Accelerating Recursive Partition-Based Causal Structure Learning

Md Musfiqur Rahman1, Ayman Rasheed1, Md. Mosaddek Khan1, Mohammad Ali Javidian2, Pooyan Jamshidi3, Md. Mamun-Or-Rashid1
1 Department of Computer Science and Engineering, University of Dhaka
2 School of Electrical and Computer Engineering, Purdue University
3 Department of Computer Science and Engineering, University of South Carolina

ABSTRACT
Causal structure discovery from observational data is fundamental to the causal understanding of autonomous systems such as medical decision support systems, advertising campaigns and self-driving cars. This is essential to solve well-known causal decision making and prediction problems associated with those real-world applications. Recently, recursive causal discovery algorithms have gained particular attention among the research community due to their ability to provide good results by using Conditional Independent (CI) tests in smaller sub-problems. However, each of such algorithms needs a refinement function to remove undesired causal relations of the discovered graphs. Notably, with the increase of the problem size, the computation cost (i.e., the number of CI-tests) of the refinement function makes an algorithm expensive to deploy in practice. This paper proposes a generic causal structure refinement strategy that can locate the undesired relations with a small number of CI-tests, thus speeding up the algorithm for large and complex problems. We theoretically prove the correctness of our algorithm. We then empirically evaluate its performance against the state-of-the-art algorithms in terms of solution quality and completion time in synthetic and real datasets.

KEYWORDS
Causal discovery; High-dimensionality; Constraint-based method; Conditional independence tests

1 INTRODUCTION
Causal structure discovery has emerged as a powerful computational method of identifying causal relationships from large quantities of data. Unlike the state-of-the-art statistical learning approaches, causal discovery examines the data generation procedure instead of inspecting the joint distribution of observed variables. Understanding and predicting causality in such a way has received significant interest in a large number of real-life application such as gene regulatory network [9, 29], advertising campaign [5], feature selection [1], self-driving cars [16], medical decision support systems [7] and many more besides [19, 24].

Causal discovery is generally formulated as a probabilistic graphical model (i.e., causal graph) where each edge represents the causal relationship between variables [12–15]. If controlled experiments are not possible, inferring the causal relations becomes challenging. Constraint-based methods can identify these relations by exploiting conditional independence tests (CI-tests) [10] when experiment samples are difficult to manipulate. In a CI-test, when two variables are independent given a conditioning set (i.e., d-separated by the conditioning set, see definitions), then we can conclude that there is no direct connection (i.e. causal relation) between them. This helps to disconnect the variables during the construction of the causal graph. Thus, following the faithfulness assumption (i.e., directed edges indicate causal relations) [17], existing constraint-based methods such as IC-algorithm [23] and PC algorithm [25] can discover partially directed acyclic graphs [21].

Now, constraint-based methods find it very difficult to deal with d-separators in large problems. When the number of variables increases, the number of possible conditioning set grows exponentially, making the exploration of d-separators [4, 21] computationally expensive. Another major challenge is, as the size of the conditioning set grows (i.e., high-order) with the number of variables, CI-tests become unreliable and may experience Type II errors (i.e., false CI hypothesis is accepted) [5, 8, 31]. As a result, we observe (i) some edges go missing, although they should exist, (ii) some undesired false edges are introduced.

To deal with these issues, researchers concentrated on recursive split-and-merge strategies [5, 11, 18, 26, 27]. These methods divide the original variable set into multiple subsets such that each subset can be solved recursively as a sub-problem by using the existing causal discovery algorithms. The results of these sub-problems are later merged to recover the causal graph corresponding to the main variable set. To learn the causal graphs, in place of using constraint-based algorithms, we can employ these recursive approaches since they avoid redundant CI-tests [30]. In effect, they provide more accurate result in less amount of time.

One notable algorithm named SADA is proposed in [5], which searches for causal cut over the variables in a sparse causal structure. Thus, it enables an efficient partitioning of the variables into subsets, and as such, produces a causal graph with a smaller number of samples. Nevertheless, SADA violates the d-separation (see Definition 2.1), and the causal cut searching process is costly since they are generated randomly and repeatedly for getting better decomposition. To address these issues, a new partitioning scheme, CAPA,
decomposes the original variable set into three smaller subsets, and can operate with low-order CI-tests to detect more causal directions than its predecessors [30]. However, the CAPA algorithm experiences severe execution time overhead, as well as a large number of CI-tests are required due to an additional partition (i.e., the third one). Recently, a split-and-merge strategy, named CP, has been developed that is reported to keep the run-time and the number of CI-tests at a lower margin. These are obtained by dividing the variable set into two partitions and executing a refinement process during the merge phase. This process is used to remove undesired false edges [28]. It is worth noting that this process takes a major portion of the execution time and a large amount of CI-tests since CP cannot distinguish false edges separately and conducts CI-tests for all of them.

