

Fast Causal Orientation Learning in Directed Acyclic Graphs

Ramin Safaeian

*Department of Electrical Engineering
Sharif University of Technology
Tehran, Iran*

RAMIN.SAFAEIAN@EE.SHARIF.EDU

Saber Salehkaleybar

*Department of Electrical Engineering
Sharif University of Technology
Tehran, Iran*

SALEH@SHARIF.EDU

Mahmoud Tabandeh

*Department of Electrical Engineering
Sharif University of Technology,
Tehran, Iran*

TABANDEH@SHARIF.EDU

Abstract

Causal relationships among a set of variables are commonly represented by a directed acyclic graph. The orientations of some edges in the causal DAG can be discovered from observational/interventional data. Further edges can be oriented by iteratively applying so-called Meek rules. Inferring edges' orientations from some previously oriented edges, which we call Causal Orientation Learning (COL), is a common problem in various causal discovery tasks. In these tasks, it is often required to solve multiple COL problems and therefore applying Meek rules could be time consuming. Motivated by Meek rules, we introduce Meek functions that can be utilized in solving COL problems. In particular, we show that these functions have some desirable properties, enabling us to speed up the process of applying Meek rules. In particular, we propose a dynamic programming (DP) based method to apply Meek functions. Moreover, based on the proposed DP method, we present a lower bound on the number of edges that can be oriented as a result of intervention. We also propose a method to check whether some oriented edges belong to a causal DAG. Experimental results show that the proposed methods can outperform previous work in several causal discovery tasks in terms of running-time.

Keywords: Causal Discovery, Structural Causal Models, Meek Rules, Causal Orientation Learning, Experiment Design

1. Introduction

Recovering causal relationships from data is one of the ultimate goal in the empirical sciences. The behavior of a complex system is not fully understood unless one can infer how different variables in the system influence each other. Without such complete understanding, it might be infeasible to predict how the system behaves in response to some target interventions.

In the literature, the causal relationships among a set of variables in a system are commonly represented by a directed acyclic graph (DAG) where there is a directed edge from variable X to variable Y if X is the direct cause of Y . From observational data, under causal sufficiency and faithfulness assumptions, the true causal graph can be identified up to a Markov equivalence class (MEC) which is the set of all DAGs representing the same set of conditional independences among the variables. It has been shown that all DAGs

in an MEC has the same skeleton¹ and the set of v-structures² (Verma and Pearl, 1992). Given the skeleton and set of v-structures, Meek (1995) proposed four rules, called Meek rules, to orient further edges in the graph by making no directed cycles or new v-structures. These rules are applied iteratively on the causal graph until no further edges can be oriented by applying any of these rules. However, some edges might be remained undirected even after applying Meek rules. The resulted graph is commonly called the essential graph which represented a summary of all DAGs in MEC in the sense that if there is a directed edge from X to Y in the essential graph, the same orientation holds in all DAGs in MEC. Otherwise, there exist at least two DAGs with the opposite directions of edge between X and Y .

In order to identify the whole causal structure, the golden standard procedure is to perform interventions in the system. For instance, by performing perfect randomized intervention on a variable, we are able to recover orientations of all edges incident with the intervened variable (He and Geng, 2008). Based on these new oriented edges, we can apply Meek rules again to discover further edges' orientations.

In both scenarios mentioned above (obtaining essential graph from observational data or discovering orientation of undirected edges from interventional data), we try to solve the problem of inferring new edges' orientations based on our current knowledge about the causal structure which might come from observational data, interventional data, or some prior knowledge. The discovered orientations should not make any new cycle or v-structure in the causal graph. We call this problem, "causal orientation learning (COL)" problem. Meek rules can be utilized as a basis to solve COL problem. However, in various applications of causal structure learning (see the related work), it is required to solve several COL problems. Unfortunately, executing Meek rules for solving multiple COL problems might be time-consuming since these rules are applied for each problem separately in an iterative manner. Moreover, in some of the applications, the COL problems have structural similarities and the solution for one of the problems can be reused in other ones. In this paper, our main goal is to devise a solution to solve multiple COL problems efficiently in time. Moreover, the proposed solution can be used as a subroutine in various settings for causal structure learning. In the next part, we review some of these settings in which COL problem arises along with the solutions that have been proposed in the literature. Two main problems that we review here, are counting size of MEC and experiment design.

First, we consider the problem of counting size of MEC. The number of DAGs in an MEC can give an indication of the "causal complexity" of the underlying true causal structure. For instance, in order to determine the range of possible causal strengths of variables on a target variable, in one of the algorithms in (Maathuis et al., 2009), it has been proposed to enumerate all DAGs in an MEC and estimate the causal strength of each parent of the target variable in the corresponding DAG. Herein, the size of MEC shows the time complexity of the estimation procedure. As another example, He and Geng (2008) proposed multiple scores, based on the possible sizes of an MEC after performing intervention on each variable. The variable with the highest score is selected for doing intervention (we will discuss this problem in more detail in the next part). In fact, remaining too many DAGs in the resulted

1. Two directed graphs have the same skeleton if they have the same set of vertices and edges regardless of their orientations.
2. Three variables X, Y , and Z form a v-structure in a graph if X and Y are the direct cause of Z and they are not connected to each other.

MEC after performing an intervention indicates the candidate variable is not suitable to be intervened on.

He et al. (2015) proposed the first efficient algorithm for counting the size of MEC instead of enumerating all DAGs. They first presented five closed-form formulas that can be used to calculate size of MECs having some specific relations between the number of nodes and edges. Next, they showed that counting size of an MEC can be partitioned into smaller sub-classes and the size of each of them can be computed in an iterative manner. In particular, the components corresponding to sub-classes (which are commonly called chain components) are obtained by removing the directed edges from the essential graph of MEC. In order to find the size of each sub-class, it is required to consider each variable in the corresponding component to be root and solve a COL problem to obtain further chain components in it. This procedure should be done recursively on each variable in the resulted components. Thus, we need to solve several COL problems and it may become time-consuming as the number of nodes increases. He et al. (2015) proposed an efficient algorithm to solve each COL problem separately which resembles applying Meek rules carefully according to properties of sub-classes. However, this solution still needs to solve multiple COL problems and does not reuse the results of a COL problem in the another one. He and Yu (2016) introduced a special structure of “core graphs” and showed that the size of MEC is a polynomial of number of vertices given its core graph. Moreover, they proposed an algorithm in order to obtain this polynomial. Inside this algorithm, similar to their previous work, it is required to solve multiple COL problems in order to find chain components in the essential graphs. Ghassami et al. (2019) and Talvitie and Koivisto (2019) proposed a dynamic programming based solution in order to compute the size of MEC. In this solution, whenever size of a chain component is computed, it is stored in memory and will be used later if we encounter the same chain component. In both works, it is necessary to solve multiple COL problems to obtain chain components and clique trees were utilized to efficiently orient edges in each COL problem.

Another important problem related to COL problem is experiment design problem. The whole causal structure can be recovered with sufficient number of interventions if it is feasible to intervene in the system. However, in most applications, performing interventions is too time-consuming or costly. The problem of experiment design is to devise a set of experiments where each experiment consists of multiple interventions performed simultaneously. The maximum number of interventions in each experiment is usually set to a fixed value (in some previous work, this value is equal to one (Hauser and Buhlmann, 2014; He and Geng, 2008; Ghassami et al., 2018)). The experiment design problem can be studied in two settings of passive or active learning. In the active setting, the experiments are performed sequentially and the result of an experiment is used in designing latter ones. More specifically, it is commonly assumed that the orientations of all edges incident with an intervened variable can be recovered by performing perfect randomized interventions. Based on these newly oriented edges, we can solve a COL problem to find resulted chain components after doing intervention. The goal is to recover the whole causal structure with minimum number of interventions. In contrary, in the passive setting, we have a limited budget for performing experiments. All the experiments are designed based on the MEC obtained from observational data and performed in parallel. According to the results of experiments, we solve a COL problem to orient as much edges as possible. Here, the goal

is to design a set of limited number of experiments (the number of interventions in each experiment is usually set to one) to recover the most part of the causal structure.

In the passive setting, He and Geng (2008) proposed a naive approach to enumerate all DAGs in an MEC and select a set of variables for interventions with minimum size such that all possible underlying DAGs can be recovered from the results of interventions after solving a COL problem. Ghassami et al. (2018) proposed a greedy algorithm to select the target interventions by sampling DAGs from the MEC and estimating the average number of edges oriented as a result of intervention. Again, in the sampling procedure, it is required to solve several COL problems similar to counting size of MECs. Kocaoglu et al. (2017) proposed an algorithm for experiment design where there exists a cost for intervening each variable. They showed that the optimal solution can be obtained in polynomial time in the case of causal structure being a tree or a clique tree.

In the active setting, He and Geng (2008) introduced an entropy based score to select the intervened variable in each step. In this score, it is required to compute the size of MEC for any possible orientations of edges incident with a given variable. Hauser and Buhlmann (2014) proposed a minimax criterion in order to select the target intervention. First, for any given variable, it is required to obtain the maximum number of undirected edges in the essential graphs that are consistent with different possible orientations of edges incident with the given variable. Next, from these variables, the one that minimizes this criterion is selected. In order to compute this criterion, Hauser and Buhlmann (2014) presented an algorithm where, for any feasible orientations of edges connected to a given variable, the number of oriented edges as a result of these newly oriented edges is obtained by solving a COL problem through running LexBFS algorithm. In fact, one can utilize LexBFS algorithm to find an ordering over a chain component and orient edges based on this ordering without making directed cycles or v-structures. Shanmugam et al. (2015) proposed a score based on graph coloring and separating system in order to select the node for the intervention. After observing the result of an intervention, a COL problem is solved to orient further edges by simply applying Meek rules. Ness et al. (2017) focused on Bayesian approach to learn the causal structure based on a prior probability distribution on the underlying graph. They introduced an information gain to select the variables for performing interventions. Similarly, Agrawal et al. (2019) studied the problem of experiment design from Bayesian perspective considering limited number of samples. They proposed a greedy algorithm with guarantee on approximation quality. Greenewald et al. (2019) also considered a Bayesian approach with the assumption of all chain components being trees. They proposed an algorithm with an approximation ratio of 2, in the sense that the number of interventions is at most twice the minimum achievable number. Teshnizi et al. (2020) proposed an efficient algorithm, called LazyIter, for iterating over MECs for all possible orientations of edges incident with a variable. Hence, it is required to solve multiple COL problems simultaneously. The authors exploited the similarities among possible essential graphs to avoid the recalculation of edges' orientations for each of them separately. The proposed algorithm has been utilized to solve problems of counting size of MEC and experiment design. Besides counting size of MEC and experiment design problems, there exist some other related works to COL problem. For instance, Chickering (1995) proposed an algorithm to recover oriented edges in the MEC of a DAG, given its skeleton and v-structures. Ramsey et al. (2017) considered a limited version of faithfulness assumption and presented a search algo-

rithm to considerably speed up the process of finding the essential graph from the samples, at the expense of allowing graphs to violate Markov factorization, i.e., for the conditional dependence and independence relations estimated from the samples. They also evaluated this model in large graphs with multi-million nodes.

The contributions of this paper are threefold:

- We introduce rigorous mathematical representations of Meek rules, which we call Meek functions. Equipped with these functions, we present key properties of these functions that enable us to execute them separately on a causal graph and aggregate their outputs in order to obtain the final result. Based on these properties, we can speed up the process of applying Meek rules for solving a COL problem.
- As mentioned before, in some cases, we are required to solve multiple COL problems with similar causal structures (such as COL problems in the experiment design problem). By exploiting Meek functions’ properties, we can keep the results of some computations and reuse them in other similar COL problem. Based on this idea, we propose a method to solve experiment design problem in the active setting in an efficient manner.
- We present a method to check efficiently whether a set of edges’ orientations (which might come from some prior knowledge) are consistent in the sense that there is a DAG with the same orientations for the given edges.
- We propose an efficient method to compute a lower bound on the number of oriented edges as a result of intervention. This lower bound can be utilized to select variables for intervention according to the minimax criterion (Hauser and Buhlmann, 2014). Experiments show that the gap between the lower bound and the true value is small and the same set of variables are selected if we use lower bounds instead of true ones.

The rest of this paper is organized as follows: We begin with some background and terminologies in Section 2. We then introduce a rigorous representation of Meek rules and present some their key properties in Section 3. In Section 4, based on these properties, we present methods for applying Meek rules on causal graphs and checking the consistency of some edges’ orientations in a DAG. We also propose a lower bound on the number of oriented edges as a result of intervention. In Section 5, we report our experimental results for the proposed methods and the lower bound. Finally, in Section 6, we conclude the paper and discuss some possible directions for future research.

2. Background and Terminology

A graph $G = (\mathbf{V}, \mathbf{E})$ is represented by a set of nodes \mathbf{V} and a set of edges \mathbf{E} . We denote the undirected edge between a pair of nodes $v_1, v_2 \in \mathbf{V}$, by $v_1 - v_2$. A directed edge from node v_1 to node v_2 is also denoted by $v_1 \rightarrow v_2$. We assume that there exists at most one edge either directed or undirected between each two nodes. We denote the set of all nodes that are connected to node v by $neigh(v)$. We also define $Neigh(v) = neigh(v) \cup \{v\}$. We call a subset of nodes of an undirected graph as clique if every two nodes in that subset are adjacent. A clique is maximal if it is not a subset of a larger clique. We denote the set of maximal cliques in the neighborhood of node v by $\mathbf{C}(v)$.

We say a graph G is partially directed acyclic graph (PDAG) if it does not have any directed cycle (Peters et al., 2017). A graph G is called directed acyclic graph (DAG) if it is PDAG and all the edges in G are directed. We say that node v_1 is a parent of node v_2 if there is a directed edge from v_1 to v_2 in graph G . Moreover, the descendants of a node v_1 is the set of nodes that there are directed paths from v_1 to them. An induced sub-graph of graph G containing vertices $\mathbf{T} \subseteq \mathbf{V}$ is defined to be a set of edges that contains all those edges in \mathbf{E} with both end points in \mathbf{T} and it is denoted by $\mathbf{E}[\mathbf{T}]$. We call an induced sub-graph in the form of $v_1 \rightarrow v_2 \leftarrow v_3$ as a v-structure. If a graph has no partially directed cycle, it is called a chain graph. Removing all directed edges in a chain graph leaves some undirected disjoint chain graphs. These chain graphs are called, chain components. An undirected graph is said to be chordal if there exists a cord in any cycle with length more than three.

Consider the set of variables $\mathcal{X} = \{X_1, \dots, X_n\}$ in the system. Causal relationships between these variables are usually modeled by a DAG G , where each variable in \mathcal{X} is mapped to one of the nodes in G , and an arrow between two nodes, like $v_1 \rightarrow v_2$, shows the corresponding variable of v_1 in \mathcal{X} as a direct cause of the corresponding variable of v_2 in \mathcal{X} . We call this DAG, which represents causal relationships between variables, as a “causal DAG”. We say that a joint distribution P over \mathcal{X} satisfies Markov property with respect to G if any variable of G is independent of its non-descendants given its parents. Under causal sufficiency and faithfulness assumptions, any conditional independence in P can be inferred by Markov property (Spirtes et al., 2000). The set of all DAGs that encode the same conditional independence assertions are called Markov equivalence class (MEC). An essential graph corresponding to an MEC is a graph that has the same skeleton as all DAGs in that MEC and an edge is directed in essential graph if that edge has the same direction in those all DAGs. It can be shown that the essential graph is a chain graph with chordal chain components, where each chain component is an undirected and connected chordal graph (UCCG for short) (Andersson et al., 1997). We denote chain components of a graph G by $\mathcal{C}(G)$. Note that the chain components are obtained by removing all the directed edges from the essential graph, where whole v-structures are identified. Thus, in any chain component, there is no undiscovered v-structure. In this manuscript, we use the term UCCG for denoting one of the chain components of an essential graph.

From the observational distribution P , the essential graph of underlying ground truth DAG can be recovered by performing conditional independence tests (Spirtes et al., 2000). For further orienting the undirected edges, we need to intervene in the system. We use the same notion of hard intervention as in Eberhardt et al. (2005) and Pearl (2009). The procedure of intervention on a random variable X is to force this variable to get its values from an independent randomized distribution, regardless of the values of its parents. For a set of interventions I , two DAGs G_1 and G_2 are called \mathcal{I} -Markov equivalent if they are statistically indistinguishable under interventions in I . Interventional MEC (\mathcal{I} -MEC) is the set of all DAGs that are \mathcal{I} -Markov equivalent. Moreover, the summary of all DAGs in an \mathcal{I} -MEC can be presented by an essential graph, which we call \mathcal{I} -essential graph.

In this paper, we assume that infinite samples from any observational or interventional distributions are available. Thus, the true essential graph of underlying causal model is available.

3. Causal Edge Orientation

In this section, we first describe Meek rules (Meek, 1995). These rules enable us to orient further edges since we know the underlying causal graph is a DAG. Then, in the next section, we present a mathematical representation for Meek rules. Specifically, we define Meek functions that help us to formulate the Meek rules in more rigorous way. In the last part, we present some properties of Meek functions. As we will see later, these properties can be utilized to solve COL problems in an efficient manner.

3.1 Meek rules

As we mentioned in the previous section, the essential graph is a partially directed acyclic graph that will be identified from the observational distribution (Spirtes et al., 2000). Note that the skeleton and all the v-structures of the essential graph are the same as the skeleton and v-structures of underlying causal DAG and they can be identified by performing some conditional independence tests (Verma and Pearl, 1992). Additional edges in the essential graph can be oriented based on these two facts: (a) there is no more undiscovered v-structure, and (b) the underlying causal DAG is acyclic. Every such additional edges can be identified by repeatedly applying Meek rules (Meek, 1995). It can be shown that Meek rules are complete and sound in recovering the essential graph (Meek, 1995).

There are four Meek rules that are given in Table 1. In this table, we illustrate how an edge will be oriented after applying each Meek rule. Each column corresponds to one of the Meek rules. The graph in the first row in each column will be converted to the graph in the second row in that column after applying the corresponding Meek rule. Thus, some undirected edges in the graph in the first row have been oriented in the graph in the second row. By considering all possible orientations for corresponding sub-graphs of Meek rules 3 and 4, it can be shown that there is no need to consider orientations of dashed lines for applying these Meek rules as long as the dashed lines do not belong to a v-structure.

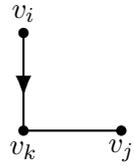
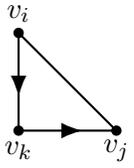
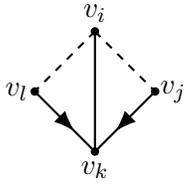
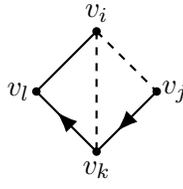
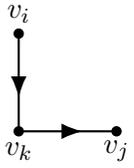
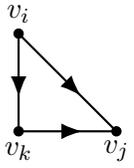
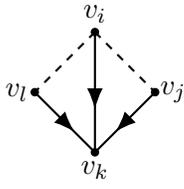
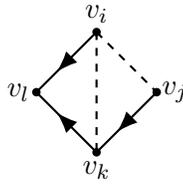
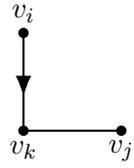
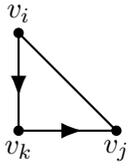
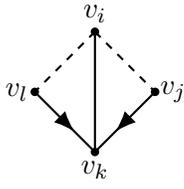
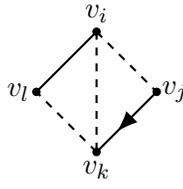
In addition to applying Meek rules to obtain essential graph, we can utilize them to orient further edges after performing an intervention on a node to obtain \mathcal{I} -essential graph. In this case, after performing a perfect randomized intervention, the orientations of all edges between the intervened node and its neighbors will be recovered (He and Geng, 2008). Applying Meek rules on the graph with these new oriented edges, results in orienting additional edges. This procedure will be continued till no edge can be discovered. The resulted graph would be the \mathcal{I} -essential graph after the intervention.

To wrap up, applying Meek rules on a graph with some new directed edges (for instance, oriented edges in the neighborhood of a node after performing an intervention), may recover the orientations of further edges in the graph. Having more directed edges means that more causal relations have been revealed from the underlying causal graph. Thus, Meek rules can be seen as a tool to infer new causal orientations based on some prior knowledge about the causal graph. We will present a more formal representation of these rules in the next part.

3.2 Meek Functions

In this part, we define Meek functions. These functions take a set of edges as their inputs and return another set containing already directed edges in the input set and some new edges

Table 1: Meek rules

Rule	Meek Rule 1	Meek Rule 2	Meek Rule 3	Meek Rule 4
Corresponding sub-graph				
After applying Meek rule on corresponding sub-graph				
Corresponding candidate sub-graph				

that are oriented as a result of repeatedly applying one of the Meek rules. Furthermore, we define a “candidate sub-graph” for each of these functions. Candidate sub-graph is an induced sub-graph, with minimal number of nodes, that applying particular Meek rule on that sub-graph, results in orienting one or more edges inside that sub-graph. We give a concise definition of candidate sub-graphs for each of the Meek functions in the following. We first provide some definitions.

Definition 1 (Mixed-edge union operator) Let \mathbf{A} and \mathbf{B} be sets of some directed and undirected edges. We define $\mathbf{A} \sqcup \mathbf{B}$ as a set of edges with the following property:

$$\mathbf{A} \sqcup \mathbf{B} = \mathbf{C} \setminus \{v_i - v_j \mid v_i \rightarrow v_j \in \mathbf{C} \text{ or } v_j \rightarrow v_i \in \mathbf{C}, \forall i, j \in \mathbf{V}\},$$

where $\mathbf{C} = \mathbf{A} \cup \mathbf{B}$. In other words, $\mathbf{A} \sqcup \mathbf{B}$ keeps the directed edge $v_i \rightarrow v_j$ if it exists in \mathbf{A} or \mathbf{B} . Moreover, it keeps the undirected edge $v_i - v_j$ if this edge has not been oriented in either \mathbf{A} or \mathbf{B} .

Definition 2 (Skeleton of a graph) Let $G = (\mathbf{V}, \mathbf{E})$ be a PDAG, \mathbf{V} be the set of nodes and \mathbf{E} be the set of edges. Assume that we replace all directed edges in this partially directed graph with the undirected ones. We denote the set of all edges in this undirected graph with $\overline{\mathbf{E}}$. Thus, we have:

$$\overline{\mathbf{E}} = \{v_i - v_j \mid v_i \rightarrow v_j \in \mathbf{E} \text{ or } v_i - v_j \in \mathbf{E}, \forall i, j \in \mathbf{V}\}.$$

Definition 3 (Set of directed edges) Let $G = (\mathbf{V}, \mathbf{E})$ be a PDAG. We denote the set of all directed edges in \mathbf{E} by $\vec{\mathbf{E}}$:

$$\vec{\mathbf{E}} = \{v_i \rightarrow v_j | v_i \rightarrow v_j \in \mathbf{E}, \forall i, j \in \mathbf{V}\}.$$

Now, we define a candidate sub-graph for each Meek rule. These candidate sub-graphs are depicted in the third row in Table 1. Examples for Meek candidate sub-graphs are shown in Figure 1. The candidate sub-graphs for Meek rules 1, 2 and 3 are as the same of their corresponding sub-graphs, while Meek rule 4 candidate sub-graph deviates from its corresponding sub-graph. In the following, we show that applying Meek rules are sound and complete. The proof of all lemmas and theorems are given in the appendix.

Theorem 4 (Orientation soundness) Considering candidate sub-graphs in Table 1, the four orientation rules are sound.

Based on the above definitions, we are now ready to define Meek functions. More specifically, let $G = (\mathbf{V}, \mathbf{E})$ be a PDAG with set of nodes \mathbf{V} and set of edges \mathbf{E} .

Definition 5 (Meek functions) We define M_i as a function that takes \mathbf{E} as input and returns the set of edges \mathbf{E}_{M_i} as the result of applying Meek rule i repeatedly on the corresponding candidate sub-graphs of Meek rule i in \mathbf{E} until no candidate sub-graph can be found and as a result no more edge can be oriented by this rule. In other words, we have: $M_i(\mathbf{E}) = \mathbf{E}_{M_i}$.

Remark 6 Since there exists no candidate sub-graph for Meek rule i in the result of $M_i(\mathbf{E})$, we have: $M_i(M_i(\mathbf{E})) = M_i(\mathbf{E})$.

Furthermore, in some parts of the paper, we overload the notation of Meek function M_1 with the second argument \mathbf{E}_f , which is called as “forbidden edges”. The overloaded function is in the form of $M_1(\mathbf{E}, \mathbf{E}_f)$. Forbidden edges are the edges that Meek function M_1 does not allow to orient those edges. We use this type of function M_1 for the recovering essential graphs in Section 5. In the following, we give examples of both types of function M_1 .

Example 1 Given a graph $G = (\mathbf{V}, \mathbf{E})$, where $\mathbf{V} = \{v_1, v_2, v_3, v_4\}$ and $\mathbf{E} = \{v_1 \rightarrow v_2, v_2 \rightarrow v_3, v_3 \rightarrow v_4\}$, we have:

$$\begin{aligned} M_1(\mathbf{E}) &= \{v_1 \rightarrow v_2, v_2 \rightarrow v_3, v_3 \rightarrow v_4\} \\ M_1(\mathbf{E}, \{v_2 \rightarrow v_3\}) &= \{v_1 \rightarrow v_2, v_2 \rightarrow v_3, v_3 \rightarrow v_4\} \end{aligned}$$

In practice, we need to apply multiple Meek functions to discover more edges’ orientation. Any combination of Meek rules can be considered. For example, we define $M_{123}(\mathbf{E}) = \mathbf{E}_{M_{123}}$ as a function that takes set of edges \mathbf{E} as input and returns the set of edges $\mathbf{E}_{M_{123}}$ as the result, by applying Meek rules 1, 2, and 3 in any order, repeatedly, till no new edge can be discovered by applying any of these Meek rules. Again, the function M_{123} has the property that $M_{123}(M_{123}(\mathbf{E})) = M_{123}(\mathbf{E})$.

