SHIELD-INDUCED PARTITIONING FOR PARALLEL BAYESIAN NETWORK STRUCTURE LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Bayesian network structure learning is computationally challenging because conditional-independence testing and graph search scale poorly with the number of variables. We introduce Shield-Induced Partitioning (SIP), a recursive framework for scalable Bayesian network structure learning. SIP decomposes the variable set into two main blocks and a shield, then separates the shield into a conditioning component that d-separates the blocks and a residual component used during merging. This decomposition enables independent learning on smaller local domains of the form $\mathbf{A}_i \cup \mathbf{C}_0$, followed by a merge step that recovers the remaining shield-mediated dependencies. We provide a seed-and-expand procedure for discovering SIPs and show that the resulting recursive learning procedure recovers the target P-map class under standard Markov and faithfulness assumptions. Experiments on discrete Bayesian-network benchmarks show that an optimized implementation achieves substantial runtime improvements over a broad set of structure-learning baselines while maintaining competitive structural accuracy. These results suggest that shield-induced recursive partitioning is an effective principle for scalable Bayesian network structure learning.

1 INTRODUCTION

Understanding relationships among variables is a central problem in applied statistics Pearl (2009); Hastie et al. (2009). Probabilistic graphical models, particularly Bayesian networks, provide a principled framework by representing probabilistic relationships in a directed acyclic graph (DAG) with variables as nodes and dependencies as edges Pearl (2014). Structure learning in Bayesian networks refers to the task of inferring this DAG from data van den Boom et al. (2023). Existing structure learning approaches are broadly categorized into constraint-based and score-based methods. Constraint-based algorithms, such as the PC algorithm Spirtes et al. (2000), rely on the sufficiency, Markov, and faithfulness assumptions to identify edges through conditional independence (CI) tests. In contrast, score-based methods formulate structure learning as an optimization problem, typically maximizing a likelihood-based criterion such as the Bayesian Information Criterion (BIC) over the space of DAGs.

Exhaustively performing CI tests or optimizing a score over the space of all possible graphs quickly becomes infeasible as the number of variables grows. Pearl (2009); Koller & Friedman (2009). Consequently, exact structure learning is NP-hard for moderate to large networks Ramsey et al. (2017); Peters et al. (2017). A natural approach to improve scalability is to decompose the problem into smaller subproblems that can be solved independently. For example, Gu & Zhou (2020) proposes an approximate distributed framework based on clustering variables, while Talvitie et al. (2019) introduces distributed score-based methods that approximate centralized solutions. Although these approaches improve scalability, they often sacrifice structural exactness or rely on restrictive assumptions such as Gaussianity.

The core component of these distributed approaches is the node-partition strategy. Some methods rely on expert-driven decompositions (Liu et al., 2017; Xie et al., 2006), while others use optimization-based partitioning schemes (Zhang et al., 2020; Hong et al., 2023). The SADA algorithm (Cai et al., 2013) introduces a causal cut-based decomposition given a shared cut set. However, existing decompositions often require high-order CI tests, provide limited control over the conditioning component used for local learning, or trade scalability for structural exactness. This motivates

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

a partitioning principle that explicitly separates the variables used to d-separate subproblems from those retained only for merging.

We propose Shield-Induced Partitioning (SIP), a recursive framework that decomposes the node set into two main blocks and a shield. The shield is divided into a d-separating component $\mathbf{C}_0$ and a residual component $\mathbf{C}_1$, with $\mathbf{A}_1 \perp_{\mathcal{G}} \mathbf{A}_2 \mid \mathbf{C}_0$. This structure allows the local domains $\mathbf{A}_i \cup \mathbf{C}_0$ to be learned independently, while the remaining shield-mediated dependencies are recovered in a merge step. We give a seed-and-expand procedure for discovering SIPs and prove that recursive SIP learning recovers the P-map class under Markov and faithfulness assumptions. We also evaluate an optimized implementation, AdjustedSIP, against a broad suite of Bayesian-network structure learners. The results show that SIP-based decomposition delivers large computational gains while maintaining competitive, though not uniformly superior, structural accuracy.

2 Background

A Bayesian network represents a joint distribution P over a set of random variables $\mathcal{V} = \{X_1, \ldots, X_n\}$ using a directed acyclic graph (DAG) $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. Each node in $\mathcal{V}$ corresponds to a random variable, and the edge set $\mathcal{E}$ encodes the direct probabilistic dependencies among them. In structure learning, the goal is to recover the graphical structure associated with P from data, up to the equivalence class of DAGs that encode the same conditional independence relationships. We write $\mathcal{I}(P)$ for the collection of conditional independence statements that hold in P. To relate these probabilistic independencies to the graph $\mathcal{G}$, we use the standard notions of active trails and d-separation.

Definition 2.1 (an active trail Koller & Friedman (2009))**.** *Consider the DAG $\mathcal{G}$ over the node set $\mathcal{V}$. Let $\mathbf{Z} \subseteq \mathcal{V}$ be the set of observed variables. A trail $X_1 \rightleftharpoons \cdots \rightleftharpoons X_n$ in $\mathcal{G}$ is **active** given $\mathbf{Z}$ if:* (i) *for any collider $X_{i-1} \to X_i \leftarrow X_{i+1}$, meaning a three-node pattern in which two arrows meet head-to-head at the middle node X_i, either X_i or one of its descendants is in $\mathbf{Z}$, and* (ii) *no other node along the trail is in $\mathbf{Z}$. A trail that is not active is called inactive or blocked.*

Definition 2.2 (d-separation Koller & Friedman (2009))**.** *Consider the DAG $\mathcal{G}$ over the node set $\mathcal{V}$. Let $\mathbf{Z} \subseteq \mathcal{V}$ be the set of observed variables. Two subsets of nodes $\mathbf{X}_1, \mathbf{X}_2 \subseteq \mathcal{V}$ are said to be* d-separated *given $\mathbf{Z}$, denoted $\mathbf{X}_1 \perp_{\mathcal{G}} \mathbf{X}_2 \mid \mathbf{Z}$, if there exists no active trail connecting any node $X_i \in \mathbf{X}_1$ to any node $X_j \in \mathbf{X}_2$ conditional on $\mathbf{Z}$.*

Let $\mathcal{I}(\mathcal{G})$ denote the set of all d-separations in a DAG $\mathcal{G}$. We assume that for the distribution P, there exists a DAG $\mathcal{G}$ satisfying the following conditions: (i) *Markovness*, i.e., $\mathcal{I}(\mathcal{G}) \subseteq \mathcal{I}(P)$; and (ii) *faithfulness*, i.e., $\mathcal{I}(P) \subseteq \mathcal{I}(\mathcal{G})$. Under these assumptions, $\mathcal{I}(P) = \mathcal{I}(\mathcal{G})$, meaning that all conditional independencies in P are exactly captured by the d-separations in $\mathcal{G}$, and vice versa. In this case, $\mathcal{G}$ is called a P-map, or perfect map, for P. For a given distribution P, there may be multiple P-maps: different DAGs can encode exactly the same set of conditional independence statements. These DAGs form an independence-equivalent, or I-equivalent, class. This equivalence class is commonly represented by a completed partially directed acyclic graph (CPDAG), in which directed edges have the same orientation in every DAG in the class, while undirected edges correspond to adjacencies whose orientation may vary across equivalent DAGs. Thus, structure learning is often formulated as the problem of recovering the CPDAG representing the P-map class of P.

Throughout, a *P-map learner* denotes any Bayesian-network structure-learning subroutine that, given data on a specified node set, returns a DAG or CPDAG for the corresponding P-map class under Markov and faithfulness assumptions. Applied to $\mathcal{V}$, it is global structure learning; applied to a subset of $\mathcal{V}$, it uses only those variables.

3 Node partitioning

Consider the example in Figure 1. Figure 1(a) shows a P-map over the node set $\mathcal{V} = \{1, 2, \ldots, 11\}$ corresponding to a distribution P. Our goal is to divide $\mathcal{V}$ into smaller subsets that can be learned separately, so that we do not need to learn the entire DAG at once. We begin with a separator in the skeleton of $\mathcal{G}$, where the skeleton is the undirected graph obtained by ignoring edge directions.

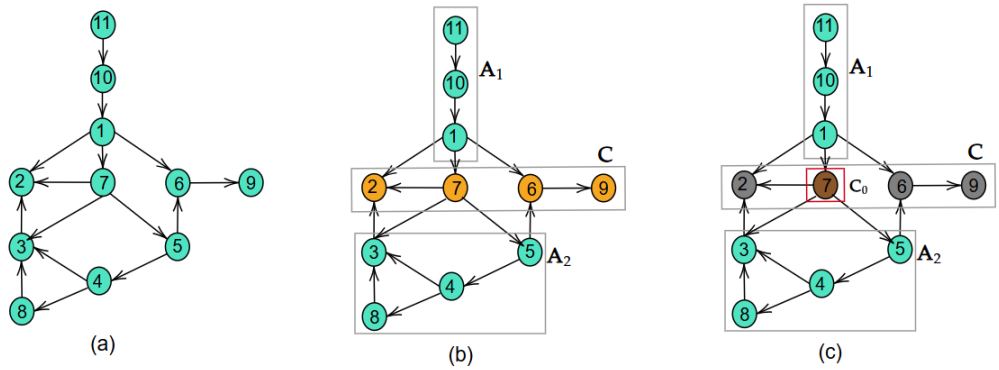

Figure 1: (a) a P-map for $\{1, 2, \ldots, 11\}$. (b) $\mathcal{V} = (\mathbf{A_1}, \mathbf{A_2}, \mathbf{C})$ is a partition of $\mathcal{V}$. Here $\mathbf{A_1} = \{1, 10, 11\}$ and $\mathbf{A_2} = \{3, 4, 5, 8\}$ are the main partitions separated by $\mathbf{C} = \{2, 6, 7, 9\}$. (c) $\mathcal{V} = (\mathbf{A_1}, \mathbf{A_2}, \mathbf{C_0}, \mathbf{C_1})$ is a valid partition of $\mathcal{V}$ such that $\mathbf{A_1} \perp_{\mathcal{G}} \mathbf{A_2} \mid \mathbf{C_0}$ where $\mathbf{C_0} = 7$ and $\mathbf{C_1} = \{2, 6, 9\}$.

Definition 3.1 (Separation)**.** *Let $\mathcal{G}$ be a graph and let $\mathbf{A_1}, \mathbf{A_2}, \mathbf{C} \subseteq \mathcal{V}$ be pairwise disjoint node sets. We say that* $\mathbf{C}$ separates $\mathbf{A_1}$ *and* $\mathbf{A_2}$ *if every trail in $\mathcal{G}$ from a node in $\mathbf{A_1}$ to a node in $\mathbf{A_2}$ contains at least one node in* $\mathbf{C}$*.*

Given such a separator, we form a partition of the node set $\mathcal{V}$: a collection of disjoint node sets $(\mathbf{A}_1, \mathbf{A}_2, \mathbf{C})$ such that $\mathbf{A}_1 \cup \mathbf{A}_2 \cup \mathbf{C} = \mathcal{V}$. Figure 1(b) gives one such partition, where the "main blocks" $\mathbf{A}_1$ and $\mathbf{A}_2$ are separated by $\mathbf{C}$. However, separation alone is not sufficient for our purpose. We seek a partition that supports local structure learning: for each block $\mathbf{A}_i$, applying a P-map learner to the restricted node set $\mathbf{A}_i \cup \mathbf{C}$ should correctly recover all edges with both endpoints in $\mathbf{A}_i$. This motivates the key question addressed next: how to identify a separator $\mathbf{C}$ that makes the resulting blocks $\mathbf{A}_i$ learnable separately?

