Journal of Machine Learning Research 24 (2023) 1-32 Submitted 11/21; Revised 6/23; Published 7/23

Clustering and Structural Robustness in Causal Diagrams

Santtu Tikka SANTTU.TIKKA@JYU.FI
Jouni Helske JOUNI.HELSKE@JYU.FI
Juha Karvanen JUHA.T.KARVANEN@JYU.FI
Department of Mathematics and Statistics
P.O.Box 35 (MaD) FI-40014 University of Jyvaskyla, Finland

Editor: Francis Bach

Abstract

Graphs are commonly used to represent and visualize causal relations. For a small number of variables, this approach provides a succinct and clear view of the scenario at hand. As the number of variables under study increases, the graphical approach may become impractical, and the clarity of the representation is lost. Clustering of variables is a natural way to reduce the size of the causal diagram, but it may erroneously change the essential properties of the causal relations if implemented arbitrarily. We define a specific type of cluster, called transit cluster, that is guaranteed to preserve the identifiability properties of causal effects under certain conditions. We provide a sound and complete algorithm for finding all transit clusters in a given graph and demonstrate how clustering can simplify the identification of causal effects. We also study the inverse problem, where one starts with a clustered graph and looks for extended graphs where the identifiability properties of causal effects remain unchanged. We show that this kind of structural robustness is closely related to transit clusters.

Keywords: causal inference, graph theory, algorithm, identifiability, directed acyclic graph

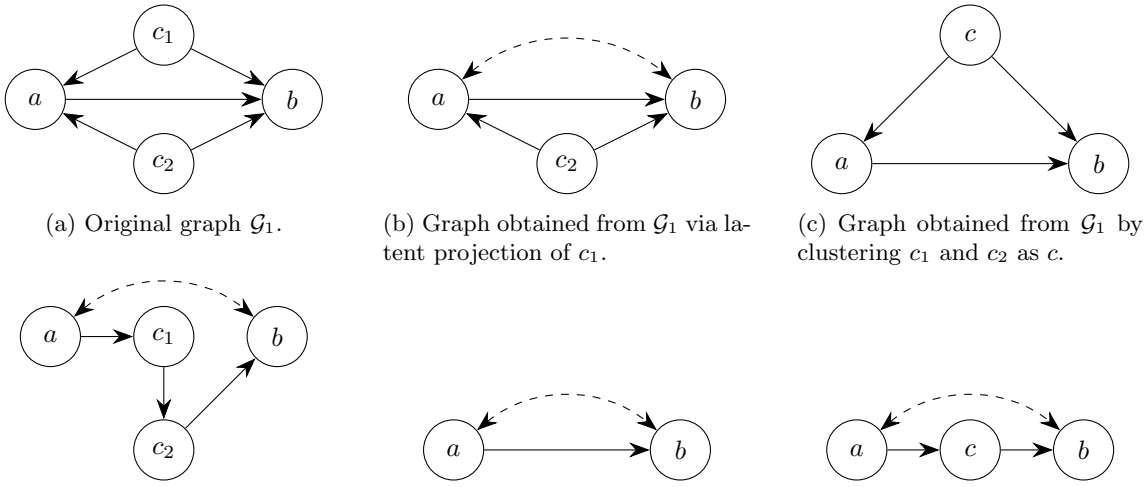
1. Introduction

Directed acyclic graphs (DAGs) and their extensions are commonly used to describe causal relations between variables in epidemiology and other fields (Pearl, 1995; Greenland et al., 1999; Tennant et al., 2021). The power of graphs lies in their ability to visualize the assumed structure, and at the same time, to serve as well-defined inputs for algorithms such as those that solve the nonparametric identifiability of causal effects (Shpitser and Pearl, 2006; Lee et al., 2019; Lee and Shpitser, 2020; Tikka et al., 2021). The graphical approach has been criticized by proponents of potential outcome framework (Rubin, 1974) for its impracticality when a large number of variables is considered (Imbens, 2020). This criticism is partially justified: the visual clarity of a graph is easily lost when the number of vertices is more than a few, especially in the case of several crossing edges (Purchase, 1997). Moreover, in some settings the identifiability of causal effects is an NP-hard problem (Tikka et al., 2019), which makes it impractical to consider large graphs. A possible remedy for these difficulties is to cluster the variables to reduce the size of the graph. A question then arises whether the clustered graph and the original graph are equivalent with respect to the identifiability properties of causal effects.

The idea of clustering is natural and has been used in causal inference explicitly and implicitly. The back-door criterion (Pearl, 1993), the front-door criterion (Pearl, 1995), and ignorability assumptions in the potential outcome framework (Rosenbaum and Rubin, 1983) impose conditions upon a set (i.e., a cluster) of variables and the structure inside the set is not important. Explicitly, clusters have been constructed starting from structural equations (Skorstad, 1990) or multivariate data (Entner and Hoyer, 2012; Parviainen and Kaski, 2017; Nisimov et al., 2021). Outside causal inference, many clustering methods for directed graphs have been proposed under varying premises (Malliaros and Vazirgiannis, 2013).

Clustering can be also viewed from a different starting point as a way to construct causal models where the causal relationships between clusters of variables are specified instead of the relationships between the variables themselves. For instance, a recent review (Tennant et al., 2021) found that many DAGs in applied health research included so-called "super-nodes" (Kornaropoulos and Tollis, 2013) which represent multiple variables with the implicit assumption of strong connectivity of the corresponding vertices. This viewpoint emphasizes the uncertainty of structural assumptions and the fact that we may not possess sufficient knowledge about the domain under study to fully specify individual relationships between variables. The goal in this type of clustering is structural robustness; inferences made with the clustered graph can be safely applied in any graph that is compatible with the clustering, but not necessarily vice versa. This approach was considered in a formal setting by Anand et al. (2021).

Clustering is different from latent projection (Verma, 1993) that also can be used to simplify the structure of DAG in causal inference (Tikka and Karvanen, 2018). Figure 1 demonstrates that a cluster of vertices is not always equivalent to a latent projection in terms of identification. We consider the identifiability of a query $p(x_b \,|\, \mathrm{do}(x_a))$ from observations $p(x_a, x_b, x_{c_1}, x_{c_2})$. In this task, we may apply clustering $C = \{c_1, c_2\}$ and obtain an identifying formula similar to the original solution. However, if we use a latent projection to consider either c_1 or c_2 as unobserved in graph $\mathcal{G}_1$, a bidirected edge between a and b appears, and the query is not identifiable anymore. In the graph $\mathcal{G}_2$, the query is not identifiable if a latent projection is used to consider both c_1 and c_2 as unobserved, although projecting only either c_1 or c_2 retains the identifiability. On the other hand, arbitrary clustering of variables in a DAG does not necessarily retain the identifiability either.



(a) Original graph $\mathcal{G}_1$.

(b) Graph obtained from $\mathcal{G}_1$ via latent projection of c_1.

(c) Graph obtained from $\mathcal{G}_1$ by clustering c_1 and c_2 as c.

(d) Original graph $\mathcal{G}_2$.

(e) Graph obtained from $\mathcal{G}_2$ via latent projection of c_1 and c_2.

(f) Graph obtained from $\mathcal{G}_2$ by clustering c_1 and c_2 as c.

Figure 1: Two examples on clustering of vertices.

As the first contribution, we introduce a specific type of cluster, called transit cluster, and present conditions for the equivalence of causal effect identifiability between the original and the clustered graph. We consider clustering as an operation that transforms a DAG into a new DAG where the cluster is represented by a single vertex. Our approach toward clustering builds on the intuitive idea that information flows through a cluster and the detailed structure inside the cluster is often irrelevant. We assume that the DAG being clustered is fully specified.

As the second contribution, we provide a sound and complete algorithm for finding all transit clusters in a given graph and demonstrate how clustering can simplify the identification of causal

effects. While polynomial-time algorithms exist for many important causal identification problems, the resulting identifying functional can be complicated (Tikka and Karvanen, 2017b). Clustering vertices in the graph can reduce the computational burden and lead to identifying functionals with a simpler structure.

As the third contribution, we study the inverse problem, where one starts with a clustered graph and looks for extended graphs where the identifiability properties of causal effects remain unchanged. This problem is related to the top-down causal modeling where one starts by creating the DAG with concepts, such as "work history", "socio-economic background" or "genetic factors", and only later divides these concepts into actual variables. Here a transit cluster represents this kind of concept. We present an iterative procedure that can be used to extend a single vertex to an arbitrary transit cluster. We show that transit clusters are structurally robust in the sense that under certain conditions the structure inside the cluster is irrelevant for identification. A schematic illustration of contributions of the paper is presented in Figure 2.

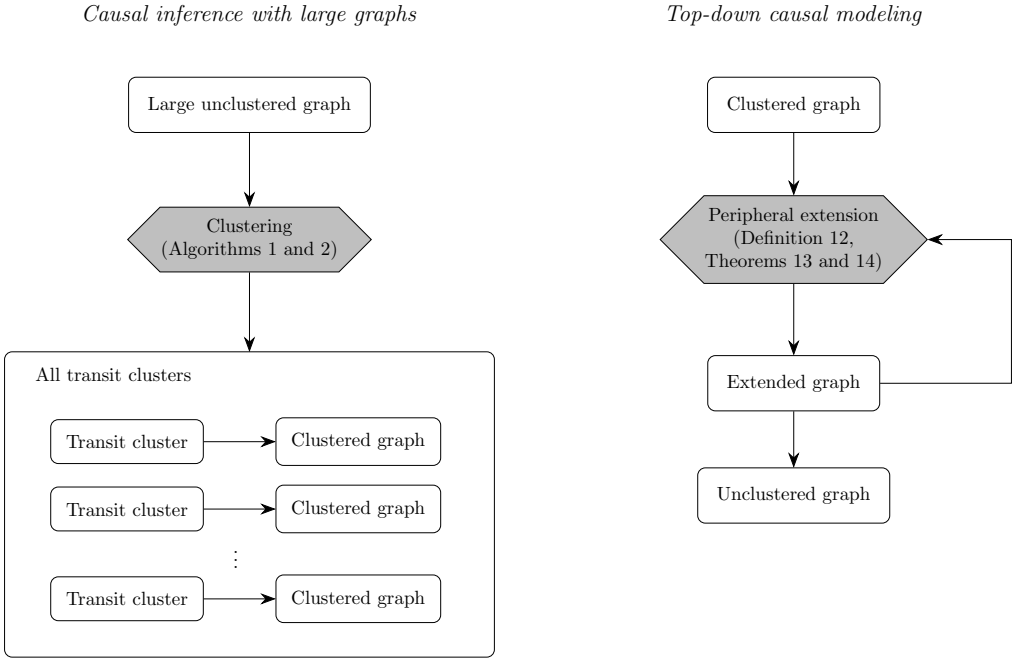


Figure 2: A schematic illustration of the contributions of the paper. In causal inference with large graphs (left), the application of the proposed clustering algorithm to a large graph results in a collection of all transit clusters, each of which corresponds to a clustered graph. In top-down causal modeling (right), we start with a clustered graph and iteratively apply peripheral extension to obtain an unclustered graph that shares the key properties with the clustered graph.

The rest of the paper is organized as follows. In Section 2, we define the transit cluster and prove its key properties. In Section 3, we present an algorithm for finding all transit clusters of a DAG and prove that it is sound and complete. After considering clustering from a purely graphical point of view in Sections 2 and 3, we then proceed to consider clustering in causal diagrams in Section 4, where we provide results on the identifiability of causal effects for specific transit clusters. In Section 5, we consider structural robustness and its connection to transit clusters. Illustrative examples on the clustering and structural robustness are given Section 6. Section 7 concludes the paper. Code for the clustering algorithms and examples are available in a GitHub repository: `https://github.com/santikka/transit_cluster`.

2. Clustering Vertices in DAGs

We begin by introducing the notation used for directed graphs. A DAG $\mathcal{G} = (V, E)$ is an ordered pair of two sets where V is a set of indices (vertices), i.e., $V = \{1, \ldots, n\}$, and E is a set of ordered pairs (directed edges) $E \subseteq \{(i, j) \mid i, j \in V\}$. Vertices and edges are denoted with small letters.

In a DAG $\mathcal{G} = (V, E)$, $\mathrm{Pa}_{\mathcal{G}}(A)$, $\mathrm{Ch}_{\mathcal{G}}(A)$, $\mathrm{An}_{\mathcal{G}}(A)$ and $\mathrm{De}_{\mathcal{G}}(A)$ denote the parents, children, ancestors and descendants of vertex set $A \subseteq V$ including A, respectively. The neighbors of a vertex set A including A is denoted by $\mathrm{Ne}_{\mathcal{G}}(A) \equiv \mathrm{Ch}_{\mathcal{G}}(A) \cup \mathrm{Pa}_{\mathcal{G}}(A)$. Vertices connected to A including A is denoted by $\mathrm{Co}_{\mathcal{G}}(A)$. The corresponding sets of the previous that exclude A are denoted by $\mathrm{Pa}^*_{\mathcal{G}}(A)$, $\mathrm{Ch}^*_{\mathcal{G}}(A)$, $\mathrm{An}^*_{\mathcal{G}}(A)$, $\mathrm{De}^*_{\mathcal{G}}(A)$, $\mathrm{Ne}^*_{\mathcal{G}}(A)$, and $\mathrm{Co}^*_{\mathcal{G}}(A)$. If there is only one relevant graph $\mathcal{G}$ in a given context, we will sometimes omit the subscript from these sets for clarity, and simply write $\mathrm{Pa}(A)$ or $\mathrm{Pa}^*(A)$, for example.