Now, we define minimal PDAG (for short MPDAG), which is the minimal graph, in terms of number of oriented edges, for representing equivalence class of a DAG.

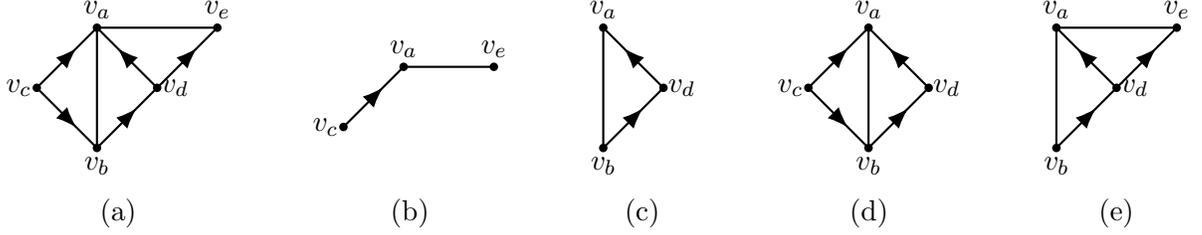


Figure 1: (a): A partially directed acyclic graph (G). (b): Induced candidate sub-graph of G for Meek rule 1. (c): Induced candidate sub-graph of G for Meek rule 2. (d): Induced candidate sub-graph of G for Meek rule 3. (e): Induced candidate sub-graph of G for Meek rule 4.

Definition 7 (Minimal PDAG) A minimal PDAG is a PDAG which is obtained by replacing some directed edges in a DAG with undirected one, those edges that do not participate in a v -structure. We denote a minimal PDAG by $G_o = (\mathbf{V}, \mathbf{E})$.

The following two theorems guarantee that applying Meek functions on the corresponding candidate sub-graphs are complete. One of these theorems is on the completeness of applying Meek functions on MPDAG, and the other one is on the completeness of applying Meek functions on MPDAG which has the some further oriented edge, which are selected from one of the DAGs in the corresponding MEC.

Definition 8 (Maximally oriented graph) A PDAG $G = (\mathbf{V}, \mathbf{E})$ in the MEC of a MPDAG G_o is a maximally oriented graph such that for each undirected edge $v_i - v_j$ in G , if there exist DAGs $D1 = (\mathbf{V}, \mathbf{E}_1)$ and $D2 = (\mathbf{V}, \mathbf{E}_2)$ in the MEC of G_o such that we have $v_i \rightarrow v_j \in \mathbf{E}_1$ and $v_j \rightarrow v_i \in \mathbf{E}_2$, where $\vec{\mathbf{E}} \subseteq \mathbf{E}_1$ and $\vec{\mathbf{E}} \subseteq \mathbf{E}_2$.

Theorem 9 (Orientation completeness on MPDAG) Let $G_o = (\mathbf{V}, \mathbf{E})$ be a MPDAG. The result of applying M_{123} on \mathbf{E} is a maximally oriented graph.

Theorem 10 (Orientation completeness) Let $G = (\mathbf{V}, \mathbf{E})$ be a MPDAG, including some further directed edges, which are selected from one of the DAGs in MEC of its MPDAG. The result of applying M_{1234} on \mathbf{E} is a maximally oriented graph with respect to $\vec{\mathbf{E}}$.

In the next part, we present some key properties of these functions.

3.3 Meek functions Properties

In the previous part, we defined four Meek functions inspired by Meek rules (Meek, 1995). In this part, we describe key properties of these functions. By exploiting these properties, we can accelerate executing Meek functions and utilize it as tools to design efficient algorithms for the problem of experiment design or checking the consistency of prior knowledge with the observational or interventional data (see Section 4). Before presenting the properties,

we define the partially directed and connected chordal graph (for short PCCG). In fact, PCCG is a representation of one of the chain components of an essential graph, where some of its edges are oriented according to one of the DAGs in the corresponding MEC.

Definition 11 (Partially directed and connected chordal graph) A PDAG $G = (\mathbf{V}, \mathbf{E})$ is a PCCG if:

- Skeleton of G is a UCCG.
- $\vec{\mathbf{E}}$ is not empty.
- There is a DAG in the MEC of G with essential graph equal to skeleton of G , which is also consistent with $\vec{\mathbf{E}}$ ³.

In the following lemma, we will provide some properties of Meek functions on PCCGs.

Lemma 12 (Meek functions properties on PCCGs) Let $G = (\mathbf{V}, \mathbf{E})$ be a PCCG. The following properties hold for Meek functions:

1. For any $\mathbf{T} \subseteq \mathbf{V}$, we have: $M_{1234}(\mathbf{E}[\mathbf{T}]) = M_{124}(\mathbf{E}[\mathbf{T}])$
2. For $M_i \in \{M_1, M_4, M_{14}\}$, we have: $M_i(\mathbf{E}) = \bigsqcup_{v_i \rightarrow v_j \in \vec{\mathbf{E}}} M_i(\{v_i \rightarrow v_j\} \sqcup \overline{\mathbf{E}})$.
3. $M_{124}(\mathbf{E}) = M_{14}(M_2(\mathbf{E}))$.
4. For any subset $\mathbf{S} = \{v_i \rightarrow v_k, v_k \rightarrow v_j, v_i - v_j\} \subseteq \mathbf{E}$, we have:

$$M_{14}(M_2(\mathbf{S}) \sqcup \mathbf{E} \setminus \mathbf{S}) = M_{14}(\mathbf{E}) \sqcup M_2(\mathbf{S}).$$

In the following, we provide some intuitions of the above properties. Based on property 1, function M_3 does not change the result when it is applied on a PCCG. This is due to the fact that there will be no v-structure in these type of graphs, and as a result there will be no candidate sub-graph for Meek rule 3. Thus, there is no need to apply this function on the mentioned graphs.

The property 2 states that all the information in order to reveal additional edges' orientations as a result of applying one of the functions M_1, M_4 or M_{14} are in the $\vec{\mathbf{E}}$. In other words, if an edge is oriented after applying Meek functions M_1, M_4 or M_{14} on set \mathbf{E} , consequently, this edge will be oriented by applying Meek functions M_1, M_4 or M_{14} on $e \sqcup \overline{\mathbf{E}}$, for some $e \in \vec{\mathbf{E}}$. This property has two appealing consequences. First, the result of applying mentioned Meek functions can be computed once and used multiple times such as in the problem of experiment design (see Section 4). Second, we can devise a dynamic programming method to apply Meek functions which reduces the running times (see Section 4).

By applying Meek function M_{124} on set of edges \mathbf{E} , some undirected edges will be oriented in \mathbf{E} . Property 3 asserts that we can recover all these edges, by consecutively applying function M_{14} after function M_2 on \mathbf{E} .

3. As the PCCG is a representation for partially oriented chain component, this constraint enforces to consider only DAGs with consistent skeleton and no v-structure.

A candidate sub-graph for Meek rule 2 is in the form of $\{v_i \rightarrow v_k, v_k \rightarrow v_j, v_i - v_j\}$. After applying function M_2 on this candidate sub-graph, $v_i - v_j$ edge will be oriented as $v_i \rightarrow v_j$. Property 4 states that discovering this edge's orientation cannot help in orienting further edges, even we apply function M_{14} on the set of edges including this directed edge.

In the next lemma, we express two properties for MPDAGs.

Lemma 13 (Meek functions properties on MPDAGs) *Let $G_o = (\mathbf{V}_o, \mathbf{E}_o)$ be a MPDAG. The following properties hold for Meek functions:*

1. $M_{1234}(\mathbf{E}_o) = M_{12}(M_3(\mathbf{E}_o))$.
2. $M_1(\mathbf{E}_o) = \bigsqcup_{v_i \rightarrow v_j \in \overrightarrow{\mathbf{E}_o}} M_1(\{v_i \rightarrow v_j\} \sqcup \overline{\mathbf{E}_o}, \overline{\mathbf{E}_o} \setminus \{v_i - v_j\})$.

Based on the Theorem 9, we know that Meek function M_4 is not necessary to be applied in order to obtain the essential graph. Thus, we can omit this function from the functions that must be applied. Moreover, candidate sub-graph for Meek rule 3 cannot be occurred as a result of applying other Meek functions. This is because there is a v-structure in the candidate sub-graph for Meek rule 3 and applying any of other Meek functions cannot generate a new v-structure. Hence, from property 1, we can imply that it is just needed to first apply function M_3 and then apply the other two functions.

Similar to Property 2 in Lemma 12, in Property 2 in Lemma 13, we use only directed edges in \mathbf{E}_o for orienting further edges in graph G_o . In this property, we take v-structures as the forbidden edges argument of Meek function M_1 to avoid orient them in the reverse direction.

After performing an intervention on an arbitrary node v in G , we can identify the direction of all the edges incident with node v (He and Geng, 2008; Eberhardt et al., 2005). We know that these edges are enough to recover the \mathcal{I} -essential graph (Hauser and Buhlmann, 2014). Moreover, according to Theorem 10, all the recoverable orientations can be identified by applying Meek functions. Furthermore, various possible orientations of incident edges yield different possible \mathcal{I} -essential graphs. For instance, Hauser and Buhlmann (2014) used this idea to compute a score function to select a node for performing an intervention.

In the following, we define intervened and connected chordal graph (for short ICCG). Suppose a node v in a UCCG $G = (\mathbf{V}, \mathbf{E})$ has been intervened on. After intervention, all the edges connected to this node, will be oriented. We denote the sets of nodes that have incoming edges toward and outgoing edges from node v by \mathbf{I} and \mathbf{O} , respectively. Furthermore, we have $\mathbf{O} = \text{neigh}(v) \setminus \mathbf{I}$.

Definition 14 (Intervened connected chordal graph) *We say PCCG $G = (\mathbf{V}, \mathbf{E})$ is an ICCG if there exists only one node $v \in \mathbf{V}$ such that $\overrightarrow{\mathbf{E}} = \mathbf{S}$, where $\mathbf{S} = \{v_i \rightarrow v | v_i \in \mathbf{I}\} \sqcup \{v \rightarrow v_o | v_o \in \mathbf{O}\}$ and $\mathbf{O} = \text{neigh}(v) \setminus \mathbf{I}$.*

An ICCG is a representation for \mathcal{I} -essential graph which is obtained after performing intervention on a node in a UCCG. This is because, based on the Hauser and Buhlmann (2014), knowing all the in-going edges toward intervened node, is a complete representation of \mathcal{I} -essential graph. According to Proposition 6 in Hauser and Buhlmann (2014), set \mathbf{I} is a subset of a maximal clique in the neighborhood of node v . An example of ICCG is shown in Example 2.

Remark 15 *An ICCG is a special case of PCCG. Thus, the properties for PCCGs hold for ICCGs.*

In the Lemma 16, we provide one properties for ICCGs.

Lemma 16 (Meek function property on ICCGs) *Let $G = (\mathbf{V}, \mathbf{E})$ be an ICCG. Denoting all the induced candidate sub-graphs for Meek rule 2 in $\mathbf{E}[\text{Neigh}(v)]$ by $\mathbf{C}_{M_2}(v)$, we have:*

$$M_2(\mathbf{E}) = \left(\bigsqcup_{\mathbf{C}_i \in \mathbf{C}_{M_2}(v)} M_2(\mathbf{C}_i) \right) \sqcup \mathbf{E}.$$

This property asserts that, after performing an intervention, the candidate sub-graphs that consists of edges in $\vec{\mathbf{E}}$ as their directed edges are sufficient to compute result of applying function M_2 . Hence, we can obtain the result of this part in two steps: (1) extracting candidate sub-graphs for function M_2 , (2) apply this function to only extracted candidate sub-graphs instead of applying to set of all edges. This procedure accelerates the execution of function M_2 .

In some applications of COL problems, such as those mentioned in Section 3, we need to apply function M_{124} on a given ICCG. This is because we do not have any v-structure in those types of graphs and as a result we do not need to apply Meek function M_3 . The conventional method is to repeatedly apply Meek rules 1, 2 and 4 on sub-graphs corresponding to these rules, till no more edge can be oriented. The following theorem states one of the key properties of Meek functions where applying function M_{124} can be decomposed to applying function M_{14} and M_2 . This theorem ensures that the result of decomposition method is the same as the conventional one of applying Meek rules.

Theorem 17 (Decomposition of applying M_{124} on ICCGs) *Let $G = (\mathbf{V}, \mathbf{E})$ be an ICCG. Applying function M_{124} on set \mathbf{E} can be decomposed as follows:*

$$M_{124}(\mathbf{E}) = \left(\bigsqcup_{e \in \vec{\mathbf{E}}} M_{14}(\{e\} \sqcup \vec{\mathbf{E}}) \right) \sqcup \left(\bigsqcup_{\mathbf{C}_i \in \mathbf{C}_{M_2}(v)} M_2(\mathbf{C}_i) \right),$$

where $\mathbf{C}_{M_2}(v)$ is collection of all induced candidate sub-graphs for Meek rule 2 in the sub-graph $\mathbf{E}[\text{Neigh}(v)]$.

In the next section, we exploit these properties of Meek functions in solving COL problems in different applications.

4. Applications of COL Problems

In this section, we utilize Meek function properties in different problems such as experiment design or checking the edge orientation consistency with a prior knowledge. In particular, in Section 4.1, we introduce a new method that accelerates computing Meek functions. In Section 4.2, we propose a lower bound on number of edges that can be oriented as a result

of performing an intervention on a variable. This lower bound can be used as a measure for designing experiments. In the last part, given a graph with set of combined directed and undirected edges, we check whether directed edges in the given graph belong to a DAG or not.

4.1 Fast Computation of \mathcal{I} -essential Graphs

In this part, we propose a novel method for efficiently discovering the orientations of additional edges that can be identified after performing an intervention⁴. This method uses mentioned properties of Meek functions in the previous section. An \mathcal{I} -essential graph is characterized by oriented edges in the neighborhood of intervened node (Hauser and Buhlmann, 2014). In order to recover orientations of further edges, Meek functions must be applied. As we mentioned in the previous section, it suffices to apply Meek function M_{124} . Based on Theorem 17, we can decompose the output of Meek function M_{124} computation into two parts: (a) The output of applying Meek function M_{14} , (b) The output of applying Meek function M_2 . Hence, accelerating the computation of Meek functions M_{14} and M_2 will reduce the running time of executing Meek function M_{124} . In the following, we describe the procedure in which we accelerate these functions.

We first describe our method for applying Meek function M_{14} in an efficient manner. Let $G = (\mathbf{V}, \mathbf{E})$ be a PCCG. Based on property 2 of Lemma 12, we can imply that Meek function M_{14} can be computed by performing mixed edge union on the results of applying this function on graphs that each of them has only one directed edge. We take advantage of this property to accelerate this function. Let $v_s - v_d$ be an undirected edge in the set of edges $\overline{\mathbf{E}}$. For any edge $v_s - v_d \in \overline{\mathbf{E}}$, we define $DP[v_s \rightarrow v_d]$ as follows:

$$DP[v_s \rightarrow v_d] = M_{14}(\{v_s \rightarrow v_d\} \sqcup \overline{\mathbf{E}}). \quad (1)$$

In the following, we propose a dynamic programming method for computing the above function.

Proposition 18 *The following equation holds for DP function in (1):*

$$DP[v_s \rightarrow v_d] = \left(\bigsqcup_{v_l \rightarrow v_k \in \overline{\mathbf{E}}'} DP[v_l \rightarrow v_k] \right) \sqcup \{v_s \rightarrow v_d\},$$

where $\mathbf{E}' = M_{14}(\{v_s \rightarrow v_d\} \sqcup \overline{\mathbf{E}}[Neigh(v_d)]) \setminus \{v_s \rightarrow v_d\}$.

Proposition 18 suggests a method to compute the result of applying Meek function M_{14} on a graph with one directed edge based on a dynamic programming method. According to Proposition 18, we need to apply Meek function M_{14} results in the neighborhood of v_d . Hence, it suffices to check two candidate sub-graphs for Meek rules 1 and 4 in the neighborhood of v_d for further edges orientation.

Figure 2 illustrates these two sub-graphs. Note that there is no edge between nodes v_s and v_j . So, in order to find candidate sub-graphs that are shown in Figure 2, we attempt

4. Similar to the method proposed in this part, we can accelerate the procedure of obtaining the essential graph. We will describe this method in Section 5.

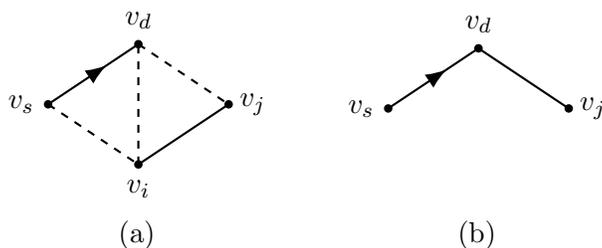


Figure 2: (a): Candidate sub-graph for Meek rule 4. (b): Candidate sub-graph for Meek rule 1.

to find some node $v_j \notin \text{neigh}(v_s)$. In the following, we design an algorithm that computes and stores the DP results using these sub-graphs.

In Algorithm 1, we suggest a recursive implementation that computes and stores DP results. This algorithm shows how one null entry of DP table can be filled. We first initialize $DP[v_s \rightarrow v_d]$ with $v_s \rightarrow v_d$ in Line 4. According to the sub-graphs in Figure 2, there is no edge between nodes v_s and v_j . Thus, we search for node $v_j \in \text{neigh}(v_d) \setminus \text{neigh}(v_s)$ in Line 5. For any such node v_j , using Meek function M_1 , we must orient the edge $v_d - v_j$ as $v_d \rightarrow v_j$. For the next step, we must compute the union of $DP[v_s \rightarrow v_d]$ and $DP[v_d \rightarrow v_j]$. In Lines 6-8, we check whether the entry $v_d \rightarrow v_j$ in DP is null or not. If DP is already calculated, we can compute the union. Otherwise, we go through another function call to calculate this entry. Lines 9-12 correspond to the case in Figure 2(a). In this case, we search for a node v_i in the neighborhood of all three nodes v_s , v_d and v_j . For any such node v_i , using Meek function M_4 , we must orient the edge $v_i - v_j$ as $v_i \rightarrow v_j$ and then compute the union of $DP[v_s \rightarrow v_d]$ and $DP[v_i \rightarrow v_j]$. After all these recursive function calls are completed, the DP result for $v_s \rightarrow v_d$ will be available. Note that in the procedure of obtaining one entry of DP table, multiple indices of this table might be filled.

Theorem 19 *Computational complexity of filling all entries of DP table using Algorithm 1 is in the order of $\mathcal{O}(|\mathbf{E}|\Delta^2)$, where Δ is the maximum degree in the graph.*

Since we compute DP for each edge in both directions, DP table size is two times of the number of edges. In the rest of this paper, we will assume that DP entries are available. In the next section, we will see that having DP table is helpful in applications that have high-demands of applying Meek functions M_{14} .

Having the DP table, now we show how it can be utilized to apply M_{14} function on a graph whose some edges are oriented as a result of intervention on a node. In the following lemma, we will show that the result of applying M_{14} can be accelerated by both using already calculated DP table and also taking advantage of property 2 in Lemma 12.

Lemma 20 (Applying M_{14} on PCCGs) *Let $G = (\mathbf{V}, \mathbf{E})$ be a ICCG, which is obtained after intervention on node v . The result of applying M_{14} on \mathbf{E} can be obtained as follows:*

$$M_{14}(\mathbf{E}) = \bigsqcup_{v_l \rightarrow v_k \in \mathbf{S}'} DP[v_l \rightarrow v_k],$$

where $\mathbf{S}' = \{v_i \rightarrow v | v_i \in \mathbf{I}\} \sqcup \{v \rightarrow v_o | v_o \in \mathbf{O}, \mathbf{I} \subseteq \text{neigh}(v_o)\}$.

Algorithm 1 Computation of $M_{14}(\{v_s \rightarrow v_d\} \sqcup \bar{\mathbf{E}})$

```

1: Input:  $G = (\mathbf{V}, \mathbf{E})$ , a PCCG
2: Output: Oriented edges as a result of applying function  $M_{14}$  on  $\{v_s \rightarrow v_d\} \sqcup \bar{\mathbf{E}}$  which
   is equal to  $DP[v_s \rightarrow v_d]$ 
3: function OrientOneEdge( $DP, v_s, v_d$ )
4:    $DP[v_s \rightarrow v_d] \leftarrow \{v_s \rightarrow v_d\}$ 
5:   for  $v_j \in \text{neigh}(v_d) \setminus \text{neigh}(v_s)$  do
6:     if  $DP[v_d \rightarrow v_j] = \text{NULL}$  then
7:        $DP \leftarrow \text{OrientOneEdge}(DP, v_d, v_j)$ 
8:      $DP[v_s \rightarrow v_d] \leftarrow DP[v_s \rightarrow v_d] \sqcup DP[v_d \rightarrow v_j]$ 
9:     for  $v_i \in \text{neigh}(v_s) \cap \text{neigh}(v_d) \cap \text{neigh}(v_j)$  do
10:      if  $DP[v_i \rightarrow v_j] = \text{NULL}$  then
11:         $DP \leftarrow \text{OrientOneEdge}(DP, v_i, v_j)$ 
12:       $DP[v_s \rightarrow v_d] \leftarrow DP[v_s \rightarrow v_d] \sqcup DP[v_i \rightarrow v_j]$ 
13: Return  $DP$ 

```

Based on the previous lemma, we can obtain the result of Meek function M_{14} by merely combining some entries of DP table. Thus, for further acceleration of function M_{124} , it suffices to accelerate applying Meek function M_2 . Using properties 3 and 4 in Lemma 12 allows us to obtain the result of Meek function M_2 on a graph without explicitly applying it. The next theorem suggests a fast method to discover new edges' orientations after applying Meek function M_{124} .

Theorem 21 (Applying M_{124} to obtain \mathcal{I} -essential graph) *Let graph $G = (\mathbf{V}, \mathbf{E})$ be an ICCG, which is obtained after intervention on node v . The result of applying M_{124} on set \mathbf{E} can be obtained as follows:*

$$M_{124}(\mathbf{E}) = \left\{ \bigsqcup_{v_l \rightarrow v_k \in \mathbf{S}'} DP[v_l \rightarrow v_k] \right\} \sqcup \{v_i \rightarrow v_o | v_i \in \mathbf{I}, v_o \in \mathbf{V}_c\}$$

where $\mathbf{S}' = \{v_i \rightarrow v | v_i \in \mathbf{I}\} \sqcup \{v \rightarrow v_o | v_o \in \mathbf{V}_c\}$, and $\mathbf{V}_c = \{v_o | v_o \in \mathbf{O}, \mathbf{I} \subseteq \text{neigh}(v_o)\}$.

Example 2 *Figure 3(a) illustrates a UCCG $G = (\mathbf{V}, \mathbf{E})$. Assume that an intervention was performed on node v_3 and the ICCG in Figure 3(b) was obtained. Entries of DP table for all possible edges' orientations are given in Figure 3(c). From the graph, we have $\mathbf{I} = \{v_4\}$, $\mathbf{O} = \{v_2\}$ and $\mathbf{S} = \{v_4 \rightarrow v_3, v_3 \rightarrow v_2\}$. Furthermore, we have $\mathbf{S}' = \mathbf{S}$. Therefore, according to Theorem 21, the result of applying Meek function M_{124} can be obtained as follows:*

$$\begin{aligned}
M_{124}(\mathbf{E} \sqcup \mathbf{S}) &= \left(\bigsqcup_{v_l \rightarrow v_k \in \mathbf{S}} DP[v_l \rightarrow v_k] \right) \sqcup \{v_4 \rightarrow v_2\} \\
&= \{v_3 \rightarrow v_2, v_4 \rightarrow v_1, v_2 \rightarrow v_1, v_1 \rightarrow v_5, v_4 \rightarrow v_3, v_4 \rightarrow v_2\}.
\end{aligned}$$



Index	DP Value	Index	DP Value
$v_2 \rightarrow v_1$	$\{v_2 \rightarrow v_1, v_1 \rightarrow v_5, v_4 \rightarrow v_1, v_1 \rightarrow v_5\}$	$v_1 \rightarrow v_2$	$\{v_1 \rightarrow v_2, v_2 \rightarrow v_3, v_4 \rightarrow v_3\}$
$v_4 \rightarrow v_1$	$\{v_4 \rightarrow v_1, v_1 \rightarrow v_5\}$	$v_1 \rightarrow v_4$	$\{v_1 \rightarrow v_4, v_2 \rightarrow v_3, v_4 \rightarrow v_3\}$
$v_3 \rightarrow v_2$	$\{v_3 \rightarrow v_2, v_4 \rightarrow v_1, v_2 \rightarrow v_1, v_1 \rightarrow v_5\}$	$v_2 \rightarrow v_3$	$\{v_2 \rightarrow v_3\}$
$v_4 \rightarrow v_2$	$\{v_4 \rightarrow v_2\}$	$v_2 \rightarrow v_4$	$\{v_2 \rightarrow v_4\}$
$v_4 \rightarrow v_3$	$\{v_4 \rightarrow v_3\}$	$v_3 \rightarrow v_4$	$\{v_3 \rightarrow v_4, v_4 \rightarrow v_1, v_1 \rightarrow v_5, v_2 \rightarrow v_1\}$
$v_5 \rightarrow v_1$	$\{v_5 \rightarrow v_1, v_1 \rightarrow v_2, v_2 \rightarrow v_3, v_1 \rightarrow v_4, v_4 \rightarrow v_3\}$	$v_1 \rightarrow v_5$	$\{v_1 \rightarrow v_5\}$

(c)

Figure 3: (a): UCCG $G = (\mathbf{V}, \mathbf{E})$. (b): Obtained ICCG after intervention on node v_3 . (c): DP values with respect to graph G .