As far as we know, no previous research has investigated false edge detection due to the d-separation violation. This paper proposes a recursive algorithm named Dsep-CP (d-separation preserving Causal Partition) to improve the scalability of causal discovery without sacrificing the solution quality. The first contribution of our paper includes the exploration of the conditions of d-separation violation and the graphical structure analysis for detecting the false edges. Dsep-CP employs this knowledge to refine the false edges of causal structure with a parsimonious number of CI-tests. This is the second key contribution of our work. Our theoretical analysis proves that our algorithm returns the correct causal structure under reliable CI-tests. Our empirical evaluation illustrates a substantial reduction in execution time up to 14% compared to the state-of-the-art without sacrificing the solution quality. In addition, during the refinement phase of our approach, it reduces the number of CI-tests up to 86% in different experimental settings.

2 BACKGROUND AND PROBLEM DEFINITION

This section formulates the causal discovery problem and discuss the background necessary to understand our proposed method.

A causal graph is a directed acyclic graph (DAG), \( G = (V, E) \) where \( V = \{v_1, v_2, \ldots, v_n\} \) is a set of \( n \) variables\(^1\) and \( E \) is a set of directed edges. Each edge \( u \rightarrow v \) represents the cause and effect relations between variables \( u \) and \( v \). In the DAG, let \( D = \{x_1, x_2, \ldots, x_m\} \) represents a data sample set where each sample is a vector, i.e., \( x_i = \{x_{i1}, x_{i2}, \ldots, x_{in}\} \). Here \( x_{ij} \) defines the value of variable \( v_j \) in sample \( x_i \). The sample data \( D \) is generated from a causal graph \( T \) with \( n \) nodes (we define it as a true graph). We consider that during the generation of data samples, the Causal Sufficiency [5] (i.e., the non-existence of latent confounders of any two observed variables) and the Faithfulness condition [17] (i.e., directed edges indicating causal relations) assumptions are followed. However, we aim to recover the causal graph \( G^* \) from \( D \) (Equation 1). Here \( \text{score}(G, T) \) indicates how close \( G \) is compared to \( T \) with respect to the causal relations between variables. Ideally, \( G^* \) is identical to \( T \), if not, as close as possible. However, finding exact \( T \) is computationally infeasible due to Markov equivalence classes [6] (i.e., different graphs with the same conditional independence relations among the variables).

\[
G^* = \arg\max_G \text{Score}(G, T)
\]

\(^1\)Throughout this paper, we use the terms variables and nodes interchangeably.

Definition 2.1 (D-separation). Two variables \( u \) and \( v \) are called d-separated with respect to a conditioning set \( Z \) if at least one of the following two conditions hold: (i) the path (i.e., a consecutive sequence of edges) between \( u \) and \( v \), contains a mediator \( (u \rightarrow w \rightarrow v) \) or a confounder \( (u \leftarrow w \rightarrow v) \) where \( w \in Z \), (ii) the path between \( u \) and \( v \), contains a collider \( (u \leftarrow w \leftarrow v) \) where \( w \) and its descendants are not in \( Z \) [20]. We employ CI-tests to determine d-separation\(^2\) in DAGs utilizing the sample set \( D \). During causal partitioning, we denote that d-separation is preserved if (i) adjacent variable pairs cannot be d-separated, (ii) non-adjacent variable pairs are either already divided into different subsets or d-separable in at least one of the subsets [30]. Otherwise, d-separation is violated during that partitioning process. Notably, if two variables \( u \) and \( v \) are conditionally independent given a variable set \( Z \), we represent it with \( u \perp \!\!\!\!\!\!\!\!\!\!\!\perp_{\mathcal{Z}} v \).

Definition 2.2 (Y-structure and Independence matrix). We define the formation of nodes as Y-structure where a collider exists with its parents and descendants. Whereas, \( M \) denotes an independence matrix. Here, \( M_{ij} = 1 \) indicates that \( v_i \perp \!\!\!\!\!\!\!\!\!\!\!\perp_{\mathcal{Z}} v_j \) for some \( Z \subseteq V \setminus \{v_i, v_j\} \) and \( |Z| \leq k_{\text{order}} \). Otherwise, they are not yet d-separated for \( k_{\text{order}} \) conditioning set. Here, \( k_{\text{order}} \) indicates the size of the conditioning set, and it increases from 0 to \( k_{\text{thresh}} \) which is a pre-specified maximum limit. Figure 1 shows an example of Y-structure causal graph \( G \) and independence matrix \( M \) of \( k_{\text{order}} = 1 \). It can be seen from this figure that in \( G \), variable 2 is a collider for variables 1 and 3, variable 3 a confounder for variables 2 and 4, variable 4 a mediator for variables 2 and 5 according to Definition 2.1. Here, \( 1 \perp \!\!\!\!\!\!\!\!\!\!\!\perp \{3\}, 1 \perp \!\!\!\!\!\!\!\!\!\!\!\perp \{4\}, 2 \perp \!\!\!\!\!\!\!\!\!\!\!\perp \{4\} \) and \( 3 \perp \!\!\!\!\!\!\!\!\!\!\!\perp \{4\} \) are represented in \( M \).