We use the notation $\mathcal{G}[W]$ to denote an induced subgraph (W, F) of $\mathcal{G} = (V, E)$, where $W \subseteq V$ and F contains those edges of E with both endpoints in W. Similarly, $\mathcal{G}[\overline{A}, \underline{B}]$ denotes an induced edge subgraph obtained from $\mathcal{G}$ by removing incoming edges to $A \subseteq V$ and outgoing edges of $B \subseteq V$. The collection of vertex sets that induce the components of $\mathcal{G}$ is denoted by $\mathcal{C}(\mathcal{G})$.

By a cluster we mean a subset of vertices of a DAG. Note that there are different definitions of "clustering" and "cluster graph" in other contexts. The motivation for the name "cluster" becomes evident when we consider a graph that represents the cluster as a single vertex:

Definition 1 (Clustering) *Clustering of a set of vertices $T \subset V$ in a DAG $\mathcal{G} = (V, E)$ induces a graph $\mathcal{G}' = (V', E')$ obtained from $\mathcal{G}$ by removing vertices T and adding a new vertex t that has parents $Pa^*_{\mathcal{G}}(T)$ and children $Ch^*_{\mathcal{G}}(T)$. In addition, sets $W \subset V$ and $W' \subset V'$ are clustering equivalent if $W \setminus T = W' \setminus \{t\}$ and $T \subset W$ if and only if $t \in W'$.*

Definition 1 captures the intuitive idea of clustering where the incoming and outgoing edges of the cluster are the same as the incoming and outgoing edges of its representative in the induced graph. However, without any constraints on the set T being clustered, this definition is too general in the sense that the properties of the induced graph may be drastically different from the original graph. For example, the induced graph is not necessarily a DAG or it may contain paths that were not present in the original graph.

Next, we will present conditions for the clustered vertices T that guarantee the usefulness of the clustering. Our approach is based on the intuitive notion that the effects flow through the cluster and the edges between the clustered vertices do not matter. Only those edges that connect to vertices outside the cluster are relevant. For this purpose, we define two special sets of vertices.

Definition 2 (Receiver) *For a set of vertices T in a DAG $\mathcal{G} = (V, E)$, the set of* receivers *is the set*

$$Re_{\mathcal{G}}(T) \equiv \{v \in T \mid Pa_{\mathcal{G}}(v) \cap (V \setminus T) \neq \emptyset\}.$$

The set of receivers for a set of vertices $T \subset V$ are those members of T that have parents outside of T in $\mathcal{G}$. To complement the receivers, we also define the following.

Definition 3 (Emitter) *For a set of vertices T in a DAG $\mathcal{G} = (V, E)$, the set of* emitters *is the set*

$$Em_{\mathcal{G}}(T) \equiv \{v \in T \mid Ch_{\mathcal{G}}(v) \cap (V \setminus T) \neq \emptyset\}.$$

We will use shortcut notation $\mathcal{G}[T^{=}]$ to denote a subgraph induced by T such that the incoming edges to receivers of T and outgoing edges from emitters of T are removed. We are now ready to define a cluster that preserves the fundamental structure of the graph.

Definition 4 (Transit cluster) *A non-empty set $T \subset V$ is a* transit cluster *in a connected DAG $\mathcal{G} = (V, E)$ if the following conditions hold*

1. $Pa(r_i) \setminus T = Pa(r_j) \setminus T$ *for all pairs* $r_i, r_j \in Re(T)$,
2. $Ch(e_i) \setminus T = Ch(e_j) \setminus T$ *for all pairs* $e_i, e_j \in Em(T)$,
3. *For all vertices* $t_i \in T$, *there exists a receiver* r *or an emitter* e *such that* t_i *and* r *or* t_i *and* e *are connected via an undirected path in* $\mathcal{G}[T^{=}]$.
4. *If* $Em(T) \neq \emptyset$ *then for all* $r \in Re(T)$ *there exists* $e \in Em(T)$ *such that* $e \in De(r)$,
5. *If* $Re(T) \neq \emptyset$ *then for all* $e \in Em(T)$ *there exists* $r \in Re(T)$ *such that* $r \in An(e)$.

In other words, a transit cluster is a set of vertices such that any member of its receivers has the same parents outside of the set, and any member of its emitters has the same children outside of the set. Additionally, we disallow those vertices from belonging to the cluster that are disconnected from the receivers or emitters when incoming edges of receivers and outgoing edges of emitters have been removed. Finally, for any receiver, there is always a directed path connecting that receiver to an emitter, and conversely, for any emitter, there is always a directed path connecting a receiver to that emitter. Together, these features endow the grouped set of vertices with several desirable properties. The set of all transit clusters of $\mathcal{G}$ is denoted by $\Upsilon_{\mathcal{G}}$.

The purpose behind the first two conditions in Definition 4 is to ensure that no new paths from the parents of the receivers to the children of emitters are created by performing the clustering. The third condition ensures that a transit cluster is characterized by its receivers and emitters, as we will show later. The last two conditions enforce the idea of information flow through the cluster. Examples of transit clusters are presented in Figure 3.

Definitions 1–4 allow us to characterize the graph induced by a transit cluster as follows:

Corollary 5 *The induced graph* $\mathcal{G}'$ *of transit cluster* T *is constructed from* $\mathcal{G}$ *by replacing* T *with a single vertex* t *such that* $Pa^*_{\mathcal{G}'}(t) = Pa^*_{\mathcal{G}}(Re_{\mathcal{G}}(T)) \setminus T$ *and* $Ch^*_{\mathcal{G}'}(t) = Ch^*_{\mathcal{G}}(Em_{\mathcal{G}}(T)) \setminus T$.

We consider some desirable basic properties of transit clusters. We delegate the proofs of all results to Appendix A. First, we must ensure that the graph induced by a transit cluster does not contain cycles.

Lemma 6 *Graph* $\mathcal{G}'$ *induced by a transit cluster* T *in a DAG* $\mathcal{G}$ *is a DAG.*

Next, we note that transit clusters are uniquely defined by their receivers and emitters.

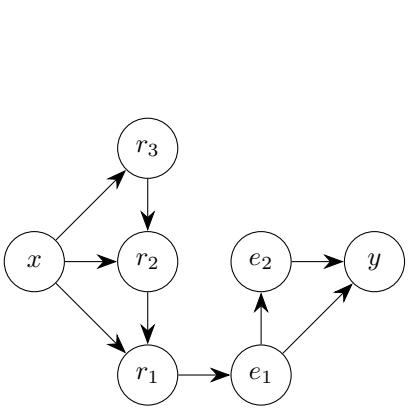
Lemma 7 *Let* T *and* S *be transit clusters in a DAG* $\mathcal{G} = (V, E)$. *If* $Re(T) = Re(S)$ *and* $Em(T) = Em(S)$, *then* $T = S$.

Intuitively, if clustering is carried out for multiple vertex sets in sequence, the order in which the clustering is carried out should not matter in terms of the graph obtained after the last set has been clustered. This notion is captured by the next two theorems. The first one states that a transit cluster remains a transit cluster even if a disjoint transit cluster is clustered.

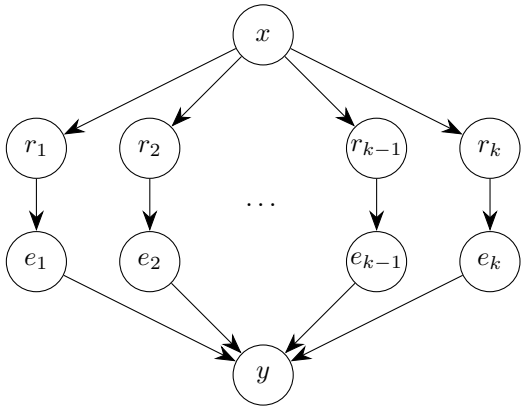
Theorem 8 (Invariance of transit clusters) *Let* T *be a transit cluster in* $\mathcal{G} = (V, E)$ *and let* $\mathcal{G}'$ *be the induced graph where* T *is replaced by a single vertex* t. *The set* $S \subset V \setminus T$ *is a transit cluster in* $\mathcal{G}$ *if and only if it is a transit cluster in* $\mathcal{G}'$.

A complementary result to the previous theorem guarantees that a transit cluster will still be a transit cluster even if its subset is clustered.

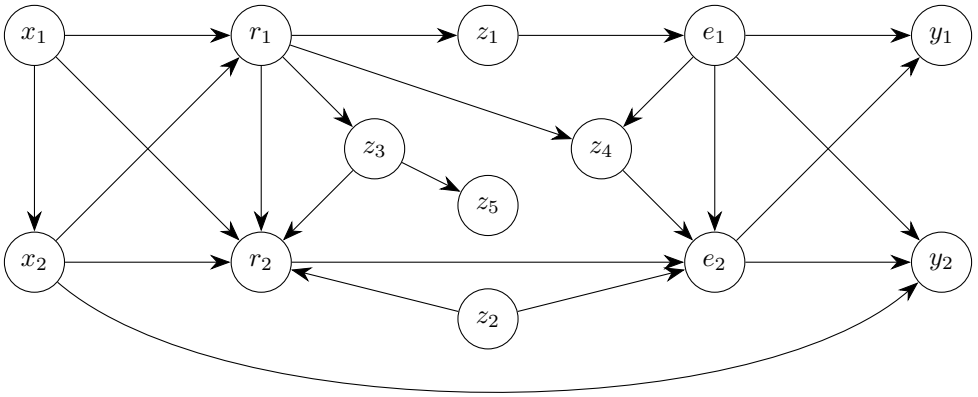
Theorem 9 (Modularity of transit clusters) *Let* T *be a transit cluster in graph* $\mathcal{G} = (V, E)$ *and* $\mathcal{G}'$ *the induced graph where* T *is replaced by a single vertex* t. *Let* $S \subset V \setminus T$. *The set* $\{t\} \cup S$ *is a transit cluster in* $\mathcal{G}'$ *if and only if* $T \cup S$ *is a transit cluster in* $\mathcal{G}$.



(a) Transit cluster $\{r_1, r_2, r_3, e_1, e_2\}$.



(b) Transit cluster $\{r_1, r_2, \ldots, r_k, e_1, e_2, \ldots, e_k\}$.



(c) Transit cluster $\{r_1, r_2, e_1, e_2, z_1, z_2, z_3, z_4, z_5\}$.

Figure 3: Examples of transit clusters. In addition to presented transit clusters, there are other transit clusters, for instance transit cluster $\{r_1, e_1\}$ in panel (a).

Theorem 9 helps us to characterize the conditions for valid unions of transit clusters. The following corollary plays a key role in the algorithmic approach to finding transit clusters in Section 3.

Corollary 10 (Union of transit clusters) *Let disjoint sets S and T be transit clusters in $\mathcal{G}$. $S \cup T$ is a transit cluster in $\mathcal{G}$ if $Pa^*(Re(S)) = Pa^*(Re(T))$ and $Ch^*(Em(S)) = Ch^*(Em(T))$.*

While other types of unions of transit clusters can result in valid transit clusters, it turns out that the particular union specified by Corollary 10 is the only one we actually need.

In a practical setting, there may be vertices that we cannot or do not want to include in the same cluster with other vertices. Thus the set of possible clusters may be restricted.

Definition 11 (Restricted transit cluster) *Let $\mathcal{G} = (V, E)$ be a DAG and $R \subseteq V$. A restricted transit cluster $T \subset V$ with respect to R in $\mathcal{G}$ is a transit cluster in $\mathcal{G}$ such that $T \subseteq R$.*

We denote the set of all restricted transit clusters with respect to R by $\Upsilon_{\mathcal{G}|R} \equiv \{T \mid T \in \Upsilon_{\mathcal{G}}, T \subseteq R\}$. The next definition specifies the operations that can be applied to a DAG so that a transit cluster remains a transit cluster.