4.2 Lower Bound on the Number of Discovered Edges' Orientations

In the literature of experiment design, several objective functions have been proposed to determine which node should be intervened on in order to recover as many edges' orientations as possible. One of the main objective function is in the form of minimax where we try to find that for each node how many edges' orientations will be discovered in the worst case scenario if we decide to intervene on that node. A naive solution is to consider all the possible orientations for the edges incident with a node and then compute the number of oriented edges can be discovered in the worst case for each of these cases. However, this solution may become very time consuming and it cannot be applied in medium-size or large graphs. Here, we propose a method to calculate a lower bound on number of edges that can be oriented as a result of intervening on each node.

We consider a UCCG $G = (\mathbf{V}, \mathbf{E})$ with set of nodes \mathbf{V} and set of edges \mathbf{E} . We denote the set of all maximal cliques in the neighborhood of a node $v \in \mathbf{V}$ by $\mathbf{C}(v)$. For computing the lower bound on a node such as v , we partition all the possible edges orientations that are connected to this node, into two cases: (1) There exists a maximal clique $\mathbf{C}_k \in \mathbf{C}(v)$, where at least one edge from a node $v_i \in \mathbf{C}_k$ is in-going toward node v and all the edges between node v and $neigh(v) \setminus \mathbf{C}_k$ are outgoing from node v , (2) All edges that are connected to node v are outgoing from this node. For the first case, we compute a lower bound for each maximal clique $\mathbf{C}_k \in \mathbf{C}(v)$ which means how many edges, at least, will be oriented after performing intervention on node v , considering edges' orientations in this case. For the second case, we obtain the true value of number of edges that can oriented. Finally, we compute the lower bound based on these two cases.

Here, first we describe the lower bound on a maximal clique. We will obtain the lower bound on a maximal clique by partitioning the problem into two settings: (a) computing number of edges that will be oriented when all edges are from maximal clique to node v . We denote this number by L_I . (b) computing a lower bound on the number of edges when at least one edge is from a node in the maximal clique to node v and at least one edge is oriented in the reverse direction, i.e., from node v to a node in the maximal clique. We denote this lower bound by L_C .

For the first setting, we can exactly find the number of oriented edges as a result of intervention. In the second setting, there exists an edge $v_i \rightarrow v$ that is from node $v_i \in \mathbf{C}_k$ toward node v and an edge $v \rightarrow v_o$ that is from node v to a node $v_o \in \mathbf{C}_k$. Consider the set of all in-going edges for the nodes $v_i \in \mathbf{V}$ and all out-going edges for the nodes $v_o \in \mathbf{V}$. Every edge in this set will be separately considered for orienting further edges and other edges in this set do not contribute to this process. Next, by post-processing these results, we obtain a lower for this maximal clique \mathbf{C}_k .

In order to compute L_C , we first obtain the set of edges that are oriented as a result of applying Meek functions on edges from node v to nodes in $neigh(v) \setminus \mathbf{C}_k$, and we denote this set by \mathbf{R} . Next, we obtain a lower bound on the number of edges that can be oriented, when we know there exist exactly j number of edges from nodes in clique \mathbf{C}_k to node v , excluding those that are in set R . We denote this lower bound by P_j . For computing P_j , we consider all possible cases that there are exactly j number of edges from clique \mathbf{C}_k to node v . Let \mathbf{I} be the set of these edges in one of these cases where $|\mathbf{I}| = j$. Now, for any $v_i \in \mathbf{I}$, we consider the orientations of edges from v to $neigh(v) \setminus \mathbf{C}_k$ and $v_i \rightarrow v$ as the input and apply Meek functions in order to recover the orientations of further edges. By removing the set \mathbf{R} from the resulted oriented edges, we can deduce the set of edges that can be oriented based on the edge $v_i \rightarrow v$. We repeat this procedure for any node $v_i \in \mathbf{I}$, and by considering the best case (i.e., picking the case resulting in maximum number of directed edges), we can obtain a lower bound on the number of edges that can be oriented by the whole edges $\{v_i \rightarrow v | v_i \in \mathbf{I}\}$. This is due to the fact that we just use the orientation of one of the edges in $\{v_i \rightarrow v | v_i \in \mathbf{I}\}$ in each of the cases for deriving the lower bound. Moreover, there might be multiple options for the set \mathbf{I} with size j . Hence, we need to consider all these options and pick the worst one as the lower bound P_j . Similarly, we can obtain a lower bound when we know there exist exactly j number of edges from v to nodes in clique \mathbf{C}_k . We denote this lower bound by Q_j .

Next, we try to orient edges in induced sub-graph $\mathbf{E}[\mathbf{C}_k]$. We can exactly compute the number of oriented edges in this set, if we assume that there exist exactly j edges from nodes in clique \mathbf{C}_k to node v . We will show in Lemma 22 that this value is equal to $j|\mathbf{C}_k - j|$. Note that having j ingoing edges to v means that we will have $|\mathbf{C}_k| - j$ outgoing edges from node v to clique \mathbf{C}_k .

Finally, we consider the edges that we did not take into account in the previous steps but they will be oriented as a result of intervention. In particular, after performing intervention on v , all the edges between nodes in clique \mathbf{C}_k and node v will be oriented. We consider one ingoing edge toward v in computing P_j and one outgoing edge from v in computing Q_j . Thus, we need to consider $|\mathbf{C}_k - 2|$ remaining oriented edges in our lower bound.

In the following, based on P_j 's and Q_j 's, we present a lower bound on number of edges that can be oriented by intervening on node v , constraint to have at least one ingoing edge toward node v .

Lemma 22 (Lower bound for the case of a maximal clique) *Consider the set of IC-CGs that can be obtained after intervention on node v , such that we have $\mathbf{I} \subseteq \mathbf{C}_k \in \mathbf{C}(v)$, $\mathbf{I} \neq \emptyset$ and $\mathbf{O} = \text{neigh}(v) \setminus \mathbf{C}_k$. The lower bound on number of oriented edges for all possible sets \mathbf{I} , $L(\mathbf{C}_k, v)$, can be obtained as follows:*

$$L(\mathbf{C}_k, v) = \min(L_I, L_C),$$

where,

$$\begin{aligned} L_I &= \left| \bigsqcup_{v_i \in \mathbf{C}_k} DP[v_i \rightarrow v] \right| \\ L_C &= |\mathbf{R}| + \min_{l \in \{1, \dots, |\mathbf{C}_k| - 1\}} P_l + Q_{|\mathbf{C}_k| - l} + |l|(|\mathbf{C}_k| - |l|) + (|\mathbf{C}_k| - 2) \\ \mathbf{R} &= \bigsqcup_{v_o \in \text{neigh}(v) \setminus \mathbf{C}_k} DP[v \rightarrow v_o] \\ P_j &= \min_{|\mathbf{I}|=j, \mathbf{I} \subseteq \mathbf{C}_k} \max_{v_i \in \mathbf{I}} |(DP[v_i \rightarrow v] \sqcup \{v_i \rightarrow v_o \mid v_o \in \text{neigh}(v_i) \cap \text{neigh}(v) \setminus \mathbf{C}_k\}) \setminus \mathbf{R}| \\ Q_j &= \min_{|\mathbf{O}|=j, \mathbf{O} \subseteq \mathbf{C}_k} \max_{v_o \in \mathbf{O}} |DP[v \rightarrow v_o] \setminus \mathbf{R}|. \end{aligned}$$

In Lemma 22, the minimum value of settings (a) and (b) is considered as a lower bound for the setting where the maximal clique \mathbf{C}_k has at least one outgoing edge toward the target node. In this bound, the computation of L_I is straight forward. In order to compute L_C , we need to consider all the cases of having $l = 1, \dots, |\mathbf{C}_k| - 1$ number of edges from the maximal clique \mathbf{C}_k to node v and compute P_l and $Q_{|\mathbf{C}_k| - l}$ (note that we have $|\mathbf{C}_k| - l$ number of edges from node v to the maximal clique). In each case, we also need to take into account the size of the set \mathbf{R} , the number of oriented edges in the induced sub-graph $E[\mathbf{C}_k]$, i.e., $|l|(|\mathbf{C}_k| - |l|)$, and the other edges that are oriented as the result of intervention on v , i.e., $|\mathbf{C}_k| - 2$. By picking the worst case scenario for different values of l , we can obtain the lower bound L_C .

In order to compute P_l and $Q_{|\mathbf{C}_k| - l}$, we can enumerate all the possible sets \mathbf{I} of size l which might be time consuming. Instead, we present Algorithm 2 which can compute $L(\mathbf{C}_k, v)$ in an efficient manner. In Line 5, we compute L_I and we compute set \mathbf{R} in Line 6. In Lemma 22, we need to compute the number of oriented edges as a result of applying Meek function M_{14} in the worst case for two scenarios; In the first scenario, only one directed edge exists from a node in maximal clique \mathbf{C}_k to node v while in the second scenario, only one directed edge exists from node v to maximal clique \mathbf{C}_k . In Line 9, we obtain the number of edges that can be oriented as a result of applying Meek function M_{14} where there exists a directed edge from maximal clique \mathbf{C}_k to node v , i.e., $DP[v_r \rightarrow v]$. In Line 10, we obtain the number of edges that can be oriented as a result of applying Meek function M_{14} where there exists a directed edge from node v toward maximal clique \mathbf{C}_k , i.e., $DP[v \rightarrow v_r]$. In Line 11, we sort the obtained values of P and Q in the ascending order. In Lines 12-13, we

search for minimum number of oriented edges where the number of ingoing edges toward node v can be varied from 1 to $|\mathbf{C}_k| - 1$. Selecting l -th item from sorted variables P and Q is the same as considering values of P_l and Q_l in Lemma 22, respectively. To see why this is true, note that for achieving the lower bound P_l , we just need to consider l ingoing edges toward node v corresponding to the l lowest values in array P as the set \mathbf{I} . In the same vein, it can be seen that l -th entry of array Q is equal to Q_l .

Algorithm 2 Computation of maximal clique lower bound

```

1: Input: A UCCG  $G = (\mathbf{V}, \mathbf{E})$ 
2: Output: lower bound on number of oriented edges after applying Meek functions
3: function  $L(G, \mathbf{C}_k, v, DP)$ 
4:    $r \leftarrow 0$ 
5:    $\mathcal{L} \leftarrow \left| \bigsqcup_{v_i \in \mathbf{C}_k} DP[v_i \rightarrow v] \right|$ 
6:    $\mathbf{R} = \bigsqcup_{v_o \in \text{neigh}(v) \setminus \mathbf{C}_k} DP[v \rightarrow v_o]$ 
7:   for  $v_r \in \mathbf{C}_k$  do
8:      $r \leftarrow r + 1$ 
9:      $P[r] = \left| \left( DP[v_r \rightarrow v] \bigsqcup_{v_o \in \text{neigh}(v_i) \cap \text{neigh}(v) \setminus \mathbf{C}_k} \{v_r \rightarrow v_o\} \right) \setminus \mathbf{R} \right|$ 
10:     $Q[r] = |DP[v \rightarrow v_r] \setminus \mathbf{R}|$ 
11:     $P \leftarrow \text{sort}(P), Q \leftarrow \text{sort}(Q)$ 
12:    for  $l \leftarrow 1 : |\mathbf{C}_k| - 1$  do
13:       $\mathcal{L} = \min(\mathcal{L}, |\mathbf{R}| + P[l] + Q[|\mathbf{C}_k| - l] + l(|\mathbf{C}_k| - l)) + (|\mathbf{C}_k| - 2)$ 
14: Return  $\mathcal{L}$ 

```

Remark 23 *Having access to entries of DP table, the computational complexity of Algorithm 2 for a node v and maximal clique \mathbf{C}_k is $\mathcal{O}(\omega \log \omega)$, where ω is the clique number.*

In the following, we propose a lower bound on number of oriented edges as a result of performing an intervention on a node. We do not consider any assumption on edges' orientations. Possible orientations of edges incident with the intervened node can be partitioned into two groups: In the first group, all the edges in the neighborhood of the target node are outgoing from it. In the second group, there is an edge (or possibly edges) from a maximal clique to the target node. In Lemma 22, we investigated the lower bound for a maximal clique with at least one ingoing edge toward the target node. Hence, the lower bound in second group can be achieved by calculating the minimum value of lower bounds for all maximal cliques in the neighborhood of the target node. Finally, the lower bound on a target node can be obtained as given in the following lemma.

Theorem 24 (Lower bound for the case of a node) *The lower bound on the number of edges that will be oriented after performing an intervention on node v , $L(v)$, in an arbitrary UCCG G can be obtained as follows:*

$$L(v) = \min \left(\left| \bigsqcup_{v_o \in \text{neigh}(v)} DP[v \rightarrow v_o] \right|, \min_{\mathbf{C}_k \in \mathbf{C}(v)} L(\mathbf{C}_k, v) \right).$$

Remark 25 *The computational complexity of lower bound in Theorem 24 for a node v is $\mathcal{O}(M\omega \log \omega)$, where ω is the clique number, and M is the maximum number of maximal cliques in the neighborhood of a node.*

4.3 Orientation Consistency

In this part, we assume that there is an expert who reveals the orientations of some undirected edges in a PDAG. This prior knowledge can also be obtained from a partial ordering on nodes (Hauser and Buhlmann, 2012; Wang et al., 2017) or restricting the causal relationships to a certain model (Eigenmann et al., 2017; Hoyer et al., 2012; Rothenhausler et al., 2018). We will take advantage of Meek functions' properties in the previous section to validate the correctness of these orientations in an efficient manner.

Suppose that we are trying to recover the underlying ground truth graph, and our identification algorithm discovered a PCCG. In this stage, an expert suggests to orient some further undirected edges in the obtained PCCG based on a domain knowledge. We are interested in checking whether the combination of already directed edges and recently added orientations are consistent or not. As underlying ground truth graph is a DAG, we just need to find a DAG being compatible with all the given directed edges. To do so, in the following, we first provide some definitions.

Definition 26 (Consistent edges) *Consider a graph $G = (\mathbf{V}, \mathbf{E})$ with set of nodes \mathbf{V} and set of edges \mathbf{E} . We say that in a graph with skeleton $\overline{\mathbf{E}}$, two directed edges $v_i \rightarrow v_j, v_r \rightarrow v_t$ in $\overrightarrow{\mathbf{E}}$ are consistent edges if for any $v_k \rightarrow v_l \in DP[v_i \rightarrow v_j]$, we have: $v_l \rightarrow v_k \notin DP[v_r \rightarrow v_t]$.*

Definition 27 (Consistent set) *Consider a PCCG $G = (\mathbf{V}, \mathbf{E})$ with set of nodes \mathbf{V} and set of edges \mathbf{E} . We say that the set of directed edges $\overrightarrow{\mathbf{E}}$ in structure $\overline{\mathbf{E}}$ is a consistent set if each pair of directed edges in this set are consistent.*

For a DAG G with the set of edges \mathbf{E} , any set of $\mathbf{E}' \subseteq \overrightarrow{\mathbf{E}}$ is a consistent set on a graph with structure $\overline{\mathbf{E}}$, where there is no v-structure in G . This is because applying Meek function M_{14} on $\mathbf{E}' \sqcup \overline{\mathbf{E}}$ does not make any conflict in edges orientation.

Now, we will use the definition of edge consistency to validate the domain knowledge of the expert. Consider a graph that consists of a combination of directed and undirected edges. The following theorem presents two conditions to check whether these directed edges are subset or equal to edges in a directed acyclic graph or not.

Theorem 28 *Consider a PCCG $G = (\mathbf{V}, \mathbf{E})$ with set of nodes \mathbf{V} and set of edges \mathbf{E} . There exists a DAG $G' = (\mathbf{V}', \mathbf{E}')$ which is in the MEC of graph G , such that $\overrightarrow{\mathbf{E}} \subseteq \overrightarrow{\mathbf{E}'}$ and $\overline{\mathbf{E}} = \overline{\mathbf{E}'}$ if and only if:*

- *There is no directed cycle in \mathbf{E} .*
- *\mathbf{E} is a consistent set.*

We can utilize Theorem 28 to validate the domain knowledge of the expert. To do so, we check whether any pair of directed edges in \mathbf{E} are consistent from the DP table. For detecting a directed cycle, we can execute DFS algorithm from all vertices on a graph that is extracted from G by removing its undirected edges.

5. Experiments

In this section, we provide some experiments for various application of COL problems based on our results in previous sections. We propose different algorithms in each subsection in the following. We utilize DP table in designing almost all of these algorithms. Hence, we call the class of the proposed algorithms “Causal Orientation Learning with Dynamic programming (COLD)”. In our experiments, we define the edge density as the ratio of edges to the maximum number of possible edges in a graph of size n , i.e., $n(n - 1)/2$. Moreover, we define the average degree as the ratio of edges to the number of the nodes.

In all experiments, we generated chordal graphs by the method which has been presented in Markenzon et al. (2008). For each point in all figures, we averaged over 100 instances of the chordal graphs, otherwise we explicitly mention it. Our generated graphs are based on the similar networks given in bnlearn⁵. The codes for proposed methods in this part are available in <https://github.com/raminsafaeian/COLD>.

5.1 Recovering Essential Graphs

One of the well-known methods for obtaining the essential graph from the observational data is PC algorithm (Spirtes et al., 2000). This algorithm has three main steps: (1) performing conditional independence tests between random variables in order to construct skeleton (2) identifying v-structures from separation sets (3) discovering more edges’ orientations by applying Meek rules on graph including v-structures. According to Theorem 9, for further recovering edges’ orientations, it suffices to apply Meek rules 1, 2 and 3 in third step of PC algorithm. In the following we propose a method to accelerate obtaining essential graph.

Similar to our previous DP table construction in Algorithm 1, and considering Property 2 in Lemma 13, we propose DPO method for the case that there exist v-structures in the graph. We show how one entry of DPO table is filled in Algorithm 3. The main difference between this algorithm and what we proposed in Algorithm 1 is that the Meek function M_4 is removed and v-structures orientations are considered for further recovering other edges’ orientations. Using DPO table enables us to accelerate Meek function M_1 which in turn will accelerate execution time of getting the essential graph from the MPDAG.

Consider an MPDAG $G = (\mathbf{V}, \mathbf{E})$ with set of nodes \mathbf{V} and set of edges \mathbf{E} . We propose Algorithm 4 for discovering further edges’ orientations in order to obtain the essential graph. Based on Property 1 in Lemma 13, we apply Meek function M_3 once, in the beginning of algorithm in Line 7. In Lines 8-14, we repetitively orient further edges by applying Meek functions M_1 and M_2 . Applying Meek function M_1 is based on our DPO table in Lines 9-12. We call this algorithm “COLD (Essential)”.

We compared the execution time of our proposed COLD algorithm with the conventional method for the third step of PC algorithm. Conventional method is the one that we apply Meek rules in no particular order to orient further edges, until no more edge can be oriented. In order to have a fair comparison, we implemented a function for the applying Meek rules similar to the one in Kalisch et al. (2012) library in python and considered it as the conventional method. For generating MPDAGs, we first constructed random chordal graphs. Then, we use lexicographic BFS or LexBFS (Rose, 1970) to consider an ordering

5. <https://www.bnlearn.com/>

on the vertices and obtain a DAG. Then, we remove one of the edges such that at least one v-structure be created. Then, we will get the MPDAG from this generated DAG.

Algorithm 3 Computation of $M_1(\{v_s \rightarrow v_d\} \sqcup \overline{\mathbf{E}})$ using dynamic programming method, considering v-structures.

```

1: Input:  $G = (\mathbf{V}, \mathbf{E})$ , a MPDAG
2: Output: Oriented edges as a result of applying function  $M_1$  on  $\{v_s \rightarrow v_d\} \sqcup \mathbf{E}$  which
   is in  $DPO[v_s \rightarrow v_d]$ 
3:  $\mathbf{E}_v \leftarrow$  set of v-structure sub-graphs in  $\mathbf{E}$ 
4: function OrientOneEdge( $DPO, v_s, v_d, \mathbf{E}_v$ )
5:    $DPO[v_s \rightarrow v_d] \leftarrow \{v_s \rightarrow v_d\}$ 
6:   for  $v_j \in \text{neigh}(v_d) \setminus \text{neigh}(v_s)$  do
7:     if  $v_j \rightarrow v_d \notin \mathbf{E}_v$  then
8:       if  $DPO[v_d \rightarrow v_j] = \text{NULL}$  then
9:          $DPO \leftarrow \text{OrientOneEdge}(DPO, v_d, v_j, \mathbf{E}_v)$ 
10:       $DPO[v_s \rightarrow v_d] \leftarrow DPO[v_s \rightarrow v_d] \sqcup DPO[v_d \rightarrow v_j]$ 
11: Return  $DPO$ 

```

Algorithm 4 COLD (Essential)

```

1: Input: MPDAG graph  $G = (\mathbf{V}, \mathbf{E})$ 
2: Output: Essential graph of  $G$ 
3: Fill all  $DPO$  entries with  $\text{NULL}$  value
4: function ESSENTIAL( $G$ )
5:    $\mathbf{E}_v \leftarrow$  set of v-structure sub-graphs in  $\mathbf{E}$ 
6:    $\mathbf{E}_o \leftarrow \mathbf{E}$ 
7:    $\mathbf{E}_t = M_3(\mathbf{E})$ 
8:   while  $|\overline{\mathbf{E}}_t| > 0$  do
9:     for  $v_s \rightarrow v_d \in \overline{\mathbf{E}}_t$  do
10:      if  $DPO[v_s \rightarrow v_d] = \text{NULL}$  then
11:         $DPO \leftarrow \text{OrientOneEdge}(DPO, v_s, v_d, \mathbf{E}_v)$ 
12:       $\mathbf{E}_o = \mathbf{E}_o \sqcup DPO[v_s \rightarrow v_d]$ 
13:       $\mathbf{E}_t = M_2(\mathbf{E}_o) \setminus \overline{\mathbf{E}}_o$ 
14:   end
15: Return  $G = (\mathbf{V}, \mathbf{E}_o)$ 

```

Figure 4(a) shows the execution time of COLD (Essential) and conventional method versus number of edges for graphs with 3000 nodes. As can be seen, the COLD (Essential) is outperform the conventional method by a factor up to 20788. For instance, for a graph with 3000 nodes and 4000 edges, the execution time of COLD (Essential) is about 0.0335 seconds, while the execution time of conventional method is about 696.4 seconds. Comparison of these two methods for graphs with fixed average degree, 2.2, versus number of graph nodes is depicted in 4(b). Again, the proposed algorithm has much lower execution time with

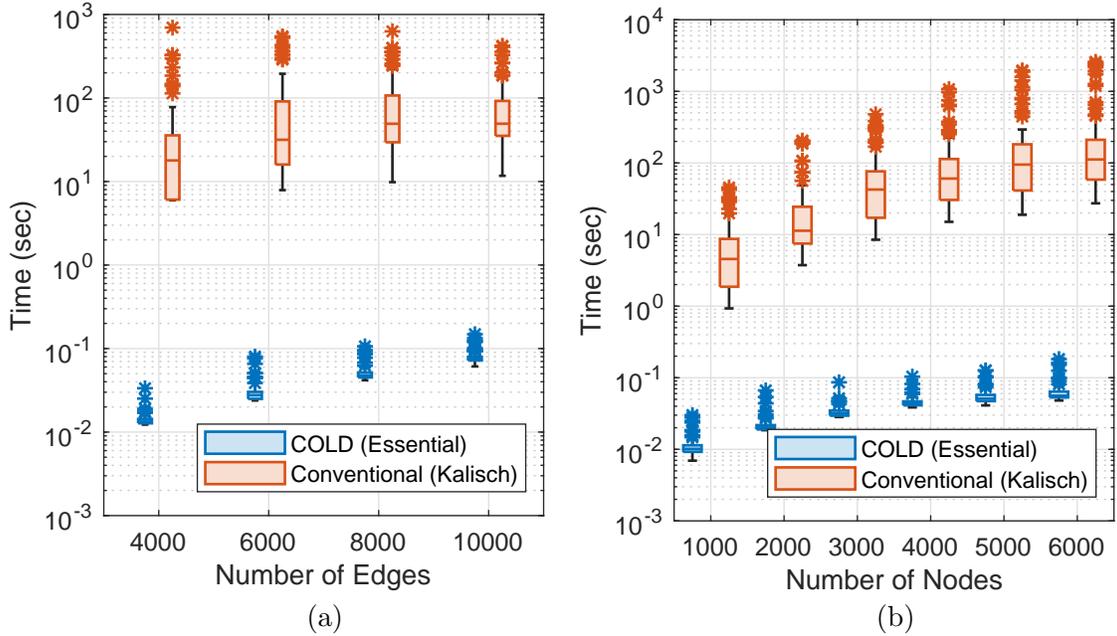


Figure 4: (a): Comparison between execution times of COLD (Essential) and conventional method, given a MPDAG, versus the number of edge for graphs with 3000 number of nodes (b): Comparison between execution times of COLD (Essential) and conventional method versus the number of nodes for graphs with average degree of 2.2.

respect to the conventional method. For instance, for graphs of size 6000, the speed up ratio is about 24910, where the execution times of COLD (Essential) and conventional method are 0.0994 and 2476 seconds, respectively.

5.2 Accelerating MEC Size Computation

The size of a MEC is the number of DAGs it contains. Main approach in previous work is to partition existing DAGs in an MEC in order to count them. Total number of DAGs can be computed by summation over the number of DAGs in each partition. There are different methods for such a partitioning. For instance, He et al. (2015) performed partitioning of DAGs in an MEC based on their roots⁶. Additionally, they proposed some closed-form formula to compute number of DAGs in an MEC when there is a specific relation between number of edges and number of nodes in the graph. They extended their work by considering some special structure “core graphs” in He and Yu (2016) to efficiently count the number of DAGs in an MEC. Ghassami et al. (2019) proposed to partition the MEC based on its clique tree representation. Additionally, they have proposed a dynamic programming method to store the computed size of each sub-graph to avoid multiple calculation of its size. Similar to Ghassami et al. (2019), Talvitie and Koivisto (2019) proposed a dynamic programming method for MEC size calculation. Recently, Teshnizi et al. (2020) proposed to compute

6. A node is defined as a root when all the edges are outgoing from this node toward its neighbors.

the MEC size by partitioning DAGs in MEC by all possible orientations for edges that are connected to a node.

