPIT	Opt-BPIT
Cancer	10(0)	20(0)	20(0)	20(0)
Earthquake	11(1)	20(0)	24(3)	24(3)
Survey	15(0)	30(0)	30(0)	30(0)
Asia	40(1)	74(1)	74(1)	74(1)
Sachs	137(2)	212(1)	210(4)	210(4)
Child	409(2)	724(1)	680(12)	678(12)
Insurance	657(2)	1,234(2)	1,192(23)	1,196(23)
Water	566(2)	1,070(1)	1,066(22)	1,066(22)
Mildew	698(1)	1,360(1)	1,458(33)	1,388(33)
Alarm	1,250(2)	2,084(2)	2,118(28)	2,160(28)
Hepar2	2,611(2)	5,120(1)	5,080(53)	5,086(53)
Win95	3,585(2)	6,792(3)	6,838(57)	6,886(57)
Pathfinder	20,359(5)	16,956(3)	19,084(91)	19,366(91)
Andes	26,860(3)	53,152(2)	–	–
Diabetes	102,436(3)	207,950(2)	–	–
Pigs	129,429(3)	255,360(2)	–	–
Link	–	–	–	–

Table 10: SHD divided by the number of true arcs, in percent, $n = 100$.

Network	PC	PIT	BPIT	Opt-BPIT
Cancer	100	100	100	100
Earthquake	125	100	150	150
Survey	133.3	133.3	133.3	133.3
Asia	137.5	187.5	187.5	187.5
Sachs	135.3	158.8	147.1	147.1
Child	112	148	148	148
Insurance	119.2	123.1	126.9	123.1
Water	122.7	128.8	131.8	131.8
Mildew	113	145.7	178.3	200
Alarm	119.6	141.3	147.8	167.4
Hepar2	117.1	156.1	154.5	151.2
Win95	127.7	134.8	147.3	146.4
Pathfinder	131.3	149.2	161.5	188.2
Andes	113	141.7	–	–
Diabetes	124.8	165.9	–	–
Pigs	116.9	121.3	–	–
Link	–	–	–	–

C.2 Sample size 1000

Table 11: Runtime in seconds, $n = 1000$.

Network	PC	PIT	BPIT	Opt-BPIT
Cancer	0.058	0.0158	0.0542	0.0685
Earthquake	0.091	0.0865	0.0585	0.0586
Survey	0.1165	0.0312	0.0221	0.0225
Asia	26.28	0.2049	0.1908	0.3507
Sachs	9.042	10.44	10.35	10.36
Child	14.6	3.542	6.411	6.056
Insurance	18.32	12.47	16.59	16.11
Water	2.033	1.76	5.806	4.943
Mildew	56.33	9.806	70.41	48.58
Alarm	10.41	7.423	11.65	10.55
Hepar2	21.68	7.61	74.75	78.94
Win95	40.47	12.22	50.91	68.11
Pathfinder	–	7,156	8,937	21,600
Andes	116.2	142.8	–	–
Diabetes	4,637	4,226	–	–
Pigs	86,400	21,600	–	–
Link	10,351	10,961	–	–

Table 12: CI-test count with maximum conditioning-set size in parentheses, $n = 1000$.

Network	PC	PIT	BPIT	Opt-BPIT
Cancer	23(2)	20(0)	20(0)	20(0)
Earthquake	30(2)	58(1)	56(1)	56(1)
Survey	18(1)	30(0)	30(0)	30(0)
Asia	84(2)	168(2)	156(3)	160(4)
Sachs	602(5)	1,202(5)	1,204(5)	1,204(5)
Child	1,411(5)	1,498(3)	1,320(15)	1,312(15)
Insurance	4,373(5)	3,900(5)	3,360(25)	3,604(25)
Water	794(2)	1,542(2)	1,412(16)	1,410(16)
Mildew	1,124(2)	2,206(2)	2,314(30)	2,242(22)
Alarm	2,444(3)	3,894(3)	3,860(24)	3,852(24)
Hepar2	4,832(4)	6,834(2)	6,532(67)	6,680(67)
Win95	6,835(4)	11,686(4)	10,728(65)	15,150(65)
Pathfinder	–	329,574(11)	397,224(102)	–
Andes	43,296(6)	84,300(6)	94,766(215)	88,112(215)
Diabetes	796,429(5)	750,878(5)	–	–
Pigs	–	–	–	–
Link	1,459,005(11)	1,318,638(6)	–	–

Table 13: SHD divided by the number of true arcs, in percent, $n = 1000$.

Network	PC	PIT	BPIT	Opt-BPIT
Cancer	75	100	100	100
Earthquake	200	225	225	225
Survey	133.3	100	100	100
Asia	162.5	150	125	125
Sachs	147.1	158.8	158.8	158.8
Child	100	124	108	84
Insurance	98.08	92.31	92.31	96.15
Water	125.8	121.2	124.2	125.8
Mildew	82.61	110.9	147.8	197.8
Alarm	69.56	93.48	89.13	82.61
Hepar2	113.8	134.1	131.7	150.4
Win95	115.2	121.4	115.2	146.4
Pathfinder	–	165.6	167.7	–
Andes	92.01	97.04	410.1	430.8
Diabetes	153	163.8	–	–
Pigs	–	–	–	–
Link	112.5	127.5	–	–