3.1 A SHIELD

A separator $\mathbf{C}$ is useful for parallel structure learning only if it preserves the information needed to learn each main block locally. For a block $\mathbf{A}_i$, this means that every non-adjacent pair $X, Y \in \mathbf{A}_i$ must admit a d-separating set contained in the local learning domain $\mathbf{A}_i \cup \mathbf{C}$. Equivalently, no node outside $\mathbf{A}_i \cup \mathbf{C}$ should be needed to certify non-adjacencies within $\mathbf{A}_i$. We formalize this property by calling such a set $\mathbf{C}$ a *shield*.

Definition 3.2 (Shield)**.** *Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a DAG, and let $\mathbf{A}, \mathbf{C} \subseteq \mathcal{V}$ be disjoint subsets. We say that* $\mathbf{C}$ *is a* shield *for* $\mathbf{A}$ *if for every non-adjacent pair $X, Y \in \mathbf{A}$, there exists a conditioning set $\mathbf{Z} \subseteq (\mathbf{A} \cup \mathbf{C}) \setminus \{X, Y\}$ such that $X \perp_{\mathcal{G}} Y \mid \mathbf{Z}$.*

For example, in Figure 1(b), let $\mathbf{A}_2 = \{3, 4, 5, 8\}$ and $\mathbf{C} = \{2, 6, 7, 9\}$. The local domain $\mathbf{A}_2 \cup \mathbf{C}$ contains d-separating sets for non-adjacent pairs within $\mathbf{A}_2$, such as $5 \perp_{\mathcal{G}} 8 \mid \{4\}$ and $3 \perp_{\mathcal{G}} 5 \mid \{4, 7\}$. Thus, applying a P-map learner to $\mathbf{A}_2 \cup \mathbf{C}$ can recover the intra-block structure over $\mathbf{A}_2$ without using nodes outside this local domain.

A shield may still be larger than necessary. We therefore look for a smaller conditioning component $\mathbf{C}_0 \subseteq \mathbf{C}$ that d-separates the two main blocks. Conditioning on $\mathbf{C}_0$ blocks all active trails between $\mathbf{A}_1$ and $\mathbf{A}_2$, so that

$$\mathbf{A}_1 \perp_{\mathcal{G}} \mathbf{A}_2 \mid \mathbf{C}_0. \tag{1}$$

The remaining shield nodes form the residual component $\mathbf{C}_1 = \mathbf{C} \setminus \mathbf{C}_0$. These nodes are not used to d-separate the two main blocks; instead, they are retained for the merge stage, where the remaining shield-mediated dependencies are learned. In the SIPs considered here, conditioning on this residual component does not d-separate the two main blocks:

$$\mathbf{A}_1 \not\perp_{\mathcal{G}} \mathbf{A}_2 \mid \mathbf{C}_1. \tag{2}$$

Equations 1 and 2 motivate the following definition.

Definition 3.3 (Shield-Induced Partition (SIP))**.** *Let $\mathcal{G}$ be a DAG over the node set $\mathcal{V}$. A partition $\mathcal{V} = \mathbf{A}_1 \cup \mathbf{A}_2 \cup \mathbf{C}_0 \cup \mathbf{C}_1$ is called a* Shield-Induced Partition *(SIP) if $\mathbf{A}_1 \perp_{\mathcal{G}} \mathbf{A}_2 \mid \mathbf{C}_0$, and $\mathbf{A}_1 \not\perp_{\mathcal{G}} \mathbf{A}_2 \mid \mathbf{C}_0 \cup \{C\}$ for all $C \in \mathbf{C}_1$.*

The key consequence of a SIP is that the conditioning component $\mathbf{C}_0$ is sufficient for independent local learning. Since $\mathbf{C}_0$ d-separates the two main blocks, the learner does not need variables from the opposite block to certify non-adjacencies inside a block. More precisely, for each main block $\mathbf{A}_i$, non-adjacent pairs involving a node in $\mathbf{A}_i$ and another node in $\mathbf{A}_i \cup \mathbf{C}_0$ can be separated using only variables in the local domain $\mathbf{A}_i \cup \mathbf{C}_0$. This gives the independent-learning property below.

Theorem 3.1 (Independent Learning)**.** *Let $\mathcal{G}$ be a DAG over the node set $\mathcal{V}$, and suppose that $\mathcal{V} = \mathbf{A}_1 \cup \mathbf{A}_2 \cup \mathbf{C}_0 \cup \mathbf{C}_1$ forms a SIP. For each $i \in \{1, 2\}$, if $X \in \mathbf{A}_i$ and $Y \in \mathbf{A}_i \cup \mathbf{C}_0$ are non-adjacent in $\mathcal{G}$, then there exists a conditioning set $\mathbf{Z} \subseteq (\mathbf{A}_i \cup \mathbf{C}_0) \setminus \{X, Y\}$ such that $X \perp_{\mathcal{G}} Y \mid \mathbf{Z}$.*

Proof. We prove the claim for $\mathbf{A}_1$; the argument for $\mathbf{A}_2$ is symmetric.

First, let $X, Z \in \mathbf{A}_1$ be non-adjacent in $\mathcal{G}$. Suppose, for contradiction, that there is no conditioning set contained in $(\mathbf{A}_1 \cup \mathbf{C}_0) \setminus \{X, Z\}$ that d-separates X and Z. Then any d-separation of X and Z must require at least one node outside the local domain $\mathbf{A}_1 \cup \mathbf{C}_0$. Since $\mathbf{C}_0$ already d-separates $\mathbf{A}_1$ and $\mathbf{A}_2$, such a required node must lie in the residual component $\mathbf{C}_1$. Let $u \in \mathbf{C}_1$ be such a node.

Thus, there exists an active trail π between X and Z whose local blocking requires u. Because $u \notin \mathbf{A}_1 \cup \mathbf{C}_0$, the local learner cannot condition in u. Hence, on the active trail π, the node u must appear as a non-collider; otherwise the trail would be blocked at u. Therefore, at least one of the two segments of π from u to X or from u to Z meets u through an edge having a tail at u. Let $R \in \{X, Z\}$ denote the corresponding endpoint, and let π_{Ru} denote this active segment.

Now, because $u \in \mathbf{C}_1$, the second SIP condition gives $\mathbf{A}_1 \not\perp_{\mathcal{G}} \mathbf{A}_2 \mid \mathbf{C}_0 \cup \{u\}$. Consequently, adding u to the conditioning set activates at least one trail ρ between some $K_1 \in \mathbf{A}_1$ and some $K_2 \in \mathbf{A}_2$ that is blocked given $\mathbf{C}_0$ but active given $\mathbf{C}_0 \cup \{u\}$. Since conditioning on an additional node cannot activate a trail through a non-collider, the newly activated trail ρ must contain u as a collider or a descendant of a collider. Let ρ_{uK_2} denote the segment of the activated trail from u to K_2.

Now concatenate the segment π_{Ru} with the segment ρ_{uK_2}. This gives a trail from $R \in \mathbf{A}_1$ to $K_2 \in \mathbf{A}_2$. At the joining node u, the concatenated trail is not blocked: one segment π_{Ru} meets u through an edge with a tail at u, so u is not a collider on that concatenated trail. All other nodes on the concatenated trail retain their active status. Therefore, this concatenated trail is active given $\mathbf{C_0}$.

This contradicts the SIP condition $\mathbf{A}_1 \perp_{\mathcal{G}} \mathbf{A}_2 \mid \mathbf{C}_0$. Hence, no residual node in $u \in \mathbf{C}_1$ can be required to d-separate a non-adjacent pair inside $\mathbf{A}_1$. Therefore, for every non-adjacent pair $X, Z \in \mathbf{A}_1$, there exists a conditioning set $\mathbf{Z} \subseteq (\mathbf{A}_1 \cup \mathbf{C}_0) \setminus \{X, Z\}$ such that $X \perp_{\mathcal{G}} Z \mid \mathbf{Z}$.

Next, consider a non-adjacent pair $X \in \mathbf{A}_1$ and $Y \in \mathbf{C}_0$. Then Y is a descendant of u defined above. Thus, conditioning on Y activates the path between K_1 and K_2 that goes via u. Given that this path does not intersect with $\mathbf{C}_0$, it follows that $K_1 \not\perp K_2 \mid \mathbf{C}_0$. Hence, $\mathbf{A}_1 \not\perp \mathbf{A}_2 \mid \mathbf{C}_0$, which is again a contradiction, completing the proof.

□

According to Theorem 3.1, $\mathbf{C}_0$ is a shield for both $\mathbf{A}_1$ and $\mathbf{A}_2$. The Theorem actually makes a stronger statement by letting one of the endpoints to be in $\mathbf{C}_0$.

Given a SIP, structure learning proceeds in two stages. First, the relation $\mathbf{A}_1 \perp_{\mathcal{G}} \mathbf{A}_2 \mid \mathbf{C}_0$ acts as a structural prior on the skeleton: candidate edges between nodes in $\mathbf{A}_1$ and nodes in $\mathbf{A}_2$ need not be considered. Second, by Theorem 3.1, edges internal to each block $\mathbf{A}_i$, as well as edges between $\mathbf{A}_i$ and $\mathbf{C}_0$, can be recovered by applying a structure learner to the local domain $\mathbf{A}_i \cup \mathbf{C}_0$. The remaining edges, those between $\mathbf{A}_i$ and $\mathbf{C}_1$, and those internal to the shield $\mathbf{C} = \mathbf{C}_0 \cup \mathbf{C}_1$, are then recovered in a merge step using the full SIP domain $\mathbf{A}_1 \cup \mathbf{A}_2 \cup \mathbf{C}_0 \cup \mathbf{C}_1$.

Figure 2 illustrates the local-learning stage for the partition in Figure 1(c), where $\mathbf{A}_1 = \{1, 10, 11\}$, $\mathbf{A}_2 = \{3, 4, 5, 8\}$, $\mathbf{C}_0 = \{7\}$, and $\mathbf{C}_1 = \{2, 6, 9\}$. For each main block $\mathbf{A}_i$, the learner is applied

only to $\mathbf{A}_i \cup \mathbf{C}_0$. The resulting local graphs are then combined and extended to recover the remaining shield-mediated dependencies, as shown in Figure 3.

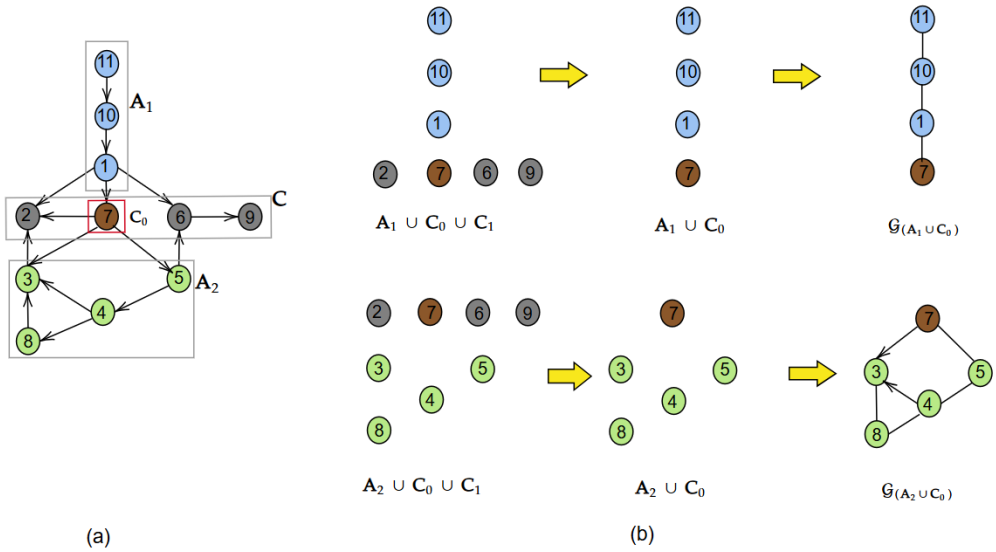

Figure 2: Independent learning under a SIP. For each main block $\mathbf{A}_i$, edges within $\mathbf{A}_i$ and edges between $\mathbf{A}_i$ and $\mathbf{C}_0$ are learned using only the local domain $\mathbf{A}_i \cup \mathbf{C}_0$.