2.1 Causal Refinement

During the merging phase of SADA, in order to hold the acyclicity assumption (i.e., the causal graph is a DAG), it removes the less significant edge(s) (according to CI-test) when a cycle is found. Whereas, if two paths are found between a pair of variables, it eliminates specific edges with CI-tests. However, SADA fails to preserve d-separation during the partitioning phase. As a result, a number of redundant edges is created. To resolve this issue, CAPA splits the problem into three smaller sub-problems. The main purpose of the third partition is to ensure the preservation of d-separation. Due to this third sub-problem, CAPA does not have to perform any refinement during the merging phase. However, this third partition created during each recursive decomposition notably degrades CAPA’s runtime performance.

\(^2\)We use the terms conditional independence and d-separation interchangeably.
The CP algorithm first arranges the variables in descending order by the number of non-adjacent variables according to the independence matrix $M$. This strategy makes the partitioning process more efficient and extremely fast [28]. After that, CP divides the variable set $V$ into three non-overlapping sets $A, B$ and $C = V \setminus (A \cup B)$ such that $V_i \in A \lor V_j \in B \lor M_{ij} = 1$. Based on this decomposition, this algorithm creates two partitions $V_1 = A \cup C, V_2 = B \cup C$. However, in case of unsuccessful decomposition, CP increases the $k_{\text{order}}$ by 1 and restarts the partitioning method. In the merge phase of CP, it combines the resulting graphs from its two sub-problems and executes a refinement procedure. During the causal partitioning, the d-separation can be violated in some cases, which may produce false edges. The aim of the refinement step is to remove those edges from the causal graph. Therefore, CP performs CI-tests for every pair of variables conditioning on their parents.

In light of the above discussion, to develop an efficient causal discovery algorithm, we have to address two challenges: (i) detect the false edges due to the d-separation violation, (ii) provide a strategy that removes those false edges efficiently without losing the solution quality. In the following section, we describe our method that deals with the above challenges.

## 3 THE DSEP-CP ALGORITHM

Dsep-CP is a recursive method that effectively performs scalable causal discovery. The algorithm initiates with an original variable set, and at each level of recursive decomposition, the algorithm performs three principal operations. Initially, the **Find Causal Partitions** step takes place to find two causal partitions of the variable set, being careful about the d-separation violation. Secondly, the **Recursive Dsep-CP** calling step proceeds to solve the two sub-problems corresponding to the two partitions. Finally, the **Dsep-CP Refinement** step refines the causal graph that is discovered and merged from the two partitions.

Algorithm 1 shows the pseudo-code for Dsep-CP. The input of Dsep-CP is the variable set $V$ for which we need to construct the causal graph. The algorithm starts by checking the size of the variable set. If the size of the variable set is smaller than or equal to a suitable threshold, $graph\_thresh\_size$, the PC algorithm is run on this variable set for constructing causal relations (Algorithm 1: lines 1-3). Next, at line 4, the Find Causal Partitions function is called for finding suitable partitions of the main variable set.

The **Find Causal Partitions** step (Procedure 1) starts with initializing partitioning sets $A, B, C$ as empty sets, independence matrix $M$ (see Definition 2.2) as a $|V| \times |V|$ size zero matrix and $k_{\text{order}}$ as 0 (line 1). Next, the matrix $M$ is calculated at lines 2-3 and $M$ is copied to $M'$ at line 4. In the next step, we partition $V$ into three non-overlapping variable sets $A, B, C$ according to the optimization process used in CP [28]. In that process (lines 5-13), we execute $|V|$ iterations, and in each iteration, we choose variable $w$ with the highest priority and assign it in either $A, B, C$. At line 6, we select the variable $w$ which is d-separated from the most number of variables according to $M'$ (a copy of $M$), i.e., $w = \arg\max_w \sum_{i \in [1, \ldots, |V|]} M'_{w, r}$. The variable $w$ is either appended to the set $B$ (or $A$) if it is independent of all the variables existing in $A$ (or $B$) (lines 7-10). Otherwise, it is added into $C$ (line 12). At last, the $M'$ matrix is updated for $w$ (line 13) by setting the column values of $w$ to zero so that other variables with lower priority can be chosen in the next iteration at line 6.

After finishing all iterations, if the partition $C$ is not smaller than the other two partitions $A$ and $B$, we do not consider these as efficient partitions and try again with higher $k_{\text{order}}$ if it is less than the allowed max value, $k_{\text{thresh}}$. In this case, the procedure increases $k_{\text{order}}$ by one and goes to line 2 to start the partitioning procedure again (Procedure 1: lines 14-16). The procedure now updates $M$ for many variable pairs that could not be d-separated earlier but becomes independent now conditioning on a higher-order variable set. On the contrary, in case of efficient partitioning, lines 17-19 merge $A$ and $C$ to form $V_1$ and also merge $B$ and $C$ to form $V_2$. Finally, these two partitions are returned from Procedure 1, and we use them in the next steps of Algorithm 1. For example, we partition the variables of Figure 2a into three sets $A = \{6, 7, 8\}, B = \{3, 4, 5\}$ and $C = \{1, 2\}$ for $k_{\text{order}} = 0$. Then we take the union of $A$ and $C$ to get $V_1 = \{1, 2, 6, 7, 8\}$ and the union of $B$ and $C$ to get $V_2 = \{1, 2, 3, 4, 5\}$ (Figure 2b). Here, the dashed directed edges indicate the causal relations between variables that are not yet discovered but exist in the true graph (details in Section 2).