Definition 12 (Peripheral extension) *Let $T = \{t_1, \ldots, t_k\}$ be a transit cluster in a DAG $\mathcal{G}$. Let $\mathcal{G}^+$ be a DAG obtained from $\mathcal{G}$ by one of the following operations:*

1. *Add an edge $t_i \to t_j$ where $t_i, t_j \in T$ and the edge does not create a cycle,*
2. *Replace the edge $t_i \to t_j$ by the path $t_i \to t_{k+1} \to t_j$ where t_{k+1} is a new vertex,*
3. *Divide vertex t_i as follows: add a new vertex t_{k+1} such that $Ch^*_{\mathcal{G}^+}(t_{k+1}) = Ch^*_{\mathcal{G}}(t_i)$, remove all outgoing edges of t_i and add edge $t_i \to t_{k+1}$.*
4. *Add a new vertex t_{k+1} and the edge $t_{k+1} \to t_i$ and where $t_i \in T \setminus Re(T)$,*
5. *Add a new vertex t_{k+1} and the edge $t_i \to t_{k+1}$ where $t_i \in T \setminus Em(T)$.*

Adding a receiver or an emitter when $Re(T) \neq \emptyset$ and $Em(T) \neq \emptyset$:

6. *Add a new vertex t_{k+1} (a new receiver) such that $Pa^*_{\mathcal{G}^+}(t_{k+1}) = Pa^*_{\mathcal{G}}(Re(T))$, and add an edge $t_{k+1} \to t_j$ where $t_j \in T$ is on a path from a receiver to an emitter in $\mathcal{G}$.*
7. *Add a new vertex t_{k+1} (a new emitter) such that $Ch^*_{\mathcal{G}^+}(t_{k+1}) = Ch^*_{\mathcal{G}}(Em(T))$, and add an edge $t_j \to t_{k+1}$ where $t_j \in T$ is on a path from a receiver to an emitter in $\mathcal{G}$.*
8. *Add two new vertices t_{k+1} (a new receiver) and t_{k+2} (a new emitter) such that $Pa^*_{\mathcal{G}^+}(t_{k+1}) = Pa^*_{\mathcal{G}}(Re(T))$ and $Ch^*_{\mathcal{G}^+}(t_{k+2}) = Ch^*_{\mathcal{G}}(Em(T))$, and add the edge $t_{k+1} \to t_{k+2}$.*

Adding a receiver when $Re(T) \neq \emptyset$ and $Em(T) = \emptyset$:

9. *Add a new vertex t_{k+1} (a new receiver) such that $Pa^*_{\mathcal{G}^+}(t_{k+1}) = Pa^*_{\mathcal{G}}(Re(T))$.*

Adding an emitter when $Re(T) = \emptyset$ and $Em(T) \neq \emptyset$:

10. *Add a new vertex t_{k+1} (a new emitter) such that $Ch^*_{\mathcal{G}^+}(t_{k+1}) = Ch^*_{\mathcal{G}}(Em(T))$.*

Now define $T^+ = T$ if operation 1 was applied, $T^+ = T \cup \{t_{k+1}\}$ if operation 2, 3, 4, 5, 6, 7, 9 or 10 was applied, and $T^+ = T \cup \{t_{k+1}, t_{k+2}\}$ if operation 8 was applied. In all cases, T^+ is a peripheral extension *of T, and $\mathcal{G}^+$ is the corresponding peripheral extension graph.*

Operations 1–5 modify the internal structure of the transit cluster. New vertices and edges can be added but new parents for receivers or new children for emitters cannot be added with these operations. Operations 6–10 add new receivers and emitters. Here the allowed operations differ for transit clusters that have only receivers, only emitters and both receivers and emitters. Special conditions are needed to make sure that T^+ fulfills the conditions of Definition 4. The following theorem shows that a peripheral extension always results in a transit cluster.

Theorem 13 *Let T^+ be a peripheral extension of a transit cluster T in a DAG $\mathcal{G} = (V, E)$ and let $\mathcal{G}'$ be the induced graph where T is replaced by a single vertex. Then T^+ is a transit cluster in the corresponding peripheral extension graph $\mathcal{G}^+$ and $\mathcal{G}'$ is the induced graph of T^+.*

A complementary result shows that any transit cluster can be constructed iteratively with operations of Definition 12.

Theorem 14 *Let $\mathcal{G}'$ be the induced graph of a transit cluster constructed from $\mathcal{G}$ by replacing set T by a single vertex t. Then $\mathcal{G}$ and T can be constructed by iteratively applying operations of Definition 12 to transit cluster $\{t\}$ in $\mathcal{G}'$.*

So far, our focus has been on transit clusters in general. It turns out that transit clusters can always be constructed from smaller "building blocks" which we call transit components. Furthermore, it is much easier to find transit components in a given DAG than transit clusters.

Definition 15 (Transit component) *A transit cluster T in a DAG $\mathcal{G}$ is a transit component if T is connected in $\mathcal{G}[T]$.*

We extend the notion of restriction to transit components: the set of all restricted transit components of a DAG $\mathcal{G}$ with respect to the set R is denoted by $\mathcal{T}_{\mathcal{G}|R}$, and when $R = V$, i.e., when the aforementioned set corresponds to components without restriction, we simply write $\mathcal{T}_{\mathcal{G}}$. The intuitive idea behind the purpose of transit components is encapsulated in the following theorem.

Theorem 16 (Transit cluster decomposition) *Let T be a transit cluster in a DAG $\mathcal{G} = (V, E)$. If T is not a transit component, then there exists a transit cluster S and a transit component R such that $T = S \cup R$ and $S \cap R = \emptyset$.*

In simpler terms, any transit cluster can always be constructed iteratively from disjoint transit components. The transit cluster decomposition plays a key role in finding transit clusters.

3. Clustering Algorithms

To find valid vertex clusters, we can always apply a naive approach and enumerate every vertex subset of a DAG $\mathcal{G}$ and determine whether the conditions of Definition 4 hold. However, this approach quickly becomes infeasible with larger graphs. In light of Theorem 16, we can instead start by constructing the set of all transit components, and then obtain the set of all transit clusters by applying Corollary 10 to the components. We begin by presenting a sound and complete algorithm for finding transit components that exploit the structure of the graph by enumerating a set of candidate receiver and emitter sets.

Algorithm 1 (FindTrComp) starts by constructing the candidate sets for potential receivers and emitters, $\mathcal{V}_{\text{Ch}}$ and $\mathcal{V}_{\text{Pa}}$, respectively on lines 3 and 4. Lemma 7 shows that we only need to consider the receivers and emitters to uniquely specify a transit component. Next, we iterate over all pairs of the candidates on lines 5 and 6. Lines 7–9 restrict the candidates into mutually ancestral sets Z and W, and further exclude those candidates that cannot satisfy the properties of a transit cluster. If at least one of the obtained candidate sets Z and W is nonempty on line 11, we move on to construct a candidate transit component A on line 12. If A obeys the restriction defined by R on line 13, we move on to the iteration over the components of A in the induced subgraph $\mathcal{G}[A]$ on line 14. Lines 15 and 16 define those members of the current candidates Z and W that belong to the current component A_k as Z_k and W_k respectively. Finally, we determine whether the members of Z_k have the same parents, and whether members of W_k have the same children during lines 17–19. If this is the case, a new transit component of $\mathcal{G}$ has been found, and it is added to the set $\mathcal{A}$ of components found so far. Finally, this set is returned on line 21 after the outermost iterations have been completed.

We proceed to show that FindTrComp always terminates.

Lemma 17 FindTrComp *always terminates for valid inputs $\mathcal{G}$ and R.*

Lemma 17 allows us to consider the output of FindTrComp. To show soundness of the algorithm, me must show that the output set only contains restricted transit components.

Theorem 18 (Soundness of FindTrComp) *Let $\mathcal{G} = (V, E)$ be a DAG, $R \subseteq V$ and $\mathcal{A} =$ FindTrComp$(\mathcal{G}, R)$, then $\mathcal{A} \subseteq \mathcal{T}_{\mathcal{G}|R}$.*

Conversely for completeness of FindTrComp, we must show that any restricted transit component of a DAG will be found by the algorithm.

Theorem 19 (Completeness of FindTrComp) *Let $\mathcal{G} = (V, E)$ be a DAG, $R \subseteq V$, and $\mathcal{A} =$ FindTrComp$(\mathcal{G}, R)$ then $\mathcal{T}_{\mathcal{G}|R} \subseteq \mathcal{A}$.*

Algorithm 1 An algorithm for finding all (restricted) transit components of a DAG. The inputs are a DAG $\mathcal{G} = (V, E)$ and a restriction set R whose members are the only vertices in V that are allowed to belong to any transit component. Returns the set of all restricted transit components in $\mathcal{G}$ with respect to R.

function FindTrComp($\mathcal{G}, R$)
  $\mathcal{A} \leftarrow \emptyset$
  $\mathcal{V}_{\text{Ch}} \leftarrow \{\text{Ch}^*(v) \cap R \mid v \in V\}$
  $\mathcal{V}_{\text{Pa}} \leftarrow \{\text{Pa}^*(v) \cap R \mid v \in V\}$
  for all $V_i \in \mathcal{V}_{\text{Ch}}$ **do**
    for all $V_j \in \mathcal{V}_{\text{Pa}}$ **do**
      if $V_j \neq \emptyset$ **then** $Z \leftarrow V_i \cap \text{An}_{\mathcal{G}}(V_j)$ **else** $Z \leftarrow V_i$
      if $V_i \neq \emptyset$ **then** $W \leftarrow V_j \cap \text{De}_{\mathcal{G}}(V_i)$ **else** $W \leftarrow V_j$
      if $(\exists z_k \in Z \text{ s.t. } \text{Pa}^*_{\mathcal{G}}(z_k) = \emptyset) \vee (\exists w_k \in W \text{ s.t. } \text{Ch}^*_{\mathcal{G}}(w_k) = \emptyset)$ **then**
        continue
      if $Z \neq \emptyset \vee W \neq \emptyset$ **then**
        $A \leftarrow \text{Co}_{\mathcal{G}[\overline{Z}, \underline{W}]}(Z \cup W)$
        if $A \subseteq R$ **then**
          for all $A_k \in \mathcal{C}(\mathcal{G}[A])$ **do**
            $Z_k \leftarrow Z \cap A_k$
            $W_k \leftarrow W \cap A_k$
            $Z_{\text{Pa}} \leftarrow \bigcap_{z \in Z_k} \text{Pa}^*_{\mathcal{G}}(z) \setminus A_k$
            $W_{\text{Ch}} \leftarrow \bigcap_{w \in W_k} \text{Ch}^*_{\mathcal{G}}(w) \setminus A_k$
            if $Z_{\text{Pa}} = \text{Pa}^*(Z_k) \setminus A_k \wedge W_{\text{Ch}} = \text{Ch}^*(W_k) \setminus A_k$ **then**
              $\mathcal{A} \leftarrow \mathcal{A} \cup \{A_k\}$
  return $\mathcal{A}$

FindTrComp operates in polynomial time with respect to the size of the graph.

Theorem 20 FindTrComp *outputs all restricted transit components of a DAG $G = (V, E)$ with respect to $R \subseteq V$ in $O(|V|^4 + |V|^3|E|)$ time.*

Theorem 20 also gives an upper bound for the number of distinct transit components of a DAG. To obtain a crude approximation, we note that there are $|V|^2$ total candidate pairs (Z, W) considered by FindTrComp, each of which can produce up to $|V|$ distinct transit clusters (the maximum amount of components in any induced subgraph), which leads to an upper bound of $|V|^3$ transit components. However, we can find the smallest possible upper bound. First, we present a utility lemma for counting transit components.

Lemma 21 *Let T be a transit component of a DAG $\mathcal{G} = (V, E)$ and let $G' = (V', E')$ be the induced graph of the clustering with t representing the set T. If there does not exist a transit component S of $\mathcal{G}$ such that $T \cap S \neq \emptyset$ and $T \setminus S \neq \emptyset$, then $|\mathcal{T}_{\mathcal{G}}| = |\mathcal{T}_{\mathcal{G}'}| + |\mathcal{T}_{\mathcal{G}[T]}|$.*

In other words, Lemma 21 states that if there exists a transit component T such that no other transit component partially intersects it, then the number of transit components in the original graph $\mathcal{G}$ is the sum of the number of transit components in the graph induced by clustering T and the number of transit components in the subgraph induced by T.

Theorem 22 *Let $\mathcal{G} = (V, E)$ be a DAG. Then $|\mathcal{T}_{\mathcal{G}}| \leq \frac{|V|(|V|+1)}{2} - 1$.*

In contrast, the upper bound for the number of transit clusters grows exponentially as the number of vertices in the graph grows, hence ruling out an efficient algorithm for listing all transit clusters.

Figure 3(b) shows an example where any combination of transit components $\{r_i, e_i\}$, $i = 1, \ldots, k$ is a transit cluster and the number of non-singleton transit clusters is $2^k - 1$.

Fortunately, by carefully combining transit components, we can construct an algorithm that lists all transit clusters with a polynomial delay. Algorithm 2 (FindTrClust) attempts to recursively combine transit components into transit clusters using Corollary 10 to detect which unions are valid. Theorem 16 guarantees that all transit clusters will be found by this approach.

Algorithm 2 An algorithm for finding all (restricted) transit clusters of a DAG. The inputs are a DAG $\mathcal{G} = (V, E)$ and the set of all transit components $\mathcal{T}_{\mathcal{G}|R}$ with respect to $R \subset V$. Returns the set of all restricted transit clusters in $\mathcal{G}$ with respect to R. The subroutine ExpandClust is used to recursively construct the clusters.

```
function FindTrClust(𝒢, 𝒯_{𝒢|R})
    𝒜 ← 𝒯_{𝒢|R}
    ℬ ← 𝒯_{𝒢|R}
    for all T ∈ 𝒯_{𝒢|R} do
        ℬ ← ℬ \ {T}
        𝒜 ← 𝒜 ∪ ExpandClust(T, 𝒜, ℬ, 𝒢)
    return 𝒜
function ExpandClust(T, 𝒜, ℬ, 𝒢)
    ℬ′ ← ℬ
    for all S ∈ ℬ do
        ℬ′ ← ℬ′ \ {S}
        if S ∪ T ∉ 𝒜 and Corollary 10 holds for S and T then
            𝒜 ← 𝒜 ∪ {S ∪ T} ∪ ExpandClust(S ∪ T, 𝒜, ℬ′, 𝒢)
    return 𝒜
```

We begin by showing that FindTrClust always terminates.