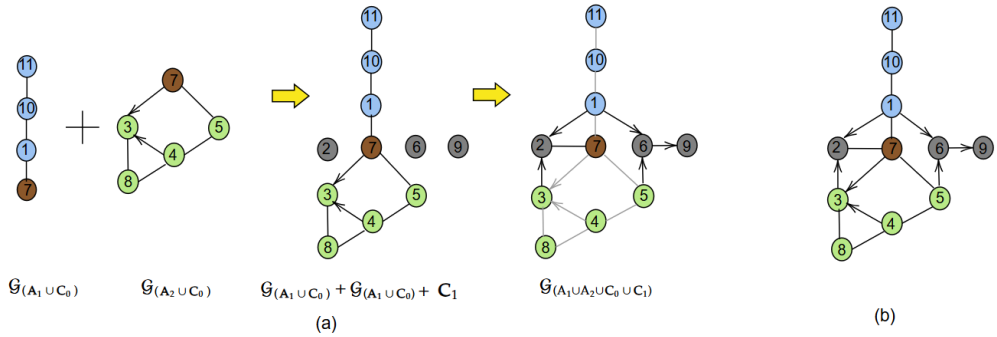

Figure 3: Merging the locally learned substructures. After local learning on $\mathbf{A}_1 \cup \mathbf{C}_0$ and $\mathbf{A}_2 \cup \mathbf{C}_0$, the learning domain is extended to $\mathbf{A}_1 \cup \mathbf{A}_2 \cup \mathbf{C}_0 \cup \mathbf{C}_1$ to recover edges involving $\mathbf{C}_1$ and edges internal to the shield.

Algorithm 1 summarizes this SIP-learning procedure. The algorithm does not create edges between the two main blocks. It learns the local domains $\mathbf{A}_i \cup \mathbf{C}_0$ independently, and finally performs the merge step to recover the remaining edges involving the residual shield component. In Algorithm 1, CI-PRUNE$(\mathbf{U}, \mathcal{Q}, \mathcal{S})$ denotes the standard CI-pruning step restricted to candidate pairs $\mathcal{Q}$: it initializes a graph over $\mathbf{U}$ with candidate edge set $\mathcal{Q}$, tests each $\{X, Y\} \in \mathcal{Q}$ using conditioning sets contained in $\mathbf{U} \setminus \{X, Y\}$, removes conditionally independent pairs, and stores the corresponding separating sets in $\mathcal{S}$.

Thus, once a SIP is available, learning can be localized to smaller domains and then completed by a merge step over the full SIP domain. The remaining challenge is to construct such a Shield-Induced Partition, which is addressed next.

3.2 SHIELD-INDUCED PARTITION (SIP) DISCOVERY

The previous subsection assumes that a SIP is available. We now describe how to construct one from data. Given a node set $\mathcal{V}$, the goal is to find disjoint subsets $(\mathbf{A}_1, \mathbf{A}_2, \mathbf{C}_0, \mathbf{C}_1)$ such that

Algorithm 1 SIP-Learning

Require: SIP $(\mathbf{A}_1, \mathbf{A}_2, \mathbf{C}_0, \mathbf{C}_1)$, data $\mathcal{D}$, CI storage $\mathcal{S}$
Ensure: PDAG over $\mathbf{A}_1 \cup \mathbf{A}_2 \cup \mathbf{C}_0 \cup \mathbf{C}_1$
$\mathbf{U} \leftarrow \mathbf{A}_1 \cup \mathbf{A}_2 \cup \mathbf{C}_0 \cup \mathbf{C}_1$
▷ Phase 1: independent local learning
for $i = 1, 2$ **in parallel do**
  $\mathbf{U}_i \leftarrow \mathbf{A}_i \cup \mathbf{C}_0$
  $\mathcal{Q}_i \leftarrow \{\{X, Y\} : X \in \mathbf{A}_i,\ Y \in \mathbf{A}_i \cup \mathbf{C}_0,\ X \neq Y\}$
  $(\widehat{\mathcal{G}}_i, \mathcal{S}) \leftarrow$ CI-PRUNE$(\mathbf{U}_i, \mathcal{Q}_i, \mathcal{S})$
end for
▷ Phase 2: merge
$\mathcal{Q}_{\text{merge}} \leftarrow \{\{X, Y\} : X \in \mathbf{A}_1 \cup \mathbf{A}_2,\ Y \in \mathbf{C}_1\} \cup \{\{X, Y\} : X, Y \in \mathbf{C}_0 \cup \mathbf{C}_1,\ X \neq Y\}$
$(\widehat{\mathcal{G}}_{\text{merge}}, \mathcal{S}) \leftarrow$ CI-PRUNE$(\mathbf{U}, \mathcal{Q}_{\text{merge}}, \mathcal{S})$
$\widehat{\mathcal{G}} \leftarrow \widehat{\mathcal{G}}_1 \cup \widehat{\mathcal{G}}_2 \cup \widehat{\mathcal{G}}_{\text{merge}}$
▷ Phase 3: orientation
Orient unshielded colliders using $\mathcal{S}$, then apply the standard CPDAG orientation rules
return $\widehat{\mathcal{G}}$

Algorithm 2 SIP Discovery

Require: Node set $\mathcal{V}$, data $\mathcal{D}$
Ensure: SIP $(\mathbf{A_1}, \mathbf{A_2}, \mathbf{C_0}, \mathbf{C_1})$
Find X, Y and minimal $\mathbf{Z} \subseteq \mathcal{V} \setminus \{X, Y\}$ such that $X \perp Y \mid \mathbf{Z}$ ▷ Seed creation
$\mathbf{A_1} \leftarrow \{X\},\ \mathbf{A_2} \leftarrow \{Y\},\ \mathbf{C_0} \leftarrow \mathbf{Z},\ \mathbf{C_1} \leftarrow \emptyset$
for all $V \in \mathcal{V} \setminus (\{X, Y\} \cup \mathbf{Z})$ **do** ▷ Seed expansion
  $\mathbf{A_1} \leftarrow \mathbf{A_1} \cup \{V\}$ if $\forall U \in \mathbf{A_2} : V \perp U \mid \mathbf{C_0}$
  $\mathbf{A_2} \leftarrow \mathbf{A_2} \cup \{V\}$ if $\big(\exists U \in \mathbf{A_2} : V \not\perp U \mid \mathbf{C_0}\big)$ and $\big(\forall U \in \mathbf{A_1} : V \perp U \mid \mathbf{C_0}\big)$
  $\mathbf{C_1} \leftarrow \mathbf{C_1} \cup \{V\}$ otherwise
end for
return $(\mathbf{A_1}, \mathbf{A_2}, \mathbf{C_0}, \mathbf{C_1})$

$\mathbf{A}_1 \perp_{\mathcal{G}} \mathbf{A}_2 \mid \mathbf{C}_0$, and $\mathbf{A}_1 \not\perp_{\mathcal{G}} \mathbf{A}_2 \mid \mathbf{C}_0 \cup \{C\}$ for all $C \in \mathbf{C}_1$. At the same time, the partition should be useful computationally: the main blocks $\mathbf{A}_1$ and $\mathbf{A}_2$ should be large enough to enable substantial independent learning, while $\mathbf{C}_0 \cup \mathbf{C}_1$ should remain small enough to keep the merge step efficient.

A brute-force approach would enumerate candidate four-way partitions of $\mathcal{V}$ and test whether each candidate satisfies the required d-separation relation. As discussed in Appendix A, this search space grows exponentially in $|\mathcal{V}|$, making exhaustive discovery infeasible except for very small graphs.

We therefore use a seed-and-expand strategy. The algorithm first identify a seed conditional independence relation $X \perp Y \mid \mathbf{Z}$, where $X, Y \in \mathcal{V}$ and $\mathbf{Z} \subseteq \mathcal{V} \setminus \{X, Y\}$. The partition is initialized as $\mathbf{A}_1 = \{X\}$, $\mathbf{A}_2 = \{Y\}$, $\mathbf{C}_0 = \mathbf{Z}$, and $\mathbf{C}_1 = \emptyset$. The remaining nodes are then assigned according to conditional dependence tests. The discovery compares each candidate against the evolving blocks $\mathbf{A}_1$ and $\mathbf{A}_2$, requiring up to $O(n^2 2^{n-2})$ CI tests, for $n = |\mathcal{V}|$.

Proposition 3.1 (Validity of SIP Discovery)**.** *Algorithm 2 outputs a SIP.*

The proof is provided in the appendix. Algorithm 2 uses the evolving blocks to check the conditional dependencies. A candidate node is assigned to $\mathbf{A}_1$ if it is conditionally independent of the current block $\mathbf{A}_2$ given $\mathbf{C}_0$, and symmetrically it is assigned to $\mathbf{A}_2$ if it is conditionally independent of $\mathbf{A}_1$ while remaining dependent on at least one node in $\mathbf{A}_2$. Nodes that cannot be assigned to either main block without violating this cross-block separation are placed in $\mathbf{C}_1$.

3.3 SHIELD-INDUCED PARTITION (SIP) ALGORITHM

Once a SIP has been discovered, we use it recursively to obtain smaller learning domains. The recursion must preserve the conditional-independence guarantee in Definition 3.3. For this reason, recursive decomposition is applied only to the local domains $\mathbf{A}_i \cup \mathbf{C}_0$. The residual component

Table 1: Recursive SIP formation for the node set in Figure 1(a). Colors indicate $\mathbf{A}_1$ in blue, $\mathbf{A}_2$ in green, $\mathbf{C}_0$ in red, and $\mathbf{C}_1$ in gray.