We can take advantage of Meek functions properties to accelerate any previous method that uses Meek rules as a subroutine. Herein, as an example, we will accelerate MEC size computation method proposed by He et al. (2015). In this method, given a UCCG G , its size can be computed as follows:

$$Size(G) = \sum_{v \in \mathbf{V}} \prod_{C_i \in \mathcal{C}(G^v)} Size(C_i),$$

where $Size(G)$ is the size of MEC corresponding to G and G^v is a graph as a result of setting node v as a root in graph G . We utilize our proposed Meek function properties, such as Proposition 18 and Theorem 21, and also dynamic programming method proposed by Ghassami et al. (2019) to accelerate UCCG size calculation. We present our proposed algorithm for counting number of DAGs in an MEC as “COLD (MECSize)”, in Algorithm 5.

In Algorithm 5, Lines 7-11 is based on what have been proposed in He et al. (2015). In these lines, we check whether the UCCG satisfies a specific relation between number of nodes and edges. We can determine the MEC size of these graphs merely from number of nodes. If we cannot calculate MEC size based on these lines, we execute Lines 12-13 to know whether we have already computed the MEC size for this UCCG or not. If that is the case, we will use the stored value in the memory. Otherwise, we execute Lines 14-17 for MEC size computation. In Line 15, we obtain the edges that can be oriented as a result of considering a node v to be a root. In Line 16, we remove these directed edges from the graph and extract the chain components of the remaining graph. For each chain component, the size function will be called in Line 17. Finally, in Line 18, we store the computed size in memory for using in the next calls.

We compared COLD (MECSize) algorithm with the previous state of the art algorithm, LazyCount (Teshnizi et al., 2020) and a former work MemoMAMO (Talvitie and Koivisto, 2019). The size of graphs in this experiment is equal to 30. As can be seen in Figure 5, our proposed algorithm performs better than the others for a wide range of number of edges. Moreover, performance of LazyCount degrades by decreasing the number of edges. While performance of MemoMAMO degrades by increasing the number of edges. For instance, for graphs with 350 edges, the speed up ratios with respect to MemoMAMO and LazyCount are 2.88 and 1.83, respectively.

5.3 Acceleration of Experiment Design in Active Learning Setting

In active learning setting, we determine the nodes for next interventions after observing the results of previous interventions. Some previous work tried to use gathered data from previous interventions to design next intervention set based on a Bayesian method (Ness et al., 2017; Greenewald et al., 2019; Agrawal et al., 2019). For instance, Greenewald et al. (2019) proposed a method to search for a node that has the maximum posterior probability to be a root variable. This method is mainly suitable for graphs with tree structures. Some other work assumed that infinite samples are available after performing intervention (Hauser and Buhlmann, 2014; He and Geng, 2008; Kocaoglu et al., 2017; Shanmugam et al.,

Algorithm 5 COLD (MECSize) Algorithm

```

1: Input: UCCG  $G = (\mathbf{V}, \mathbf{E})$ 
2: Output:  $|MEC(G)|$ 
3: SizeDP is a storage indexed with set of nodes  $\mathbf{T}$  and is initialized by NULL for each
    $\mathbf{T} \subseteq \mathbf{V}$ 
4: function  $Count(G = (\mathbf{V}, \mathbf{E}))$ 
5:    $p \leftarrow |\mathbf{V}|$ 
6:   switch  $|\mathbf{E}|$  do
7:     case  $p - 1$  return  $p$ ;
8:     case  $p$  return  $2p$ ;
9:     case  $p(p - 1)/2 - 2$  return  $(p^2 - p - 4)(p - 3)!$ ;
10:    case  $p(p - 1)/2 - 1$  return  $2(p - 1)! - (p - 2)!$ ;
11:    case  $p(p - 1)/2$  return  $p!$ ;
12:   if  $\mathbf{V}$  is in SizeDP then
13:     return  $SizeDP[\mathbf{V}]$ 
14:   for each  $v$  in  $\mathbf{V}$  do
15:      $\mathbf{E}_r = \bigsqcup_{v_o \in \text{neigh}(v)} DP[v \rightarrow v_o]$ 
16:      $G' \leftarrow G = (\mathbf{V}, \overline{\mathbf{E}} \setminus \overline{\mathbf{E}}_r)$ 
17:      $S_v = \prod_{\mathcal{C}_i \in \mathcal{C}(G')} Count(\mathcal{C}_i)$ 
18:    $SizeDP[\mathbf{V}] = \sum_{v \in \mathbf{V}} S_v$ 
19:   return  $SizeDP[\mathbf{V}]$ 

```

2015; Ghassami et al., 2018). Therefore, the edges' orientations incident to the intervened variable can be exactly determined.

In the case of infinite samples, previous works have considered different objective functions for the problem of experiment design. He and Geng (2008) proposed to select candidate intervention nodes based on mutual information criterion. Another approach that has been proposed by this work is to select a node with maximum number of neighbors. Hauser and Buhlmann (2014) considered an objective function to select a node for intervention that results in orienting maximum number of edges in the worst case scenario after the intervention. More specifically, for a given UCCG $G = (\mathbf{V}, \mathbf{E})$, the following objective function is considered:

$$v^* \in \underset{v \in \mathbf{V}}{\operatorname{argmax}} \min_{\mathbf{I} \subseteq \mathbf{C}_k, \mathbf{C}_k \in \mathbf{C}(v)} |M_{124}(\mathbf{S} \sqcup \overline{\mathbf{E}})|,$$

where $\mathbf{S} = \{v_i \rightarrow v | v_i \in \mathbf{I}\} \sqcup \{v \rightarrow v_o | v_o \in \text{neigh}(v) \setminus \mathbf{I}\}$. We call this objective function, "MinMax" function. In MinMax function, for every node in the graph, the minimum number of oriented edges, in the worst case, will be computed when we intend to perform an intervention on that node. The possible edges' orientation after intervention will be partitioned by considering different ingoing edges toward the target node (Hauser and Buhlmann, 2014). We know that the ingoing edges must be a subset of a maximal clique. Thus, we search over all possible ingoing edges sets like \mathbf{I} in every maximal clique \mathbf{C}_k . For each of these orientations, we apply Meek function M_{124} to discover further edges' orientations. As mentioned earlier, using Meek function properties, we can accelerate any function that needs Meek

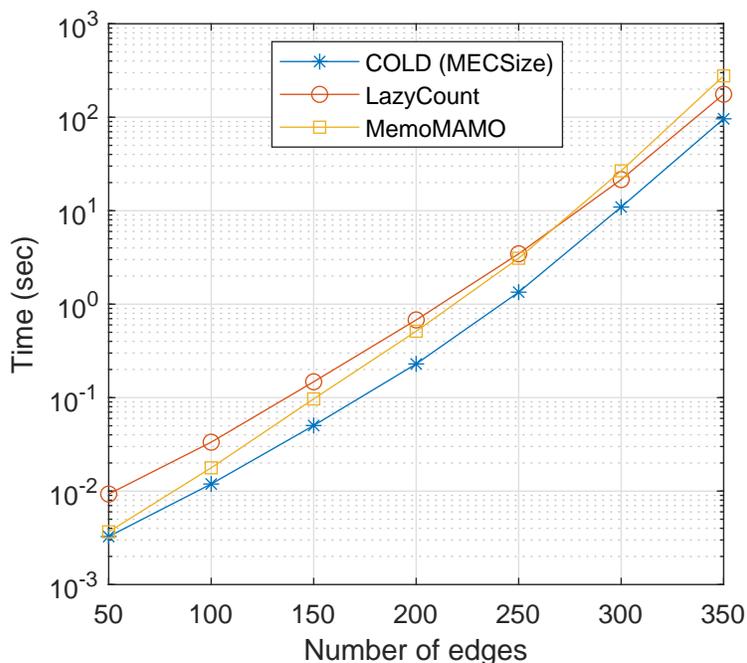


Figure 5: Running times of COLD (MECSize), LazyCount, and MemoMAMO for graphs with 30 number of nodes.

rules to be applied. In the following, we show that using our proposed method for computing Meek function results, we can accelerate the process of optimizing MinMax objective function.

We compared our proposed algorithm, which we call “COLD (MinMax)”, with LazyIter (Teshnizi et al., 2020) method, which is the state of art in solving MinMax problem. COLD (MinMax) uses DP tables and utilizes Theorem 21 to apply Meek rules, rather than applying conventional Meek rules. The execution times for these two methods are given in Figure 6. The curves are the average of execution times over 100 generated chordal graphs. Figure 6(a) shows that the execution time of finding the best node versus number of edges for graphs with 1000 number of nodes. Figure 6(b) shows the execution time of finding the best node versus number of node for graphs graphs with 1.1 average degree.

As can be seen in Figure 6, the execution time of our proposed algorithm is considerably less than the one for LazyIter Algorithm. As expected, we gain by avoiding from applying the enormous number of Meek rules using DP table.

5.4 Quality of the Proposed Lower Bound

In this section, we investigate the quality of our proposed lower bound with respect to number of edges that can be oriented in the worst case scenario for each node. We evaluated the lower bound in two cases: 1- The edge density is fixed and we increase the number of nodes, 2- The number of nodes is fixed and we increase the number of edges. We provide the summary of these simulation results for these two cases in Table 2 and 3, respectively. Note

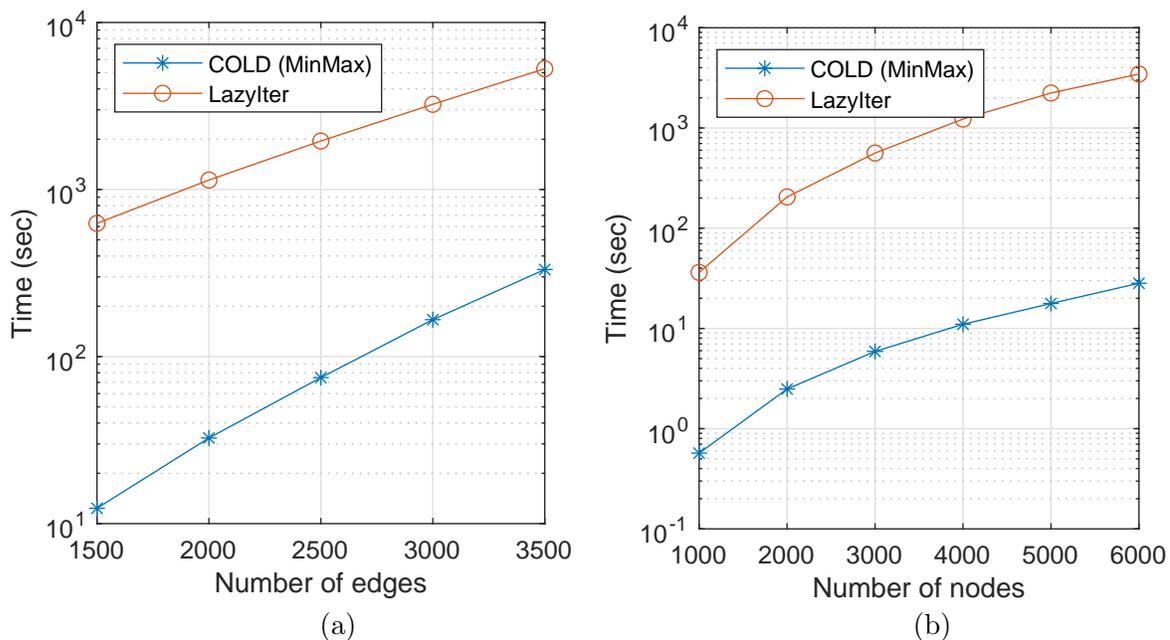


Figure 6: (a): Comparison between execution times of COLD (MinMax) and LazyIter (Teshnizi et al., 2020) versus number of edges for graphs with 1000 number of nodes (b): Comparison between execution times of COLD (MinMax) and LazyIter (Teshnizi et al., 2020) versus number of nodes for graphs with 1.1 average degree.

that for finding the minimum number of oriented edges in the worst case, we enumerated all consistent DAGs.

$ V $	Edge density	Exact %	Norm. avg. worst case	Mean gap	Avg. Time (Sec)
20	0.4	0.9915	0.0029	0.0245	0.0080
30	0.4	0.9827	0.0088	0.1190	0.0444
40	0.4	0.9722	0.0159	0.2843	0.1596
50	0.4	0.9708	0.0213	0.5328	0.4981
60	0.4	0.9627	0.0228	0.8143	1.2824

Table 2: Lower bound comparison with the true value for different number of oriented edges with fixed edge density

In these tables, the “exact %” column shows in how many cases, our proposed lower bound is equal to the true value. In “norm avg worst case” column, we considered the maximum difference between the true value and the proposed lower bound over all nodes for each graph, and then averaged over all graphs and divided it by the number of existed edges for that setting. In “mean gap” column, we computed the average difference between the true value and the lower bound for each node in all graphs. The last column shows the average execution time for computing the lower bounds for all nodes in each graph.

$ \mathbf{V} $	Edge density	Exact %	Norm. avg. worst case	Mean gap	Avg. Time (Sec)
40	0.1	0.9996	0.0002	0.0004	0.0064
40	0.2	0.9854	0.0071	0.0563	0.0319
40	0.3	0.9779	0.0120	0.1549	0.0843
40	0.4	0.9722	0.0159	0.2843	0.1596
40	0.5	0.9762	0.0157	0.3555	0.2605
40	0.6	0.9781	0.0129	0.4136	0.3654
40	0.7	0.9825	0.0104	0.4338	0.4671

Table 3: Lower bound comparison with the true value for different edge density with fixed number of nodes

The results in Table 2 and 3 show that in almost all cases, the lower bound is exactly equal to the true value. Moreover, in other few cases, the gap between the lower bound and the true value is negligible.

5.5 Intervention Design Based on the Lower Bound Criterion

In this part, we propose a new objective function to find the best node for intervention. This function reduces the computational complexity while preserving the number of interventions in order to fully identify the whole causal structure. Our objective function is defined as the following:

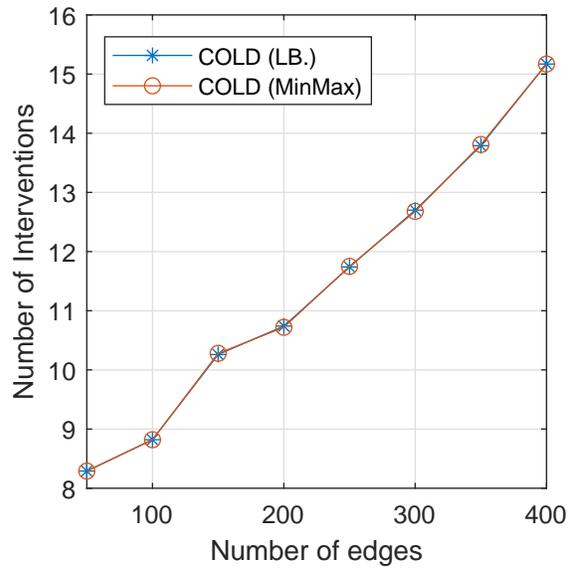
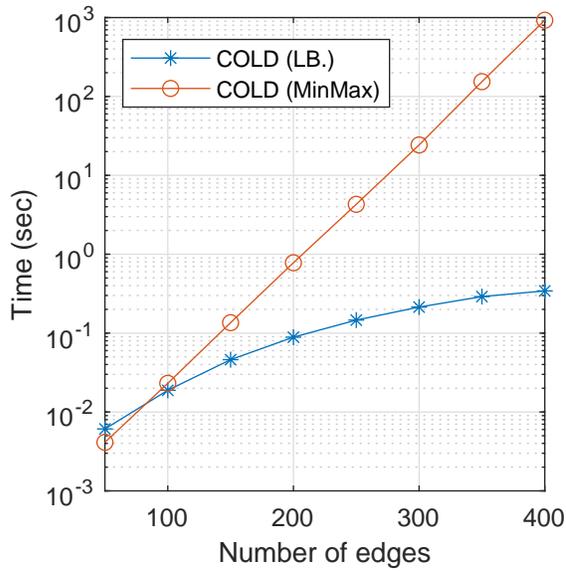
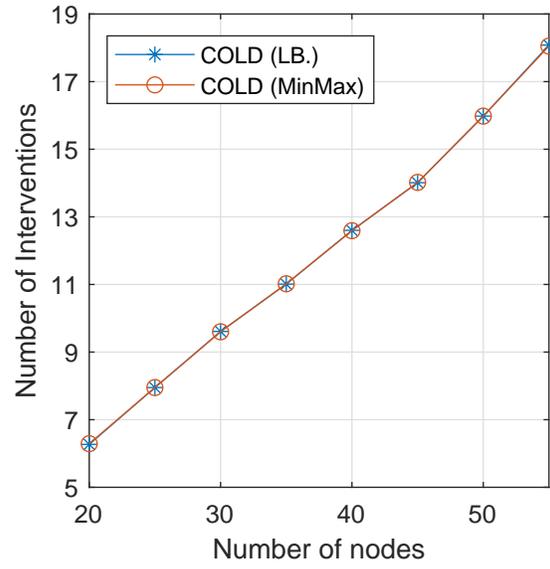
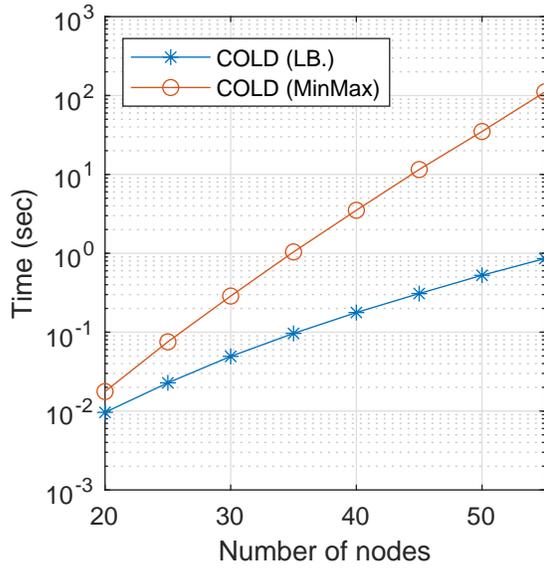
$$v^* \in \underset{v \in \mathbf{V}}{\operatorname{argmax}} L(v),$$

where the $L(v)$ is the lower bound on number of oriented edges after intervention on node v . Based on the above objective function we select a node for intervention that has maximum lower bound with respect to the other nodes.

In the previous part, we evaluated the quality of our proposed lower bound. The reason for using lower bound instead of the true value is that lower bound is a good estimator of it while saving the computation time. We compared our heuristic algorithm based on our lower bound which we call it “COLD (LB)” with the one that we proposed in Section 5.3, i.e., COLD (MinMax).

The result of comparison between two algorithms COLD (MinMax) and COLD (LB) is shown in Figure 7. We see that COLD (LB) is considerably faster than COLD (MinMax) in both cases of increasing number of nodes (Figure 7(a)) or increasing number of edges (Figure 7(c)). We plot the average number of interventions that is needed for full identification in Figure 7(b) and 7(d). We can see that in both of these plots, the average number of interventions for full identification is the same for both algorithms.

In this part, we evaluate our algorithm for the finite-sample case. Here, our main goal is to demonstrate how one can use the infinite sample based algorithms in the finite sample case. Thus, we just compared our algorithm with two baselines: 1- “Random Naive” algorithm, which always selects a random node for the next intervention from the set of all nodes, and 2- “Random Chordal” algorithm, which selects a random node from set of nodes



(c)

(d)

Figure 7: Comparison between COLD (LB) and COLD (MinMax) (a): execution time versus number of nodes for graphs with edge density of 0.35. (b): Average number of interventions for full identification versus number of nodes for graphs with edge density of 0.35. (c): execution times versus number of edges for graphs with 40 nodes. (d): Average number of interventions that is needed for full identification versus number of edges for graphs with 40 nodes.

that those nodes have at least one connected undirected edge in the obtained \mathcal{I} -essential graph after each intervention.

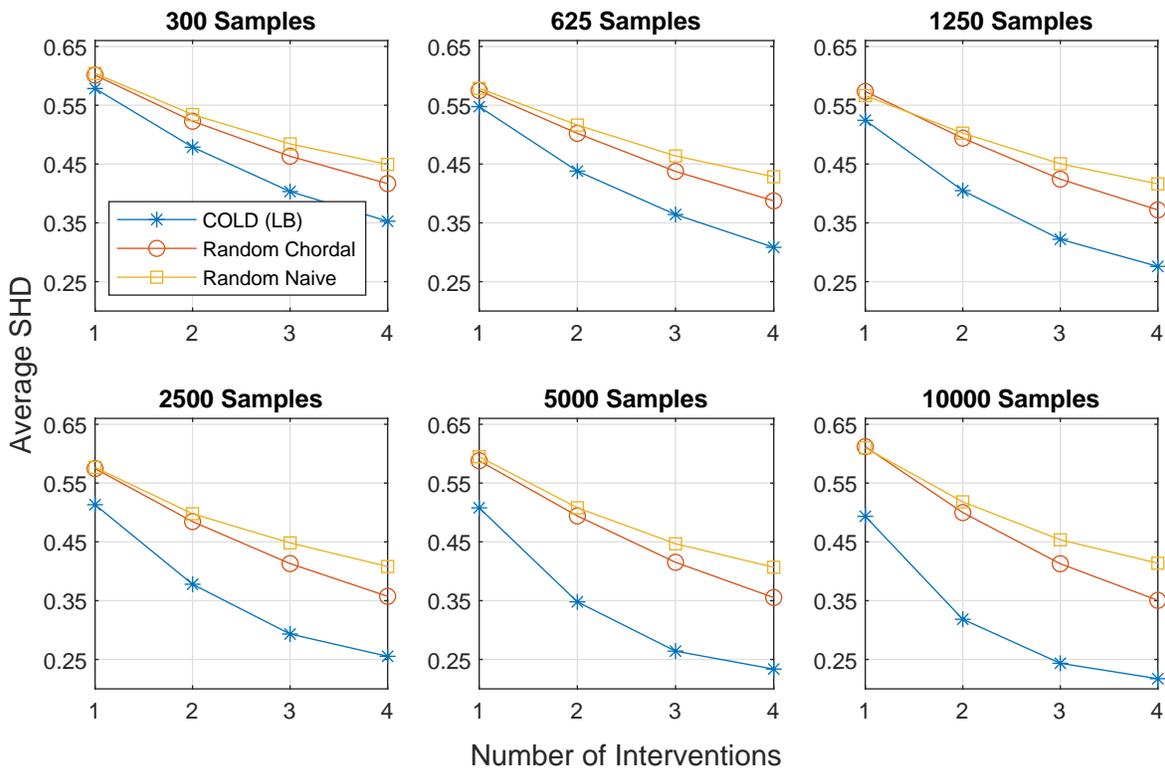


Figure 8: Average SHD values of COLD (LB), Random chordal, and Random Naive versus number of interventions for graphs with 25 nodes and 35 edges.

We considered 3000 randomly generated linear Gaussian causal model for graph with 25 nodes and 35 edges. All edges coefficients were sampled uniformly from $[-1.5, -0.5] \cup [0.5, 1.5]$ and error variances were uniformly sampled from $[0.01, .2]$. We consider a budget of at most four interventions. We compared our proposed COLD (LB) algorithm, with Random Naive and Random Chordal algorithms for the number of samples 300, 625, 1250, 2500, 5000, 10000. Figure 8 shows the result of these simulations. We use the structural hamming distance (SHD), which is defined in Brown et al. (2005), for comparing the identification rate for each of these algorithms. We divided the SHD value of each graph in each step of intervention by number of edges and averaged over all instances. Note that limited number of samples may cause to converge to a wrong DAG. In this stage, Random Chordal and COLD (LB) algorithms stop intervening on nodes. Thus, their SHD will remain unchanged after a number of interventions. In order to compare these algorithms with Random Naive algorithm, which has not any stopping rule, we consider the average SHD value in the last intervention before getting a DAG as the SHD value for the following steps of interventions. Although, the number of interventions in Random Naive algorithm is more than the ones in the other two algorithms, simulation results show that our COLD (LB) Algorithm outperforms baselines even in the small number of samples.

5.6 Practical Trick

In this part, we add a practical trick to enhance the acceleration of some of the applications that have been discussed in this section. In applications such as MinMax problem or obtaining the lower bound in previous part, we can stop the computation as early as we know that the selected node cannot be the desired solution. We call this “early stopping trick”. For example, suppose we have executed the lower bound algorithm on a part of nodes in the graph and obtain that the maximum lower bound among these nodes. We select another node such v to compute its corresponding lower bound. If in the process of computing some L_C , we find that the lower bound of the node v is less than what we have already obtained, we immediately terminate the process of this node and we do not consider it as our desired solution.