In Algorithm 1 (lines 5-7), if the largest of the resulting two partitions $V_1$ and $V_2$ is the same as the main variable set $V$, then it indicates that the partitions are not efficient even after the partitioning procedure. This situation occurs when the variables are more connected to each other and may form a dense sub-graph. In that case, we execute the **PC_Algorithm(V)** for constructing the causal structure $G$, similar to Algorithm 1: line 2. The PC algorithm discovers and stores the causal relations in an adjacency matrix $G.dir$. Here $G.dir_{i,j} = 1$ indicates a directed edge, $v_i \rightarrow v_j$ and 0 means no edge exists. We use dot operation to express specific attributes of a graph, in this case, the direction matrix of graph $G$. It is worth noting that our algorithm is compatible to deal with both directed and undirected causal relations. So, if the PC algorithm cannot detect the directions properly, then our algorithm considers the causal structure as an undirected causal skeleton. Under this circumstance, Dsep-CP is still able to perform its improved refinement procedure and provide significant result. Therefore, if the causal skeleton is undirected then $G'.dir_{i,j} = 1$ represents only the existence of an edge.

In Figure 2b, we consider $graph\_thresh\_size = 5$ and thus, we execute the PC algorithm on each partition for discovering the

| Algorithm 1: Dsep-CP(V) |
|-------------------------|
| **Input:** A finite set of variables $V = \{v_1, v_2, \ldots, v_n\}$ |
| **Output:** Discovered causal structure, $G$ |
| 1 if $\text{len}(V) \leq graph\_thresh\_size$ then |
| 2 $G \leftarrow \text{PC}_{\text{Algorithm}}(V)$ // Discover causal relation |
| 3 return $G$ |
| 4 $[V_1, V_2] \leftarrow \text{Find-Causal-Partitions}(V)$ |
| 5 if $\max(\text{len}(V_1), \text{len}(V_2)) = \text{len}(V)$ then |
| 6 $G \leftarrow \text{PC}_{\text{Algorithm}}(V)$ // Discover causal relation |
| 7 return $G$ |
| 8 else |
| 9 $graph_1 \leftarrow \text{Dsep-CP}(V_1)$ |
| 10 $graph_2 \leftarrow \text{Dsep-CP}(V_2)$ |
| 11 $G' \leftarrow \text{Merge}(graph_1, graph_2)$ // Resolving conflicts |
| 12 $G \leftarrow \text{Dsep-CP-Refinement}(G', graph_1, graph_2)$ |
| 13 return $G$ |

\[1\]We have followed CP [28]. Other constraint-based algorithms can also be used.
individual partitions. The discovered causal relations return produced from the right partition.

Similarly, another false edge inferred that d-separation is violated, and a false edge separators from the left partition, \(\rightarrow 6\) conditioning on collider node 1 of the figure also opens the paths \(\rightarrow\). CI-tests indicates that Z cannot d-separate variables 2, 6 and creates during the partitioning process. For example, in the left partition \(\rightarrow 6\), we perform CI-tests conditioning on \(\rightarrow\)={1,7,8}. The result of the partition

Notably, the d-separation may get violated in some specific situations during the partitioning process. For example, in the left partition \(\rightarrow 6\) of Figure 2b, to detect the causal relation between 2 and 6, we perform CI-tests conditioning on \(\rightarrow\). The result of the CI-tests indicates that Z cannot d-separate variables 2, 6 and creates the edge 6 \(\rightarrow 2\) in Figure 2c. However, this is a false edge since it does not exist in the true graph of \(\rightarrow 6\) in Figure 2b. In the whole true graph of Figure 2a, the paths 6 \(\rightarrow 7\) and 6 \(\rightarrow 2\) can be blocked by conditioning on 7 and 1, respectively. However, conditioning on collider node 1 of the figure also opens the paths 6 \(\rightarrow 3\) \(\rightarrow 4\) \(\rightarrow 2\) and 6 \(\rightarrow 1\) \(\rightarrow 3\) \(\rightarrow 5\) \(\rightarrow 4\) \(\rightarrow 2\). Hence, to make 6 and 2 conditionally independent, we have to condition on \(\rightarrow\) or \(\rightarrow\). But we do not have access to all of these d-separators from the left partition, \(\rightarrow 6\) in Figure 2b. Hence, it can be inferred that d-separation is violated, and a false edge 6 \(\rightarrow 2\) is created. Similarly, another false edge 2 \(\rightarrow 3\) is constructed.