Depth	Node set	SIP-discovery seed	Formed SIP	Recursive domains
1	$\{1, 10, 11, 3, 4, 5, 8, 7, 2, 6, 9\}$	$1 \perp 4 \mid \{7\}$	$(\{1, 10, 11\}, \{3, 4, 5, 8\}, \{7\}, \{2, 6, 9\})$	$\{1, 10, 11, 7\}, \{3, 4, 5, 8, 7\}$
2	$\{10, 11, 7, 1\}$	$10 \perp 7 \mid \{1\}$	$(\{10, 11\}, \{7\}, \{1\}, \emptyset)$	$\{10, 11, 1\}, \{7, 1\}$
2	$\{4, 8, 7, 5, 3\}$	$4 \perp 7 \mid \{5\}$	$(\{4, 8\}, \{7\}, \{5\}, \{3\})$	$\{4, 8, 5\}, \{7, 5\}$

Algorithm 3 Recursive SIP Algorithm

Require: Node set $\mathcal{V}$, data $\mathcal{D}$, threshold τ
Ensure: PDAG $\widehat{\mathcal{G}}$

Initialize $\widehat{\mathcal{G}}$ as an empty PDAG over $\mathcal{V}$
$\mathcal{S} \leftarrow \emptyset$ ▷ stored CI relations and separating sets
$\mathcal{R} \leftarrow \emptyset$ ▷ learned domains
function DECOMPOSETOPDOWN($\mathbf{U}$)
  $(\mathbf{A}_1, \mathbf{A}_2, \mathbf{C}_0, \mathbf{C}_1) \leftarrow$ SIP-DISCOVERY($\mathbf{U}, \mathcal{D}$)
  Create a tree node T storing $(\mathbf{A}_1, \mathbf{A}_2, \mathbf{C}_0, \mathbf{C}_1)$
  for $i = 1, 2$ **do**
    $\mathbf{U}_i \leftarrow \mathbf{A}_i \cup \mathbf{C}_0$
    if $|\mathbf{U}_i| > \tau$ and $|\mathbf{U}_i| < |\mathbf{U}|$ **then**
      Add DECOMPOSETOPDOWN($\mathbf{U}_i$) as a child of T
    end if
  end for
  return T
end function
function LEARNBOTTOMUP(T)
  for each child T' of T **do**
    LEARNBOTTOMUP(T')
  end for
  Let $(\mathbf{A}_1, \mathbf{A}_2, \mathbf{C}_0, \mathbf{C}_1)$ be the SIP stored at T
  $\mathbf{U}_T \leftarrow \mathbf{A}_1 \cup \mathbf{A}_2 \cup \mathbf{C}_0 \cup \mathbf{C}_1$
  $\widehat{\mathcal{G}}_T \leftarrow$ SIP-LEARNING($(\mathbf{A}_1, \mathbf{A}_2, \mathbf{C}_0, \mathbf{C}_1), \mathcal{D}, \mathcal{S}$), reusing any learned subgraph $\widehat{\mathcal{G}}[\mathbf{U}_i]$ with $\mathbf{U}_i \in \mathcal{R}$
  Update $\widehat{\mathcal{G}}$ with $\widehat{\mathcal{G}}_T$
  Add $\mathbf{U}_T$ to $\mathcal{R}$
end function
$T_{\text{root}} \leftarrow$ DECOMPOSETOPDOWN($\mathcal{V}$)
LEARNBOTTOMUP(T_{root})
return $\widehat{\mathcal{G}}$

$\mathbf{C}_1$ is not recursively split: it is retained for the merge step, where edges involving $\mathbf{C}_1$ and edges internal to $\mathbf{C}_0 \cup \mathbf{C}_1$ are recovered. Table 1 illustrates this recursive construction for the example in Figure 1. At each depth, SIP discovery produces a partition, and the next recursive calls are made only on the two domains $\mathbf{A}_1 \cup \mathbf{C}_0$ and $\mathbf{A}_2 \cup \mathbf{C}_0$.

The complete recursive procedure is given in Algorithm 3. The algorithm first builds a decomposition tree top-down by repeatedly applying SIP discovery. It then performs learning bottom-up: children are learned before their parent, so that previously learned induced subgraphs can be reused whenever a local domain $\mathbf{A}_i \cup \mathbf{C}_0$ has already been processed. Each bottom-up call uses Algorithm 1 to learn the corresponding SIP domain and merge the local structures into the global PDAG. After the decomposition tree has been constructed, Algorithm 3 learns the leaf domains first and then merges upward. At each internal node, the two local domains $\mathbf{A}_1 \cup \mathbf{C}_0$ and $\mathbf{A}_2 \cup \mathbf{C}_0$ have either already been learned recursively or are learned directly using Algorithm 1. The global graph $\widehat{\mathcal{G}}$ is updated after each merge, and the final output is a PDAG representing the recovered Markov-equivalence class. (See Appendix C for an illustration of Algorithm 3 applied to the example in Figure 1.)

Theorem 3.2. *Algorithm 3 outputs the P-map class PDAG for the distribution P.*

4 EXPERIMENTAL RESULTS

We evaluate AdjustedSIP on a discrete Bayesian-network benchmark suite with known ground-truth structures, enabling direct comparison between learned and target graphs. Experiments use $n \in \{100, 1000, 10000\}$ samples, generated from *bnlearn* package, across networks ranging from small benchmarks such as CANCER, EARTHQUAKE, SURVEY, and ASIA, to medium and large networks including CHILD, INSURANCE, ALARM, HEPAR2, WIN95PTS, ANDES, DIABETES, PIGS, LINK, and MUNIN. Network sizes and sample settings are listed in Appendix Table 3.

The implementation used in these experiments follows the same shield-induced decomposition principle as the SIP procedure in Algorithms 1–3, but it uses a more efficient execution strategy. We refer to this efficient version of the algorithm as AdjustedSIP, explained below. First, partition discovery uses the active SIP expansion rule: after identifying a low-order seed $X \perp Y \mid \mathbf{C_0}$, remaining variables are assigned to one of the two main blocks if independent of the opposite block given $\mathbf{C_0}$, and otherwise placed in the residual shield $\mathbf{C_1}$. In the benchmark preset, seed discovery is limited to conditioning sets of size at most one.

Second, the skeleton stage uses a PC-stable-style depth sweep over SIP-specific candidate-pair groups. Local stages learn edges in $\mathbf{A_1} \cup \mathbf{C_0}$ and $\mathbf{A_2} \cup \mathbf{C_0}$; merge stages learn edges between $\mathbf{A_i}$ and $\mathbf{C_1}$ and within $\mathbf{C_0} \cup \mathbf{C_1}$. Edges between $\mathbf{A_1}$ and $\mathbf{A_2}$ are removed immediately using separating set $\mathbf{C_0}$. No conditioning-set cap was used for the reported local side-learning stages.

Third, adjustedSIP also parallelizes CI testing within each conditioning-depth sweep and runs node-disjoint local SIP subproblems concurrently when possible. The benchmark used all cores assigned by the scheduler, $n_{\text{jobs}} = -1$, with four cores requested for AdjustedSIP.

Finally, all CI decisions use the Pearson chi-squared test with $\alpha = 0.05$, matching the categorical CI test used by the other constraint-based baselines. CI queries and separating sets are cached globally, and pairwise tests sharing a conditioning set are vectorized. Thus, AdjustedSIP differs from the scalar SIP reference mainly in scheduling, caching, vectorization, and parallelism, not in the underlying independence test.

We evaluate the SIP algorithm against 25 structure-learning algorithms: ABC ((Ji et al., 2013)), ARGES ((Nandy et al., 2018)), astar ((Yuan et al., 2011)), CB ((Singh & Valtorta, 1993)), COS ((Perrier et al., 2008)), CPBayes ((Van Beek & Hoffmann, 2015)), ConservativePC ((Ramsey et al., 2006)), ConsistentPC ((Li et al., 2019)), GES ((Chickering, 2002)), XGES ((Nazaret & Blei, 2024)), BOSS ((Andrews et al., 2023)), DAS ((Montagna et al., 2023)), GSP ((Solus et al., 2021)), Grow-Shrink ((Margaritis & Thrun, 1999)), H2PC ((Gasse et al., 2014)), HEA ((Wong & Leung, 2004)), HillClimbing ((Koller & Friedman, 2009)), K2Improved ((Behjati & Beigy, 2020)), MC3 ((Madigan et al., 1995)), MMHC ((Tsamardinos et al., 2006)), MMPC ((Tsamardinos et al., 2003)), OBS ((Teyssier & Koller, 2005)), PC ((Spirtes et al., 2000)), PCStable ((Colombo & Maathuis, 2014)), ICDCPDAG ((Rohekar et al., 2021); assuming no latent variables), SC ((Friedman et al., 2013)), WINASOBS ((Scanagatta et al., 2017)). We report wall-clock runtime and normalized structural Hamming distance (nSHD), where lower nSHD indicates a learned graph closer to the ground truth. Full runtime and nSHD tables are reported in Appendix Tables 4– 6 and 7– 9. Cells marked as unavailable are excluded from paired statistical tests; completed 24-hour timeouts are retained as 86400 seconds.

Because runtimes and graph errors are highly skewed, we use paired one-sided Wilcoxon signed-rank tests over matched algorithm-network cells. For both runtime and nSHD, lower values are better, so each paired difference is defined as "competitor minus AdjustedSIP". A positive difference therefore indicates that AdjustedSIP is better on that matched cell. We perform the test overall and separately for each sample size.

Results. AdjustedSIP is substantially faster than the other algorithms. Across all matched runtime cells, AdjustedSIP is faster in 969 of 986 comparisons, with a median speedup of $263.7\times$ and a one-sided Wilcoxon $p = 6.64 \times 10^{-159}$. The speed advantage is significant at every sample size: $196.7\times$, $308.2\times$, and $396.6\times$ median speedups for $n = 100$, 1000, and 10000, respectively. The by-network runtime ranks show the same pattern: AdjustedSIP has median rank one for every sample-size regime and is the fastest available method on 11/15, 14/14, and 10/12 networks for $n = 100$, 1000, and 10000, respectively.

Table 2: Matched-cell significance summary for AdjustedSIP versus other baselines. Runtime and SHD percent are both lower-is-better.

Metric	Samples	Matched cells	Wins/losses/ties	Median improvement	p-value
Runtime	overall	986	969/16/1	263.7× speedup	6.64×10^{-159}
Runtime	100	342	329/12/1	196.7× speedup	2.83×10^{-53}
Runtime	1000	346	346/0/0	308.2× speedup	9.03×10^{-59}
Runtime	10000	298	294/4/0	396.6× speedup	1.14×10^{-50}
SHD %	overall	800	454/243/103	13.4% reduction	1.57×10^{-34}
SHD %	100	299	158/93/48	3.4% reduction	7.85×10^{-12}
SHD %	1000	269	168/83/18	17.2% reduction	1.56×10^{-15}
SHD %	10000	232	128/67/37	28.6% reduction	5.67×10^{-11}

The structural accuracy results are more nuanced. Using SHD percent as the accuracy metric, AdjustedSIP has lower error than the competing method in 454 matched cells, higher error in 243 cells, and tied error in 103 cells. The overall paired test is significant ($p = 1.57 \times 10^{-34}$), with a median relative SHD-percent reduction of 13.4%. The pooled advantage remains significant within every sample-size regime, with median relative reductions of 3.4%, 17.2%, and 28.6% for $n = 100$, 1000, and 10000, respectively. However, the full appendix tables show that AdjustedSIP is not uniformly the most accurate method on individual networks. In particular, pairwise comparisons against strong PC-family baselines such as PC, PC-stable, ICD-CPDAG, and dual-PC do not show a significant SHD-percent improvement and are often unfavorable to AdjustedSIP. Thus, the corrected results support AdjustedSIP primarily as a substantially faster learner with competitive, but not uniformly superior, structural accuracy.

5 Discussion

The benchmark results show that AdjustedSIP substantially improves computational efficiency while maintaining competitive structural accuracy. Across matched runtime comparisons, AdjustedSIP is consistently faster than the competing methods, with statistically significant speedups overall and within each sample-size regime. The per-network runtime rankings show the same pattern, indicating that the speed advantage is not driven by a small number of isolated cases.

The accuracy results are more nuanced. AdjustedSIP more often reduces normalized SHD than increases it, and the aggregate paired tests show statistically significant improvements across all sample-size regimes. However, the appendix tables also show that AdjustedSIP is not uniformly the most accurate method on individual networks. In particular, comparisons against strong PC-family baselines can be unfavorable. Thus, the results support AdjustedSIP primarily as a substantially faster learner with competitive, and sometimes improved, structural accuracy.