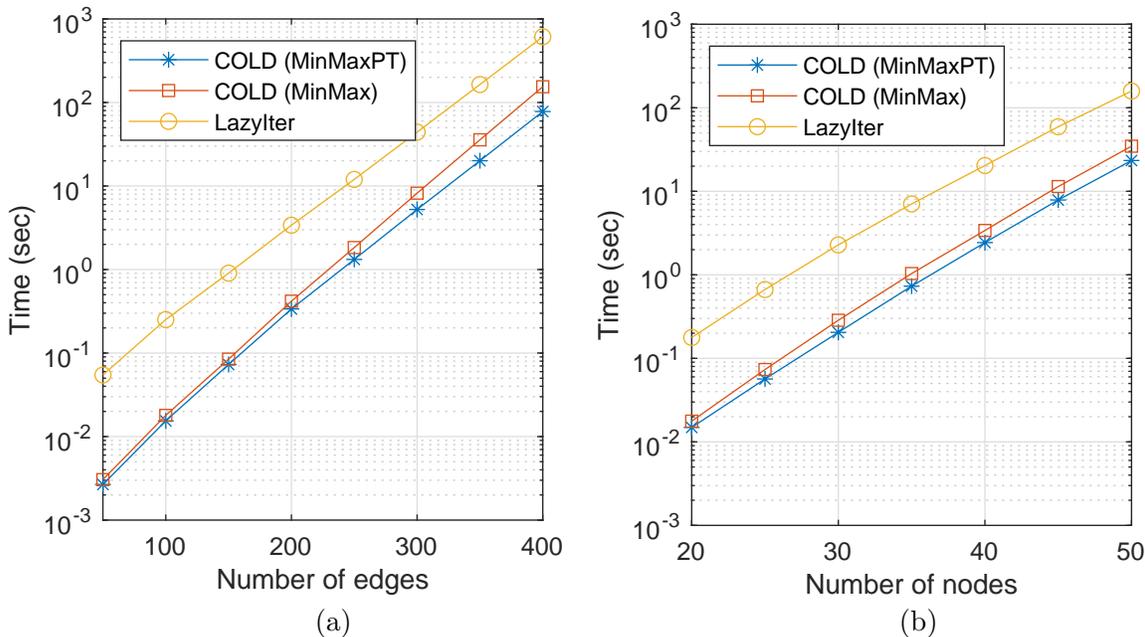


Figure 9: (a): Comparison between execution times of COLD (MinMaxPT), COLD (MinMax) and LazyIter (Teshnizi et al., 2020) versus number of edges for graphs with 40 nodes (b): Comparison between execution times of COLD (MinMaxPT), COLD (MinMax) and LazyIter (Teshnizi et al., 2020) versus number of nodes for graphs with edges density of 0.35.

In order to evaluate the gain we can get from this trick, we utilize it in selecting a target intervention by considering MinMax objective function in the active learning setting. We call the proposed algorithm using this trick “COLD (MinMaxPT)”. Here, we compare this method with COLD (MinMax) that has been proposed previously in this section. The execution times of these two algorithms are shown in Figure 9. In both cases, i.e., increasing number of edges with fixed number of nodes (Figure 9(a)) or increasing number of nodes with fixed edge density (Figure 9(b)), using practical trick improves the computation time for solving MinMax problem.

6. Conclusion

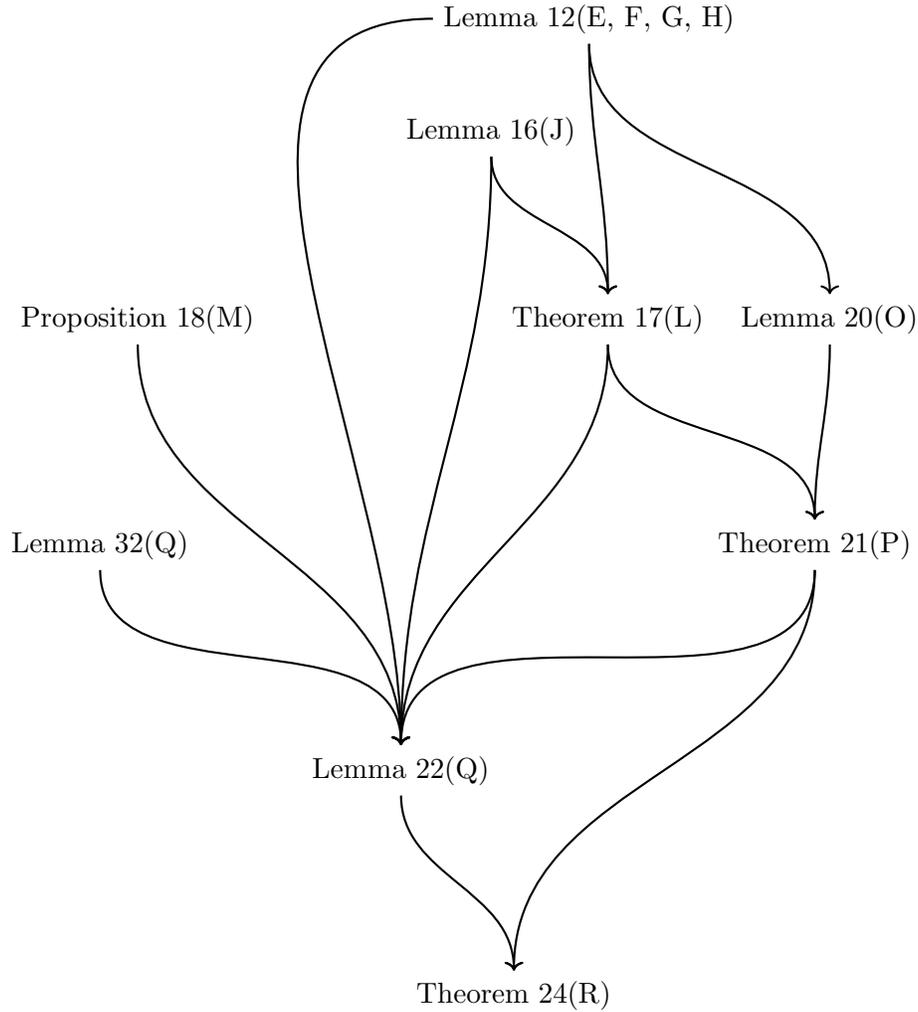
In this paper, we focused on the COL problem which may appear in many causal discovery tasks. Traditionally, Meek rules can be applied in order to solve a COL problem. Here, we presented a mathematical representation of these rules with Meek functions. Based on this representation, we provided some desirable properties for these functions. These properties enable us to simplify or accelerate solving COL problems. For example, we utilized these properties to accelerate different causal discovery problems such as solving experiment design problems where we need to apply Meek rules multiple times. Additionally, we took advantage of these properties to obtain a tight lower bound on number of oriented edges for a target intervention in a causal graph. To the best of our knowledge, this is the first lower bound on number of oriented edges for a target intervention. Furthermore, we used these rules to check whether the prior knowledge acquired from an expert in orienting some undirected edges in causal graph is compatible with already oriented edges or not. We proposed a heuristic algorithm that solves experiment design problem in active setting based on lower bound which reduce the execution time significantly.

As an interesting future research direction, one can use Meek properties to obtain an upper bound for number of oriented edges after performing intervention. Furthermore, designing algorithms based on these Meek properties is another possible research direction for experiment design in passive setting.

Appendices

Proof Sketch.

Lemma 12 provides Meek function properties on PCCGs. We prove this lemma by showing each property of that in Sections E, F, G, and H. Lemma 20 is related to applying Meek functions M_{14} on PCCGs and it is proved by using Lemma 12. According to Lemma 12 and Lemma 16, which is about Meek function properties on ICCGs, Theorem 17 is proved. This theorem asserts that Meek functions M_{124} can be decomposed to union of applying Meek functions M_{14} and M_2 . Using Theorem 17 and Lemma 20, we prove the Theorem 21 which introduces a method to obtain \mathcal{I} -essential graph. Based on the below sketch, we derive the proof of Lemma 22, lower bound for a maximal clique, by using a few lemmas and theorems depicted in the following figure. Finally, the proof of Theorem 24, lower bound for a case of node, is using Theorem 21 and Lemma 22.



A. Proof of Theorem 4

Corresponding sub-graphs for Meek rule 1, 2 and 3 are as the same of their corresponding candidate sub-graphs. Thus, based on the Meek (1995), applying these rules on their corresponding candidate sub-graphs are sound. We use the the following lemma to prove the soundness of applying Meek rule 4 on its corresponding candidate sub-graph.

Lemma 29 *The result of applying Meek rule 1 and then Meek rule 4 on corresponding candidate sub-graph for Meek rule 4 is the same as of applying Meek rule 4 on its corresponding sub-graph.*

Proof As dashed lines in Table 1 cannot be part of a v-structure, the edge $v_k - v_l$ in corresponding sub-graph of Meek rule 4 can be either directed as $v_k \rightarrow v_l$ or undirected as $v_k - v_l$ in its corresponding candidate sub-graph in Table 1. In the case of having $v_k - v_l$, this edge will be oriented as $v_k \rightarrow v_l$ by applying Meek rule 1 on sub-graph $\{v_j \rightarrow v_k, v_k - v_l\}$. Thus, after applying Meek rule 1 on the corresponding candidate sub-graph for Meek rule 4, we get the corresponding sub-graph for Meek rule 4. Therefore, the proof is complete. ■

Based on the Lemma 29, applying Meek rule 1 and 4 on corresponding candidate sub-graph for Meek rule 4 is as the same as the applying Meek rule 4 on corresponding sub-graph for Meek rule 4. As applying Meek rules are sound (Meek, 1995), applying Meek rule 4 on corresponding candidate sub-graph will be sound.

B. Proof of Theorem 9

Applying Meek rules 1, 2 and 3 on a MPDAG is complete (Meek, 1995). Additionally, corresponding sub-graphs for Meek rule 1, 2 and 3 are as the same of their corresponding candidate sub-graphs. Therefore, applying Meek functions 1, 2 and 3 on a MPDAG is complete.

C. Proof of Theorem 10

Applying Meek rules on a MPDAG, with some further oriented edges is complete (Meek, 1995). Additionally, corresponding sub-graphs for Meek rule 1, 2 and 3 are as the same of their corresponding candidate sub-graphs. Moreover, based on the Lemma 29, the result of applying Meek function M_4 on set of edges is equal or subset of the results of repeatedly applying Meek rule 1 and Meek rule 4 on that set. Therefore, applying Meek functions 1, 2, 3 and 4 on a graph is complete.

D. Proof of Property 1 in Lemma 13

According to Spirtes et al. (2000), having skeleton and the discovered v-structures, applying Meek function M_{123} is enough for discovering further edges' orientations to obtain the essential graph. Moreover, with the infinite samples from observational data, there will be no undiscovered v-structure sub-graph in the identified graph. As there is no undiscovered v-structure sub-graph, no new v-structure will be discovered by identifying more edges' orientations rather than those that exist in essential graph. Hence, it suffices to apply Meek function M_3 once before applying other Meek functions.

E. Proof of Property 1 in Lemma 12

As G is a PCCG, no v-structure sub-graph exists in G . This is because all v-structures in causal DAG are discovered in the procedure of obtaining essential graph and removed. Considering the fact that there is a v-structure in candidate sub-graph for Meek rule 3, we cannot find any candidate sub-graph for Meek rule 3 in \mathbf{E} . Thus, applying Meek function M_3 on graph G can not discover any further edges' orientations.

F. Proof of property 2 in Lemma 12

Given a PCCG $G = (\mathbf{V}, \mathbf{E})$ with set of nodes \mathbf{V} and set of edges \mathbf{E} , we will show the following holds:

$$M_i(\mathbf{E}) = \bigsqcup_{v_i \rightarrow v_j \in \vec{\mathbf{E}}} M_i(\{v_i \rightarrow v_j\} \sqcup \overline{\mathbf{E}}),$$

where M_i is one of the Meek functions M_1 , M_4 or M_{14} . We can write M_i as follows:

$$M_i(\mathbf{E}) = M_i \left(\bigsqcup_{v_i \rightarrow v_j \in \vec{\mathbf{E}}} M_i(\{v_i \rightarrow v_j\} \sqcup \overline{\mathbf{E}}) \right) = M_i \left(\bigsqcup_{v_i \rightarrow v_j \in \vec{\mathbf{E}}} \mathbf{E}_{ij} \right),$$

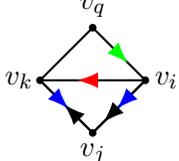
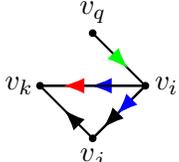
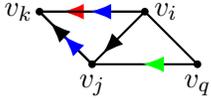
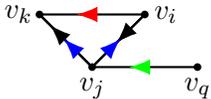
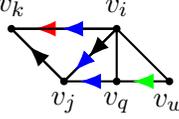
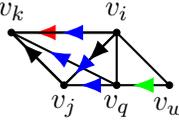
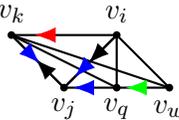
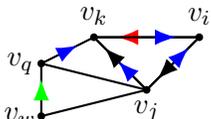
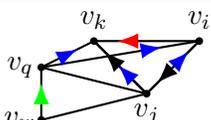
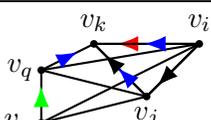
where $\mathbf{E}_{ij} = M_i(\{v_i \rightarrow v_j\} \sqcup \overline{\mathbf{E}})$. We will show that the right hand side of above equation is equal to $\mathbf{E}' = \bigsqcup_{v_i \rightarrow v_j \in \vec{\mathbf{E}}} \mathbf{E}_{ij}$. By contradiction, suppose that there are some edges which are not in \mathbf{E}' but they will be oriented by applying Meek function M_i on \mathbf{E}' . Therefore, there would be at least some edge like $v_t \rightarrow v_r$ which is in a candidate sub-graph, having directed edges in \mathbf{E}' . Moreover, it has been oriented by applying Meek function M_i on this candidate sub-graph. Note that there should be such an edge like $v_t \rightarrow v_r$, otherwise $\mathbf{E}' = M_i(\mathbf{E}')$. Now, assume that the edge $v_t \rightarrow v_r$ has been oriented by applying Meek function M_i on a candidate sub-graph for Meek rule 1, like $\{v_m \rightarrow v_t, v_t - v_r\}$, such that we have $v_m \rightarrow v_t \in \mathbf{E}_{qw}$ for some $v_q \rightarrow v_w \in \left\{ v_i \rightarrow v_j \mid v_i \rightarrow v_j \in \vec{\mathbf{E}} \right\}$. However, the edge $v_t - v_r$ has been oriented by applying Meek function M_1 on $v_q \rightarrow v_w$, and we have $v_t \rightarrow v_r \in \mathbf{E}'$, which is a contradiction. Similar to proof of candidate sub-graph for Meek rule 1, we can prove this for candidate sub-graph for Meek rule 4, and the proof is complete.

G. Candidate sub-graphs for Meek rule 2 can not be genetated as a result of applying Meek function M_{14}

Lemma 30 *Let $G = (\mathbf{V}, \mathbf{E})$ be a PCCG, \mathbf{V} be the set of nodes and \mathbf{E} be the set of edges. There exist some candidate sub-graphs for Meek rule 2 in G and no new candidate sub-graph for this rule is generated after applying Meek functions M_{14} on \mathbf{E} .*

We prove this lemma by contradiction. A candidate sub-graph for Meek rule 2 is depicted in Figure 10. We assume that one of the directed edges in the candidate sub-graph for Meek rule 2 is already oriented and the other one will be oriented by one of the Meek functions M_1 or M_4 . All possible scenarios have been depicted in Table 4. Note that in PCCGs, sub-graphs are chordal and there exist no v-structure before or after applying any Meek

Table 4: Different cases in Lemma 30 for orienting one edge in order to make a candidate sub-graph for Meek rule 2

Case 1	$\mathbf{E}' = \{v_q \rightarrow v_i\} \sqcup \overline{\mathbf{E}}$ $v_i \rightarrow v_j \in M_1(\mathbf{E}')$	$v_q \in \text{neigh}(v_k)$		contradiction on $v_j \rightarrow v_k$ $v_k \rightarrow v_j \in M_4(\mathbf{E}')$
		$v_q \notin \text{neigh}(v_k)$		$v_i \rightarrow v_k \in M_1(\mathbf{E}')$
Case 2	$\mathbf{E}' = \{v_q \rightarrow v_j\} \sqcup \overline{\mathbf{E}}$ $v_j \rightarrow v_k \in M_1(\mathbf{E}')$	$v_q \in \text{neigh}(v_i)$		$v_i \rightarrow v_k \in M_4(\mathbf{E}')$
		$v_q \notin \text{neigh}(v_i)$		contradiction on $v_i \rightarrow v_j$ $v_j \rightarrow v_i \in M_1(\mathbf{E}')$
Case 3	$\mathbf{E}' = \{v_w \rightarrow v_q\} \sqcup \overline{\mathbf{E}}$ $v_i \rightarrow v_j \in M_4(\mathbf{E}')$ $v_i \rightarrow v_j \notin M_1(\mathbf{E}')$	$v_q \notin \text{neigh}(v_k)$		$v_i \rightarrow v_k \in M_4(\mathbf{E}')$
		$v_q \in \text{neigh}(v_k)$ $v_w \notin \text{neigh}(v_k)$		$v_i \rightarrow v_k \in M_4(\mathbf{E}')$
		$v_q \in \text{neigh}(v_k)$ $v_w \in \text{neigh}(v_k)$		contradiction on $v_j \rightarrow v_k$ $v_k \rightarrow v_j \in M_4(\mathbf{E}')$
Case 4	$\mathbf{E}' = \{v_w \rightarrow v_q\} \sqcup \overline{\mathbf{E}}$ $v_j \rightarrow v_k \in M_4(\mathbf{E}')$ $v_j \rightarrow v_k \notin M_1(\mathbf{E}')$	$v_q \notin \text{neigh}(v_i)$		contradiction on $v_i \rightarrow v_j$ $v_j \rightarrow v_i \in M_4(\mathbf{E}')$
		$v_q \in \text{neigh}(v_i)$ $v_w \notin \text{neigh}(v_i)$		contradiction on $v_i \rightarrow v_j$ $v_j \rightarrow v_i \in M_4(\mathbf{E}')$
		$v_q \in \text{neigh}(v_i)$ $v_w \in \text{neigh}(v_i)$		$v_i \rightarrow v_k \in M_4(\mathbf{E}')$

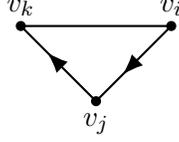


Figure 10: Meek candidate sub-graph for Meek rule 2

functions. We will see that two types of contradictions will be encountered in all of these cases. In some cases, there will be a conflict in direction of recently oriented edges with the already directed edges in the candidate sub-graph. In some other cases, we see that an undirected edge in candidate sub-graph will be oriented and there is no need to apply Meek function M_2 . Hence, for both of these cases, there will be no candidate sub-graph for Meek rule 2. In the following, we will study each of these cases in Table 4.

In depicted graphs in Table 4, we use different colors to distinguish different types of oriented edges. In all of these graphs, black oriented edges are the edges in Meek function M_2 candidate sub-graph. The red oriented edge is the one that will be oriented as a result of applying Meek function M_2 . The green oriented edge is supposed to be the edge that applying Meek functions M_1 or M_4 on that, results in orienting one further edge in order to construct Meek candidate sub-graph for function M_2 . The blue oriented edges are the result of applying Meek function M_1 , M_4 or M_{14} on green oriented edge in each graph. Note that the blue edges are a part of edges that can be oriented based on green edges and knowing their orientations is enough to prove the properties in the lemma.

In the first case, we investigate the scenario that the edge $v_i \rightarrow v_j$ has been oriented as a result of applying Meek function M_1 on an existing edge $v_q \rightarrow v_i$. As $v_i \rightarrow v_j \in M_1(\{v_q \rightarrow v_i\} \sqcup \overline{\mathbf{E}})$, we know $v_q \notin \text{neigh}(v_j)$. Thus, two different cases $v_q \notin \text{neigh}(v_k)$ and $v_q \in \text{neigh}(v_k)$ can be considered. In the case of $v_q \in \text{neigh}(v_k)$, the edge $v_k - v_j$ will be oriented in opposite direction of $v_j \rightarrow v_k$, which is a contradiction. In the second case, $v_q \notin \text{neigh}(v_k)$, the edge $v_i \rightarrow v_k$ will be oriented as a result of applying Meek function M_1 . Hence, in both cases, no candidate sub-graph for Meek rule 2 will be created.

In the second case, we consider the scenario that the edge $v_j \rightarrow v_k$ has been oriented as a result of applying Meek function M_1 on an existing edge $v_q \rightarrow v_j$. As $v_j \rightarrow v_k \in M_1(\{v_q \rightarrow v_j\} \sqcup \overline{\mathbf{E}})$, we know $v_q \notin \text{neigh}(v_k)$. Two different cases can be considered: $v_q \notin \text{neigh}(v_i)$ and $v_q \in \text{neigh}(v_i)$. In the case of $v_q \in \text{neigh}(v_i)$, the edge $v_i \rightarrow v_k$ will be oriented as a result of applying Meek function $M_4(\{v_q \rightarrow v_j\} \sqcup \overline{\mathbf{E}})$. In the other case, $v_q \notin \text{neigh}(v_i)$, the edge $v_i - v_j$ will be oriented in opposite direction of $v_i \rightarrow v_j$, which is a contradiction. Therefore, no candidate sub-graph for Meek rule 2 will be created.

In the third case, we assume that the edge $v_i \rightarrow v_j$ has been oriented as a result of applying Meek function M_4 on a set $\{v_w \rightarrow v_q\} \sqcup \overline{\mathbf{E}}$, and we also know that $v_i \rightarrow v_j \notin M_1(\{v_w \rightarrow v_q\} \sqcup \overline{\mathbf{E}})$ (see Table 4). As $v_i \rightarrow v_j \in M_4(\{v_w \rightarrow v_q\} \sqcup \overline{\mathbf{E}})$, we know $v_w \notin \text{neigh}(v_j)$. We partition the possible skeletons in three cases: (A) $v_q \notin \text{neigh}(v_k)$, (B) $v_q \in \text{neigh}(v_k)$, and $v_w \notin \text{neigh}(v_k)$ (C) $v_q \in \text{neigh}(v_k)$ and $v_w \in \text{neigh}(v_k)$. In case A, we know $v_q \rightarrow v_j \in M_4(\{v_w \rightarrow v_q\} \sqcup \overline{\mathbf{E}})$, and therefore, we have $v_i \rightarrow v_k \in M_4(\{v_q \rightarrow v_j\} \sqcup \overline{\mathbf{E}})$.

In case B, we have $v_i \rightarrow v_k \in M_4(\{v_w \rightarrow v_q\} \sqcup \overline{\mathbf{E}})$. In case C, the edge $v_j - v_k$ will be oriented in opposite direction of $v_j \rightarrow v_k$, which is a contradiction.

In the fourth case, we assume that the edge $v_j \rightarrow v_k$ has been oriented as a result of applying M_4 on a set $\{v_w \rightarrow v_q\} \sqcup \overline{\mathbf{E}}$, and also we know $v_j \rightarrow v_k \notin M_1(\{v_w \rightarrow v_q\} \sqcup \overline{\mathbf{E}})$. As $v_j \rightarrow v_k \in M_4(\{v_w \rightarrow v_q\} \sqcup \overline{\mathbf{E}})$, we know $v_w \notin \text{neigh}(v_k)$. We partition the possible skeletons for the mentioned scenario in three cases: (A) $v_q \notin \text{neigh}(v_i)$, (B) $v_q \in \text{neigh}(v_i)$, and $v_w \notin \text{neigh}(v_i)$ (C) $v_q \in \text{neigh}(v_i)$ and $v_w \in \text{neigh}(v_i)$. In case A, we know $v_q \rightarrow v_k \in M_4(\{v_w \rightarrow v_q\} \sqcup \overline{\mathbf{E}})$, and therefore, we have $v_j \rightarrow v_i \in M_4(\{v_q \rightarrow v_k\} \sqcup \overline{\mathbf{E}})$, which is a contradiction. In case B, we have $v_j \rightarrow v_i \in M_4(\{v_w \rightarrow v_q\} \sqcup \overline{\mathbf{E}})$, which is a contradiction. In the case C, we have $v_i \rightarrow v_k \in M_4(\{v_w \rightarrow v_q\} \sqcup \overline{\mathbf{E}})$. Therefore, there is no candidate sub-graph for Meek rule 2 in all of possible scenarios. Thus, the proof is complete.

H. Proof of Property 3 in Lemma 12

We write Meek function $M_{124}(\mathbf{E})$ as follows:

$$M_{124}(\mathbf{E}) = M_{124}(M_{14}(M_2(\mathbf{E}))) = M_{124}(\mathbf{E}') = \mathbf{E}' \sqcup \mathbf{E}_N,$$

where $\mathbf{E}' = M_{14}(M_2(\mathbf{E}))$. Note that $\overrightarrow{\mathbf{E}} \subseteq M_{14}(M_2(\mathbf{E}))$. It suffices to show \mathbf{E}_N is an empty set. We prove this by contradiction. Suppose $\mathbf{S} \subseteq \mathbf{E}'$ is a candidate sub-graph for Meek rule 1, 2 or 4 and M_i is one of the Meek functions M_1 , M_2 or M_4 . Assume that there exists an edge $e_n \in \mathbf{E}_N$ such that $e_n \notin \mathbf{E}'$. We have:

$$M_{124}(\mathbf{S} \sqcup \mathbf{E}' \setminus \mathbf{S}) = M_{124}(\mathbf{S} \sqcup (\mathbf{E}' \setminus \mathbf{S}) \sqcup \{e_n\}),$$

where edge e_n has been oriented in the result of applying one of the Meek functions M_1 , M_2 or M_4 on candidate sub-graph \mathbf{S} . In the case of $e_n \in M_1(\mathbf{E}')$ and $e_n \in M_4(\mathbf{E}')$, we will have $e_n \in \mathbf{E}'$ which is a contradiction. In the case of $e_n \in M_2(\mathbf{E}')$, we know that $\mathbf{S} \not\subseteq M_2(\mathbf{E})$. This is because it is not possible to have a candidate sub-graph of Meek rule 2 on the result of Meek function M_2 . Furthermore, we know from the Lemma 30 that candidate sub-graph for Meek rule 2 cannot be generated as a result of applying Meek function M_{14} . Hence, we will have $\mathbf{S} \not\subseteq \mathbf{E}'$ which is a contradiction.