In case of efficient partitions at Algorithm 1: line 4, we further proceed and recursively run the Dsep-CP algorithm for each individual partitions \(\rightarrow 6\), \(\rightarrow 2\) (lines 9-10). The causal graphs constructed at lines 2 and 6 are returned as the return value of the sub-procedures called at line 9 and line 10. The resulting causal graphs, \(\rightarrow\) and \(\rightarrow\), are combined with a Merge(\(\rightarrow\), \(\rightarrow\)) function to form \(\rightarrow\) at Algorithm 1: line 11. In this function, a new causal

\[
\text{procedure 1: Find-Causal-Partitions(V)}
\]

\[
\text{procedure 2: Dsep-CP-Refinement(G, graph1, graph2)}
\]

structure \(\rightarrow\) is created appending the variables of both graphs i.e., \(\rightarrow\), \(\rightarrow\). We merge another attribute, direction matrix, \(\rightarrow\) such that \(\rightarrow\) \(\rightarrow\) = 0 when any of the graphs has \(\rightarrow\) = 0. Otherwise, \(\rightarrow\) \(\rightarrow\) takes the same values from both graphs. Precisely, these conditions can be expressed with an AND operation in Equation 2.

\[
\text{G.dir}_{1,j} = \text{graph1.dir}_{1,j} \land \text{graph2.dir}_{1,j}
\]

The reason is that there can be some situations when we observe conflicting relations between variables in the graphs produced from two partitions. In one partition, two variables may be causally connected, and in another partition, they may be disconnected. In that case, we take the independent relation, \(\rightarrow\) \(\rightarrow\) i.e., (disconnected) as the correct one. Because, if two variables \(\rightarrow\) and \(\rightarrow\) are found independent in one partition, they are still independent after merging the sub-problems since the conditioning set also exists in the merged graph. The causal graph of Figure 2d is the result of merging two graphs produced in Figure 2c. This resulting graph G has 13 edges in total: {1 \(\rightarrow\) 2}, {2 \(\rightarrow\) 3}, {3 \(\rightarrow\) 1}, {3 \(\rightarrow\) 4}, {3 \(\rightarrow\) 5}, {4 \(\rightarrow\) 2}, {5 \(\rightarrow\) 4}, {6 \(\rightarrow\) 1}, {6 \(\rightarrow\) 2}, {6 \(\rightarrow\) 7}, {6 \(\rightarrow\) 8}, {7 \(\rightarrow\) 2}, {7 \(\rightarrow\) 8}. In some cases, there exist some undesired false edges resulting from inefficient decomposition due to d-separation violations. Here, \(\rightarrow\) and \(\rightarrow\) are false edges according to the true graph of Figure 2a (see Section 2 for more details).

The Dsep-CP_Refinement (Procedure 2) is called as a sub-procedure of Algorithm 1 at line 12. Similar to the other algorithms described in Section 2, the purpose of the refinement procedure in Dsep-CP is to refine the resulting causal graph constructed from the sub-problems. This procedure finds a set of selected edges, \(\rightarrow\) and eliminates the false ones by executing CI-tests on each of them. To measure the performance of the refinement procedure, we define a function Hit_rate(\(\rightarrow\)) in Equation 3 that indicates the proportion
Figure 2: Illustration of Dsep-CP algorithm. Find-Causal-Partitions procedure divides the variable set \( V \) of Figure 2a into \( V_1 = \{1, 2, 6, 7, 8\} \) and \( V_2 = \{1, 2, 3, 4, 5\} \) in Figure 2b. Since, partition size has reached the size threshold (=5), we can execute the causal discovery PC algorithm. It discovers \( graph_1 \) and \( graph_2 \) in Figure 2c. In Figure 2d, the merge function combines the discovered graphs to form \( G \). We observe that, \((6 \rightarrow 2) \) and \((2 \rightarrow 3) \) are false edges. The refinement phase performs CI-test for 5 (blue) edges={(1,2), (2,3), (4,2), (6,2), (7,2)} and refines 2 edges in Figure 2e. Figure 2f is the final graph.

of the number of CI-tests required for refining the false edges with respect to the total number CI-tests performed on all edges in \( E \).

\[
\text{Hit}_r\text{ate}(E) = \frac{F(E)}{F(E) + T(E)}
\]

Here, \( F(E) \) is the number of CI-tests required to detect all edges in \( E \). This is fixed for a specific causal structure. \( T(E) \) is the number of CI-tests that are required to detect which edges are not false. Note that, these CI-tests do not help us to remove undesired edges. So, they are redundantly performed in a refinement procedure. The objective is to choose an optimal edge set \( E^* \) which includes all the false edges and least number of edges that are not false such that the function \( \text{Hit}_r\text{ate}(E) \) is maximized in Equation 4. Notably, in the CP algorithm refinement, CI-tests needs to be performed on the whole edge set of the causal structure. However, in Dsep-CP, we reduce the size of \( E \) and therefore, reduce the number of redundant CI-tests. This reduction is performed by searching for Y-structure (see Definition 2.2) and utilizing the colliders in it (correctness proved in Theorem 4.2). The colliders in the Y-structure must belong to both sub-graphs (i.e., \( graph_1, graph_2 \)). It indicates the presence of the colliders in the variable set \( C \) that was formed during the execution of the Find-Causal-Partitions procedure (Procedure 1: line 12).