The main source of the runtime improvement is the combination of decomposition and execution strategy. SIP removes candidate edges between $\mathbf{A}_1$ and $\mathbf{A}_2$ once $\mathbf{C}_0$ is identified, learns the local domains $\mathbf{A}_i \cup \mathbf{C}_0$ independently, and reserves only shield-mediated dependencies for the merge step. AdjustedSIP implements these operations using PC-stable-style depth sweeps, memoized CI queries, vectorized batches of chi-squared tests, and parallel execution. Since the same Pearson chi-squared test with $\alpha = 0.05$ is used throughout, the improvement is mainly due to how CI queries are organized, reused, and distributed across smaller subproblems.

These benefits depend on the quality of the discovered partition. When the shield is small and the seed separation is well aligned with the graph structure, SIP can substantially reduce the effective learning problem. When the shield is large or useful low-order separating sets are difficult to find, the merge stage becomes more costly and the benefits of decomposition may be reduced.

Overall, the experiments support shield-induced partitioning as an effective principle for scalable Bayesian network structure learning. Future work should isolate the contributions of partition quality, caching, vectorization, and parallel scheduling through ablation studies, and should further investigate settings in which useful SIPs are difficult to discover or require higher-order conditioning sets.

AI USE STATEMENT

In this work, we used generative AI tools to assist with research execution, primarily for code development and debugging. AI-assisted code was reviewed and tested by the authors, and computational experiments were executed using Digital Research Alliance of Canada infrastructure. We also used generative AI tools to assist with the discovery of related literature and subsequently verified the identified references and their relevance against the original sources. Additionally, generative AI was used to improve the readability of the manuscript, including grammar, wording, and clarity.

Generative AI was not used to develop the research ideas or methodology, formulate or prove mathematical claims, generate synthetic datasets, or draw the scientific conclusions presented in this work. The authors reviewed and verified all AI-assisted work and take responsibility for the final content of this work, including text, claims, code and other artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We have taken several steps to support the reproducibility of this work. The assumptions underlying the theoretical results are stated in the main text and proofs are provided for all theoretical claims, with the proof for Theorem3.1 included in the main text and the proofs for Proposition3.1 and Theorem3.2 provided in AppendixD. The experimental setup, including the algorithms, parameter settings and benchmark networks, is described in the main text. The datasets used for each benchmark network were generated using the `bnlearn` package, as specified in the Experimental section. In addition, the appendix provides an illustration of the recursive SIP procedure and supplementary experimental results to further clarify the proposed method and support reproduction of the reported findings.

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. *Advances in neural information processing systems*, 36:63945–63956, 2023.

Shahab Behjati and Hamid Beigy. Improved k2 algorithm for bayesian network structure learning. *Engineering Applications of Artificial Intelligence*, 91:103617, 2020.

Ruichu Cai, Zhenjie Zhang, and Zhifeng Hao. Sada: A general framework to support robust causation discovery. In *International conference on machine learning*, pp. 208–216. PMLR, 2013.

David Maxwell Chickering. Optimal structure identification with greedy search. *Journal of machine learning research*, 3(Nov):507–554, 2002.

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

Nir Friedman, Iftach Nachman, and Dana Pe'er. Learning bayesian network structure from massive datasets: The" sparse candidate" algorithm. *arXiv preprint arXiv:1301.6696*, 2013.

Maxime Gasse, Alex Aussem, and Haytham Elghazel. A hybrid algorithm for bayesian network structure learning with application to multi-label learning. *Expert Systems with Applications*, 41 (15):6755–6772, 2014.

Jiaying Gu and Qing Zhou. Learning big gaussian bayesian networks: Partition, estimation and fusion. *Journal of machine learning research*, 21(158):1–31, 2020.

Trevor Hastie, Robert Tibshirani, and Jerome H Friedman. *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer, 2009.

Yinghan Hong, Junping Guo, Guizhen Mai, Yingqing Lin, Hao Zhang, Zhifeng Hao, and Gengzhong Zheng. High-dimensional causal discovery based on heuristic causal partitioning. *Applied Intelligence*, 53(20):23768–23796, 2023.

Junzhong Ji, Hongkai Wei, and Chunnian Liu. An artificial bee colony algorithm for learning bayesian networks. *Soft Computing*, 17(6):983–994, 2013.

Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

Honghao Li, Vincent Cabeli, Nadir Sella, and Hervé Isambert. Constraint-based causal structure learning with consistent separating sets. *Advances in neural information processing systems*, 32, 2019.

Hui Liu, Shuigeng Zhou, Wai Lam, and Jihong Guan. A new hybrid method for learning bayesian networks: Separation and reunion. *Knowledge-Based Systems*, 121:185–197, 2017.

David Madigan, Jeremy York, and Denis Allard. Bayesian graphical models for discrete data. *International statistical review/revue internationale de statistique*, pp. 215–232, 1995.

Dimitris Margaritis and Sebastian Thrun. Bayesian network induction via local neighborhoods. *Advances in neural information processing systems*, 12, 1999.

Francesco Montagna, Nicoletta Noceti, Lorenzo Rosasco, Kun Zhang, and Francesco Locatello. Scalable causal discovery with score matching. In *Conference on Causal Learning and Reasoning*, pp. 752–771. PMLR, 2023.

Preetam 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 Blei. Extremely greedy equivalence search. In *Proceedings of the Fortieth Conference on Uncertainty in Artificial Intelligence*, volume 244 of *Proceedings of Machine Learning Research*, pp. 2716–2745. PMLR, 2024.

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

Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Elsevier, 2014.

Eric Perrier, Seiya Imoto, and Satoru Miyano. Finding optimal bayesian network given a super-structure. *Journal of Machine Learning Research*, 9(10), 2008.

Jonas Peters, Dominik Janzing, and Bernhard Schölkopf. *Elements of causal inference: foundations and learning algorithms*. The MIT press, 2017.

Joseph Ramsey, Madelyn Glymour, Ruben Sanchez-Romero, and Clark Glymour. A million variables and more: the fast greedy equivalence search algorithm for learning high-dimensional graphical causal models, with an application to functional magnetic resonance images. *International journal of data science and analytics*, 3(2):121–129, 2017.

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

Raanan Y Rohekar, Shami Nisimov, Yaniv Gurwicz, and Gal Novik. Iterative causal discovery in the possible presence of latent confounders and selection bias. *Advances in Neural Information Processing Systems*, 34:2454–2465, 2021.

Mauro Scanagatta, Giorgio Corani, and Marco Zaffalon. Improved local search in bayesian networks structure learning. In *Advanced methodologies for Bayesian networks*, pp. 45–56. PMLR, 2017.

Moninder Singh and Marco Valtorta. An algorithm for the construction of bayesian network structures from data. In *Uncertainty in artificial intelligence*, pp. 259–265. Elsevier, 1993.

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

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

Topi Talvitie, Ralf Eggeling, and Mikko Koivisto. Learning bayesian networks with local structure, mixed variables, and exact algorithms. *International Journal of Approximate Reasoning*, 115: 69–95, 2019.

Marc Teyssier and Daphne Koller. Ordering-based search: A simple and effective algorithm for learning bayesian networks. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*, pp. 584–590. AUAI Press, 2005.

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*, pp. 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.

Peter Van Beek and Hella-Franziska Hoffmann. Machine learning of bayesian networks using constraint programming. In *International Conference on Principles and Practice of Constraint Programming*, pp. 429–445. Springer, 2015.

Willem van den Boom, Maria De Iorio, and Alexandros Beskos. Bayesian learning of graph substructures. *Bayesian Analysis*, 18(4):1311–1339, 2023.

Man Leung Wong and Kwong Sak Leung. An efficient data mining method for learning bayesian networks using an evolutionary algorithm-based hybrid approach. *IEEE transactions on evolutionary computation*, 8(4):378–404, 2004.

Xianchao Xie, Zhi Geng, and Qiang Zhao. Decomposition of structural learning about directed acyclic graphs. *Artificial Intelligence*, 170(4-5):422–439, 2006.

Changhe Yuan, Brandon Malone, and Xiaojian Wu. Learning optimal bayesian networks using a* search. In *IJCAI proceedings-international joint conference on artificial intelligence*, volume 22, pp. 2186, 2011.

Hao Zhang, Shuigeng Zhou, Chuanxu Yan, Jihong Guan, Xin Wang, Ji Zhang, and Jun Huan. Learning causal structures based on divide and conquer. *IEEE Transactions on Cybernetics*, 52(5): 3232–3243, 2020.

A THE EXHAUSTIVE SEARCH

When discovering a SIP given a node set $\mathcal{V}$, the exhaustive search enumerate 4-way partitions such that $\mathcal{V} = \mathbf{A_1} \cup \mathbf{A_2} \cup \mathbf{C_0} \cup \mathbf{C_1}$ satisfying $\mathbf{A_1} \perp \mathbf{A_2} \mid \mathbf{C_0}$ and $\mathbf{A_1} \not\perp \mathbf{A_2} \mid \mathbf{C_0} \cup \{C\} \quad \forall C \in \mathbf{C}_1$. In this way, we will obtain atleast one SIP, if any. (Algorithm 4)

Algorithm 4 Partition Enumeration

Require: Node set $\mathcal{V}$
Ensure: A valid SIP $(\mathbf{A}_1, \mathbf{A}_2, \mathbf{C}_0, \mathbf{C}_1)$ minimizing $|\mathbf{C}_0 \cup \mathbf{C}_1|$

```
BestPartition ← null
MinSize ← ∞
for each four-way partition (A_1, A_2, C_0, C_1) of V do
    if A_1 ⊥ A_2 | C_0 then
        if A_1 ⊥̸ A_2 | C_0 ∪ {C} for all C ∈ C_1 then
            ShieldSize ← |C_0 ∪ C_1|
            if ShieldSize < MinSize then
                MinSize ← ShieldSize
                BestPartition ← (A_1, A_2, C_0, C_1)
            end if
        end if
    end if
end for
return BestPartition
```

The exhaustive search procedure evaluates all possible partitions of the node set and retains only those that satisfy $\mathbf{A_1} \perp \mathbf{A_2} \mid \mathbf{C_0}$. However, this approach is computationally expensive. For $|\mathcal{V}| = n$, the enumeration space contains 4^n assignments. As each requires a conditional independence check of cost $O(n^2)$, the exhaustive enumeration has overall complexity $O(n^2 \cdot 4^n)$.

B Different outputs of SIP discovery

Consider the example in Figure 4 to discover different SIP discoveries given different seed creations. Figure 4 (a) shows a P-map over the node set $\mathcal{V} = \{1, 2, \ldots, 13\}$ where more than one marginal independencies are present. Thus, SIP-Discovery procedure can be initialized with any marginal independencies with $\mathbf{Z} = \emptyset$. Figure 4 (b), (c) shows SIP-Discovery outputs using Algorithm 2 for different initializations.

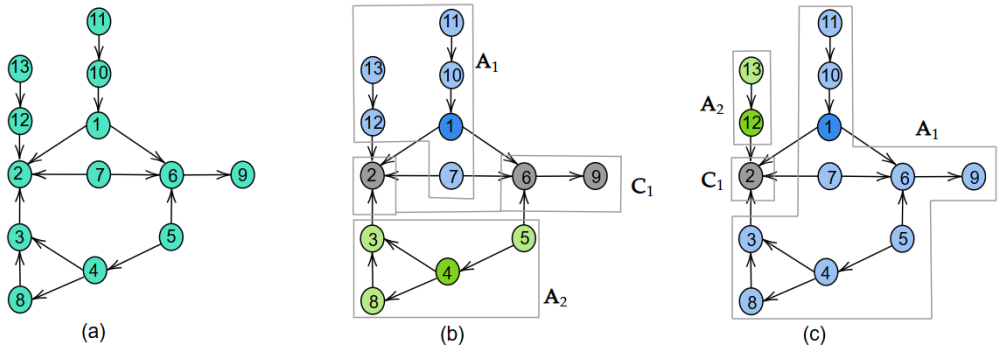

Figure 4: (a) a P-map for $\mathcal{V} = \{1, 2, \ldots, 13\}$. (b) SIP discovery of $\mathcal{V}$ with $1 \perp 4$ initialization. (c) SIP discovery of $\mathcal{V}$ with $1 \perp 12$ initialization.