Lemma 23 FindTrClust *always terminates for valid inputs* $\mathcal{G}$ *and* $\mathcal{T}_{\mathcal{G}|R}$.

Lemma 23 guarantees that the output of FindTrClust is well-defined. Next, we show that the output of the algorithm is a set of transit clusters.

Theorem 24 (Soundness of FindTrClust) *Let* $\mathcal{G} = (V, E)$ *be a DAG,* $R \subseteq V$*, and* $\mathcal{A} = \text{FindTrClust}(\mathcal{G}, \mathcal{T}_{\mathcal{G}|R})$*, then* $\mathcal{A} \subseteq \Upsilon_{\mathcal{G}|R}$.

For the inverse, we show that any transit cluster is found by FindTrClust.

Theorem 25 (Completeness of FindTrClust) *Let* $\mathcal{G} = (V, E)$ *be a DAG,* $R \subseteq V$*, and* $\mathcal{A} = \text{FindTrClust}(\mathcal{G}, \mathcal{T}_{\mathcal{G}|R})$ *then* $\Upsilon_{\mathcal{G}|R} \subseteq \mathcal{A}$.

Finally, we prove that all transit clusters of a DAG can be listed with a polynomial delay.

Theorem 26 FindTrClust *outputs all restricted transit clusters of a DAG* $\mathcal{G} = (V, E)$ *with respect to* $R \subseteq V$ *with* $O\left(|V|^5\right)$ *polynomial delay and a* $O\left(|V| + |E|\right)$ *initialization delay.*

If FindTrClust and FindTrComp are run in sequence for the same DAG, a dynamic programming approach can be applied to further eliminate the preprocessing delay of FindTrComp by caching the parent and child sets of the receivers and emitters of each transit component during the operation of FindTrClust. We also note that in practice, the worst case performance only occurs in the first iteration of the outermost recursion level, because the number of possible unions of transit components always decreases in both the number of loop iterations and recursion depth.

Naturally, it is not necessary to obtain all transit components, and the iteration can be stopped for example when a cluster with some desired properties is found. Alternatively, one can consider

transit components directly, as they are valid transit clusters themselves, without attempting to find larger transit clusters at all. Furthermore, we note that it is not necessary to consider restrictions directly on the transit components in form of the set R. The same set of transit clusters can also be obtained by first finding the unrestricted transit clusters and then simply discarding those that violate the restrictions. This type of approach can be useful when the possible restrictions are not known beforehand.

4. Transit Clusters and Causal Inference

So far, we have considered clustering from a purely graphical point of view. However, in the context of causal inference and structural causal models of Pearl (2009), the causal model defines some variables as unobserved background variables and others as observed, which has to be accounted for when constructing transit clusters in causal diagrams.

Let W be a set of vertices. We denote by $(X_w)_{w\in W}$ a collection of random variables taking values in measurable spaces $(\mathfrak{X}_w)_{w\in W}$. We assume that the measurable spaces are finite-dimensional vector spaces or finite discrete sets. For $A \subseteq W$ let $\mathfrak{X}_A \equiv \times_{a\in A}(\mathfrak{X}_a)$ denote the product space, and $X_A \equiv (X_a)_{a\in A}$ the corresponding random vector. We will use $p(\cdot|\cdot)$ to denote joint distribution, marginal distributions, and conditional distributions of random variables.

To facilitate the concept where a single vertex represents the entire cluster in the induced graph of the clustering, we adopt a definition of a causal model that explicitly makes it possible for a single vertex of the causal diagram to correspond to a multivariate random variable. Assume that a DAG $\mathcal{G} = (V, E)$ is clustered as $\mathcal{G}' = (V', E')$ and let $W' \subset V'$ be the clustering equivalent set of $W \subset V$ resulting from a clustering of a transit cluster T. Suppose now that for each $v \in V$, there is a corresponding random variable X_v. We can now define clustering equivalent random variables as follows: for any $w' \in W' \setminus \{t\}$, $X_{w'} = X_w$, and $X_t = (X_w)_{w\in W\cap T}$, i.e., the random variables corresponding to the clustered vertices are combined into a new random variable and the variables unrelated to the transit cluster remain unchanged. Thus, for any functional g of the joint distribution it holds that $g(p(x_W)) = g(p(x_{W'}))$.

We define structural causal models analogously to Pearl (2009) using our notation.

Definition 27 (Causal model) *A causal model $\mathcal{M}$ is a tuple $(X_V, X_U, \mathcal{F}, p)$, where*

- *X_V is an observed random vector indexed by the set V.*
- *X_U is an unobserved random vector indexed by the set U.*
- *$\mathcal{F}$ is a collection of functions $(f_v)_{v\in V}$ such that each f_v is a mapping from $\mathfrak{X}_{U\cup(V\setminus\{v\})}$ to $\mathfrak{X}_v$ and such that $\mathcal{F}$ forms a mapping from $\mathfrak{X}_U$ to $\mathfrak{X}_V$. Symbolically, the set of equations $\mathcal{F}$ can be represented by writing $X_v = f_v(X_{Pa(v)}, X_{U(v)})$, where $X_{Pa(v)} \subseteq X_V$ is the unique minimal set of observed variables sufficient for representing f_v. Likewise, $X_{U(v)} \subseteq X_U$ stands for the unique minimal set of unobserved variables sufficient for representing f_v.*
- *p is the joint probability distribution of X_U.*

Each causal model $\mathcal{M}$ can be associated with a directed graph $\mathcal{G}(\mathcal{M})$ where the vertices correspond to the sets V and U and directed edges point from members of $Pa(v)$ and $U(v)$ to v. We refer to this graph as the causal diagram. We consider recursive semi-Markovian causal models in this paper, meaning that $\mathcal{G}(\mathcal{M})$ is a DAG and each $u \in U$ has at most two children in $\mathcal{G}(\mathcal{M})$. For simplicity, we assume that noise terms, i.e., vertices of unobserved variables with only one child, are always clustered together with their children when clustering is carried out. This makes it unnecessary to include such unobserved variables when drawing causal diagrams.

We make a distinction between vertices of a DAG and the random variables that they represent in the causal model and equate them only when it is suitable to do so. In figures that depict DAGs

that are causal diagrams, we draw vertices V that relate to observed variables as circles, and vertices U that relate to unobserved variables as squares. The $\mathrm{do}(x_A)$ operator denotes that the variables X_A are assigned fixed values x_a irrespective of their parents in the causal diagram.

We define the identifiability of a causal effect as follows.

Definition 28 (Identifiability) *An interventional distribution (causal effect) $p(x_A|do(x_B))$ is identifiable from $p(x_V)$ in a causal diagram $\mathcal{G} = (V, E)$ if it is uniquely computable from $p(x_V)$ in every causal model that has the causal diagram $\mathcal{G}$.*

Causal effect identification is a well-known problem in causal inference, and a complete algorithm exists for the problem of determining whether $p(x_A \,|\, \mathrm{do}(x_B))$ is identifiable in a causal diagram $\mathcal{G}$ from the observational distribution $p(x_V)$ (Shpitser and Pearl, 2006; Tikka and Karvanen, 2017a). An important graphical structure related to identifiability is a confounded component, or a c-component for short (sometimes also called a "district"). C-components are typically defined for semi-Markovian causal models, that is, models where unobserved variables have exactly two children and they are represented graphically by bidirected edges instead of including the corresponding variables explicitly in the graph. Because transit clusters can contain vertices that represent unobserved variables, we provide an equivalent definition of c-components for causal diagrams where unobserved variables are present.

Definition 29 (c-component) *Let $\mathcal{G} = (V \cup U, E)$ be a causal diagram of a causal model $\mathcal{M}$. If there exists a path between every pair of vertices $i, j \in V$ such that every vertex on the path that is a member of V is a collider with the exception of i and j, and the path contains at least one vertex that is a member of U, then $\mathcal{G}$ is a c-component.*

When c-components are defined as in Definition 29, we may define maximal c-components, c-trees, c-forests and hedges analogously they are defined by Shpitser and Pearl (2006) (see Appendix B for details). Any causal diagram that is not a c-component can be uniquely partitioned into a set of its maximal c-components. A causal effect $p(x_A \,|\, \mathrm{do}(x_B))$ is identifiable from $p(x_V)$ in $\mathcal{G} = (V \cup U, E)$ if and only if $\mathcal{G}$ does not contain a hedge for any $p(x_{A'} \,|\, \mathrm{do}(x_{B'}))$ in $\mathcal{G}$, where $A' \subseteq A$ and $B' \subseteq B$. The existence of a hedge means that the graph has two c-components that fulfill specific graphical conditions, and that can be used to construct two causal models that agree on $p(x_V)$ but disagree on $p(x_A \,|\, \mathrm{do}(x_B))$.

Identifiability may not be preserved by arbitrary clusters of vertices in the causal diagram, meaning that a causal effect may be identifiable in the original graph but not in the graph induced by the cluster or vice versa. Fortunately, transit clusters can be proven to preserve identifiability under specific conditions. The first condition requires that the emitters and the parents of receivers are observed:

Definition 30 *A transit cluster T in a DAG $\mathcal{G} = (V \cup U, E)$ of a causal model $\mathcal{M}$ is* plain *if $Pa^*(Re(T)) \cup Em(T) \subseteq V$.*

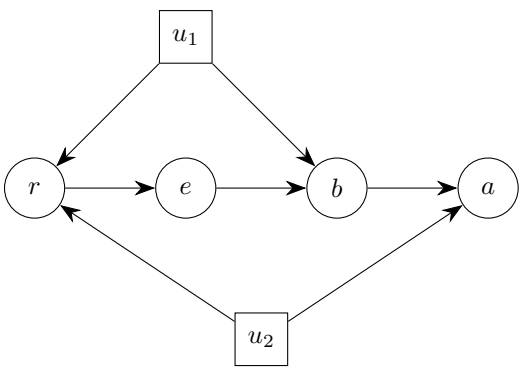
In a plain transit cluster, any latent confounders will always be clustered together with their children. The second condition requires that the entire transit cluster belongs to the same c-component:

Definition 31 *A transit cluster T in a DAG $\mathcal{G} = (V \cup U, E)$ of a causal model $\mathcal{M}$ is* congested *if all members of T belong to the same c-component in $\mathcal{G}$ and $Em(T) \subseteq V$.*

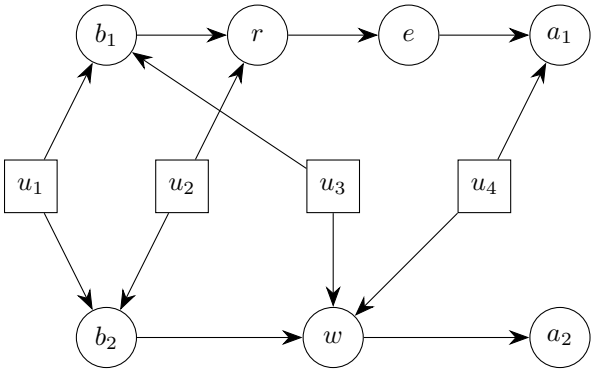
In essence, plain and congested transit clusters do not change the c-components of the causal diagram. Thus, identifiability is preserved for plain and congested transit clusters.

Theorem 32 *Let $\mathcal{G} = (V \cup U, E)$ be a DAG of a causal model $\mathcal{M}$ and let A and B be disjoint subsets of V. Let $T = \{t_1, \ldots, t_n\}$ be a restricted transit cluster with respect to $(V \cup U) \setminus (A \cup B)$ in $\mathcal{G}$, and let $\mathcal{G}' = (V' \cup U', E')$ be the induced graph of the cluster with vertex $t' \in V'$ as the representative of T. If T is plain or congested, then $p(x_A \,|\, do(x_B))$ is identifiable from $p(x_V)$ in $\mathcal{G}$ precisely when it is identifiable from $p(x_{V'})$ in $\mathcal{G}'$.*

Figure 4 demonstrates that in general identifiability may be lost in clustering even if the cluster is a transit cluster. In these examples, receiver r and emitter e form a transit cluster that is neither plain nor congested because r has unobserved variables as parents but r and e do not belong to the same c-component. Consequently, Theorem 32 does not apply for these transit clusters.



(a) Query $p(x_a \,|\, \mathrm{do}(x_b))$ is identified from $p(x_r, x_e, x_b, x_a)$.



(b) Query $p(x_{a_1}, x_{a_2} \,|\, \mathrm{do}(x_{b_1}, x_{b_2}))$ is identified from $p(x_{b_1}, x_{b_2}, x_r, x_e, x_w, x_{a_1}, x_{a_2})$.

Figure 4: Two examples where identifiability is lost when transit cluster $T = \{r, e\}$ is replaced by a single vertex. The vertices of unobserved variables are denoted by squares.