To remove the false edges, the Dsep-CP_refinement takes merged graph \( G \) and the resulting graphs (\( graph_1, graph_2 \)) from the sub-problems as input. The first step of this procedure is to create a collider_set that consists of colliders of the graph. If the resulting causal graph is directed, we find colliders by choosing such nodes that have at least two incoming edges (Procedure 2: lines 1-2). In Figure 2e, variables 1, 2, 4 and 7 are colliders, i.e., \( \text{collider}_set = \{1, 2, 4, 7\} \). On the contrary, if the graph is undirected, we first select a node \( v_k \) and a list of its neighbors. Then for each pair of its neighbors \( v_i, v_j \), we check if \( v_i \perp v_j \) for any conditioning set \( Z \). Now, if they are independent but the considered node \( v_k \) does not belong to the set \( Z \), we can admit it as a collider (Procedure 2: lines 3-5) [25]. For example, in Figure 2e, we can consider about variable 2 and its neighbors = \{1, 3, 4, 6, 7\}. Now, \( 1. \perp 2 \) but \( 2 \perp \{6\} \). So, 2 is a collider in this graph.

Next, we iterate through the colliders listed in collider_set and remove the suitable false edges (lines 6-17). If a collider does not exist in both graphs, \( graph_1 \) and \( graph_2 \), we skip it (lines 7-8). At line 9, we save all the neighbors connected to the collider in neighbor_set. After that, if the causal graph is directed, we acquire the list of collider’s parents in the parent_set with the help of directed adjacency matrix \( G.dir \). If not directed, then we use collider’s neighbor_set as the alternative of its parent_set. (Procedure 2: lines 10-13). Next, lines 14-15 checks whether the collider is a descendant of another collider that (i.e., the ancestor) has at least two of its parents in different sub-graphs ensuring the Y-structure. For example, in Figure 2e, we continue with variable 2 since its ancestor 1 (a collider) has two parents \( 6 \in \text{graph}_1 \) and \( 3 \in \text{graph}_2 \).

After that, we iterate through each cur_par in the neighbor_set of the collider (lines 16-17). We perform a conditional independence test between the collider and the cur_par where any subset (size less than pre-specified \( k\_thresh \)) of the parent_set can be used as conditioning set. If they are independent, we remove the edge between the collider and the cur_par. Finally, after removing suitable edges following the conditions, the refined graph \( G \) is returned from the Dsep-CP-Refinement procedure. The final causal graph is returned from Algorithm 1 at line 13. We can visualize the illustration of these steps with Figure 2e. Our current collider_set = \{1, 2, 4, 7\}.

Now, we exclude those variables that do not belong to both graphs and are not descendant of any nodes in Y-structures. So, the updated collider_set is \{2\}. For collider 2, we execute CI-tests on \{1, 2\}, \{2, 3\}, \{4, 2\}, \{6, 2\}, \{7, 2\}\). Altogether, we examine in total 5 distinct edges and among them two of them are removed i.e., \( 6 \perp 2 \in \{1, 3, 7\} \) and \( 2 \perp 3 \in \{1, 4, 6\} \). So, the hit rate = 2/5. On the other hand, a major drawback in CP algorithm is that, it tests every edge in the graph for refinement. So, for this graph, CP performs CI-tests for all 13 edges and refines only 2 of them. This results in a hit rate of 2/13. After the refinement step depicted in Figure 2e, we can see the final causal graph in Figure 2f.

4 THEORETICAL ANALYSIS

In this section, we prove the validity of Dsep-CP in detecting the false edges created due to d-separation violation (see Definition 2.1). The violation occurs during the partitioning process when we construct partitions \( A, B, C \) from \( V \) and create two sub-problems
\[ V_1 = A \cup C, V_2 = B \cup C. \] Notably, the following Lemma 4.1 builds the premise for proving the correctness of our refinement approach in Theorem 4.2. Lemma 4.3 verifies the optimization technique used in this approach. We also analyze the time complexity of our algorithm in this section.

**Lemma 4.1.** *If a false edge \( x \to y \) created in sub-problem \( V_1 \) due to d-separation violation, can be removed after merging in \( V \) with independence test \( x \perp\!\!\!\perp y [\{w_{A \cup C}, w_B\}] \) then we can prove that there exist \( x \in A, y \in C, w_{A \cup C} \in C \) and \( w_B \in B. \)

**Proof.** We know that any pair of variables \( x, y \) in \( A \) are d-separable in \( V_1 = A \cup C \) if they are non-adjacent [28]. So, the only d-separation violation can occur when \( x \in A \) and \( y \in C \). In the sub-problem \( V_1, x \perp\!\!\!\perp y \) but \( x \perp\!\!\!\perp y | Z \) in the merged graph, therefore, it is obvious that few variables \( w_B \subseteq Z \) must be from the other partition \( B \). So, the rest, \( w_{A \cup C} \subseteq Z \) is from set \( A \) or \( C \). \( \square \)