C Recursive SIP illustration

Figure 5 illustrates the learning over the recursively split domains shown in Table 1 for the example in Figure 1(a).

D Supporting proofs

Proof of Proposition 3.1. Suppose $X \perp_{\mathcal{G}} Y \mid \mathbf{Z}$ initialized the Algorithm and resulted a partition $(\mathbf{A_1}, \mathbf{A_2}, \mathbf{C_0}, \mathbf{C_1})$. We need to prove the assignment of nodes $\mathcal{V} \setminus \{X, Y, \mathbf{Z}\}$ to partition blocks $\mathbf{A_1}, \mathbf{A_2}, \mathbf{C_1}$ in Algorithm2 gurantees $\mathbf{A_1} \perp_{\mathcal{G}} \mathbf{A_2} \mid \mathbf{C_0}$ and $\mathbf{A_1} \not\perp_{\mathcal{G}} \mathbf{A_2} \mid \mathbf{C}_0 \cup \{C_1\}$ for all $C_1 \in \mathbf{C_1}$.

Case $\mathbf{A_1} \perp_{\mathcal{G}} \mathbf{A_2} \mid \mathbf{C_0}$: Given $X \perp_{\mathcal{G}} Y \mid \mathbf{Z}$, the set $\mathbf{C_0}$ is initialized as $\mathbf{C_0} = \mathbf{Z}$ and remains unchanged throughout the procedure. By construction, a node V is added to $\mathbf{A_1}$ only if $V \perp_{\mathcal{G}} U \mid \mathbf{C_0}$ for all $U \in \mathbf{A_2}$. Symmetrically, a node is added to $\mathbf{A_2}$ only if it is conditionally independent of all nodes in $\mathbf{A_1}$ given $\mathbf{C_0}$. Therefore, for every $V \in \mathbf{A_1}$ and $W \in \mathbf{A_2}$, we have $V \perp_{\mathcal{G}} W \mid \mathbf{C_0}$. Hence, $\mathbf{A_1} \perp_{\mathcal{G}} \mathbf{A_2} \mid \mathbf{C_0}$.

Case $\mathbf{A_1} \not\perp_{\mathcal{G}} \mathbf{A_2} \mid \mathbf{C}_0 \cup \{C_1\}$: Nodes that do not satisfy the inclusion conditions for $\mathbf{A_1}$ or $\mathbf{A_2}$ are placed in $\mathbf{C_1}$. By construction, for each such node V, there exist nodes $U_1 \in \mathbf{A_1}$ and $U_2 \in \mathbf{A_2}$ such that $V \not\perp U_1 \mid \mathbf{C_0}$ and $V \not\perp U_2 \mid \mathbf{C_0}$. If V lies on an active path between any node in $\mathbf{A_1}$ and any node in $\mathbf{A_2}$, it would contradict $\mathbf{A_1} \perp_{\mathcal{G}} \mathbf{A_2} \mid \mathbf{C_0} = \mathbf{Z}$. Hence, if $\mathbf{C}_0$ is observed, there are active trails between V and U_1 and between V and U_2. Hence, in view of $\mathbf{A}_1 \perp_{\mathcal{G}} \mathbf{A}_2 \mid \mathbf{C}_0$, these active trails should form a collider at V. Hence, $\mathbf{A_1} \not\perp_{\mathcal{G}} \mathbf{A_2} \mid \mathbf{C}_0 \cup \{V\}$, completing the proof. □

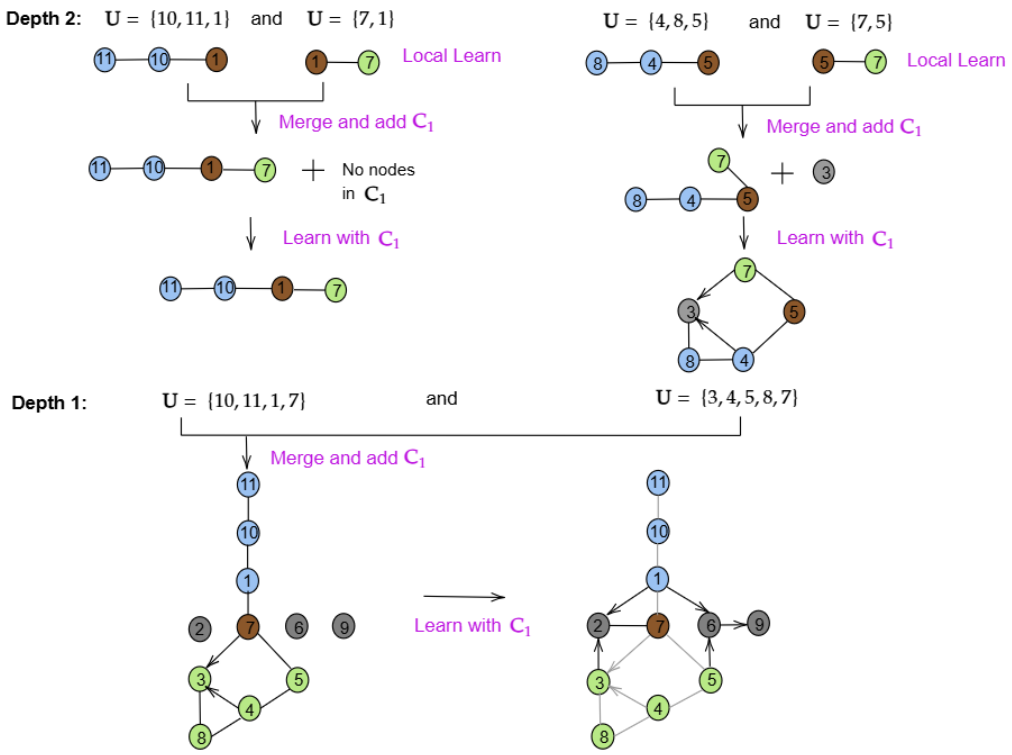

Figure 5: Recursive SIP algorithm illustration for recursive SIP formation on Table 1

Proof of Theorem 3.2. Let $\mathcal{G}$ be a P-map for P. Under Markovness and faithfulness, the CPDAG representing the P-map class is determined by the skeleton of $\mathcal{G}$ together with its unshielded colliders. It therefore suffices to show that Algorithm 3 recovers these two objects.

First consider a single call of Algorithm 1 on a SIP

$$(\mathbf{A}_1, \mathbf{A}_2, \mathbf{C}_0, \mathbf{C}_1)$$

with domain

$$\mathbf{U} = \mathbf{A}_1 \cup \mathbf{A}_2 \cup \mathbf{C}_0 \cup \mathbf{C}_1.$$

The candidate pairs over $\mathbf{U}$ fall into three types: pairs between $\mathbf{A}_1$ and $\mathbf{A}_2$, local pairs with one endpoint in $\mathbf{A}_i$ and the other in $\mathbf{A}_i \cup \mathbf{C}_0$, and the remaining shield-mediated pairs, namely pairs between $\mathbf{A}_i$ and $\mathbf{C}_1$ or pairs internal to $\mathbf{C}_0 \cup \mathbf{C}_1$.

For the first type, Algorithm 1 introduces no candidate edges between $\mathbf{A}_1$ and $\mathbf{A}_2$. This is correct because the SIP condition gives

$$\mathbf{A}_1 \perp_{\mathcal{G}} \mathbf{A}_2 \mid \mathbf{C}_0.$$

If there were a true edge between some $X \in \mathbf{A}_1$ and $Y \in \mathbf{A}_2$, then the direct trail $X - Y$ would remain active given $\mathbf{C}_0$, since neither endpoint is conditioned on. This would contradict the above d-separation. Hence all cross-block pairs are true non-edges.

For the second type, Theorem 3.1 guarantees precisely the local recoverability needed for independent learning: for every non-adjacent pair X, Y with $X \in \mathbf{A}_i$ and $Y \in \mathbf{A}_i \cup \mathbf{C}_0$, there exists a conditioning set

$$\mathbf{Z} \subseteq (\mathbf{A}_i \cup \mathbf{C}_0) \setminus \{X, Y\}$$

such that $X \perp_{\mathcal{G}} Y \mid \mathbf{Z}$. Therefore, with sound and complete CI tests, the local CI-pruning step on $\mathbf{A}_i \cup \mathbf{C}_0$ removes exactly the false local adjacencies. Conversely, if X and Y are adjacent in $\mathcal{G}$, the direct trail between them is active under every conditioning set that excludes X and Y, so faithfulness prevents the local learner from deleting this true edge.

For the third type, Algorithm 1 performs the merge step on the full SIP domain $\mathbf{U}$. Thus the remaining candidate pairs, those between $\mathbf{A}_i$ and $\mathbf{C}_1$, and those internal to $\mathbf{C}_0 \cup \mathbf{C}_1$, are tested using the

Table 3: Number of Nodes, Arcs, and Parameters

DATASET	NODES	ARCS	PARAMETERS
CANCER	5	4	10
EARTH(QUAKE)	5	4	10
SURVEY	6	6	21
ASIA	8	8	18
SACHS	11	17	178
CHILD	20	25	230
INSURANCE	27	52	1008
WATER	32	66	10083
MILDEW	35	46	540150
ALARM	37	46	509
HEPAR2	70	123	1453
WIN95	76	112	574
PATHFINDER	109	195	72079
ANDES	223	338	1157
DIABETES	413	602	429409
PIGS	441	592	5618
LINK	724	1125	1109

variables available in the full current domain. By the soundness and completeness of the CI tests, this step removes exactly the non-adjacent candidate pairs in the merge domain and retains all true adjacencies. Hence, after one call of Algorithm 1, the recovered skeleton over $\mathbf{U}$ agrees with the skeleton of $\mathcal{G}$ on all pairs considered in that SIP domain.

Algorithm 3 applies this procedure bottom-up on the decomposition tree. We argue by induction on the tree. At a leaf, the claim follows from the single-call argument above. Now suppose the claim holds for all children of an internal node. The child domains are of the form $\mathbf{A}_i \cup \mathbf{C}_0$, so their learned subgraphs already contain the correct local adjacencies by the induction hypothesis. At the parent node, Algorithm 1 reuses these local subgraphs and then performs the merge step to recover the remaining shield-mediated adjacencies in the parent SIP domain. Therefore, after processing the parent, the learned skeleton is correct over that parent domain. By induction, the skeleton is correct at the root, whose domain is the full node set $\mathcal{V}$.

It remains to justify the orientations. The algorithm stores the separating sets found during CI testing. For any unshielded triple $X - Z - Y$, the standard collider-orientation criterion applies: under faithfulness, Z is a collider on the triple if and only if Z is not contained in a separating set for X and Y. Therefore, orienting such triples as $X \to Z \leftarrow Y$ is correct. The subsequent CPDAG orientation rules orient only compelled edges and preserve Markov equivalence. Thus the final output has the same skeleton and the same unshielded colliders as $\mathcal{G}$, and consequently Algorithm 3 outputs the CPDAG representing the P-map class of P. □

E ADDITIONAL RESULTS

RUNTIME

ACCURACY: SHD PERCENT

Table 4: Runtime (seconds) for samples=100.