5. Robustness of Structural Assumptions

Consider now the top-down causal modeling where a cluster T represents observed variables $(X_{t_1}, \ldots, X_{t_n})$ and an unspecified number of unobserved background variables. The cluster is represented by a single node t in a DAG $\mathcal{G}'$ but the causal structure inside the cluster has not been specified. Assume that a causal effect $p(x_A|\mathrm{do}(x_B))$, where $T \cap (A \cup B) = \emptyset$, is identifiable from $p(x_{V'})$ under the structural assumptions coded in a causal diagram $\mathcal{G}'$ and $g(p(x_{V'}))$ is an identifying functional for $p(x_A|\mathrm{do}(x_B))$. We are interested in characterizing the internal structure of cluster T for which we can guarantee that $g(p(x_V))$ obtained from $g(p(x'_V))$ by explicitly replacing x_t by its observed components $(x_{t_1}, \ldots, x_{t_n})$, is an identifying functional for $p(x_A|\mathrm{do}(x_B))$ in $\mathcal{G}$. We will show that a plain or congested transit cluster fulfills this requirement of structural robustness.

We start with DAG $\mathcal{G}'$ where the single node transit cluster $\{t\}$ represents the random variables $X_{t_1}, \ldots, X_{t_n}, X_{u_1}, \ldots, X_{u_m}$, and the number of unobserved variables m has been chosen arbitrarily. We apply the peripheral extension of Definition 12 until all variables $X_{t_1}, \ldots, X_{t_n}, X_{u_1}, \ldots, X_{u_m}$ of the cluster are explicitly presented as vertices $\{t_1, \ldots, t_n, u_1, \ldots, u_m\}$ of graph $\mathcal{G}$. Finally, we state conditions that the identifying functional remains valid.

Theorem 33 *Let X_V be a vector of observed random variables, X_U a vector of unobserved random variables, and $\mathcal{G}' = (V' \cup U', E')$ the causal diagram of a causal model $\mathcal{M}'$ where vertex $t \in V'$ represents set $T = \{t_1, \ldots, t_n, u_1, \ldots, u_m\}$ in $\mathcal{G}'$, $t_1, \ldots, t_n \in V$, $u_1, \ldots, u_m \in U$, and $v \in V' \setminus \{t\}$ implies $v \in V$. Let $\mathcal{G} = (V \cup U, E)$ be a DAG obtained from $\mathcal{G}'$ by applying a series of peripheral extensions to vertex t such a way that $\mathcal{G}$ is a causal diagram. If T is a plain or congested transit cluster in $\mathcal{G}$, the following holds for disjoint subsets A and B of V' such that $T \cap (A \cup B) = \emptyset$.*

1. *Causal effect $p(x_A \,|\, do(x_B))$ is identifiable from $p(x_V)$ in $\mathcal{G}$ exactly when it is identifiable from $p(x_{V'})$ in $\mathcal{G}'$.*

2. *If $g(p(x_{V'}))$ is an identifying functional for $p(x_A \mid do(x_B))$ in $\mathcal{G}'$, it is also an identifying functional for $p(x_A \mid do(x_B))$ in $\mathcal{G}$.*

6. Use cases and illustrations

Transit clusters can be applied in various ways. Here we demonstrate their use in reducing the size of a causal diagram, simplification of identifying functionals, speeding up identification algorithms, and top-down causal modeling.

6.1 Reducing the size of a causal diagram

As an example of simplification of the causal diagrams and identifying functionals, as well as the robustness of causal effect estimation, we consider a graph related to the Sangiovese grapes studied earlier as a conditional linear Gaussian network by Magrini et al. (2017), where the interest was in the effect of various treatments X_b on the mean weight of grapes X_a. In addition to a and b, the graph contains vertices $z_1, \ldots, z_{13}$ which are related to different characteristics of the soil, grape plants and must (Magrini et al., 2017). Compared to the original graph, we added a latent confounder X_u between the treatment variable X_b and the mean weight of grapes X_a for illustrative purposes. We do not present the causal diagram graphically as its large number of vertices (16) and edges (57) makes it difficult to visualize clearly.

Applying algorithm 2 gives us transit cluster $T = \{z_1, \ldots, z_{13}\}$, with variables $r = \{z_1, z_2, z_4, z_{10}\}$ as receivers and $e = \{z_1, z_3, z_4, z_6, z_7, z_8, z_{10}, z_{12}, z_{13}\}$ as emitters (with variables z_5, z_9, and z_{11} being neither). This leads to a simplified graph in Figure 5.

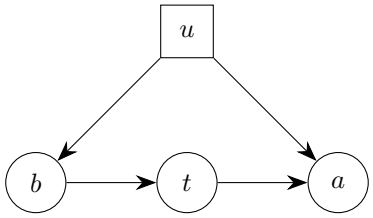


Figure 5: Induced graph of the Sangiovese graph where the transit cluster $T = \{z_1, \ldots, z_{13}\}$ is replaced with single vertex t.

6.2 Simplification of identifying functionals

The application of the ID-algorithm (Shpitser and Pearl, 2006) to original graph related to the Sangiovese grapes leads to long and complicated identifying functional:

$$
\begin{aligned}
p(x_a \mid \mathrm{do}(x_b)) = \sum_{x_{z_1}, \ldots, x_{z_{13}}} \Bigg[& p(x_{z_{13}} | x_b, x_{z_1}, \ldots, x_{z_{12}}) p(x_{z_{12}} | x_b, x_{z_1}, \ldots, x_{z_8}, x_{z_{10}}, x_{z_{11}}) \\
& \times p(x_{z_{11}} | x_b, x_{z_1}, \ldots, x_{z_8}, x_{z_{10}}) p(x_{z_{10}} | x_b, x_{z_1}, \ldots, x_{z_8}) p(x_{z_9} | x_b, x_{z_1}, \ldots, x_{z_8}) \\
& \times p(x_{z_8} | x_b, x_{z_1}, \ldots, x_{z_7}) p(x_{z_7} | x_b, x_{z_1}, \ldots, x_{z_6}) p(x_{z_6} | x_b, x_{z_1}, x_{z_2}, x_{z_3}, x_{z_5}) \\
& \times p(x_{z_5} | x_b, x_{z_1}, x_{z_2}, x_{z_3}) p(x_{z_4} | x_b, x_{z_1}) p(x_{z_3} | x_b, x_{z_1}, x_{z_2}) p(x_{z_2} | x_b, x_{z_1}) p(x_{z_1} | x_b) \\
& \times \left(\sum_{x'_b} p(x_a | x'_b, x_{z_1}, \ldots, x_{z_{13}}) p(x'_b) \right) \Bigg]
\end{aligned}
\tag{1}
$$

On the contrary, the application of the ID-algorithm to the clustered graph of Figure 5 leads to identifying functional of form

$$p(x_a \mid \mathrm{do}(x_b)) = \sum_{x_t} p(x_t|x_b) \left(\sum_{x'_b} p(x_a|x'_b, x_t) p(x'_b) \right), \tag{2}$$

i.e. front-door adjustment. This enables us to model $p(x_t|x_b)$ in an arbitrary, but consistent manner without making specific claims about the internal structure of T.

6.3 Speeding up identification algorithms

Identifying functionals obtained from the application of ID-algorithm or general do-calculus are often unnecessarily complex and could be further simplified (Tikka and Karvanen, 2017b). This can allow easier interpretation of the identifying functional and more efficient estimation of the causal effect. The R (R Core Team, 2022) package causaleffect (Tikka and Karvanen, 2017a) implements an automatic simplification algorithm for this task, however, the algorithm can be slow in case of large graphs. Alternatively if we can first simplify the graph by clustering, we can reduce the computational burden of both the identification algorithm as well as subsequent simplification algorithm. For example, in case of the Sangiovese graph, the causaleffect package returns (1) in 0.1 seconds with simplification option disabled and 132 seconds with simplification enabled (which in this case does not lead to simpler equation) on a standard laptop. On the other hand, running the clustering algorithm and subsequent identification (which returns (2)) takes only 0.5 seconds. Importantly, the simplification algorithm of Tikka and Karvanen (2017b) is NP-hard, and thus it may be possible to obtain simpler identifying functionals using transit clusters in scenarios where direct simplification is infeasible. The code for this benchmark is also available in the GitHub repository.

6.4 Top-down causal modeling

As an example of peripheral extension and the robustness of the estimation strategies, consider a causal graph shown in Figure 6, studied earlier by Helske et al. (2021), where the interest is in the causal effect of the education level X_e on income X_i. Variable X_s measures general language skills on Illinois Test of Psycholinguistic Abilities (ITPA), which is a composite of 12 subtests. Thus instead of vertex s representing a single composite variable X_s in Figure 6, we can by the peripheral extension (12) treat it as a transit cluster $T = \{s_1, \ldots, s_{12}\}$ (with s_i corresponding to the subtest i) without affecting the identifiability of the causal effect $p(x_i \mid \mathrm{do}(x_e))$. While the causal effect estimates can depend on whether we use X_s or X_T in the modelling, the obtained estimator is robust to these changes in a sense that the methodology of Helske et al. (2021) can be used to estimate the effect in both cases.

As another example of peripheral extension, we consider an epidemiological application studied earlier by Karvanen et al. (2020). The question of interest is the causal effect of salt-adding behavior on the salt intake. The high salt intake is one of the causes of hypertension (He et al., 2013).

The example is based on the National Health and Nutrition Examination Survey (NHANES, `https://wwwn.cdc.gov/nchs/nhanes/`) 2015–2016 that is an observational study on the health and nutritional status of adults and children in the United States. The NHANES variables are already divided into categories by their content and the type of the data. In the top-down modeling, these categories may correspond to transit clusters in the causal diagram. An example of a causal diagram constructed by this approach is shown in Figure 7. The peripheral extension (Definition 12) can be used to extend the clusters. For instance, the cluster represented by the vertex a, "Salt-adding behavior", may consists of the following variables measured in NHANES:

1. *How often do you add ordinary salt to your food at the table?* (Rarely 0, Occasionally 1, Very often 2)

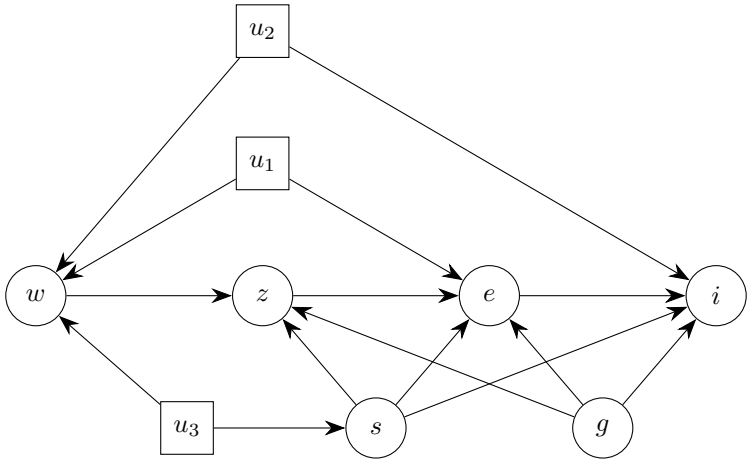


Figure 6: Causal diagram representing the effect of education level X_e on income X_i. Other variables represented in graph are gender X_g, score on Illinois Test of Psycholinguistic Abilities (ITPA) X_s, socioeconomic status of the parents X_w, and the grade point average X_z at the end of primary school. Variables $X_{u_1}, X_{u_2}, X_{u_3}$ are unobserved.

2. *Did you add any salt to your food at the table yesterday?* (No 0, Yes 1), and
3. *How often is ordinary salt or seasoned salt added in cooking or preparing foods in your household?* (Never 0, Rarely 1, Occasionally 2, Very often 3).

and the variables of the cluster represented by vertex b, "Diet behavior", may include

1. *Are you on low salt/low sodium diet?*
2. *Are you on other special diet?* (several options)
3. *Number of meals not home prepared during the past 7 days*
4. *Number of meals from fast food or pizza place during the past 7 days*

The use of transit clusters provides a formal justification for the top-down modeling. Especially, Theorem 33 states sufficient conditions for the validity of conclusions made with the clustered graph.

7. Discussion

We have considered clustering from two starting points. First, we started with an unclustered DAG that may have a large number of vertices and proposed algorithms for finding transit components and transit clusters, allowing us to simplify the representation of the DAG and the obtained identifying functional. Furthermore, we provided sufficient conditions for non-identifiability in a clustered DAG to imply non-identifiability in the original DAG. Second, we started with a clustered DAG where a single vertex represents a group of variables and presented the peripheral extension, a procedure for constructing all transit clusters that are compatible with the clustered DAG. We showed that an identifying functional for a causal effect in the clustered DAG remains valid in DAGs obtained via peripheral extension.