I. Proof of Property 4 in Lemma 12

We want to show that for any subset $\mathbf{S} = \{v_i \rightarrow v_k, v_k \rightarrow v_j, v_i - v_j\} \subseteq \mathbf{E}$, we have:

$$M_{14}(M_2(\mathbf{S}) \sqcup \mathbf{E} \setminus \mathbf{S}) = M_{14}(\mathbf{E}) \sqcup M_2(\mathbf{S}).$$

Equivalently, we want to show that:

$$M_{14} \left(\left(\begin{array}{l} v_i \rightarrow v_k \\ v_k \rightarrow v_j \\ v_i \rightarrow v_j \end{array} \right) \sqcup \mathbf{E} \right) = M_{14} \left(\left(\begin{array}{l} v_i \rightarrow v_k \\ v_k \rightarrow v_j \\ v_i - v_j \end{array} \right) \sqcup \mathbf{E} \right) \sqcup \left(\begin{array}{l} v_i \rightarrow v_k \\ v_k \rightarrow v_j \\ v_i \rightarrow v_j \end{array} \right) \quad (2)$$

$$= M_{14} \left(\left(\begin{array}{l} v_i \rightarrow v_k \\ v_k \rightarrow v_j \\ v_i - v_j \end{array} \right) \sqcup \mathbf{E} \right) \sqcup \{v_i \rightarrow v_j\}. \quad (3)$$

The last equation holds because the edges in $\vec{\mathbf{E}}$ exist in the set of edges as a result of applying any Meek function. Furthermore, according to Property 2 in Lemma 12, for any $e_1 \in \vec{\mathbf{E}}$, we have:

$$\begin{aligned} M_{14}(\mathbf{E}) &= \bigsqcup_{e \in \vec{\mathbf{E}}} M_{14}(\{e\} \sqcup \overline{\mathbf{E}}) \\ &= \left(\bigsqcup_{e \in \vec{\mathbf{E}} \setminus \{e_1\}} M_{14}(\{e\} \sqcup \overline{\mathbf{E}}) \right) \sqcup M_{14}(\{e_1\} \sqcup \overline{\mathbf{E}}) \\ &= M_{14}(\overline{\mathbf{E}} \sqcup \vec{\mathbf{E}} \setminus \{e_1\}) \sqcup M_{14}(\{e_1\} \sqcup \overline{\mathbf{E}}), \end{aligned}$$

where the third equality is due to Property 2 in Lemma 12. Based on above equation, we can decompose the left side of (2) as follows:

$$M_{14} \left(\left(\begin{array}{c} v_i \rightarrow v_k \\ v_k \rightarrow v_j \\ v_i \rightarrow v_j \end{array} \right) \sqcup \mathbf{E} \right) = M_{14} \left(\left(\begin{array}{c} v_i \rightarrow v_k \\ v_k \rightarrow v_j \\ v_i \rightarrow v_j \end{array} \right) \sqcup \mathbf{E} \right) \sqcup M_{14} \left(\left(\begin{array}{c} v_i \rightarrow v_k \\ v_k \rightarrow v_j \\ v_i \rightarrow v_j \end{array} \right) \sqcup \overline{\mathbf{E}} \right).$$

It suffices to show that:

$$M_{14} \left(\left(\begin{array}{c} v_i \rightarrow v_k \\ v_k \rightarrow v_j \\ v_i \rightarrow v_j \end{array} \right) \sqcup \overline{\mathbf{E}} \right) \setminus \{v_i \rightarrow v_j\} \subseteq M_{14} \left(\left(\begin{array}{c} v_i \rightarrow v_k \\ v_k \rightarrow v_j \\ v_i \rightarrow v_j \end{array} \right) \sqcup \mathbf{E} \right)$$

In order to show this relation, we depict all possible skeletons, containing sub-graph $\{v_i \rightarrow v_k, v_k \rightarrow v_j, v_i \rightarrow v_j\}$, for further edges' orientations in Table 5. Note that in PCCGs, sub-graphs are chordal and there exist no v-structure before or after applying any Meek functions. We intend to determine all additional edges that can be oriented as a result of applying Meek function M_{14} on $\overline{\mathbf{E}} \sqcup \{v_i \rightarrow v_j\}$. Hence, if this function can orient any further edges, then there should be a node $v_q \in \text{neigh}(v_j)$, and also, an edge $v_j \rightarrow v_q$ in all possible skeletons. This edge will be oriented as a result of applying Meek function M_{14} on $\overline{\mathbf{E}} \sqcup \{v_i \rightarrow v_j\}$. In the following, we will investigate each case in Table 5, and we will show that the above equation holds.

In case 1, the edge $v_j \rightarrow v_q$ has been oriented as a result of applying Meek function M_1 on $\overline{\mathbf{E}} \sqcup \{v_i \rightarrow v_j\}$. This edge will also be oriented as a result of applying Meek function M_1 on set $\overline{\mathbf{E}} \sqcup \{v_k \rightarrow v_j\}$. Hence, there is no new information in applying Meek function M_{14} on set $\overline{\mathbf{E}} \sqcup \{v_i \rightarrow v_j\}$.

In the case 2, the edges $v_j \rightarrow v_q$ and $v_k \rightarrow v_q$ have been oriented as a result of applying Meek function M_4 on $\overline{\mathbf{E}} \sqcup \{v_i \rightarrow v_j\}$. We have $\{v_j \rightarrow v_q, v_k \rightarrow v_q\} \subseteq M_4(\overline{\mathbf{E}} \sqcup \{v_i \rightarrow v_k\})$.

In the case 3, the edges $v_j \rightarrow v_q$ and $v_w \rightarrow v_q$ have been oriented as a result of applying Meek function M_4 on $\overline{\mathbf{E}} \sqcup \{v_i \rightarrow v_j\}$. We have $v_i \rightarrow v_w \in M_4(\overline{\mathbf{E}} \sqcup \{v_k \rightarrow v_j\})$ and $\{v_j \rightarrow v_q, v_w \rightarrow v_q\} \subseteq M_4(\overline{\mathbf{E}} \sqcup \{v_i \rightarrow v_w\})$.

In the case 4, the edges $v_j \rightarrow v_q$ and $v_w \rightarrow v_q$ have been oriented as a result of applying Meek function M_4 on $\overline{\mathbf{E}} \sqcup \{v_i \rightarrow v_j\}$. We have $\{v_j \rightarrow v_q, v_w \rightarrow v_q\} \subseteq M_4(\mathbf{E} \sqcup \{v_k \rightarrow v_j\})$.

Table 5: Applying Meek function M_{14} on candidate sub-graph for Meek rule 2

Case 1	Case 2	Case 3	Case 4
$v_q \notin \text{neigh}(v_i)$		$v_q \notin \text{neigh}(v_k)$	
$v_q \notin \text{neigh}(v_k)$	$v_q \in \text{neigh}(v_k)$	$v_w \notin \text{neigh}(v_k)$	$v_w \in \text{neigh}(v_k)$
Case 5		Case 6	
$v_q \in \text{neigh}(v_k)$			
$v_w \notin \text{neigh}(v_k)$		$v_w \in \text{neigh}(v_k)$	

In the case 5, the edge $v_w \rightarrow v_q$ has been oriented as a result of applying Meek function M_4 on $\overline{\mathbf{E}} \sqcup \{v_i \rightarrow v_j\}$. In the other hand, this edge has been oriented in the opposite direction, i.e., $v_q \rightarrow v_w$ as a result of applying Meek function M_4 on $\overline{\mathbf{E}} \sqcup \{v_k \rightarrow v_j\}$, which is a contradiction.

In the case 6, the edges $v_w \rightarrow v_q$, $v_j \rightarrow v_q$ and $v_k \rightarrow v_q$ have been oriented as a result of applying Meek function M_4 on $\overline{\mathbf{E}} \sqcup \{v_i \rightarrow v_j\}$. Herein, for the set of nodes $\mathbf{T} = \{v_i, v_k, v_j, v_q\}$, we have $\{v_k \rightarrow v_q, v_j \rightarrow v_q\} \subseteq M_4(\mathbf{E}[\mathbf{T}] \sqcup \{v_i \rightarrow v_k\})$. Additionally, for the set of nodes $\mathbf{T} = \{v_i, v_k, v_w, v_q\}$, we have $\{v_w \rightarrow v_q, v_k \rightarrow v_q\} \subseteq M_4(\mathbf{E}[\mathbf{T}] \sqcup \{v_i \rightarrow v_k\})$.

J. Proof of Lemma 16

We know $\mathbf{S} = \{v_i \rightarrow v | v_i \in \mathbf{I}\} \sqcup \{v \rightarrow v_o | v_o \in \mathbf{O}\}$ where $\mathbf{S} \subseteq \mathbf{E}$, $\vec{\mathbf{S}} = \vec{\mathbf{E}}$ and $\text{neigh}(v) = \mathbf{I} \cup \mathbf{O}$. We want to show that:

$$M_2(\mathbf{E}) = \left(\bigsqcup_{\mathbf{C}_i \in \mathbf{C}_{M_2}(v)} M_2(\mathbf{C}_i) \right) \sqcup \mathbf{E}.$$

It suffices to show that orienting new edges as a result of applying Meek function M_2 on set of edges \mathbf{E} does not make any new candidate sub-graph for Meek rule 2, more than those that exist in set $\mathbf{E}[\text{Neigh}(v)]$ before applying Meek function M_2 . We will show that no new candidate sub-graph for Meek rule 2 will be generated neither in $\mathbf{E}[\text{Neigh}(v)]$ nor in an candidate sub-graph, having at least one edge outside $\mathbf{E}[\text{Neigh}(v)]$.

For the first case, Figure 11 shows the only possible skeleton for making new candidate sub-graph for Meek rule 2 in $\mathbf{E}[\text{Neigh}(v)]$. Note that all the edges incident with v have already been oriented. As a result, in order to have a new candidate sub-graph in

$\mathbf{E}[\text{Neigh}(v)]$ it should be a triangle with three nodes like $\{v_s, v_d, v_j\}$ which are in neighbor of v . In Figure 11(a), assume the red arrow edge, i.e., $\{v_d \rightarrow v_s\}$ is oriented as a result of applying Meek function M_2 on candidate sub-graph $\{v_d \rightarrow v, v \rightarrow v_s, v_d - v_s\}$. Now, we show that the triangle $\{v_s, v_d, v_j\}$ cannot form a new candidate sub-graph for Meek rule 2. There exist two options for the orientation of edge $v - v_j$. In Figure 11(b), we orient this edge as $v \rightarrow v_j$. As a result of applying Meek function M_2 on candidate sub-graph $\{v_d \rightarrow v, v \rightarrow v_j, v_d - v_j\}$, the edge $v_d - v_j$ will be oriented as $v_d \rightarrow v_j$. Thus, no new candidate sub-graph is made. In the case of orienting the edge $v - v_j$ as $v_j \rightarrow v$, shown in Figure 11(c), we cannot obtain new candidate sub-graph for Meek rule 2. Hence, we can imply that no new candidate sub-graph will be made within sub-graph $\mathbf{E}[\text{Neigh}(v)]$.

For the second case, we show that it is not possible to make a new candidate sub-graph, containing at least one node not being in $\text{Neigh}(v)$. First we show that it is not possible to orient any edge which exactly one of its end-points is in $\text{neigh}(v)$. By showing this, as a result, no edges will be oriented outside the sub-graph $\mathbf{E}[\text{Neigh}(v)]$. We prove this by contradiction. For having a candidate sub-graph of this kind, the only possible scenario is to have a structure like $\{v_t \rightarrow v_m, v_m \rightarrow v_n, v_t - v_n\}$, such that we have $\{v_m, v_n\} \subseteq \text{neigh}(v)$ and $v_t \in \mathbf{V} \setminus \text{Neigh}(v)$. Thus, we have $\{v_t - v_m, v_t - v_n\} \subseteq \overline{\mathbf{E}} \setminus \overline{\mathbf{E}}[\text{Neigh}(v)]$ and we intend to orient the only undirected edge in this candidate sub-graph by applying Meek function M_2 . Now, we study whether such directed edge $v_t \rightarrow v_m$ exists in our scenario. To orient $v_t \rightarrow v_m$, we need another candidate sub-graph of Meek rule 2, having two edges outside $\mathbf{E}[\text{Neigh}(v)]$ (note that one of them is $v_t - v_m$). The other edge should have already oriented by another candidate sub-graph for Meek rule 2 with the same property that it has two edges outside $\mathbf{E}[\text{Neigh}(v)]$. However, this continues relying on some other edges outside $\mathbf{E}[\text{Neigh}(v)]$, while the number of edges of this type is finite, and it gives a contradiction.

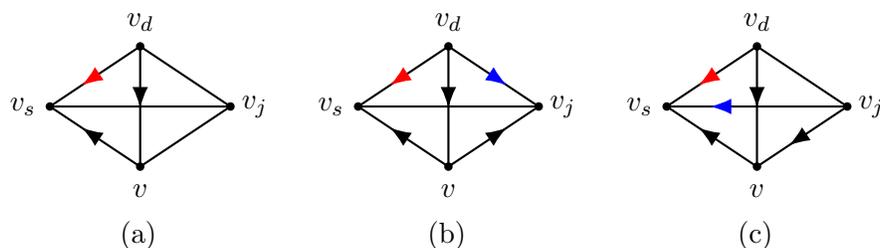


Figure 11: Possible orientations for making new candidate sub-graph for Meek rule 2

K. Proof of Property 2 in Lemma 13

The proof of Property 2 is exactly the same as the proof of Property 2 in Lemma 12 as adding some forbidden edges does not affect the proof. Note that we do not assume that the graph is chordal or there is no v-structures for proving Property 2 in Lemma 12.

L. Proof of Theorem 17

From Property 3 of Lemma 12, we have:

$$M_{124}(\mathbf{E}) = M_{14}(M_2(\mathbf{E}))$$

Substituting $M_2(\mathbf{E})$ from Lemma 16 in the above equation, we have:

$$M_{124}(\mathbf{E}) = M_{14} \left(\bigsqcup_{\mathbf{C}_i \in \mathbf{C}_{M_2}(v)} M_2(\mathbf{C}_i) \sqcup \mathbf{E} \right),$$

where the $\mathbf{C}_{M_2}(v)$ is set of all candidate sub-graphs for Meek rule 2 in $\mathbf{E}[Neigh(v)]$. From Property 4, we can infer that:

$$M_{14} \left(\bigsqcup_{\mathbf{C}_i \in \mathbf{C}_{M_2}(v)} M_2(\mathbf{C}_i) \sqcup \mathbf{E} \right) = M_{14}(\mathbf{E}) \sqcup \left(\bigsqcup_{\mathbf{C}_i \in \mathbf{C}_{M_2}(v)} M_2(\mathbf{C}_i) \right).$$

Thus, we can write:

$$M_{124}(\mathbf{E}) = M_{14}(\mathbf{E}) \sqcup \left(\bigsqcup_{\mathbf{C}_i \in \mathbf{C}_{M_2}(v)} M_2(\mathbf{C}_i) \right).$$

Finally, plugging $M_{14}(\mathbf{E})$ from Property 2 of Lemma 12 into the above equation, we have:

$$M_{124}(\mathbf{E}) = \left(\bigsqcup_{e \in \vec{\mathbf{E}}} M_{14}(\{e\} \sqcup \vec{\mathbf{E}}) \right) \sqcup \left(\bigsqcup_{\mathbf{C}_i \in \mathbf{C}_{M_2}(v)} M_2(\mathbf{C}_i) \right).$$

Therefore, the proof is complete.

M. Proof of Proposition 18

Consider a PCCG $G = (\mathbf{V}, \mathbf{E})$ with set of nodes \mathbf{V} and set of edges \mathbf{E} . Suppose that $v_s \rightarrow v_d$ is the only oriented edge in the set \mathbf{E} , i.e., $\vec{\mathbf{E}} = \{v_s \rightarrow v_d\}$. By defining $\mathbf{T} \triangleq Neigh(v_d)$, we have:

$$M_{14}(\{v_s \rightarrow v_d\} \sqcup \vec{\mathbf{E}}) = M_{14}(\{v_s \rightarrow v_d\} \sqcup \mathbf{E}' \sqcup \vec{\mathbf{E}}) = \mathbf{E}_{M_{14}},$$

where $\mathbf{E}' = M_{14}(\{v_s \rightarrow v_d\} \sqcup \vec{\mathbf{E}}[\mathbf{T}]) \setminus \{v_s \rightarrow v_d\}$. From Property 2 of Lemma 12, we can write:

$$M_{14}(\{v_s \rightarrow v_d\} \sqcup \mathbf{E}' \sqcup \vec{\mathbf{E}}) = M_{14}(\mathbf{E}' \sqcup \vec{\mathbf{E}}) \sqcup M_{14}(\{v_s \rightarrow v_d\} \sqcup \vec{\mathbf{E}}).$$

Furthermore, we have:

$$\begin{aligned} M_{14}(\{v_s \rightarrow v_d\} \sqcup \vec{\mathbf{E}}) \setminus \{v_s \rightarrow v_d\} &= \mathbf{E}_{M_{14}} \setminus \{v_s \rightarrow v_d\} \\ M_{14}(\mathbf{E}' \sqcup \vec{\mathbf{E}}) &= \mathbf{E}_{M_{14}} \setminus \{v_s \rightarrow v_d\}. \end{aligned}$$

Thus, we have:

$$M_{14}(\{v_s \rightarrow v_d\} \sqcup \overline{\mathbf{E}}) = M_{14}(\mathbf{E}' \sqcup \overline{\mathbf{E}}) \sqcup \{v_s \rightarrow v_d\}.$$

Finally, we obtain from Property 2 in Lemma 12:

$$DP[v_s \rightarrow v_d] = \left(\bigsqcup_{v_l \rightarrow v_k \in \overrightarrow{\mathbf{E}'}} DP[v_l \rightarrow v_k] \right) \sqcup \{v_s \rightarrow v_d\}.$$

N. Proof of Theorem 19

We know that calling function in Algorithm 1, may cause multiple another calls of this function in a recursive manner. Here, we prove that calling this function with directed edge $v_s \rightarrow v_d$ as input cannot occurred inside in calling another function with the same input as $v_s \rightarrow v_d$. We prove this by using the following lemma:

Lemma 31 *Given a PCCG $G = (\mathbf{V}, \mathbf{E})$ with set of nodes \mathbf{V} and set of edges \mathbf{E} . For any $v_s \in \mathbf{V}$ and for any edge $v_r \rightarrow v_k \in M_{14}(\overline{\mathbf{E}} \sqcup \{v_s \rightarrow v_d\})$, there exists a directed path from v_s to v_r .*

Proof We prove this statement by an induction on the number of steps taken by Meek function M_{14} . In the base case, if we apply Meek rule 1, then we orient some edge $v_d \rightarrow v_j$. Thus, there is the directed path $v_s \rightarrow v_d \rightarrow v_j$ from v_s to v_j . Moreover, if we apply Meek rule 4 in the first step, two edges like $v_d \rightarrow v_j$ and $v_k \rightarrow v_j$ will be oriented (see Table 1) and there is the directed path $v_s \rightarrow v_d \rightarrow v_j$ from v_s to v_j . Now, for the induction step, assume that induction hypothesis holds up to step r , i.e., for any oriented edge like $v_i \rightarrow v_r$ from Meek function M_1 , there is a directed path from v_s to v_r . Suppose that in step $r+1$, a new edge like $v_r \rightarrow v_k$ is oriented by applying Meek rule 1 on some edge $v_i \rightarrow v_r$. Since we know that there is a directed path from v_s to v_r , the same holds from v_s to v_k . In the case that new edge is oriented by applying Meek function M_4 on some previous oriented edge like $v_i \rightarrow v_r$, it can be easily seen that (see Table 1), there would be a directed edge from v_r to the head vertex of the newly oriented edge. Based on the induction hypothesis we can infer that there is a directed path from v_s to head vertex of the new oriented edge and the proof is complete. \blacksquare

Based on Lemma 31, we will get a directed cycle if such a mentioned recursive call is happened. In this scenario, we will have a directed path from node v_s to itself. As the underlying graph is a causal DAG, that cannot be happened. Thus, we have not multiple function calls with the same input. Additionally, as we have stored the resulted oriented edges after applying Meek rules on that edge, we did not call that edge again. Therefore, we call this function for each edge once. For filling all entries of DP table, we must call this function $2|\mathbf{E}|$ times. Moreover, we have a nested for-loop in each call which takes $\mathcal{O}(\Delta^2)$ operations. Hence, in total, the computational complexity would be in the order of $\mathcal{O}(|\mathbf{E}|\Delta^2)$.

O. Proof of Lemma 20

We know $\mathbf{S} = \{v_i \rightarrow v | v_i \in \mathbf{I}\} \sqcup \{v \rightarrow v_o | v_o \in \mathbf{O}\} \subseteq \mathbf{E}$. From Property 2 of Lemma 12, we have:

$$M_{14}(\mathbf{E}) = \left(\bigsqcup_{v_i \in \mathbf{I}} M_{14}(\{v_i \rightarrow v\} \sqcup \overline{\mathbf{E}}) \right) \sqcup \left(\bigsqcup_{v_o \in \mathbf{O}} M_{14}(\{v \rightarrow v_o\} \sqcup \overline{\mathbf{E}}) \right).$$

We define set of nodes \mathbf{O}_c as $\{v_o | v_o \in \mathbf{O}, \mathbf{I} \subseteq \text{neigh}(v_o)\}$. Hence, we rewrite the above equation as the following:

$$M_{14}(\mathbf{E}) = \left(\bigsqcup_{v_i \in \mathbf{I}} M_{14}(\{v_i \rightarrow v\} \sqcup \overline{\mathbf{E}}) \right) \sqcup \left(\bigsqcup_{v_o \in \mathbf{O}_c} M_{14}(\{v \rightarrow v_o\} \sqcup \overline{\mathbf{E}}) \right) \\ \sqcup \left(\bigsqcup_{v_o \in \mathbf{O} \setminus \mathbf{O}_c} M_{14}(\{v \rightarrow v_o\} \sqcup \overline{\mathbf{E}}) \right).$$

For each node $v_o \in \mathbf{O} \setminus \mathbf{O}_c$, there exists a node $v_k \in \mathbf{I}$ such that $v_k \notin \text{neigh}(v_o)$, thus we have $v \rightarrow v_o \in M_{14}(\{v_k \rightarrow v\} \sqcup \overline{\mathbf{E}})$. Therefore, we have the following relation:

$$M_{14}(\{v \rightarrow v_o\} \sqcup \overline{\mathbf{E}}) \subseteq M_{14}(\{v_k \rightarrow v\} \sqcup \overline{\mathbf{E}}) \subseteq \bigsqcup_{v_i \in \mathbf{I}} M_{14}(\{v_i \rightarrow v\} \sqcup \overline{\mathbf{E}}).$$

Thus, we have:

$$\bigsqcup_{v_o \in \mathbf{O} \setminus \mathbf{O}_c} M_{14}(\{v \rightarrow v_o\} \sqcup \overline{\mathbf{E}}) \subseteq \bigsqcup_{v_i \in \mathbf{I}} M_{14}(\{v_i \rightarrow v\} \sqcup \overline{\mathbf{E}}).$$

Hence, we can write:

$$M_{14}(\mathbf{E}) = \left(\bigsqcup_{v_i \in \mathbf{I}} M_{14}(\{v_i \rightarrow v\} \sqcup \overline{\mathbf{E}}) \right) \sqcup \left(\bigsqcup_{v_o \in \mathbf{O}_c} M_{14}(\{v \rightarrow v_o\} \sqcup \overline{\mathbf{E}}) \right) \\ = \bigsqcup_{v_l \rightarrow v_k \in \mathbf{S}'} M_{14}(\{v_l \rightarrow v_k\} \sqcup \overline{\mathbf{E}}) \\ = \bigsqcup_{v_l \rightarrow v_k \in \mathbf{S}'} DP[v_l \rightarrow v_k],$$

where $\mathbf{S}' = \{v_i \rightarrow v | v_i \in \mathbf{I}\} \sqcup \{v \rightarrow v_o | v_o \in \mathbf{O}, \mathbf{I} \subseteq \text{neigh}(v_o)\}$.