Algorithm	CANCER	EARTH	SURVEY	ASIA	CHILD	INSURANCE	WATER	MILDEW	ALARM	HEPAR2	WIN95	PATH	ANDES	DIABETES	PIGS
ABC	4.3261	0.9811	2.1175	2.7238	17.12	47.22	28.77	50.97	52.63	247	147.52	251.81	1644.2	11290	17938
ARGES	0.1525	3.689	0.8489	4.5237	6.7054	15.46	98.06	62.3	682.2	5568.7	1139.5	14007	NA	NA	86400
astar	0.3094	2.4757	0.8039	4.1824	NA	NA	86400	NA	86400	86400	86400	NA	NA	NA	86400
CB	0.0171	4.3756	0.9419	7.741	24.39	17.98	5.2216	46.06	56.71	86	144.72	1004.1	12944	86400	86400
COS	8.441	2.7069	0.236	1.8177	6.5472	11.81	39.28	41.74	86400	86400	86400	NA	86400	86400	86400
CPBayes	24.23	0.2368	0.7528	4.7194	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
ConservativePC	0.0128	0.0074	0.0082	0.0337	1.4356	1.3194	1.0031	2.3795	2.9512	3.6907	5.7998	42.97	34.71	249.22	274.6
ConsistentPC	0.0049	0.0042	0.0081	0.0366	1.1114	1.3285	0.8838	2.4752	3.9352	3.7903	5.2544	37.34	36.19	373.96	368.4
dual_PC	10.25	0.0294	0.0208	2.4226	1.334	4.525	0.9601	32.84	37.74	11.74	12.29	29.7	36.65	242.02	232.42
GES	0.031	0.1064	1.1485	1.7659	28.3	40.54	817.61	910.6	562.76	3305.3	2545.4	8351.4	39574	86400	NA
XGES	0.3637	11.21	2.1932	362.65	399.68	18577	47192	886.32	40236	86400	86400	86400	86400	86400	86400
BOSS	0.0394	0.0439	0.1007	0.3625	3.1404	15.13	370.27	7.3384	140.19	3964.8	18889	NA	NA	NA	NA
DAS	0.6625	0.6264	4.186	2.2645	30.94	336.54	557.34	703.55	800.57	5552.7	6991.5	22028	86400	86400	86400
GSP	5.351	15.28	256.03	16.34	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
GrowShrink	20.9	0.1083	0.0592	0.2007	4.7904	11.37	162.87	18.77	56.86	26.61	160.82	834.24	717.06	11362	6864.5
H2PC	0.0628	3.9324	0.9168	0.5842	9.8495	32.61	86400	NA	NA	510.68	86400	NA	86400	NA	86400
HEA	26.67	5.0429	3.1946	6.9513	24.52	50.25	58.5	254.26	96.78	496.5	525.98	2266.1	13278	NA	86400
HillClimbing	20.14	9.449	28.39	46.66	212.26	409.73	1197.2	644.91	936.19	7817.2	9000.4	20858	86400	NA	86400
K2Improved	0.9578	2.0857	2.0219	6.9233	35967	NA	86400	NA	86400	86400	NA	NA	NA	NA	86400
MMHC	10.55	5.5992	5.9734	20.37	97.82	225.62	467.11	364.23	670.45	6071.3	3707.3	16447	NA	NA	NA
MMPC	61.88	0.6678	0.2139	46.25	6.5356	12.07	25.5	22.23	142.26	1198.2	486.39	3819.5	NA	NA	NA
OBS	68.86	0.327	3.8386	16.33	NA	NA	NA	86400	86400	86400	NA	NA	NA	NA	86400
PC	8.2214	0.384	0.0214	0.2363	1.2977	1.7416	1.8388	2.0867	2.9817	16.64	10.86	34.51	38.94	285.58	222.89
PC_stable	8.127	0.0186	0.0205	7.6998	1.5597	2.1036	1.7935	2.1973	3.5222	7.0554	7.5851	36.31	35.12	332.86	284.01
ICD_CPDAG	0.0046	0.0061	0.0097	0.0283	0.7291	1.9684	0.9645	2.126	3.0657	3.6437	4.8426	28.33	34.22	283.06	274.62
SC	0.2755	0.7214	1.6324	7.5041	71.4	114.57	125.6	181.99	212.3	885.09	473.31	1382.1	8015.3	35650	14644
WINASOBS	0.5336	0.4863	34.15	8.737	969.77	10384	8643.5	14059	41295	86400	86400	NA	86400	NA	86400
AdjustedSIP	0.0067	0.0166	0.0082	0.0137	0.1345	0.1553	0.0989	0.2683	0.3229	0.6115	0.9105	170.14	8.3737	57.69	62.05

Table 5: Runtime (seconds) for samples=1000.

Algorithm	CANCER	EARTH	SURVEY	ASIA	CHILD	INSURANCE	WATER	MILDEW	ALARM	HEPAR2	WIN95	ANDES	DIABETES	LINK
ABC	14.17	10.01	18	24	256.9	495.11	376.75	715.57	526.9	2362.2	1444.4	16796	44053	86400
ARGES	5.7928	7.9498	3.8861	38.3	100.01	233.46	1110.7	246.1	6138.3	22239	9852.7	86400	NA	86400
astar	2.9708	2.3579	8.3185	43.03	NA	NA	86400	86400	NA	86400	86400	86400	86400	86400
CB	2.8948	0.1154	0.045	9.8832	22.03	48.89	10.12	1217.9	627.91	1270.1	1007.5	26336	86400	86400
COS	22.68	2.8609	2.2143	15.12	63.6	124.95	222.17	214.82	NA	86400	86400	86400	NA	86400
CPBayes	8.3051	2.2762	7.0918	40.06	NA	NA	NA	NA	NA	NA	86400	NA	NA	86400	86400
ConservativePC	0.0747	0.0873	0.0333	0.4591	92.75	68.21	2.6355	19.32	15.41	39.66	38.82	386.32	9383.6	16995
ConsistentPC	0.0173	0.0857	0.0227	0.247	16.91	23.94	2.9071	19.11	15.68	33.97	22.45	179.36	8550.5	15127
dual_PC	22.43	0.0761	0.285	9.5222	9.82	18.42	2.1793	53.79	11.2	23.51	15.81	118.36	5176.9	9554.9
GES	6.4132	7.3709	5.9437	36.88	2301	3924.2	6773.9	8164.6	6974.2	19551	38877	86400	86400	86400
XGES	37.89	58.5	42.12	3830.6	86400	41954	86400	20320	86400	86400	86400	86400	86400	86400
BOSS	1.5753	1.5565	1.7951	7.9876	90.7	174.23	5686	114.74	470.1	697.81	40688	20012	NA	86400
DAS	36.18	35.38	59.1	130.24	2048.1	5199.4	8845.7	11774	14100	86400	86400	86400	86400	86400
GSP	24.02	0.9618	9.0323	23.23	NA	NA	NA	NA	NA	NA	NA	NA	NA	86400
GrowShrink	22.46	0.086	0.1267	10.02	31.91	53.12	20.25	221.14	72.71	100.09	814.55	3567.2	86400	86400
H2PC	10.2	1.9335	8.5948	86400	103.73	NA	86400	86400	203.44	86400	86400	86400	86400	86400
HEA	28.08	19.27	22.8	47.55	130.19	200.98	241.11	594	316.06	1059.6	1255.6	39988	86400	86400
HillClimbing	189.25	101.75	284.86	365.62	1788.5	3370.9	7483.7	6319.1	8278.6	51875	68587	86400	86400	86400
K2Improved	24.72	1.5676	87.64	41.1	NA	NA	86400	86400	86400	86400	86400	86400	86400	86400
MMHC	70.13	9.2345	63.35	187.74	1052.3	2098.9	5829.2	4553.3	5820.8	38106	30386	86400	NA	86400
MMPC	1.5382	1.4733	1.8599	9.3585	56.68	123.81	230.96	169.79	1399.4	11060	4755.7	86400	NA	86400
OBS	3.59	2.8207	35.72	152.89	NA	NA	86400	86400	NA	86400	86400	86400	NA	86400
PC	0.0581	0.0911	0.1165	26.28	14.6	18.32	2.0334	56.33	10.41	21.68	40.47	116.15	4637.5	10351
PC_stable	19.93	0.1129	0.0971	26.33	14.84	21.34	2.6046	23.14	15.69	31.64	22.21	161.01	7536.4	14857
ICD_CPDAG	0.0657	0.0644	0.027	0.2347	20.67	71.87	2.1179	18.53	13.33	30.84	32.24	445.24	5737	13980
SC	10.12	8.0458	11.25	51.6	1130.8	1827	1321.9	3444	2537.3	3688.7	7009.3	85116	86400	86400
WINASOBS	3.2913	2.5101	6.1547	21.8	2266.4	15073	20077	9839.2	45778	86400	86400	86400	86400	86400
AdjustedSIP	0.0155	0.0336	0.0125	0.0404	0.698	0.641	0.1949	0.8988	1.0344	2.6186	2.0291	20.38	397.73	595.72

Table 6: Runtime (seconds) for samples=10000.

Algorithm	CANCER	EARTH	SURVEY	ASIA	CHILD	INSURANCE	WATER	MILDEW	ALARM	HEPAR2	WIN95	ANDES
ABC	151.73	130.9	190.82	392.37	3419.8	7281.4	4051.8	2641	5218.1	11943	18236	86400
ARGES	59.81	106.87	26.7	460.29	1182.3	6034.4	4308.6	5525.3	20846	86400	86400	86400
astar	29.24	23.67	69.2	431.91	NA	NA	86400	86400	NA	86400	86400	86400
CB	1.8982	0.3434	0.4491	2.2304	133.82	219.49	95	2215.8	2619.8	4676.2	28676	86400
COS	20.36	22.24	23.43	107.99	598.91	1253.2	2373.6	1722.1	11707	86400	86400	86400
CPBayes	28.53	22.54	72.42	387.1	NA	NA	NA	NA	NA	NA	NA	86400
ConservativePC	0.1763	0.1746	0.5976	1.0661	405.88	3162.6	43.9	1118.4	124.74	16827	819.35	35527
ConsistentPC	0.0484	0.2028	0.1828	0.4839	87.9	124.07	14.26	153.6	47.23	1790.2	184.51	2170.6
dual_PC	0.1908	0.1946	0.3356	0.4417	111.92	110.2	10.03	112.99	28.9	1263.2	139.09	1510
GES	106.72	86.17	192.27	447.93	5937.2	18699	24637	29306	51384	86400	86400	86400
XGES	371.17	7542	727.05	57664	86400	86400	86400	86400	86400	86400	86400	86400
BOSS	16.19	18.7	27.74	87.5	1379.4	3870.3	55723	2137.6	8741.9	13527	40817	86400
DAS	1187.2	1371.6	2020.9	4974.5	86400	86400	86400	86400	86400	86400	86400	86400
GSP	1.2431	1.2237	3.6732	23.19	NA	NA	86400	NA	NA	NA	NA	NA
GrowShrink	0.0727	0.1472	0.4219	1.1363	186.16	377.2	91.04	4488.4	359.92	376.01	3004.4	22026
H2PC	86.82	22.71	172.56	86.14	NA	NA	13678	86400	2059.7	86400	86400	86400
HEA	254.21	188.59	309.34	434.09	1205.1	2174.8	2146.3	2970.8	2726.7	7123.3	8329.7	86400
HillClimbing	1775.9	298.26	2000.6	3325	28564	56651	83855	41410	NA	86400	86400	86400
K2Improved	19.42	14.78	56.67	295.74	NA	NA	86400	86400	NA	86400	86400	86400
MMHC	576.27	111.81	645.15	1400.4	10436	22327	36959	6263	51778	86400	86400	86400
MMPC	15.23	14.33	18.77	98.46	578.48	1218.7	5194.2	1735.4	12159	86400	46707	86400
OBS	35.63	26.61	349.52	1589.9	NA	NA	86400	86400	NA	86400	86400	86400
PC	0.12	0.1843	0.3615	0.4876	73.03	101.74	9.2968	110.05	31.46	1263.5	124.11	1553.4
PC_stable	0.1321	0.1822	0.3543	0.5348	85.13	125.21	13.02	132.57	41.16	1415.6	202.5	1935.9
ICD_CPDAG	0.142	0.186	0.9187	0.7183	186.42	4049.7	23.49	519.37	53.9	7501	1259	86400
SC	109.82	80.21	214.96	572.11	3677.9	9351.9	5145.9	10166	33832	58077	86400	86400
WINASOBS	28.21	21.57	59.97	197.03	10389	39216	81442	86400	86400	86400	86400	86400
AdjustedSIP	0.1206	0.1274	0.1277	0.1815	4.4817	9.1961	1.6383	20.49	5.5026	384.29	19.89	107.78

Table 7: SHD (%) for samples=100.