A transit cluster was deliberately defined for a DAG without any reference to a causal model. This allows us to cluster vertices even before it is known which data will be available. The division into observed and unobserved variables is however hard-coded into the definition of a structural

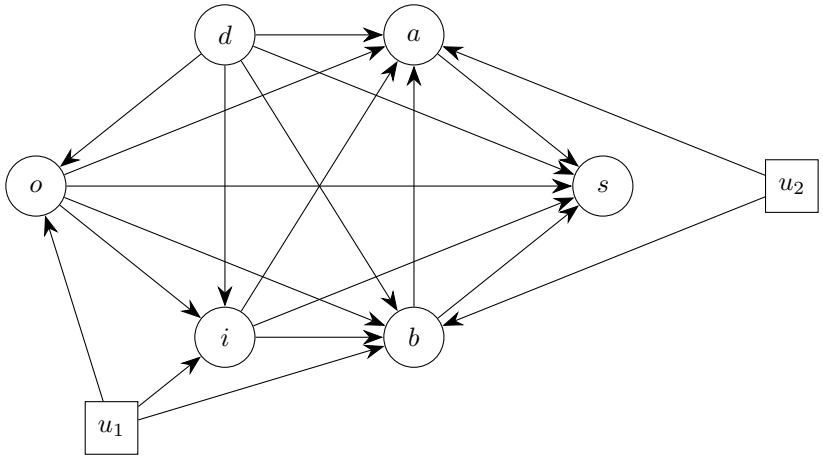


Figure 7: Causal model for the salt intake example. The vertices represent the following transit clusters of variables: Salt-adding behavior X_a (represented by a single vertex a), salt intake X_s, diet behavior X_b, demographic variables X_d, occupation X_o, and income X_i.

causal model where an unobserved variable cannot have parents. This restriction is taken into account in Theorems 32 and 33.

The DAG-based definition of a transit cluster makes it possible to apply a workflow where Algorithm 1 is first run for the whole graph in order to find all transit components. Restrictions may then be applied to these transit components before transit clusters are constructed by Algorithm 2. The same transit components can be re-used when the causal effect in the focus is changed to a new one that implies different restrictions for the transit clusters.

The examples presented in Section 6 illustrate the use of transit clusters in reducing the size of a causal diagram, simplifying identifying functionals, and speeding up identification algorithms. The identifying functional defined using the representative vertex in place of transit cluster allows a researcher to focus on the overall structure of the functional when choosing suitable estimation methods for the causal effect. Transit clusters also provide a justification for the top-down causal modeling.

In addition to the use cases considered, clustering could be beneficial also in causal discovery (Spirtes et al., 2000, 2001). If a set of variables can be assumed to form a transit cluster, we may, at least theoretically, use any single variable of the set as representative of the whole cluster when considering whether the cluster and a variable outside the cluster are d-separated. In general, causal discovery methods can construct the underlying DAG only up to an equivalence class and additional challenges with finite samples may occur due to a variety of reasons, such as measurement error (Zhang et al., 2017), selection bias (Zhang et al., 2016), or missing data (Tu et al., 2019). The assumption on a transit cluster could in some cases provide the information needed to reduce these ambiguities.

In future work, we would like to extend the results of Sections 4 and 5 to more general identifiability problems with multiple data sources consisting of a mix of observational and interventional distributions. We hypothesize, that at least plain transit clusters can be used to retain identifiability in more complex settings. It may also be possible to extend the definition of transit clusters to graphs where the direction of some edges is unknown.

Acknowledgments

This work was supported by Academy of Finland grant numbers 311877 and 331817.

Appendix A. Proofs

We restate and prove all results of the paper.

Lemma 6 *Graph $\mathcal{G}'$ induced by a transit cluster T in a DAG $\mathcal{G}$ is a DAG.*

Proof Let t be the single vertex in $\mathcal{G}'$ that corresponds to T in $\mathcal{G}$. We show that if there exists a directed path from v_1 to v_2 in $\mathcal{G}'$, there cannot also exist a directed path from v_2 to v_1. Assume first that both directed paths exist and neither of them contains t. This is a contradiction because then both paths would exist in a DAG $\mathcal{G}$ as well. Next, assume without loss of generality that the directed path from v_1 to v_2 contains t. It follows that $\mathcal{G}$ has a directed path $v_1 \to \ldots \to r \to \ldots \to e \to \ldots \to v_2$, where the existence of vertices $r \in \mathrm{Re}(T)$ and $e \in \mathrm{Em}(T)$ is guaranteed by the definition of transit cluster. Similarly, if the directed path from v_2 to v_1 contains t, there would a directed path from v_2 to v_1, which together with directed path from v_1 to v_2 would create a cycle in $\mathcal{G}$. If the directed path from v_2 to v_1 does not contain t, it will exist also in $\mathcal{G}$ and form a cycle in $\mathcal{G}$. We conclude that $\mathcal{G}'$ cannot have cycles and is thus a DAG. ■

Lemma 7 *Let T and S be transit clusters in a DAG $\mathcal{G} = (V, E)$. If $Re(T) = Re(S)$ and $Em(T) = Em(S)$, then $T = S$.*

Proof Suppose instead that there exists $t \in T$ such that $t \notin S$ and that $t \notin \mathrm{Re}(T) \cup \mathrm{Em}(T)$. By condition 3, t is connected to a receiver or an emitter in $\mathcal{G}[T^{=}]$. Assume first that t is connected to receiver r in $\mathcal{G}[T^{=}]$. As $\mathrm{Re}(T) = \mathrm{Re}(S)$, r is also a receiver for S. Follow the path from r to t and let s be the last vertex that belongs S and q the next vertex that does not belong to S. If there is an edge $s \to q$, s is an emitter for S and if there is an edge $q \to s$, s is a receiver for S. As $\mathrm{Re}(T) = \mathrm{Re}(S)$ and $\mathrm{Em}(T) = \mathrm{Em}(S)$, s is also a receiver or an emitter for T. This leads to a contradiction because by definition the edges incoming to receivers and outgoing from emitters are cut in $\mathcal{G}[T^{=}]$ and the path between t and r cannot exist. The case where t is connected to an emitter in $\mathcal{G}[T^{=}]$ proceeds analogously. ■

Theorem 8 (Invariance of transit clusters) *Let T be a transit cluster in $\mathcal{G} = (V, E)$ and let $\mathcal{G}'$ be the induced graph where T is replaced by a single vertex t. The set $S \subset V \setminus T$ is a transit cluster in $\mathcal{G}$ if and only if it is a transit cluster in $\mathcal{G}'$.*

Proof As S and T are disjoint, the vertices of S as well as the edges between vertices of S are unaffected by the clustering of T. It follows that $S' = S$, where S' is the clustering equivalent set of S. We will show that $\mathrm{Re}_{\mathcal{G}'}(S') = \mathrm{Re}_{\mathcal{G}}(S)$, $\mathrm{Em}_{\mathcal{G}'}(S') = \mathrm{Em}_{\mathcal{G}}(S)$, and S' fulfills the conditions of Definition 4 both in $\mathcal{G}$ and $\mathcal{G}'$. As the edges between S and $V \setminus (S \cup T)$ are unaffected by the clustering, it suffices to consider only edges between S and T. If a parent of $\mathrm{Re}_{\mathcal{G}}(S)$ belongs to T, condition 1 applied to S in $\mathcal{G}$ guarantees that vertex t will be a parent of all vertices in $\mathrm{Re}_{\mathcal{G}}(S)$ in $\mathcal{G}'$. If t is a parent of $\mathrm{Re}_{\mathcal{G}'}(S')$, condition 1 applied to S' in $\mathcal{G}'$ guarantees that any member of T that is a parent of a receiver in $\mathcal{G}$ will be a parent of all vertices in $\mathrm{Re}_{\mathcal{G}}(S)$ in $\mathcal{G}$. Similarly, if a child of $\mathrm{Em}_{\mathcal{G}}(S)$ belongs to T, vertex t will be a children of all vertices in $\mathrm{Em}_{\mathcal{G}}(S)$ in $\mathcal{G}'$ and if t is a child of $\mathrm{Em}_{\mathcal{G}'}(S')$, any member of T that is a child of an emitter in $\mathcal{G}$ will be a child of all vertices in $\mathrm{Em}_{\mathcal{G}}(S)$ in $\mathcal{G}$. It follows that $\mathrm{Re}_{\mathcal{G}'}(S') = \mathrm{Re}_{\mathcal{G}}(S)$, $\mathrm{Em}_{\mathcal{G}'}(S') = \mathrm{Em}_{\mathcal{G}}(S)$ and conditions 1 and 2 are fulfilled for $S' = S$ both in $\mathcal{G}$ and $\mathcal{G}'$. Conditions 3, 4 and 5 are fulfilled as well as they consider only paths inside S'. ■

Theorem 9 (Modularity of transit clusters) *Let T be a transit cluster in graph $\mathcal{G} = (V, E)$ and $\mathcal{G}'$ the induced graph where T is replaced by a single vertex t. Let $S \subset V \setminus T$. The set $\{t\} \cup S$ is a transit cluster in $\mathcal{G}'$ if and only if $T \cup S$ is a transit cluster in $\mathcal{G}$.*

Proof Denote $Q = T \cup S$ and $Q' = \{t\} \cup S$.

First, we assume that $\{t\} \cup S$ is transit cluster in $\mathcal{G}'$ and show that $T \cup S$ is transit cluster in $\mathcal{G}$.

Condition 1: We will show for all $r \in \mathrm{Re}_{\mathcal{G}}(Q)$ that $\mathrm{Pa}_{\mathcal{G}}(r) \setminus Q = \mathrm{Pa}_{\mathcal{G}'}(\mathrm{Re}_{\mathcal{G}'}(Q')) \setminus Q'$. First let $v \notin Q$ to be a parent of receiver r in $\mathcal{G}$. It follows that in $\mathcal{G}'$, vertex v is a parent of r if $r \in S$ or a parent of t if $r \in T$ because $v \notin Q$ in $\mathcal{G}$. It follows that r or t is a receiver in $\mathcal{G}'$ and $v \in \mathrm{Pa}_{\mathcal{G}'}(\mathrm{Re}_{\mathcal{G}'}(Q')) \setminus Q'$.

Now let $v \in \mathrm{Pa}_{\mathcal{G}'}(\mathrm{Re}_{\mathcal{G}'}(Q')) \setminus Q'$ and consider two cases: a) If v is a parent of t in $\mathcal{G}'$ then t is a receiver in $\mathcal{G}'$. Further, v is a parent of some $t_i \in T$ in $\mathcal{G}$ because t is a single vertex corresponding to transit cluster T in $\mathcal{G}$. It holds $t_i \in \mathrm{Re}_{\mathcal{G}}(Q)$ because $v \notin Q$. Condition 1 of transit cluster guarantees that v is also a parent of r. b) If v is a parent of $s_i \in S$ in $\mathcal{G}'$ then v is a parent of $s_i \in S$ also in $\mathcal{G}$ because $v \notin Q'$ in $\mathcal{G}'$. Thus s_i is a receiver in $\mathcal{G}$ and $v \in \mathrm{Pa}(\mathrm{Re}(Q)_{\mathcal{G}})_{\mathcal{G}} \setminus Q$.

Condition 2: The proof is analogous to condition 1.

Condition 3: Consider $q_i \in Q$. Assume first that $q_i \in T$. There exist in $\mathcal{G}'$ vertex v that is a receiver for Q' or an emitter for Q' and is connected to t. Applying conditions 1, 2, 4 and 5 for transit cluster T guarantees that there a path between q_i and v in $\mathcal{G}$. If $v \in S$, it is the required receiver or emitter for Q. If $v = t$, applying conditions 1, 2, 4 and 5 to transit cluster T guarantees that set T has the required receiver or emitter.

Assume next that $q_i \in S$. There exist in $\mathcal{G}'$ vertex v that is a receiver for Q' or an emitter for Q' and is connected to q_i. If $v \in S$, it satisfies the condition 3 for Q because applying conditions 4 and 5 to transit cluster T guarantees that t can be replaced by a path consisting of vertices in T. If $v = t$, applying conditions 1, 2, 4 and 5 for transit cluster T guarantees that set T has the required receiver or emitter.

Condition 4: Assume $\mathrm{Em}_{\mathcal{G}}(Q) \neq \emptyset$ and let $r \in \mathrm{Re}_{\mathcal{G}}(Q)$. a) If $r \in T$ then t is a receiver in $\mathcal{G}'$ by the proof of condition 1. It follows that there exists $e' \in \mathrm{Em}_{\mathcal{G}'}(Q')$ such that $e' \in \mathrm{De}_{\mathcal{G}'}(t)$. If $e' \neq t$, it directly fulfills condition 4. If $e' = t$, there exists $e \in \mathrm{Em}_{\mathcal{G}}(T)$ such that $e \in \mathrm{De}_{\mathcal{G}}(r)$, which fulfills condition 4. b) If $r \in S$ then $r \in \mathrm{Re}_{\mathcal{G}'}(Q')$ and there exists $e \in \mathrm{Em}_{\mathcal{G}'}(Q')$ such that $e \in \mathrm{De}_{\mathcal{G}'}(r)$. If $e \in S$, it fulfills condition 4 because Q' is a transit cluster. If $e = t$, there exists an emitter in T that fulfills condition 4 because T is a transit cluster.

Condition 5: The proof is analogous to condition 4.

Next, we assume that $T \cup S$ is transit cluster in $\mathcal{G}$ and show that $\{t\} \cup S$ is transit cluster in $\mathcal{G}'$. Conditions 1 and 2 are already covered above when we showed that $\mathrm{Pa}(\mathrm{Re}(Q)_{\mathcal{G}})_{\mathcal{G}} \setminus Q = \mathrm{Pa}_{\mathcal{G}'}(\mathrm{Re}_{\mathcal{G}'}(Q')) \setminus Q'$ and $\mathrm{Ch}(\mathrm{Em}(Q)_{\mathcal{G}})_{\mathcal{G}} \setminus Q = \mathrm{Ch}_{\mathcal{G}'}(\mathrm{Em}_{\mathcal{G}'}(Q')) \setminus Q'$.