P. Proof of Theorem 21

Having $\mathbf{S} = \{v_i \rightarrow v | v_i \in \mathbf{I}\} \sqcup \{v \rightarrow v_o | v_o \in \mathbf{O}\}$ and $\vec{\mathbf{S}} = \vec{\mathbf{E}}$, from Theorem 17, we have:

$$M_{124}(\mathbf{E}) = \left(\bigsqcup_{e \in \vec{\mathbf{E}}} M_{14}(\{e\} \sqcup \overline{\mathbf{E}}) \right) \sqcup \left(\bigsqcup_{\mathbf{C}_i \in \mathbf{C}_{M_2}(v)} M_2(\mathbf{C}_i) \right).$$

Substituting the first term with its equivalent term in Lemma 20, we have:

$$M_{124}(\mathbf{E}) = \left(\bigsqcup_{v_l \rightarrow v_k \in \mathbf{S}'} DP[v_l \rightarrow v_k] \right) \bigsqcup \left(\bigsqcup_{\mathbf{C}_i \in \mathbf{C}_{M_2}(v)} M_2(\mathbf{C}_i) \right),$$

where $\mathbf{S}' = \{v_i \rightarrow v | v_i \in \mathbf{I}\} \sqcup \{v \rightarrow v_o | v_o \in \mathbf{O}_c\}$ and $\mathbf{O}_c = \{v_o | v_o \in \mathbf{O}, \mathbf{I} \subseteq \text{neigh}(v_o)\}$. By defining $\mathbf{I}_{\text{neigh}(v_o)} = \{v_i | v_i \in \mathbf{I} \cap \text{neigh}(v_o)\}$, we have the following equation:

$$\bigsqcup_{\mathbf{C}_i \in \mathbf{C}_{M_2}(v)} M_2(\mathbf{C}_i) = \{v_i \rightarrow v_o | v_i \in \mathbf{I}_{\text{neigh}(v_o)}, v_o \in \mathbf{O}_c\} \bigsqcup \{v_i \rightarrow v_o | v_i \in \mathbf{I}_{\text{neigh}(v_o)}, v_o \in \mathbf{O} \setminus \mathbf{O}_c\}.$$

From the definition of \mathbf{O}_c , we know that for each $v_o \in \mathbf{O}_c$, we have $\mathbf{I} \subseteq \text{neigh}(v_o)$. Hence, for each $v_o \in \mathbf{O}_c$, we have $\mathbf{I}_{\text{neigh}(v_o)} = \mathbf{I}$. Thus, for the first term in the above equation, we have:

$$\{v_i \rightarrow v_o | v_i \in \mathbf{I}_{\text{neigh}(v_o)}, v_o \in \mathbf{O}_c\} = \{v_i \rightarrow v_o | v_i \in \mathbf{I}, v_o \in \mathbf{O}_c\}.$$

Therefore, it suffices to show that:

$$\{v_i \rightarrow v_o | v_i \in \mathbf{I}_{\text{neigh}(v_o)}, v_o \in \mathbf{O} \setminus \mathbf{O}_c\} \subseteq \bigsqcup_{v_l \rightarrow v_k \in \mathbf{S}'} DP[v_l \rightarrow v_k]$$

Having $v_i \in \mathbf{I}_{\text{neigh}(v_o)}$ and $v_o \in \mathbf{O} \setminus \mathbf{O}_c$, for each edge $v_i - v_o \in \overline{\mathbf{E}}$, there exists a node $v_t \in \mathbf{I} \setminus \mathbf{I}_{\text{neigh}(v_o)}$, and a sub-graph $\mathbf{E}[\{v_i, v_o, v_t, v\}] = \{v_t \rightarrow v, v \rightarrow v_o, v_i \rightarrow v, v_t - v_i, v_i - v_o\}$ in the graph. Thus, we will have:

$$v_i \rightarrow v_o \in DP[v_t \rightarrow v].$$

Furthermore, we have $\{v_t \rightarrow v\} \subset \mathbf{S}'$. Thus, we can write:

$$\{v_i \rightarrow v_o\} \subseteq DP[v_t \rightarrow v] \subseteq \bigsqcup_{v_l \rightarrow v_k \in \mathbf{S}'} DP[v_l \rightarrow v_k].$$

Hence, we can conclude that:

$$M_{124}(\mathbf{E}) = \left\{ \bigsqcup_{v_l \rightarrow v_k \in \mathbf{S}'} DP[v_l \rightarrow v_k] \right\} \bigsqcup (v_i \rightarrow v_o | v_i \in \mathbf{I}, v_o \in \mathbf{O}_c)$$

where $\mathbf{S}' = \{v_i \rightarrow v | v_i \in \mathbf{I}\} \sqcup \{v \rightarrow v_o | v_o \in \mathbf{O}, \mathbf{I} \subseteq \text{neigh}(v_o)\}$.

Q. Proof of Lemma 22

We denote the obtained ICCGs with $G = (\mathbf{V}, \mathbf{E})$. We know that $\overrightarrow{\mathbf{E}} = \mathbf{S} = \{v_i \rightarrow v | v_i \in \mathbf{I}\} \sqcup \{v \rightarrow v_o | v_o \in \mathbf{O}\}$ in which $\mathbf{I} \subseteq \mathbf{C}_k$. We divide the problem of calculating the lower bound into two cases. In the first case, all the edges are from nodes in maximal clique toward the intervened node, v , i.e., $\mathbf{I} = \mathbf{C}_k$. In the second case, we consider that there is an edge from a node in maximal clique toward node v and also, there is an edge from intervened node v toward a node in maximal clique, i.e., $\mathbf{I} \subset \mathbf{C}_k$.

We use Theorem 21 for calculating number of oriented edges in the first case. We have $\mathbf{I} = \mathbf{C}_k$. Thus, we know $\mathbf{O}_c = \emptyset$ and $\mathbf{S} = \{v_i \rightarrow v | v_i \in \mathbf{C}_k\}$. Hence, in this case, number of oriented edges after applying Meek function M_{124} , $|M_{124}(\mathbf{E})|$, is equal to the following:

$$|M_{124}(\mathbf{E})| = L_I = \left| \bigsqcup_{v_i \in \mathbf{I}} DP[v_i \rightarrow v] \right|.$$

Now, we will investigate the second case. For sake of simplicity, first, we assume that the set of nodes that have edges from nodes in maximal clique \mathbf{C}_k to intervened node v is known. Then, we calculate the minimum number of oriented edges for any possible orientation of edges between maximal clique \mathbf{C}_k and intervened node v . After intervention, we have:

$$M_{124}(\mathbf{E}) = M_{124} \left(\left(\bigsqcup_{v_i \in \mathbf{I}} \{v_i \rightarrow v\} \right) \sqcup \left(\bigsqcup_{v_o \in \text{neigh}(v) \setminus \mathbf{I}} \{v \rightarrow v_o\} \sqcup \overline{\mathbf{E}} \right) \right)$$

Using Theorem 17, we decompose Meek function M_{124} to Meek functions M_2 and M_{14} :

$$M_{124}(\mathbf{E}) = M_{14} \left(\left(\bigsqcup_{v_i \in \mathbf{I}} \{v_i \rightarrow v\} \right) \sqcup \left(\bigsqcup_{v_o \in \text{neigh}(v) \setminus \mathbf{I}} \{v \rightarrow v_o\} \sqcup \overline{\mathbf{E}} \right) \right) \sqcup M_2 \left(\left(\bigsqcup_{v_i \in \mathbf{I}} \{v_i \rightarrow v\} \right) \sqcup \left(\bigsqcup_{v_o \in \text{neigh}(v) \setminus \mathbf{I}} \{v \rightarrow v_o\} \sqcup \overline{\mathbf{E}} \right) \right).$$

Based on Property 2 in Lemma 12, we extract mixed edge union from inside of Meek function M_{14} :

$$M_{124}(\mathbf{E}) = \left(\bigsqcup_{v_i \in \mathbf{I}} M_{14}(\{v_i \rightarrow v\} \sqcup \overline{\mathbf{E}}) \right) \sqcup \left(\bigsqcup_{v_o \in \text{neigh}(v) \setminus \mathbf{I}} M_{14}(\{v \rightarrow v_o\} \sqcup \overline{\mathbf{E}}) \right) \sqcup M_2 \left(\left(\bigsqcup_{v_i \in \mathbf{I}} \{v_i \rightarrow v\} \right) \sqcup \left(\bigsqcup_{v_o \in \text{neigh}(v) \setminus \mathbf{I}} \{v \rightarrow v_o\} \sqcup \overline{\mathbf{E}} \right) \right).$$

We know that $\mathbf{I} \subseteq \mathbf{C}_k$, therefore we have:

$$\text{neigh}(v) \setminus \mathbf{I} = \text{neigh}(v) \setminus \mathbf{C}_k \cup \mathbf{C}_k \setminus \mathbf{I}.$$

Thus, we can write:

$$\bigsqcup_{v_o \in \text{neigh}(v) \setminus \mathbf{I}} M_{14}(\{v \rightarrow v_o\} \sqcup \overline{\mathbf{E}}) = \left(\bigsqcup_{v_o \in \text{neigh}(v) \setminus \mathbf{C}_k} M_{14}(\{v \rightarrow v_o\} \sqcup \overline{\mathbf{E}}) \right) \sqcup \left(\bigsqcup_{v_o \in \mathbf{C}_k \setminus \mathbf{I}} M_{14}(\{v \rightarrow v_o\} \sqcup \overline{\mathbf{E}}) \right).$$

Substituting M_{14} function with the DP value in Proposition 18, we will have:

$$\bigsqcup_{v_o \in \text{neigh}(v) \setminus \mathbf{I}} M_{14}(\{v \rightarrow v_o\} \sqcup \overline{\mathbf{E}}) = \left(\bigsqcup_{v_o \in \text{neigh}(v) \setminus \mathbf{C}_k} DP[v \rightarrow v_o] \right) \sqcup \left(\bigsqcup_{v_o \in \mathbf{C}_k \setminus \mathbf{I}} DP[v \rightarrow v_o] \right).$$

In the other hand, we have:

$$\bigsqcup_{v_i \in \mathbf{I}} M_{14}(\{v_i \rightarrow v\} \sqcup \overline{\mathbf{E}}) = \bigsqcup_{v_i \in \mathbf{I}} DP[v_i \rightarrow v].$$

Therefore, we can write:

$$M_{124}(\mathbf{E}) = \left(\bigsqcup_{v_o \in \text{neigh}(v) \setminus \mathbf{C}_k} DP[v \rightarrow v_o] \right) \sqcup \left(\bigsqcup_{v_o \in \mathbf{C}_k \setminus \mathbf{I}} DP[\{v \rightarrow v_o\}] \right) \sqcup \left(\bigsqcup_{v_i \in \mathbf{I}} DP[v_i \rightarrow v] \right) \sqcup M_2 \left(\left(\bigsqcup_{v_i \in \mathbf{I}} \{v_i \rightarrow v\} \right) \sqcup \left(\bigsqcup_{v_o \in \text{neigh}(v) \setminus \mathbf{I}} \{v \rightarrow v_o\} \sqcup \overline{\mathbf{E}} \right) \right).$$

According to Lemma 16, we can write:

$$M_2 \left(\left(\bigsqcup_{v_i \in \mathbf{I}} \{v_i \rightarrow v\} \right) \sqcup \left(\bigsqcup_{v_o \in \text{neigh}(v) \setminus \mathbf{I}} \{v \rightarrow v_o\} \sqcup \overline{\mathbf{E}} \right) \right) = \{v_i \rightarrow v_o \mid v_i \in \mathbf{I}, v_o \in \mathbf{C}_k \setminus \mathbf{I}\} \sqcup \{v_i \rightarrow v_o \mid v_i \in \mathbf{I}, v_o \in \text{neigh}(v) \cap \text{neigh}(v_i) \setminus \mathbf{C}_k\}.$$

Therefore, we can obtain the result of applying Meek function M_{124} on set \mathbf{E} with the following equation:

$$M_{124}(\mathbf{E}) = \mathbf{T}_1 \sqcup \mathbf{T}_2 \sqcup \mathbf{T}_3 \sqcup \mathbf{T}_4.$$

where,

$$\begin{aligned} \mathbf{T}_1 &= \bigsqcup_{v_o \in \text{neigh}(v) \setminus \mathbf{C}_k} DP[v \rightarrow v_o], \\ \mathbf{T}_2 &= \{v_i \rightarrow v_o \mid v_i \in \mathbf{I}, v_o \in \mathbf{C}_k \setminus \mathbf{I}\}, \\ \mathbf{T}_3 &= \left(\bigsqcup_{v_i \in \mathbf{I}} DP[v_i \rightarrow v] \sqcup \{v_i \rightarrow v_o \mid v_i \in \mathbf{I}, v_o \in \text{neigh}(v) \cap \text{neigh}(v_i) \setminus \mathbf{C}_k\} \right) \setminus \mathbf{T}_1, \\ \mathbf{T}_4 &= \left(\bigsqcup_{v_o \in \mathbf{C}_k \setminus \mathbf{I}} DP[v \rightarrow v_o] \right) \setminus \mathbf{T}_1. \end{aligned}$$

Now, our goal is to calculate $|M_{124}(\mathbf{E})|$. To do so, we want to prove that the intersection of any two sets selected from the sets $\mathbf{T}_1, \mathbf{T}_2, \mathbf{T}_3, \mathbf{T}_4$ is empty. In order to show this, we first state the following Lemma 32.

Lemma 32 Consider a UCCG $G = (\mathbf{V}, \mathbf{E})$ with set of nodes \mathbf{V} and set of edges \mathbf{E} . Suppose that there exists a maximal clique \mathbf{C}_k in the neighborhood of node v . The following relations hold:

- For any $v_1 - v_2 \in \overline{\mathbf{E}}[\mathbf{C}_k]$ and $v_i \in \mathbf{C}_k$, we have $v_1 \rightarrow v_2 \notin DP[v_i \rightarrow v]$.
- For any $v_1 - v_2 \in \overline{\mathbf{E}}[Neigh(v)]$ and $v_o \in neigh(v)$, we have $v_1 \rightarrow v_2 \notin DP[v \rightarrow v_o] \setminus \{v \rightarrow v_o\}$.
- For any $\{v_1 - v_2, v_3 - v_4\} \subseteq \{v - v_t | v_t \in \mathbf{C}_k\}$, we have $v_1 \rightarrow v_2 \notin DP[v_3 \rightarrow v_4]$.

Proof It can be shown that there exist a DAG in the Markov equivalence class of graph G , such that this DAG consists of some edges that are oriented from nodes in a clique in the neighborhood of node v to this node (see Proposition 6 in Hauser and Buhlmann (2014)). We call such orientations in which there exist a DAG in the MEC as valid orientations. Based on this fact, we prove the relations by contradiction. For proving the first relation, assume that we have $v_1 \rightarrow v_2 \in DP[v_i \rightarrow v]$. Thus, according to Lemma 31, there exists a directed path from node v to node v_2 . As v_2 and v_i are in the maximal clique \mathbf{C}_k , we can have valid orientations of $v_2 \rightarrow v$ and $v_i \rightarrow v$ without making a new v-structure. Hence, there would be a directed cycle $v \rightarrow \dots \rightarrow v_2 \rightarrow v$ which is a contradiction. Therefore, there exists no DAG and we can imply that: $v_1 \rightarrow v_2 \notin DP[v_i \rightarrow v]$.

In order to show the second relation, suppose that we have $v_1 \rightarrow v_2 \in DP[v \rightarrow v_o]$. We partition the problem in three cases: $v_2 = v$, $v_2 = v_o$ and $v_2 \in neigh(v) \setminus v_o$. In the first case, according to Lemma 31, there will be a directed path $v \rightarrow v_o \dots \rightarrow v$ which is a directed cycle. In the second case, according to Lemma 31, there will be a directed path $v_o \rightarrow \dots \rightarrow v_1 \rightarrow v_o$ which is a directed cycle. In the third case, having the valid orientations $v_2 \rightarrow v$ and $v \rightarrow v_o$, we have a directed path $v \rightarrow v_o \dots \rightarrow v_2 \rightarrow v$ which is a directed cycle and there exists no DAG. This is a contradiction.

For proving the third relation, assume we have $v_1 \rightarrow v_2 \in DP[v_3 \rightarrow v_4]$. This means that orienting the edge $v_3 \rightarrow v_4$ orients the edge $v_1 \rightarrow v_2$, and we cannot have a DAG including directed edges $v_3 \rightarrow v_4$ and $v_2 \rightarrow v_1$. However, according to Proposition 6 in Hauser and Buhlmann (2014), such orientations are valid orientations and there exists a DAG, including those directed edges. Thus, this is a contradiction and the proof is complete. ■

According to Lemma 32, we will show that intersection of sets $\mathbf{T}_1, \mathbf{T}_2, \mathbf{T}_3, \mathbf{T}_4$ is empty. Hence, the number of elements in $M_{124}(\mathbf{E})$, i.e., $|M_{124}(\mathbf{E})|$, can be computed by adding the number of elements in each of the sets $\mathbf{T}_1, \mathbf{T}_2, \mathbf{T}_3, \mathbf{T}_4$.

First, it can be seen that $\overline{\mathbf{T}}_2 \subseteq \overline{\mathbf{E}}[\mathbf{C}_k]$. Furthermore, based on the second relation in Lemma 32, we cannot find any edge $v_1 \rightarrow v_2 \in \mathbf{T}_1$ for nodes $v_1, v_2 \in \mathbf{C}_k$. Therefore, we have $\overline{\mathbf{T}}_1 \cap \overline{\mathbf{E}}[\mathbf{C}_k] = \emptyset$. Additionally, we already removed all \mathbf{T}_1 elements from the sets \mathbf{T}_3 and \mathbf{T}_4 . Thus, we can write:

$$\mathbf{T}_1 \cap (\mathbf{T}_2 \sqcup \mathbf{T}_3 \sqcup \mathbf{T}_4) = \emptyset.$$

In the following we prove that $\overline{\mathbf{T}}_3 \cap \overline{\mathbf{E}}[\mathbf{C}_k] = \emptyset$ and $\overline{\mathbf{T}}_4 \cap \overline{\mathbf{E}}[\mathbf{C}_k] = \emptyset$. Based on the second relation in Lemma 32, we cannot find any edge $v_1 \rightarrow v_2 \in \mathbf{T}_4$ for set of nodes $v_1, v_2 \in \mathbf{C}_k$.

Moreover, we can define the following two sets \mathbf{T}_3^1 and \mathbf{T}_3^2 such that $\mathbf{T}_3 = \mathbf{T}_3^1 \sqcup \mathbf{T}_3^2$:

$$\begin{aligned}\mathbf{T}_3^1 &= \left(\bigsqcup_{v_i \in \mathbf{I}} DP[v_i \rightarrow v] \right) \setminus \mathbf{T}_1, \\ \mathbf{T}_3^2 &= \left(\bigsqcup \{v_i \rightarrow v_o \mid v_i \in I, v_o \in \text{neigh}(v) \cap \text{neigh}(v_i) \setminus \mathbf{C}_k\} \right) \setminus \mathbf{T}_1.\end{aligned}$$

We know that $\overline{\mathbf{T}}_3^2 \cap \overline{\mathbf{E}}[\mathbf{C}_k] = \emptyset$. It suffices to show $\overline{\mathbf{T}}_3^1 \cap \overline{\mathbf{E}}[\mathbf{C}_k] = \emptyset$. Again based on the first relation in Lemma 32, we cannot find any edge $v_1 \rightarrow v_2 \in \mathbf{T}_3^1$ for nodes $v_1, v_2 \in \mathbf{C}_k$. Hence, we have $\overline{\mathbf{T}}_3 \cap \overline{\mathbf{E}}[\mathbf{C}_k] = \emptyset$ and we can write:

$$\mathbf{T}_2 \cap (\mathbf{T}_3 \sqcup \mathbf{T}_4) = \emptyset.$$

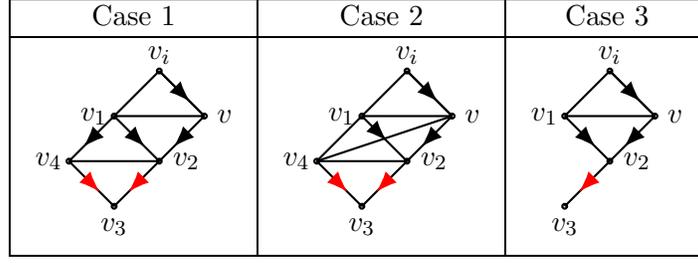
Finally, we will show that: $\mathbf{T}_3 \cap \mathbf{T}_4 = \emptyset$. To do so, we prove the following two relations:

$$\overline{\mathbf{T}}_3 \subseteq \overline{\mathbf{E}}[\text{Neigh}(v)] \setminus \{v - v_o \mid v_o \in \mathbf{C}_k \setminus \mathbf{I}\}, \quad (4)$$

$$\overline{\mathbf{T}}_4 \subseteq \{v - v_o \mid v_o \in \mathbf{C}_k \setminus \mathbf{I}\} \sqcup \overline{\mathbf{E}} \setminus \overline{\mathbf{E}}[\text{Neigh}(v)]. \quad (5)$$

For proving the relation (5), consider set \mathbf{T}_3 . We know $\overline{\mathbf{T}}_3^2 \subseteq \overline{\mathbf{E}}[\text{Neigh}(v)] \setminus \{v - v_o \mid v_o \in \mathbf{C}_k \setminus \mathbf{I}\}$. Thus, it suffices to show: $\overline{\mathbf{T}}_3^1 \subseteq \overline{\mathbf{E}}[\text{Neigh}(v)] \setminus \{v - v_o \mid v_o \in \mathbf{C}_k \setminus \mathbf{I}\}$. Based on Proposition 18, in the first step of DP calculation for each edge $v_i \rightarrow v$, we orient the edges inside the sub-graph $\overline{\mathbf{E}}[\text{Neigh}(v)]$. The edges that are in $DP[v_i \rightarrow v]$ and $\overline{\mathbf{E}}[\text{Neigh}(v)]$, can be categorized into two types. In the first type, at least one of the endpoint of the edge is v . According to third relation in Lemma 32, for these edges like $v \rightarrow v_2$, we cannot orient edges $v - v_2$ for any $v_2 \in \mathbf{C}_k$. Thus, we conclude that: $M_{14}(\{v \rightarrow v_2\} \sqcup \overline{\mathbf{E}}) \subseteq \mathbf{T}_1$. For the second type of the edges, none of the endpoints of edges is the node v . For this type of edges like $v_1 \rightarrow v_2$, we cannot have $\{v_1, v_2\} \subseteq \mathbf{C}_k$ due to the first relation in Lemma 32. Furthermore, we cannot have $v_2 \in \mathbf{C}_k$. Since if we have $v_2 \in \mathbf{C}_k$, then $v_1 \notin \mathbf{C}_k$. Moreover, based on Proposition 6 in Hauser and Buhlmann (2014), by considering the valid orientation $v_2 \rightarrow v$, we will have a cycle as $\{v \rightarrow v_1, v_1 \rightarrow v_2, v_2 \rightarrow v\}$. Additionally, for each edge of this type there is always an edge $v \rightarrow v_2$, where $v_2 \in \text{neigh}(v) \setminus \mathbf{C}_k$. Therefore, it suffices to show that, $\mathbf{E}_1 \setminus \mathbf{E}_1[\text{Neigh}(v)] \subseteq M_{14}(\{v \rightarrow v_2\} \sqcup \overline{\mathbf{E}})$, where $\mathbf{E}_1 = M_{14}(\{v_1 \rightarrow v_2\} \sqcup \overline{\mathbf{E}})$. We depict all possible skeletons in Table 6. Note that we consider the cases in which applying Meek function orients edges outside the sub-graph $\overline{\mathbf{E}}[\text{Neigh}(v)]$. Therefore, in all of these cases $v_3 \notin \text{neigh}(v)$. Additionally, in all of these cases, the red edges are the ones that have been oriented as a result of applying Meek function M_{14} on $v_1 \rightarrow v_2$, and as can be seen, these edges are also oriented as a result of applying Meek functions M_{14} on $v \rightarrow v_2$. Based on the Proposition 18, for discovering more edges' orientation outside the sub-graph $\overline{\mathbf{E}}[\text{Neigh}(v)]$, we must compute DP function in the sub-graph $\overline{\mathbf{E}}[\text{Neigh}(v_3)]$, where only red edges are oriented. Thus, as we depict all possible cases, we conclude that every edges that will be oriented outside the sub-graph $\overline{\mathbf{E}}[\text{Neigh}(v)]$ as a result of $M_{14}(\{v_1 \rightarrow v_2\} \sqcup \overline{\mathbf{E}})$ is also in $M_{14}(\{v \rightarrow v_2\} \sqcup \overline{\mathbf{E}}) \subseteq \mathbf{T}_1$. Thus, we have $\overline{\mathbf{T}}_3 \subseteq \overline{\mathbf{E}}[\text{Neigh}(v)]$. In addition, based on the

Table 6: Different orientations for applying Meek function on $v_i \rightarrow v$



third relation in Lemma 32, no edge that is between node v and clique \mathbf{C}_k will be oriented in \mathbf{T}_3^1 . Thus, we can write:

$$\overline{\mathbf{T}}_3 \subseteq \overline{\mathbf{E}}[Neigh(v)] \setminus \{v - v_o | v_o \in \mathbf{C}_k \setminus \mathbf{I}\}$$

For relation (6), based on Lemma 32, the result of applying Meek function M_{14} on set $\{v \rightarrow v_o\} \cup \overline{\mathbf{E}}$ cannot orient any edge inside the sub-graph $\overline{\mathbf{E}}[Neigh(v)] \setminus \{v \rightarrow v_o\}$. According to these two relations, we can write:

$$\mathbf{T}_3 \cap \mathbf{T}_4 = \emptyset.$$

To wrap up, the following equation holds:

$$\begin{aligned} \mathbf{T}_1 \cap (\mathbf{T}_2 \sqcup \mathbf{T}_3 \sqcup \mathbf{T}_4) &= \emptyset, \\ \mathbf{T}_2 \cap (\mathbf{T}_3 \sqcup \mathbf{T}_4) &= \emptyset, \\ \mathbf{T}_3 \cap \mathbf{T}_4 &= \emptyset. \end{aligned}$$

Hence, we will have:

$$|M_{124}(\mathbf{E})| = |\mathbf{T}_1| + |\mathbf{T}_2| + |\mathbf{T}_3| + |\mathbf{T}_4|.$$

Now, we can exactly compute the number of oriented edges after applying Meek function M_{124} on set \mathbf{E} . We can compute \mathbf{T}_1 from the DP table. \mathbf{T}_2 can also be obtained easily. Now, we want to find the minimum value for the size of sets \mathbf{T}_3 and \mathbf{T}_4 . First, we consider the set \mathbf{T}_3 . Recall that the definitions of P_j and Q_j are:

$$\begin{aligned} P_j &\triangleq \min_{|\mathbf{I}|=j, \mathbf{I} \subset \mathbf{C}_k} \max_{v_i \in \mathbf{I}} \left| \left(DP[v_i \rightarrow v] \sqcup_{v_o \in neigh(v_i) \cap neigh(v) \setminus \mathbf{C}_k} \{v_i \rightarrow v_o\} \right) \setminus \mathbf{T}_1 \right| \\ Q_j &\triangleq \min_{|\mathbf{O}|=j, \mathbf{O} \subset \mathbf{C}_k} \max_{v_o \in \mathbf{O}} |DP[v \rightarrow v_o] \setminus \mathbf{T}_1|. \end{aligned}$$

Now, it can be seen that:

$$\begin{aligned} \max_{v_i \in \mathbf{I}} \left| \left(DP[v_i \rightarrow v] \sqcup_{v_o \in neigh(v_i) \cap neigh(v) \setminus \mathbf{C}_k} \{v_i \rightarrow v_o\} \right) \setminus \mathbf{T}_1 \right| &\leq \\ \left| \left(\sqcup_{v_i \in \mathbf{I}} DP[v_i \rightarrow v] \sqcup \{v_i \rightarrow v_o | v_i \in \mathbf{I}, v_o \in neigh(v) \cap neigh(v_i) \setminus \mathbf{C}_k\} \right) \setminus \mathbf{T}_1 \right|. \end{aligned}$$

But the right hand side of the above inequality is $|\mathbf{T}_3|$. Hence, we can imply that $P_j \leq |\mathbf{T}_3|$. Similarly, we can show that $Q_j \leq |\mathbf{T}_4|$.