**Theorem 4.2.** *Existence of false edge \( x \to y \) can be detected through Y-structures.*

**Proof.** From Lemma 4.1, we can infer that the variable \( w_B \) blocks the causal effect from \( x \to y \) that flows through partition \( B \). However, we know that \( x \perp\!\!\!\perp w_B | C \subseteq C \) since \( x \in A \) and \( w_B \subseteq B \) are divided during the partitioning process. That means, the path between \( x \) and \( w_B \) gets open when we condition on a variable \( w' \in w_{A \cup C} \). This situation occurs only when we condition on variable \( w' \) to block the chain \( x \to w' \to \cdots \to y \) (path 1) and \( w' \) also acts as a collider in the path \( x \to w' \to w_B \to \cdots \to y \) (path 2). Since, \( w' \) is a collider and a false edge \( x \to y \) exists, path 1 must be a chain. We cannot replace path 1 with another path \( x \to w' \to \cdots \to y \), because in that case, \( x, y \) would be independent and no false edge would exist in the sub-problem. However, \( y \) is a descendant of collider \( w' \in C \) and \( w_B \subseteq B \). So, the arrangement of the variables forms a Y-structure. That concludes we can locate the possible false edges by searching for Y structure without testing all the edges of the merged graph. As a result, if the PC algorithm and the CI-tests are both reliable, Dsep-CP returns the actual causal graph by removing the false edges. \( \square \)

**Lemma 4.3.** *To check the independence between \( x \) and \( y \) in Y-structure, CI-tests with only conditioning on the parents of the collider \( y \) is sufficient.*

**Proof.** From Theorem 4.2, we can locate the position of the variable \( y \) that may have false edges but not sure about which of its connected edges is false. So, we have to perform CI-test for each of its edges. However, in Theorem 4.2, we prove that the direction of the causal effect is from \( x \) to \( y \). Now, a path \( y \to \cdots \to x \) is not possible because it creates a cycle which violates the acyclicity assumption. It indicates that the causal effect of \( x \) passes through the parents of \( y \). Therefore, we can perform CI-test conditioning on only the parent variables of \( y \) instead of conditioning on parents of both \( x \) and \( y \). \( \square \)

We now consider time complexity of our algorithm. During the partitioning procedure, we use a conditioning set of size maximum \( k_{order \_thresh} = \sigma \). So, in the worst case, we have to calculate the independence matrix \( M \) (see Definition 2.2) for \( \sigma \) order. For variable set \( V = \{v_1, v_2, \ldots, v_n\} \) with conditioning set of size \( k_{order} = \{0, \ldots, \sigma\} \), we have to perform \( n^2 \cdot \left( \binom{n}{0} + \binom{n}{1} + \ldots + \binom{n}{\sigma} \right) = n^2 + n^\sigma \) number of CI-tests to construct independence matrix \( M \). During the refinement step, Dsep-CP performs CI-tests only for those edges that are connected to any collider of the Y-structure and remain in the middle partition, \( C \) (see Theorem 4.2 for the proof). If the size of the variable set is \( |V| = n \), the size of final selected node set is, \( n' = n + d + p \) where, \( d \) is the ratio of the variables in \( C \) to \( V \) and \( p \) is the ratio of the Y-structure colliders in \( C \) to all variables in \( C \). Both of these ratios help lowering \( n \). Therefore, if the average degree is \( e \), the required number of CI-tests is \( (n' + e) \cdot \left( \binom{n}{0} + \binom{n}{1} + \ldots + \binom{n}{\sigma} \right) \approx n \cdot m_{\max}^2 \) (Here, \( m_{\max} \) denotes the max number of neighbors). Finally, the complexity of the PC algorithm for solving sub-problems is \( O(s_{\max}^2 + 2^s_{\max}-2) \) where \( s \) is the number of sub-problems and \( s_{\max} \) is size of the largest of them. So, the total time complexity is \( O(n^2 s_{\max} + n \cdot m_{\max} + s_{\max} \cdot 2^s_{\max}-2) \cdot CT \) where \( CT \) is the time complexity for a single CI-test. However, the time complexity for solving the complete problem with PC is \( O(n^2 2^s-2) \cdot CT \). Since the parameter \( s_{\max} \) used in Dsep-CP is much lower than \( n \), Dsep-CP experiences notably reduced time complexity than PC.

**5 EMPIRICAL RESULTS**

In this section, we empirically evaluate Dsep-CP compared to three different state-of-the-art causal learning algorithms in terms of solution quality and scalability. We run our experiments on different simulated causal structures and on eight real-world causal networks. We show that Dsep-CP performs better than CAPA [30], SADA [5] and CP [28] in terms of scalability while yielding solutions of similar or better quality. To conduct all these experiments, we run parallel instances of all the competing algorithms in an Intel Xeon 20 Core machine with 92GB RAM (code: github.com/softsys4ai/Dsep-CP.)