Condition 3: Consider $q_i \in Q'$. Assume first that $q_i = t$. For any $t_i \in T$ there exist some vertex v that is the required receiver or emitter in $\mathcal{G}$. By the definition of clustering, t and v are connected in $\mathcal{G}'$. Assume next that $q_i \in S$. There exist in $\mathcal{G}$ vertex v that is a receiver for Q or an emitter for Q and is connected to q_i. If $v \in S$, it satisfies the condition 3. If $v \in T$, vertex t is the required receiver or emitter for q_i in $\mathcal{G}'$.

Condition 4: Assume $\mathrm{Em}_{\mathcal{G}'}(Q) \neq \emptyset$ and let $r \in \mathrm{Re}_{\mathcal{G}'}(Q')$. a) If $r = t$, set $\mathrm{Re}_{\mathcal{G}}(Q) \cap T$ is non-empty and all members of this set fulfill condition 4 in $\mathcal{G}$. Let there be a path from $r_i \in \mathrm{Re}_{\mathcal{G}}(Q) \cap T$ to $e \in \mathrm{Em}_{\mathcal{G}}(Q)$ in $\mathcal{G}$. It follows that either $e \in T$ and t is the requested emitter in $\mathcal{G}'$ or $e \in S$ and there is a path from t to e in $\mathcal{G}'$ and e is the requested emitter.

Condition 5: The proof is analogous to condition 4. ■

Corollary 10 (Union of transit clusters) *Let disjoint sets S and T be transit clusters in $\mathcal{G}$. $S \cup T$ is a transit cluster in $\mathcal{G}$ if $Pa^*(Re(S)) = Pa^*(Re(T))$ and $Ch^*(Em(S)) = Ch^*(Em(T))$.*

Proof Consider graph $\mathcal{G}'$ where the transit cluster S is replaced by vertex s and graph $\mathcal{G}''$ where transit clusters S and T are replaced by vertices s and t, respectively. It is easy to check that set $\{s, t\}$ is a transit cluster in $\mathcal{G}''$ under the assumption that $\mathrm{Pa}^*(\mathrm{Re}(S)) = \mathrm{Pa}^*(\mathrm{Re}(T))$ and $\mathrm{Ch}^*(\mathrm{Em}(S)) = \mathrm{Ch}^*(\mathrm{Em}(T))$. By applying Theorem 9 to $\mathcal{G}''$ we conclude that $\{s\} \cup T$ is a transit cluster in $\mathcal{G}'$. By applying Theorem 9 then to $\mathcal{G}'$, we conclude that $S \cup T$ is a transit cluster in $\mathcal{G}$. ■

Theorem 13 *Let T^+ be a peripheral extension of a transit cluster T in a DAG $\mathcal{G} = (V, E)$ and let $\mathcal{G}'$ be the induced graph where T is replaced by a single vertex. Then T^+ is a transit cluster in the corresponding peripheral extension graph $\mathcal{G}^+$ and $\mathcal{G}'$ is the induced graph of T^+.*

Proof Let $\mathcal{G}^+$ be the peripheral extension graph of T^+. A new receiver is created by operations 6, 8 and 9. A new emitter is created by operations 7, 8 and 10. A new emitter can be created also by operation 3. Operations 6, 7, 8, 9 and 10 explicitly copy parents and children so that the conditions 1 and 2 of transit cluster hold for T^+. If t_i is an emitter in operation 3, it will not be an emitter in $\mathcal{G}^+$. Copying the children to new vertex t_{k+1} ensures that condition 2 is fulfilled for t_{k+1}.

Before an operation, condition 3 hold for all existing vertices in T. Condition 3 holds also in $\mathcal{G}^+$ for all existing vertices because the operations do not change the existence of the current paths. Condition 3 holds for the new vertex t_{k+1} added by operations 3, 4 and 5 because condition 3 holds for t_i, and t_{k+1} is connected to t_i. Condition 3 directly holds for a receiver or an emitter added by operation 6, 7, 8, 9 or 10.

We also conclude that if there exist a path in $\mathcal{G}$ from $r \in \mathrm{Re}_{\mathcal{G}}(T)$ to $e \in \mathrm{Em}_{\mathcal{G}}(T)$, there also exist a path from r to e in $\mathcal{G}^+$ because operation 1 does not remove edges or vertices, operations 2 and 3 preserve directed paths, operations 4 and 5 do not affect receivers and emitters, operations 6, 7 and 8 explicitly add an edge to create the required directed path, and operations 9 and 10 do not apply to cases where T has both receivers and emitters. It follows that the conditions 4 and 5 of transit cluster hold for T^+. Graph $\mathcal{G}'$ is the induced graph of $\mathcal{G}^+$ because $\mathrm{Pa}_{\mathcal{G}^*}(\mathrm{Re}_{\mathcal{G}^*}(T^*)) \setminus T^* = \mathrm{Pa}_{\mathcal{G}}(\mathrm{Re}_{\mathcal{G}}(T)) \setminus T$ and $\mathrm{Ch}_{\mathcal{G}^+}(\mathrm{Em}_{\mathcal{G}^+}(T^+)) \setminus T^+ = \mathrm{Ch}_{\mathcal{G}}(\mathrm{Em}_{\mathcal{G}}(T)) \setminus T$. ■

Theorem 14 *Let $\mathcal{G}'$ be the induced graph of a transit cluster constructed from $\mathcal{G}$ by replacing set T by a single vertex t. Then $\mathcal{G}$ and T can be constructed by iteratively applying operations of Definition 12 to transit cluster $\{t\}$ in $\mathcal{G}'$.*

Proof At the beginning, all vertices in T are unmarked, i.e., they are not yet included in the graph to be constructed. Assume first $\mathrm{Re}(T) \neq \emptyset$ and $\mathrm{Em}(T) \neq \emptyset$. Apply operation 3 to t to create set A_0 that has exactly one receiver r and one emitter e. Choose r to be an arbitrary member of $\mathrm{Re}(T)$ to r and choose e to be an arbitrary member of $\mathrm{Em}(T)$ that is a descendant of r. Apply operation 2 iteratively to construct a path corresponding to the path from r to e in T. Now all vertices of T that belong to this path are marked.

Form a set A_2 of such receiver-emitter pairs in T that there is a directed path from the receiver to the emitter. While there are unprocessed pairs in A_2, do the following operations: If both the receiver and the emitter are unmarked, apply operation 8 to create the receiver-emitter pair and apply operation 2 to create the directed path between them. If the receiver is unmarked and the emitter is marked, apply operation 6 to connect the receiver to a vertex that is on the receiver-emitter path and is an ancestor of all marked vertices on this path. Then apply operation 2 iteratively to create all vertices of the receiver-emitter path. If the receiver is marked and the emitter is unmarked, apply operation 7 to connect the emitter to a vertex that is on the receiver-emitter path and is a descendant of all marked vertices on this path. Then apply operation 2 iteratively to create all vertices of the receiver-emitter path. If both the receiver and the emitter are marked, apply first operation 1 and then iteratively operation 2 to create the directed path between them.

Next process all vertices of T that have not been marked yet. Apply operations 4 and 5 to connect them to a vertex that is their child or parent. Repeat this until there are no vertices left. Finally, process all edges of T and use operation 1 to add the missing edges.

Next assume $\mathrm{Re}(T) = \emptyset$ and $\mathrm{Em}(T) \neq \emptyset$. Apply operation 9 to add all receivers. Then process all vertices of T that have not been marked yet similar way as above. Finally process all edges of T and use operation 1 to add the missing edges. The case $\mathrm{Re}(T) = \emptyset$ and $\mathrm{Em}(T) \neq \emptyset$ proceed analogously. ■

Theorem 16 (Transit cluster decomposition) *Let T be a transit cluster in a DAG $\mathcal{G} = (V, E)$. If T is not a transit component, then there exists a transit cluster S and a transit component R such that $T = S \cup R$ and $S \cap R = \emptyset$.*

Proof Because T is not a transit component, there exists a set $R \subset T$ such that $G[R]$ is connected and such that R is not connected to $S = T \setminus R$ in $G[T]$. Note that such a set necessarily exists, because at least one vertex t in T is not connected to $T \setminus \{t\}$ in $G[T]$ due to T not being a transit component. Next, we show that S and R are transit clusters. If T has receivers, then the receivers of S and R must have the same parents as the receivers of T because S and R are disconnected in $G[T]$, and because T is a transit cluster, thus satisfying condition 1. Analogously, if T has emitters then the children of the emitters of S and R must be the same, satisfying condition 2. Conditions 3 through 5 are satisfied for S and R because any path from an emitter to a receiver in T exists either entirely in S or R because S or R are disconnected in $G[T]$. Thus S and R are disjoint transit clusters such that $T = S \cup R$ and R is a transit component because it is connected in $G[R]$. ■

Lemma 17 FindTrComp *always terminates for valid inputs $\mathcal{G}$ and R.*

Proof The sets $\mathcal{V}_{\mathrm{Ch}}$ and $\mathcal{V}_{\mathrm{Pa}}$ are finite, and for any set A constructed on line 12, there is only a finite number of possible components $\mathcal{C}(\mathcal{G}[A])$ in the innermost for-loop. Thus, there is finite number of iterations in total across all for-loops, and all other operations are well-defined and nonrecursive. ■

Theorem 18 (Soundness of FindTrComp) *Let $\mathcal{G} = (V, E)$ be a DAG, $R \subseteq V$ and $\mathcal{A} =$ FindTrComp$(\mathcal{G}, R)$, then $\mathcal{A} \subseteq \mathcal{T}_{\mathcal{G}|R}$.*

Proof We show that if FindTrComp$(\mathcal{G}, R)$ reaches line 20, then the set being added to $\mathcal{A}$ is a transit component in $\mathcal{G}$. Because $\mathcal{T}_{\mathcal{G}|R} \subseteq \mathcal{T}_{\mathcal{G}}$ for all $R \subset V$, we can assume that $R = V$. Let V_i, V_j be a pair defined on lines 5 and 6 such that line 20 is triggered in the same iteration. Let Z and W be defined as dictated by lines 7 and 8, respectively. The condition on line 9 must not have been fulfilled, which means that if Z is non-empty, all of its members have parents, and if W is non-empty, all of its members have children. Because the condition on line 11 is fulfilled, we know that at least one of the sets Z and W is non-empty.

We summarize the construction of A on line 12. The set contains all vertices connected to $Z \cup W$ when incoming edges of Z and outgoing edges of W have been removed in $\mathcal{G}$. The intuition is to construct a set A such that Z would be equal to $\mathrm{Re}(A)$ and W would be equal to $emi(A)$. The construction together with lines 7 and 8 ensures that there will be a path from any member of Z to some member of W and vice versa, which is required to satisfy conditions 4 and 5 of Definition 4. The line 12 directly enforces condition 3. However, this construction alone does not guarantee that Z and W will be the set of receivers and emitters of A, respectively, because Z might not have the same parents outside of A, or W might not have the same children outside of A. It might also be the case that A is not connected in $\mathcal{G}[A]$.

Next, we break A into its components in $\mathcal{G}[A]$ and iterate over them on line 14. Conditions 3, 4 and 5 remain valid for each component. Because line 20 is reached, there must be at least one component A_k for which line 19 evaluates to true. This means that for such a set A_k, the sets Z_k and W_k have the same set of parents and children outside of A_k as any of their members, respectively. This means that the remaining conditions 1 and 2 of Definition 4 are satisfied by A_k, making A_k a transit component. ■

Theorem 19 (Completeness of FindTrComp) *Let $\mathcal{G} = (V, E)$ be a DAG, $R \subseteq V$, and $\mathcal{A} =$ FindTrComp$(\mathcal{G}, R)$ then $\mathcal{T}_{\mathcal{G}|R} \subseteq \mathcal{A}$.*

Proof We show that if $T \subset V$ is a transit component in $\mathcal{G}$, then it will be a member of the set $\mathcal{A}$ returned by FindTrComp. Because $\mathcal{T}_{\mathcal{G}|R} \subseteq \mathcal{T}_{\mathcal{G}}$ for all $R \subset V$, we can assume that $R = V$. Definition 4 implies that there exists a pair of vertices $v_i, v_j \in V$ such that $\mathrm{Re}(T) \subseteq \mathrm{Ch}^*(v_i)$ and $\mathrm{Em}(T) \subseteq \mathrm{Pa}^*(v_j)$ by conditions 1 and 2. Denote $V_i = \mathrm{Ch}^*(v_i)$ and $V_j = \mathrm{Pa}^*(v_j)$ with respect to the definitions on lines 5 and 6, respectively. Conditions 4 and 5 further imply that $\mathrm{Re}(T) \subseteq \mathrm{An}(\mathrm{Em}(T))$ and $\mathrm{Em}(T) \subseteq \mathrm{De}(\mathrm{Re}(T))$. Therefore $\mathrm{Re}(T) \subseteq V_i \cap \mathrm{An}(V_j)$ and $\mathrm{Em}(T) \subseteq V_j \cap \mathrm{De}(V_i)$. At this point, we make an important choice; if there exist multiple v_i, v_j pairs that satisfy these conditions, we choose one that minimizes the corresponding intersections $V_i \cap \mathrm{An}(V_j)$ and $V_j \cap \mathrm{De}(V_i)$. More precisely, we assume that for our choice v_i, v_j there does not exist v_i', v_j' such that $V_i' \cap \mathrm{An}(V_j') \subset V_i \cap \mathrm{An}(V_j)$ and $V_j' \cap \mathrm{De}(V_i') \subset V_j \cap \mathrm{De}(V_i)$. We call this the minimal representative choice in the context of this proof and illustrate the meaning of this choice with an example.