Finally, we will have:

$$\begin{aligned} |\mathbf{T}_1| &= |\mathbf{R}|, \\ |\mathbf{T}_2| &= |\mathbf{I}|(|\mathbf{C}_k| - |\mathbf{I}|), \\ |\mathbf{T}_3| &\geq P_j, \\ |\mathbf{T}_4| &\geq Q_j. \end{aligned}$$

Thus, when the set \mathbf{I} is known, we will have:

$$|M_{124}(\mathbf{E})| \geq |\mathbf{R}| + P_{|I|} + Q_{(|\mathbf{C}_k| - |I|)} + |I|(|\mathbf{C}_k| - |I|) + (|\mathbf{C}_k| - 2).$$

Note that for obtaining $P_{|I|}$ and $Q_{(|\mathbf{C}_k| - |I|)}$, we just consider two edges between the \mathbf{C}_k and v . Therefore, the last term is for those remaining $|\mathbf{C}_k| - 2$ edges that are between clique and node v and they will definitely be oriented after intervention. Note that, based on the second and third relation, these edges have not been considered in computing Q_j and P_j values and also they have not been oriented in set \mathbf{R} . If we consider all possible sets for set \mathbf{I} , where $|\mathbf{I}| = l$, we will have:

$$|M_{124}(\mathbf{E})| \geq L_C = |\mathbf{R}| + \min_{l=\{1, \dots, |\mathbf{C}_k| - 1\}} P_l + Q_{|\mathbf{C}_k| - l} + |l|(|\mathbf{C}_k| - |l|) + (|\mathbf{C}_k| - 2).$$

Finally we can infer that:

$$L(\mathbf{C}_k, v) = \min(L_I, L_C).$$

R. Proof of Theorem 24

After intervention on node v , we will discover orientations of edges in the neighborhood of intervened node v . We denote the obtained ICCG by $G' = (\mathbf{V}, \mathbf{E}')$. Additionally, we denote set of nodes that have edges toward node v with \mathbf{I} . Furthermore, we denote the set of all maximal cliques in the neighborhood of node v by $\mathbf{C}(v)$. As edges that are orientated from intervention do not generate a new v-structure in the graph, there exists a maximal clique $\mathbf{C}_k \in \mathbf{C}(v)$, such that $\mathbf{I} \subseteq \mathbf{C}_k$.

After intervention, two cases can be considered: (a) All edges are outgoing from node v toward the nodes in the neighborhood of node v (b) At least there exists one edge from a node in the neighborhood of node v to node v . In the first case, i.e., $\mathbf{I} = \emptyset$, using Theorem 21, the number of oriented edges after intervention is equal to the following equation:

$$|M_{124}(\mathbf{E}')| = \left| \bigsqcup_{v_o \in \text{neigh}(v)} DP[v \rightarrow v_o] \right|.$$

In the second case, i.e., $\mathbf{I} \subseteq \mathbf{C}_k$ and $|\mathbf{I}| > 0$, based on Lemma 22, the lower bound on number of oriented edges can be written as follows:

$$|M_{124}(\mathbf{E}')| \geq \min_{\mathbf{C}_k \in \mathbf{C}(v)} L(\mathbf{C}_k, v).$$

Hence, the lower bound on number of oriented edges after intervention on node v , $L(v)$, can be calculated as the following:

$$|M_{124}(\mathbf{E}')| \geq L(v) = \min \left(\left| \bigsqcup_{v_o \in \text{neigh}(v)} DP[v \rightarrow v_o] \right|, \min_{\mathbf{C}_k \in \mathbf{C}(v)} L(\mathbf{C}_k, v) \right).$$

S. Proof of Theorem 28

We know the PCCG G consists of the combination of both directed and undirected edges. For the “only if” part, we know that there is a causal DAG with the same skeleton as graph G , including directed edges in \mathbf{E} . We can imply that there is no cycle in \mathbf{E} and all edges in \mathbf{E} are consistent. For the ”if” part, we know that there is no cycle in \mathbf{E} and the set \mathbf{E} is consistent. We want to show that there will be a causal DAG with the same skeleton as graph G , including directed edges in \mathbf{E} .

To prove this, we will use an algorithm that constructs a DAG from our graph G , if the mentioned constraints are satisfied. Algorithm 6 takes G as its input and returns a causal DAG, including no v-structure if the constraints are satisfied. This algorithm guarantees that generated DAG contains all directed edges in \mathbf{E} and has the same skeleton as $\overline{\mathbf{E}}$ and it has no v-structure. In Line 5 in Algorithm 6, we apply Meek functions M_{124} on set \mathbf{E} . Through Lines 6-9, we orient undirected edges that are existed in set \mathbf{E} . In Line 7, we select an arbitrary node, and in Line 8, we orient all undirected edges in the neighborhood of this node as out-going edges. Thus, all the edges in the neighborhood of this node will be oriented and we denote the set of these oriented edges by \mathbf{S} . In Line 9, we obtain the result of applying Meek functions M_{124} on set $\mathbf{S} \sqcup \overline{\mathbf{E}}$. Also, we add the oriented edges to the output DAG \mathcal{G} .

Algorithm 6 Causal DAG Construction Algorithm

```

1: Input: PCCG  $G = (\mathbf{V}, \mathbf{E})$ 
2: Output: DAG  $\mathcal{G} = (\mathbf{V}_{\mathcal{G}}, \mathbf{E}_{\mathcal{G}})$ 
3: function PDAG2DAG( $G$ )
4:    $\mathbf{V}_{\mathcal{G}} \leftarrow \mathbf{V}$ 
5:    $\mathbf{E}_{\mathcal{G}} \leftarrow M_{124}(\mathbf{E})$ 
6:   while  $|\mathbf{E}| \neq |\overrightarrow{\mathbf{E}}_{\mathcal{G}}|$  do
7:      $v \leftarrow$  an arbitrary node from  $\{v_l | \exists v_k \in \mathbf{V}_{\mathcal{G}}, v_l - v_k \in \mathbf{E}_{\mathcal{G}}\}$ 
8:      $\mathbf{S} \leftarrow \{v_i \rightarrow v | v_i \rightarrow v \in \mathbf{E}_{\mathcal{G}}\} \sqcup \{v \rightarrow v_o | v \rightarrow v_o \in \mathbf{E}_{\mathcal{G}} \text{ or } v - v_o \in \mathbf{E}_{\mathcal{G}}\}$ 
9:      $\mathbf{E}_{\mathcal{G}} \leftarrow M_{124}(\mathbf{E}_{\mathcal{G}} \sqcup \mathbf{S})$ 
10:  return  $\mathcal{G}$ 

```

Based on our assumptions, we know that PDAG G has no cycle. Additionally, the following lemma expresses it has no v-structure.

Lemma 33 For any graph $G = (\mathbf{V}, \mathbf{E})$, there is no v -structure in \mathbf{E} if \mathbf{E} is a consistent set.

Proof We prove this by contradiction. Suppose a v -structure like $\{v_i \rightarrow v_j, v_j \leftarrow v_k\}$ exists in \mathbf{E} . Thus, we have $v_j \rightarrow v_k \in DP[v_i \rightarrow v_j]$. This is a contradiction because \mathbf{E} is a consistent set. ■

It suffices to show no new cycle and no new v -structure will be generated during running Algorithm 6. First, we will show that no v -structure will be appeared in the output graph of Algorithm 6.

To do so, we will prove that orienting further edges, either by applying Meek function M_{124} on set \mathbf{E} or adding new directed edges such as those in the constructing set \mathbf{S} in Algorithm 6, does not generate any new v -structure. Before providing the proof, we state the following lemma that helps in proving mentioned statement.

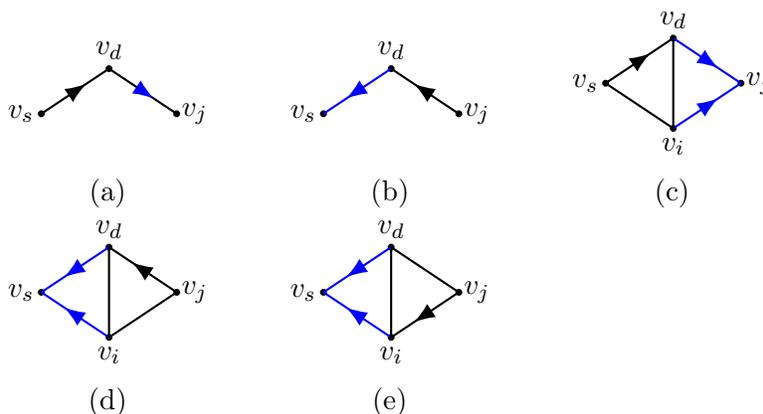


Figure 12: reverse orientation of edges oriented by applying function M_1 or M_4

Lemma 34 Consider a graph $G = (\mathbf{V}, \mathbf{E})$ with set of nodes \mathbf{V} and set of edges \mathbf{E} . If we have $v_i \rightarrow v_j \in DP[v_k \rightarrow v_l]$, we will have $v_l \rightarrow v_k \in DP[v_j \rightarrow v_i]$.

Proof First, we prove that this lemma holds for candidate sub-graphs for Meek rule 1 and 4. Then, we extend this result for applying Meek function M_{14} on a general graph. In Figure 12, the blue edges are oriented as the result of applying Meek function on black edges. We depict the candidate sub-graph for Meek rule 1 in the Figure 12(a). In this graph, the directed edge $v_s \rightarrow v_d$ orients the edge $v_d \rightarrow v_j$. According to Figure 12(b), if we orient the edge $v_d \rightarrow v_j$ in reverse direction, i.e., $v_j \rightarrow v_d$, the edge $v_s \rightarrow v_d$ will be oriented in the reverse direction, i.e., $v_d \rightarrow v_s$.

The candidate sub-graph for Meek rule 4 is depicted in Figure 12(c). In this sub-graph, by applying Meek function M_4 , directed edge $v_s \rightarrow v_d$ will orient two edges $v_i \rightarrow v_j$ and $v_d \rightarrow v_j$. According to the graphs in Figure 12(d)(Figure 12(e)), if we orient the edge $v_d \rightarrow v_j$ ($v_i \rightarrow v_j$) in reverse direction, i.e., $v_j \rightarrow v_d$ ($v_j \rightarrow v_i$), the edge $v_s \rightarrow v_d$ will be

oriented in the reverse direction, i.e., $v_d \rightarrow v_s$. Hence, we proved that this lemma holds for candidate sub-graphs of Meek function M_1 and M_4 .

Based on the above observation, we will show the statement in the lemma holds in a general graph. Assume we know $v_i \rightarrow v_j \in DP[v_k \rightarrow v_l]$. The edge $v_i \rightarrow v_j$ has been oriented in a repetitive sequence of applying Meek function M_1 and M_4 in the corresponding candidate sub-graphs. Based on the above results, we can reverse the sequence of applying Meek functions on each candidate sub-graphs in order to orient the edge $v_l - v_k$ as $v_l \rightarrow v_k$, and the proof is complete. \blacksquare

Based on the Lemma 33, if we show that the set of directed edges after execution of Algorithm 6 is consistent, we can conclude no v-structure exists in the output DAG. To do so, we prove following three facts: 1) The set of directed edges as a result of applying Meek function M_{124} in Line 5 is consistent. 2) The set \mathbf{S} is consistent with already directed edges. 3) The set $M_{124}(\mathbf{E}_g \sqcup \mathbf{S})$ is consistent.

1) We prove this by induction on the number of steps taken by applying Meek function M_{124} . For the base case, it is obvious that the edges in \mathbf{E} are consistent. For the induction step, suppose that the directed edges up to step r are consistent. We show that the directed edges will be remained consistent after step $r + 1$. We prove this by contradiction. Assume that we orient new edge $v_i \rightarrow v_j$ in step $r + 1$ by applying one of the Meek functions and there exists an already directed edge $v_r \rightarrow v_t$, such that for some edges $v_l - v_k \in \overline{\mathbf{E}}$ we have $v_l \rightarrow v_k \in DP[v_i \rightarrow v_j]$, but $v_k \rightarrow v_l \in DP[v_r \rightarrow v_t]$. Two cases can be considered:

- In the first case, we have: $v_i \rightarrow v_j \in M_{14}(\mathbf{E})$. Thus, there exists an edge $v_q \rightarrow v_w$ such that $v_i \rightarrow v_j \in DP[v_q \rightarrow v_w]$ and as a result $v_l \rightarrow v_k \in DP[v_q \rightarrow v_w]$. This is a contradiction, because we know the edges $v_q \rightarrow v_w$ and $v_r \rightarrow v_t$ are consistent.
- In this case, there exists a Meek rule 2 candidate sub-graph, such that we have: $v_i \rightarrow v_j \in M_2(\{v_i \rightarrow v_m, v_m \rightarrow v_j, v_i - v_j\})$. It can be shown that if $v_l \rightarrow v_k \in DP[v_i \rightarrow v_j]$, we will have $v_l \rightarrow v_k \in DP[v_i \rightarrow v_m] \sqcup DP[v_m \rightarrow v_j]$ according to Property 4 in Lemma 12. This is a contradiction, because the edges $v_i \rightarrow v_m$, $v_m \rightarrow v_j$ and $v_r \rightarrow v_t$ are consistent.

2) In this case, we want to prove that the oriented edges in \mathbf{S} are consistent with already directed edges. We prove this by contradiction. Assume we orient the edge $v - v_j$ as $v \rightarrow v_j$ in the procedure of orienting edges in \mathbf{S} and there exists an already directed edge $v_r \rightarrow v_t$, such that $v_l \rightarrow v_k \in DP[v \rightarrow v_j]$ and $v_k \rightarrow v_l \in DP[v_r \rightarrow v_t]$. Based on the Lemma 34, if we have $v_l \rightarrow v_k \in DP[v \rightarrow v_j]$, we will have $v_j \rightarrow v \in DP[v_k \rightarrow v_l]$. Having the relations $v_j \rightarrow v \in DP[v_k \rightarrow v_l]$ and $v_k \rightarrow v_l \in DP[v_r \rightarrow v_t]$, $v_j - v$ should be already oriented as $v_j \rightarrow v$ when we apply Meek function M_{124} in Line 5 and Line 9, which is a contradiction.

3) In part 2, we stated that the union of set \mathbf{S} and set \mathbf{E}_g is consistent. Thus, based on part 1, oriented edges as a result of applying Meek function M_{124} on $\mathbf{E}_g \sqcup \mathbf{E}$ is also consistent. Therefore, the proof is complete.

Next, we will show that no cycle will be generated in the output graph \mathbf{E}_g during the execution of Algorithm 6. In the following lemma, we first prove that we do not have any directed cycle in this graph if there is no directed cycle of length three.

Lemma 35 *There is no directed cycle in a chordal graph if there exists no directed cycle of length three in that chordal graph.*

Proof Given a graph $G = (\mathbf{V}, \mathbf{E})$ with set of nodes \mathbf{V} and set of edges \mathbf{E} . Suppose there is a directed cycle $(v_1, \dots, v_i, \dots, v_j, \dots, v_1)$ with length N in graph G . Without loss of generality, we assume that there is a chord between nodes v_i and v_j . Thus, based on the direction of edges between nodes v_i and v_j , we know one of the paths $(v_1, \dots, v_i, v_j, \dots, v_1)$ or (v_i, \dots, v_j, v_i) will be a directed cycle with length less than N . Thus, we can obtain a cycle with length less than N from the original directed cycle. We can repeat this procedure until end up with a directed cycle of size three. Hence we can conclude that a cycle of size three exists in a chordal graph with directed cycle. \blacksquare

We know there is no cycle of length 3 in the set \mathbf{E} , and we will show that no such cycle will be generated during the execution of Algorithm 6. To do so, we need to prove three facts: 1) No cycle will be generated in Line 5. 2) No cycle will be generated during constructing \mathbf{S} in Line 8. 3) No cycle will be created as the result of applying M_{124} on $\mathbf{E}_G \sqcup \mathbf{S}$. We will prove them in the sequel.

Recall that the directed edges in \mathbf{E}_G remain consistent during execution of Algorithm 6. We will use this statement in the following proofs.

1) We prove this by induction on the number of steps taken by applying Meek function M_{124} . For the base case, it is obvious that there is no cycle of length three in the set \mathbf{E} . For the induction step, suppose that there is no directed cycle of length three up to step r . We show that no such a cycle will be generated in step $r + 1$. We show this by contradiction. First, consider Meek function M_2 . Applying this function on a Meek candidate sub-graph does not generate a cycle on that sub-graph. Therefore, we only consider the case in which the oriented edge as the result of applying Meek function M_2 in a candidate sub-graph makes cycle in another sub-graph. In that case we will have the following structure as $\{v_i \rightarrow v_j, v_j \rightarrow v_k, v_k \rightarrow v_l, v_l \rightarrow v_i, v_j - v_l\}$. In such sub-graph, we have $v_l \rightarrow v_k \in DP[v_i \rightarrow v_j]$. This means the mentioned orientations violates the consistency, which is a contradiction.

The all sub-graphs that are able to make a new cycle after applying Meek function M_1 or M_4 are depicted in Table 7. We check each of these cases and show that none of them can make a cycle. Case 1 and 2 belong to Meek function M_1 , and the remaining two cases are for Meek function M_4 . Black edges are already oriented and the red edge is the one that applying specific Meek function on that edge will orient green edges. We intend to check whether the green edge makes a cycle or not. As the first case violates the consistency, it cannot be happened. In case 2, no cycle is generated. In case 3, similar to case 1, the consistency is violated. Finally, no cycle will be generated in case 4.

2) In this case, we orient some edges connected to node v . Thus, for making a cycle C by orienting some edges during the \mathbf{S} construction, two case can be considered. In addition of already directed edges, one (the first case) or two (the second case) edges are oriented in C in the procedure of \mathbf{S} construction. The first case cannot be occurred because it means we have a Meek candidate sub-graph M_2 while the Meek functions M_{124} in Line 5 or Line 9 has already been applied. In the second case, if both edges are oriented, they are out-going edges and cannot make a cycle.

Table 7: Different cases that make cycle after applying Meek function M_1 or M_4

Case 1	Case 2	Case 3	Case 4
Meek function M_1		Meek function M_4	

3) In this part, we prove that the set of directed edges in the result of applying Meek function M_{124} on $\mathbf{S} \sqcup \mathbf{E}_g$ cannot generate a cycle. To do so, we recall from part 1 that applying Meek function M_{124} on a set with the consistent directed edges cannot make a cycle. As we showed, the union of set \mathbf{S} and set \mathbf{E}_g is consistent, no cycle exists in the result of applying Meek function M_{124} on this set.

All in all, we proved that for any \mathbf{E} as an input that is satisfying our assumptions, the set of directed edges after each iteration of the algorithm are subset or equal of a causal DAG in true MEC. As the number of edges are finite, the algorithm terminates and return a causal DAG in true MEC and the proof is complete.

References

- Raj Agrawal, Chandler Squires, Karren Yang, Karthikeyan Shanmugam, and Caroline Uhler. Abcd-strategy: Budgeted experimental design for targeted causal structure discovery. *Proceedings of Machine Learning Research*, 89, 2019.
- Steen A. Andersson, David Madigan, and Michael D. Perlman. A characterization of markov equivalence classes for acyclic digraphs. *Annals of Statistics*, 25(2):505–541, 1997.
- Laura E. Brown, Ioannis Tsamardinos, and Constantin F. Aliferis. A comparison of novel and state-of-the-art polynomial bayesian network learning algorithms. *Proceedings of the 20th National Conference on Artificial Intelligence*, page pp.739–745, 2005.
- David Maxwell Chickering. A transformational characterization of equivalent bayesian-network structures. *Proceedings of Eleventh Conference on Uncertainty in Artificial intelligence, Montreal, QU*, pages pages 87–98, 1995.
- Frederick Eberhardt, Clark Glymour, and Richard Scheines. On the number of experiments sufficient and in the worst case necessary to identify all causal relations among n variables. *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence (UAI)*, page 178–184, 2005.
- Marco F Eigenmann, Preetam Nandy, and Marloes H Maathuis. Structure learning of linear gaussian structural equation models with weak edges. *arXiv preprint arXiv:1707.07560*, 2017.
- AmirEmad Ghassami, Saber Salehkaleybar, Negar Kiyavash, and Elias Bareinboim. Budgeted experiment design for causal structure learning. *International Conference on Machine Learning*, page 1724–1733, 2018.
- AmirEmad Ghassami, Saber Salehkaleybar, Negar Kiyavash, and Kun Zhang. Counting and sampling from markov equivalent dags using clique trees. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33:3664–3671, Jul 2019.
- Kristjan Greenewald, Dmitriy Katz, Karthikeyan Shanmugam, Sara Magliacane, Murat Kocaoglu, Enric Boix-Adsera, and Guy Bresler. Sample efficient active learning of causal trees. *33rd Conference on Neural Information Processing Systems (NeurIPS)*, 2019.
- Alain Hauser and Peter Buhlmann. Characterization and greedy learning of interventional markov equivalence classes of directed acyclic graphs. *Journal of Machine Learning Research*, 13(1):2409–2464, August 2012.
- Alain Hauser and Peter Buhlmann. Two optimal strategies for active learning of causal models from interventional data. *International Journal of Approximate Reasoning*, 55(4): 926–939, Jun 2014.
- Yangbo He and Zhi Geng. Active learning of causal networks with intervention experiments and optimal designs. *The Journal of Machine Learning Research*, 9(Nov):2523–2547, 2008.

- Yangbo He and Bin Yu. Formulas for counting the sizes of markov equivalence classes of directed acyclic graphs. *arXiv preprint arXiv:1610.07921*, 2016.
- Yangbo He, Jinzhu Jia, and Bin Yu. Counting and exploring sizes of markov equivalence classes of directed acyclic graphs. *The Journal of Machine Learning Research*, 16(1): 2589–2609, 2015.
- Patrik O Hoyer, Aapo Hyvarinen, Richard Scheines, Peter L Spirtes, Joseph Ramsey, Gustavo Lacerda, and Shohei Shimizu. Causal discovery of linear acyclic models with arbitrary distributions. *arXiv preprint arXiv:1206.3260*, 2012.
- Markus Kalisch, Martin Machler, Diego Colombo, Marloes H. Maathuis, and Peter Bühlmann. Causal inference using graphical models with the R package pcalg. *Journal of Statistical Software*, 47(11):1–26, 2012. URL <https://www.jstatsoft.org/article/view/v047i11>.
- Murat Kocaoglu, Alex Dimakis, and Sriram Vishwanath. Cost-optimal learning of causal graphs. In *International Conference on Machine Learning*, 2017.
- Marloes H. Maathuis, Markus Kalisch, and Peter Buhlmann. Estimating high-dimensional intervention effects from observational data. *The Annals of Statistics*, page 37(6A):3133–3164, 2009.
- Lilian Markenzon, Oswaldo Vernet, and Luiz Henrique Araujo. Two methods for the generation of chordal graphs. *Ann. of Op. Res.*, 157(1):47–60, 2008.
- Christopher Meek. Causal inference and causal explanation with background knowledge. *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, page 403–410, 1995.
- Robert O Ness, Karen Sachs, Parag Mallick, and Olga Vitek. A bayesian active learning experimental design for inferring signaling networks. In *International Conference on Research in Computational Molecular Biology, Springer*, page 134–156, 2017.
- Judea Pearl. *Causality*. Cambridge university press, 2009.
- Jonas Peters, Dominik Janzing, and Bernhard Scholkopf. Elements of causal inference: foundations and learning algorithms. *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*, page pages 1–9, 2017.
- Donald J. Rose. Triangulated graphs and the elimination process. *J. Math. Anal. Appl.*, page 597–609, 1970.
- Dominik Rothenhausler, Jan Ernest, and Peter Buhlmann. Causal inference in partially linear structural equation models: identifiability and estimation. *Ann. Stat. To appear*, 2018.

- Karthikeyan Shanmugam, Murat Kocaoglu, Alexandros, G. Dimakis, and Sriram Vishwanath. Learning causal graphs with small interventions. *In Advances in Neural Information Processing Systems*, page 3195–3203, 2015.
- Peter Spirtes, Clark Glymour, and Richard Scheines. *Causation, prediction, and search*. MIT press, 2000.
- Topi Talvitie and Mikko Koivisto. Counting and sampling markov equivalent directed acyclic graphs. *In The Thirty-Third AAAI Conference on Artificial Intelligence*, 9(Nov): 7984–7991, 2019.
- Ali Ahmadi Teshnizi, Saber Salehkaleybar, and Negar Kiyavash. Lazyiter: A fast algorithm for counting markov equivalent dags and designing experiments. *In International Conference on Machine Learning*, page 1663–1671, 2020.
- Thomas Verma and Judea Pearl. An algorithm for deciding if a set of observed independencies has a causal explanation. *Uncertainty in Artificial Intelligence*, pages 323–330, 1992.
- Yuhao Wang, Liam Solus, Karren Yang, and Caroline Uhler. Permutation-based causal inference algorithms with interventions. *In Advances in Neural Information Processing Systems*, page 5822–5831, 2017.