Algorithm	CANCER	EARTH	SURVEY	ASIA	CHILD	INSURANCE	WATER	MILDEW	ALARM	HEPAR2	WIN95	PATH	ANDES	DIABETES	PIGS
ABC	100	75	100	50	164	146.15	150	256.52	136.96	196.75	120.54	109.23	160.06	213.62	268.75
ARGES	100	125	133.33	100	116	123.08	110.61	113.04	110.87	137.4	128.57	138.46	NA	NA	NA
astar	100	75	100	50	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
CB	100	100	100	175	156	130.77	106.06	117.39	126.09	121.14	119.64	117.44	125.44	NA	NA
COS	100	125	66.67	150	80	101.92	112.12	104.35	NA	NA	NA	NA	NA	NA	NA
CPBayes	100	75	100	50	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
ConservativePC	100	125	133.33	137.5	152	132.69	133.33	121.74	134.78	118.7	138.39	124.62	123.67	129.24	145.61
ConsistentPC	100	100	133.33	200	252	217.31	134.85	208.7	319.57	169.11	250.89	508.21	273.96	917.77	1135.3
dual_PC	100	125	133.33	150	116	109.62	122.73	108.7	123.91	119.51	127.68	125.13	112.13	124.25	117.57
GES	100	175	166.67	50	164	144.23	131.82	239.13	147.83	151.22	137.5	88.72	109.47	NA	NA
XGES	125	150	133.33	50	164	107.69	136.36	119.57	167.39	NA	NA	NA	NA	NA	NA
BOSS	0	275	133.33	50	164	186.54	336.36	119.57	226.09	233.33	758.04	NA	NA	NA	NA
DAS	100	75	116.67	112.5	152	146.15	122.73	158.7	184.78	131.71	140.18	137.44	NA	NA	NA
GSP	150	125	100	125	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
GrowShrink	100	225	133.33	212.5	208	226.92	474.24	184.78	539.13	279.67	1583	1861	1079.6	3259.6	271.79
H2PC	100	100	100	137.5	92	94.23	NA	NA	NA	103.25	NA	NA	NA	NA	NA
HEA	100	125	100	125	124	132.69	118.18	241.3	147.83	156.91	119.64	144.62	131.07	NA	NA
HillClimbing	125	75	166.67	212.5	396	340.38	415.15	650	734.78	1048	1303.6	1561	NA	NA	NA
K2Improved	125	150	83.33	162.5	116	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
MMHC	100	150	100	125	248	350	425.76	623.91	673.91	1021.1	1365.2	1531.3	NA	NA	NA
MMPC	100	100	133.33	225	120	126.92	140.91	108.7	710.87	1182.1	536.61	1228.7	NA	NA	NA
OBS	325	300	283.33	287.5	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
PC	100	125	133.33	137.5	112	119.23	122.73	113.04	119.57	117.07	127.68	131.28	113.02	124.75	116.89
PC_stable	100	100	133.33	137.5	116	117.31	122.73	104.35	117.39	109.76	133.93	111.28	111.83	113.62	107.6
ICD_CPDAG	100	125	133.33	137.5	108	109.62	121.21	106.52	106.52	113.82	126.79	116.92	113.02	124.92	103.72
SC	100	75	100	100	152	148.08	148.48	236.96	158.7	191.87	125.89	96.92	153.85	208.14	100.51
WINASOBS	100	75	100	50	188	146.15	146.97	241.3	154.35	NA	NA	NA	NA	NA	NA
AdjustedSIP	100	75	133.33	137.5	112	125	125.76	113.04	126.09	114.63	133.04	139.49	125.15	132.72	110.47

Table 8: SHD (%) for samples=1000.

Algorithm	CANCER	EARTH	SURVEY	ASIA	CHILD	INSURANCE	WATER	MILDEW	ALARM	HEPAR2	WIN95	ANDES	DIABETES	LINK
ABC	125	50	100	125	240	198.08	136.36	358.7	169.57	198.37	106.25	110.36	146.68	NA
ARGES	150	200	133.33	100	148	125	130.3	132.61	121.74	87.8	119.64	NA	NA	NA
astar	125	50	100	37.5	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
CB	250	100	100	150	124	123.08	103.03	139.13	152.17	135.77	139.29	130.77	NA	NA
COS	75	75	100	162.5	88	101.92	100	106.52	NA	NA	NA	NA	NA	NA
CPBayes	125	50	100	37.5	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
ConservativePC	175	200	133.33	112.5	156	71.15	131.82	115.22	121.74	127.64	130.36	77.22	144.85	116
ConsistentPC	100	125	83.33	175	168	332.69	190.91	406.52	489.13	247.15	428.57	501.48	4100.3	1627.3
dual_PC	100	175	133.33	162.5	96	96.15	121.21	78.26	89.13	113.01	113.39	87.87	153.32	115.64
GES	175	200	133.33	100	188	178.85	142.42	121.74	126.09	84.55	79.46	NA	NA	NA
XGES	175	200	133.33	87.5	NA	155.77	NA	152.17	NA	NA	NA	NA	NA	NA
BOSS	175	200	133.33	62.5	124	94.23	301.52	150	80.43	77.24	170.54	47.93	NA	NA
DAS	100	250	100	262.5	228	196.15	145.45	169.57	269.57	NA	NA	NA	NA	NA
GSP	125	100	66.67	137.5	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
GrowShrink	200	275	133.33	275	208	213.46	369.7	206.52	306.52	205.69	1188.4	694.38	NA	NA
H2PC	100	150	116.67	NA	104	NA	NA	NA	65.22	NA	NA	NA	NA	NA
HEA	125	50	100	75	188	130.77	121.21	123.91	167.39	102.44	128.57	137.28	NA	NA
HillClimbing	75	175	83.33	100	320	342.31	395.45	680.43	782.61	1013	1341.1	NA	NA	NA
K2Improved	100	175	83.33	200	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
MMHC	150	75	66.67	162.5	268	303.85	398.48	671.74	686.96	1030.1	1309.8	NA	NA	NA
MMPC	150	200	133.33	200	132	132.69	127.27	121.74	263.04	1261.8	386.61	NA	NA	NA
OBS	300	250	283.33	287.5	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
PC	75	200	133.33	162.5	100	98.08	125.76	82.61	69.57	113.82	115.18	92.01	152.99	112.53
PC_stable	75	200	133.33	162.5	100	92.31	122.73	91.3	93.48	112.2	120.54	85.21	142.69	107.29
ICD_CPDAG	75	200	133.33	150	100	90.38	130.3	84.78	89.13	111.38	112.5	81.36	129.4	105.78
SC	125	50	100	125	248	196.15	124.24	330.43	180.43	90.24	101.79	92.9	NA	NA
WINASOBS	125	50	100	37.5	188	103.85	104.55	117.39	36.96	NA	NA	NA	NA	NA
AdjustedSIP	75	25	133.33	112.5	128	105.77	122.73	95.65	104.35	126.02	109.82	109.76	163.29	107.82

Table 9: SHD (%) for samples=10000.

Algorithm	CANCER	EARTH	SURVEY	ASIA	CHILD	INSURANCE	WATER	MILDEW	ALARM	HEPAR2	WIN95	ANDES
ABC	100	0	0	100	292	246.15	171.21	117.39	186.96	75.61	90.18	NA
ARGES	175	0	133.33	100	156	128.85	122.73	132.61	136.96	NA	NA	NA
astar	0	0	0	37.5	NA	NA	NA	NA	NA	NA	NA	NA
CB	200	200	166.67	200	124	126.92	109.09	121.74	152.17	127.64	166.07	NA
COS	75	0	100	87.5	88	107.69	92.42	117.39	86.96	NA	NA	NA
CPBayes	0	0	0	37.5	NA	NA	NA	NA	NA	NA	NA	NA
ConservativePC	175	0	0	62.5	44	88.46	118.18	34.78	69.57	81.3	75.89	42.6
ConsistentPC	100	0	16.67	62.5	328	182.69	143.94	408.7	234.78	382.11	341.96	540.53
dual_PC	125	0	66.67	62.5	40	55.77	87.88	56.52	26.09	104.88	69.64	64.5
GES	0	0	0	100	164	86.54	116.67	91.3	78.26	NA	NA	NA
XGES	0	0	0	87.5	NA	NA	NA	NA	NA	NA	NA	NA
BOSS	0	0	0	62.5	96	113.46	292.42	115.22	32.61	45.53	38.39	NA
DAS	125	250	100	262.5	NA	NA	NA	NA	NA	NA	NA	NA
GSP	200	225	50	162.5	NA	NA	NA	NA	NA	NA	NA	NA
GrowShrink	200	225	216.67	262.5	228	209.62	410.61	291.3	260.87	226.02	801.79	555.03
H2PC	125	0	100	25	NA	NA	104.55	NA	54.35	NA	NA	NA
HEA	100	0	66.67	62.5	196	257.69	139.39	384.78	271.74	126.83	139.29	NA
HillClimbing	50	50	166.67	212.5	364	313.46	381.82	769.57	NA	NA	NA	NA
K2Improved	75	125	100	137.5	NA	NA	NA	NA	NA	NA	NA	NA
MMHC	125	125	66.67	162.5	340	301.92	389.39	736.96	710.87	NA	NA	NA
MMPC	175	225	133.33	212.5	144	134.62	134.85	132.61	160.87	NA	307.14	NA
OBS	250	125	233.33	212.5	NA	NA	NA	NA	NA	NA	NA	NA
PC	125	0	66.67	62.5	80	59.62	86.36	76.09	52.17	104.07	73.21	69.82
PC_stable	125	0	66.67	62.5	80	59.62	89.39	67.39	26.09	102.44	71.43	58.88
ICD_CPDAG	125	0	83.33	100	52	75	115.15	65.22	56.52	105.69	70.54	NA
SC	100	0	0	112.5	36	121.15	110.61	115.22	160.87	69.11	NA	NA
WINASOBS	0	0	0	37.5	156	38.46	116.67	NA	NA	NA	NA	NA
AdjustedSIP	100	0	66.67	62.5	92	75	109.09	54.35	69.57	130.89	81.25	70.12