In all the experiments, we use the linear non-Gaussian model for sample data generation from any causal structures, according to [5]. Briefly, for each node \( i \) in topologically sorted order, we generate a linear function \( v_i = \sum_{j \in P_i} w_{ij} v_j + r_{ei} \) and evaluate it for a number of times called sample size to produce the sample set. Here \( P_i \) is the set of parents of node \( i \), \( w_{ij} \) is a weight denoting the effect of \( v_j \) on \( v_i \), \( r_{ei} \) is the non-Gaussian noise term and \( r = 0.3 \) is a constant value denoting the effect of the noise term on \( v_i \). During the generation of these linear functions, for each node \( i \), we ensure that the arithmetic mean of \( r_{ei} \) is 0 and the variance is 1, as well as \( \sum_{j \in P_i} w_{ij} \) is 1. Finally, we shift all the values of \( v_i \) by their arithmetic mean and scale them by their variance (i.e., normalize the values) such that the new arithmetic mean and the new variance of these values also become 0 and 1, respectively. We set the maximum size of conditioning sets for CI-tests to 3. We do this because most practical causal networks are sparse and this threshold is sufficient to discover them. Also, size > 3 may produce type II errors [30]. Moreover, we terminate the causal partitioning process when the size of the corresponding subset is smaller than or equal to max([N/10], 3), where \( N \) is the total number of nodes. Finally, each algorithm deploys the PC algorithm to discover causal graphs from the smallest sub-problems.

In our experiments, we employ the algorithms on the generated sample data to find causal skeletons (without directions) instead of finding the causal structures (with directions). We do this based on the recommendation in [28]. However, we can use V-structure based methods [3, 4], Additive Noise Models [22], Information Geometric Causal Inference [2], etc. to acquire the directions of causal skeletons. To report each of the results, we run each algorithm 20 times for a different number of nodes (or samples). In so doing, we
Note that, we are particularly concerned about the larger networks, as the smaller ones are trivial to handle by any of the algorithms.

5.2 Experiments on Real-World Structures

In this group of experiments, we compare SADA, CP and Dsep-CP on linear non-Gaussian sample data generated by some real-world causal structures. We select these causal structures because they cover a variety of applications, including causal inference (Asia), protein signaling network (Sachs), waste water treatment (Water), disease risk forecasting (Mildew), diagnosis of liver disorders (Hepatitis), printer troubleshooting (Win95pts), intelligent tutoring system (Andes) and the pedigree of breeding pigs (Pigs). Moreover, we particularly select these networks to cover different sizes of networks. Table 1 and 2 show the names and some structural information about each network, in their first three columns.

We run SADA, CP and Dsep-CP using different sample sizes \{250, 500, 1000, 1500, 2000\} for each of the real-world causal structures following [28]. However, for two very large networks i.e. Andes and Pigs, we cannot run all algorithms 20 times. Instead, we run all algorithms on Andes and Pigs networks 18 and 5 times, respectively. The results for Mildew network is presented in Figures 4a - 4c. These figures show comparative run-times (in seconds), numbers of refining CI-tests and refining times (in seconds), respectively. From these figures, we see that Dsep-CP runs faster than the other algorithms while producing solutions of similar quality for varying sample sizes for real-world networks. Notably, we observe similarities between the figures of number of refining CI-tests and refining time (in Figure 3e and 3f, or Figure 4b and 4c) since they are proportionate to each other. Additionally, in Figure 4b - 4c, SADA gets closer to Dsep-CP with increasing number of samples, but it is outperformed by Dsep-CP due to its solution quality.

We observe comparable results by running our experiments on other real-world networks. Due to lack of space, we present these results for a fixed sample size (i.e. 500) in Table 1 and Table 2. Table 1 shows the solution quality of the three algorithms in terms of F1 scores, recall scores and precision scores. From this table, we again see that Dsep-CP yields solutions of similar quality compared to its competitors.
to SADA and CP. Table 2 shows the performance of the three algorithms in terms of total run-time, numbers of refining CI-tests and refining times. We also observe from the table that for very large real-world networks (200+ nodes), Dsep-CP runs around 37-58% faster than SADA and around 11-13% faster than CP while reducing both refining time and CI-tests by 51-81% compared to SADA and by 81-86% compared to CP. So, we can conclude that Dsep-CP is more applicable to causal discovery in high-dimensional cases than the state-of-the-art algorithms.

6 CONCLUSIONS AND FUTURE WORK

This paper introduces a recursive causal structure learning algorithm, Dsep-CP, to support effective and efficient causal discovery for large sets of variables. To incorporate our recursive method, we present an improved refinement mechanism that can reduce the algorithm’s execution time without compromising the solution quality. In our theoretical section, we prove the correctness of Dsep-CP. Finally, our extensive empirical observation illustrates that Dsep-CP outperforms the state-of-the-art algorithms in both synthetic and real-world structures. To be precise, Dsep-CP runs up to 92% faster than SADA and up to 14% faster than the CP algorithm. In the future, we intend to investigate whether this algorithm can be improved for faster causality discovery while increasing the recall value (i.e., recovering the falsely removed edges) by analyzing the characteristics of CI-tests.

ACKNOWLEDGMENTS

This research is mainly supported by the ICT Innovation Fund of Bangladesh Government. We also acknowledge the use of the BdREN High Performance Computing Facility. Moreover, the work done by Pooyan Jamshidi is supported by NASA (RASPERRY-SI Grant No. 80NSSC20K1720) and NSF (SmartSight Award 2007202).