It is easy to verify that $B = \{r_1, e_1\}$ is a transit cluster in the graph of Figure 3(a). Suppose that we had chosen $v_i = x$ and $v_j = y$ resulting in $V_i = \mathrm{Ch}^*(x) = \{r_1, r_2, r_3\}$ and $V_j = \mathrm{Pa}^*(y) = \{e_1, e_2\}$. This is a valid choice because $\mathrm{Re}(B) \subseteq V_i$, $\mathrm{Em}(B) \subseteq V_j$, and for the intersections we have that $V_i \cap \mathrm{An}(V_j) = V_i$, $V_j \cap \mathrm{De}(V_i) = V_j$. However, x, y is not the pair that minimizes these intersections. Choosing instead $V_i' = \mathrm{Ch}^*(r_2) = \{r_1\}$ and $V_j' = \mathrm{Pa}^*(e_2) = \{e_1\}$ we have that $\mathrm{Re}(B) = V_i'$, $\mathrm{Em}(B) = V_j$ and for the intersections it holds $V_i' \cap \mathrm{An}(V_j') = V_i'$, $V_j' \cap \mathrm{De}(V_i') = V_j'$. Now $V_i' \subset V_i$ and $V_j' \subset V_j$ which shows that our initial choice x, y was not minimal, and r_2, e_2 is actually the minimizing pair.

Let $Z = V_i \cap \mathrm{An}(V_j)$ and $W = V_j \cap \mathrm{De}(V_i)$ as defined on lines 7 and 8, respectively. Because T is a transit component, it must have either receivers or emitters, which by definition have parents and children outside of T. This means that at least one of the sets Z and W is non-empty and Z has parents in $\mathcal{G}$ or W has children in $\mathcal{G}$. Thus, the for-loop does not continue on line 9, and we move on to line 11, which is satisfied for the same reason.

Next, we construct the set A on line 12. Importantly, we must show that $T \subseteq A$. Suppose instead that there exists $t \in T$ such that $t \notin A$ and t is not a receiver or emitter of T. Because T is a transit component, then t must be connected to $\mathrm{Re}(T) \cup \mathrm{Em}(T)$ when incoming edges of $\mathrm{Re}(T)$ and outgoing edges of $\mathrm{Em}(T)$ have been removed. Due to the construction of A, this leaves the only option that t is connected to $\mathrm{Re}(T) \cup \mathrm{Em}(T)$ only via paths that intersect $Z \setminus \mathrm{Re}(T)$ or $W \setminus \mathrm{Em}(T)$ and is no longer connected to $\mathrm{Re}(T) \cup \mathrm{Em}(T)$ when incoming edges of Z and outgoing edges of W are removed. Let Z' and W' denote those subsets of Z and W that only contain vertices that intersect such paths, respectively. This means that it must also be the case that $Z' \subset T$ and $W' \subset T$ because T is connected in $\mathcal{G}[T]$ and thus entire connecting path is in T. Suppose that the path has an incoming edge to $Z' \setminus \mathrm{Re}(T)$ and let $t_z \in Z' \setminus \mathrm{Re}(T)$ be a vertex on this path. Because $t_z \in T$ also, we have a contradiction, because t_z is not a receiver of T but Z' is not empty which means that t_z must have parents that are not members of T. The case for the path intersecting $W' \setminus \mathrm{Re}(T)$ is analogous. Thus we affirm that $T \subseteq A$.

Next, we must show that T is a component of $\mathcal{G}[A]$. Because $T \subseteq A$, there must be a component A_k of $\mathcal{G}[A]$ such that $T \subseteq A_k$. Let Z_k and W_k be defined according to lines 15 and 16, and suppose instead that there exists $a \in A_k \setminus T$ such that a is connected to T in $\mathcal{G}[A_k]$. Let a' be a vertex

in $A_k \setminus T$ on the path from a to T in $\mathcal{G}[A_k]$ such that it is either parent or a child of T. As T and A_k are both connected, it follows from the construction of A that if a' is a parent of T, then $a' \in Z_k \setminus \mathrm{Re}(T)$ or if a' is a child of T, then $a' \in W_k \setminus \mathrm{Em}(T)$. Suppose that $a' \in Z_k \setminus \mathrm{Re}(T)$. Now, because a' is a parent of T, it must a parent of all of its receivers. Furthermore, $\mathrm{Ch}_{\mathcal{G}[A]}(a') \subset Z_k$, i.e., a' necessarily has at least one fewer child than the representative v_i (mainly, a' itself). Then $\mathrm{Ch}_{\mathcal{G}[A]}(a') \cap \mathrm{An}_{\mathcal{G}[A]}(V_j) \subset V_i \cap \mathrm{An}_{\mathcal{G}[A]}(V_j)$, which contradicts the minimal representative choice. The case for $a' \in W_k \setminus \mathrm{Em}(T)$ is analogous. Hence, T is a component of $\mathcal{G}[A]$.

We can now deduce that $Z_k \cap T = \mathrm{Re}_{\mathcal{G}}(T)$ and $W_k \cap T = \mathrm{Em}_{\mathcal{G}}(T)$. If there existed $z \in Z_k \cap T$ that is not a receiver of T, we would have a contradiction, because T is a transit component, and $z_k \in \mathrm{Ch}^*_{\mathcal{G}}(v_i)$ meaning that T would be connected to $\mathrm{Pa}^*(T)$ via a vertex that is not its receiver. The case for W_k is once again analogous. The sets Z_{Pa} and W_{Ch} constructed on lines 17 and 18 are simply the common parents of $\mathrm{Re}_{\mathcal{G}}(T)$ and common children $\mathrm{Em}_{\mathcal{G}}(T)$, and because T is a transit component, the check of line 19 evaluates to true as all receivers have the same parents and all emitters have the same children in $\mathcal{G}$. Finally, FindTrComp adds T to the set $\mathcal{A}$ on line 20. ■

Theorem 20 FindTrComp *outputs all restricted transit components of a DAG $G = (V, E)$ with respect to $R \subseteq V$ in $O(|V|^4 + |V|^3|E|)$ time.*

Proof Let $n = |V|$ and $m = |E|$. Any restrictions on the vertices that are allowed to be members of transit clusters will only lead to a decrease in runtime, so we assume that $R = V$. Because the algorithm repeatedly accesses sets of parents, children, ancestors, and descendants of the vertex sets in the input graph $\mathcal{G}$, we assume that these are derived as a preprocessing step, which evaluates each vertex and edge once in the worst case, thus taking $O(n+m)$ time to construct the sets (for example, via a depth-first search). Thus any future access to these sets can be carried out in constant time.

The maximum number of unique parent and child sets in the collections $\mathcal{V}_{\mathrm{Pa}}$ and $\mathcal{V}_{\mathrm{Pa}}$ occurs when the parents and children of each vertex are unique. Thus there are at most n iterations in both of the two outermost for-loops on lines 5 and 6, leading to n^2 iterations in total. Each of the constructions and verifications on lines 7–11 can be evaluated in $O(n)$ time with help of the preprocessing step.

The construction of the candidate set A and its components $\mathcal{C}(\mathcal{G}[A]))$ on lines 12 and 14 takes $O(n+m)$ time in the worst case, when the entire graph has to be traversed (again, for example by a single depth-first search for both tasks simultaneously, when the search reaches a vertex v through an incoming edge to Z or an outgoing edge from W, it simply immediately returns to the previous vertex without discovering v.). There are at most n components of $\mathcal{G}[A]$, which leads to at most n iterations in the innermost for-loop on line 14. During lines 15–20, each operation can be carried out in $O(n)$ time, once again taking advantage of the preprocessing step.

Combining all of the previous observations gives us

$$O\left((n+m) + n^2(n + (n+m) + n^2)\right) = O(n^4 + n^3 m) = O(|V|^4 + |V|^3|E|).$$

■

Lemma 21 *Let T be a transit component of a DAG $\mathcal{G} = (V, E)$ and let $G' = (V', E')$ be the induced graph of the clustering with t representing the set T. If there does not exist a transit component S of $\mathcal{G}$ such that $T \cap S \neq \emptyset$ and $T \setminus S \neq \emptyset$, then $|\mathcal{T}_{\mathcal{G}}| = |\mathcal{T}_{\mathcal{G}'}| + |\mathcal{T}_{\mathcal{G}[T]}|$.*

Proof From the assumptions it follows that for any transit component S of $\mathcal{G}$, we have that either $T \subset S$ or $T \cap S = \emptyset$. Thus, by Theorem 8, there must be an equal number of transit components that do not contain T in $\mathcal{G}$ and those that do not contain t in $\mathcal{G}'$. Similarly by Theorem 9 there must be an equal number of transit components that contain T for $\mathcal{G}$ and those contain t for $\mathcal{G}'$. In this

case we apply the theorem to $S' = S \setminus T$ and T to make the previous observation. This means that the only difference in the number of transit components of $\mathcal{G}$ and $\mathcal{G}'$ is that in $\mathcal{G}$, the set T induces $|\mathcal{T}_{\mathcal{G}[T]}|$ transit components, whereas in $\mathcal{G}'$ the set $\{t\}$ induces a single transit component. However, the difference is offset by one because T is not a transit component in $\mathcal{G}[T]$. ■

Theorem 22 *Let $\mathcal{G} = (V, E)$ be a DAG. Then $|\mathcal{T}_{\mathcal{G}}| \leq \frac{|V|(|V|+1)}{2} - 1$.*

Proof Because FindTrComp is sound and complete, we take advantage of the algorithm, and consider pairs of sets V_i and V_j defined on lines 6 and 5, and the corresponding sets Z and W. We consider cases: 1) each pair (V_i, V_j) produces at most one transit component, 2) at least one pair (V_i, V_j) produces more than one transit component, and in addition 3) show that the equality $|\mathcal{T}_{\mathcal{G}}| = |V|(|V|+1)/2 - 1$ is attainable.

1. In the first case, the maximum number of pairs (V_i, V_j) such that $V_i \cap \mathrm{An}(V_j) \neq \emptyset$ and $V_j \cap \mathrm{De}(V_i) \neq \emptyset$ is clearly $\frac{|V|(|V|+1)}{2} - 1$, because $\mathcal{G}$ is a DAG. Assuming that each pair where at least one of the aforementioned intersections is nonempty produces a distinct transit component, the claim follows.

2. In the second case, we must consider the repercussions of multiple transit components being induced by the same pair (V_i, V_j). Also, we will associate each transit component T with a unique representative pair (V_i, V_j) as follows: if it occurs that a transit component T is induced by two distinct pairs (V_i, V_j) and (V_i', V_j'), we choose the pair that induces the smallest number of transit components. In the case that there are still multiple such pairs, the choice is arbitrary. Next, we must consider two separate scenarios: a) the sets Z and W corresponding to the pair (V_i, V_j) defined on lines 7 and 8 are both nonempty, and b) one of the sets Z and W is empty.

   a) Suppose that the pair (Z, W) induces k transit components, $A_1, \ldots, A_k$, defined according to line 4 which are the components of A, defined according to line 12. Let $n = |V|$, $n_i = |A_i|$ for each $i = 1, \ldots, k$, and $n_0 = |V| - |A|$. Suppose that for some A_j, there exists a transit component T of $\mathcal{G}$ such that $A_j \cap T \neq \emptyset$ and $A_j \setminus T \neq \emptyset$. Because T is a transit component and thus connected, it must contain at least one child of an emitter of A_j or a parent of a receiver of A_j. Without loss of generality, assume that a child c of an emitter of A_j is a member of T. However, this also makes c a child of an emitter e of at least one other transit component A_i, $i \neq j$. It cannot be the case that there would exist a child of an emitter of A_j such that it is not a child of an emitter of $A_i \neq A_j$ because we have assumed that the current pair (V_i, V_j) produces the smallest number of transit components. Now, if $e \notin T$, we have a contradiction, because $A_j \setminus T \neq \emptyset$, there must a path from A_j to a receiver of T, but there cannot be a path from e to the same receiver, because A_i and A_j are not connected in $\mathcal{G}[A]$. If $e \in T$, it follows that $A_i \subset T$ using the same argument with nodes along any path from a receiver of A_i to e that always exists due to the definition of a transit cluster. Now, because Z is not empty, there is a parent p of a receiver of A_j that is also a parent of a receiver of A_i. If $p \notin T$ we have a contradiction, because there is a receiver of T in A_i which has different parents than a receiver of T in A_j. If $p \in T$, this makes p an emitter of T and we have again a contradiction, because there would be a cycle in the induced graph of T, as there is a directed path from an emitter to a receiver in T. We conclude that no such transit component T can exist. The case where T contains instead a parent of a receiver of A_j is analogous.

   We can now apply Lemma 21 repeatedly to each set $A_1, \ldots, A_k$. Let $\mathcal{G}'$ denote the graph obtained after clustering each A_i, $k = 1, \ldots, k$. We have that

$$|\mathcal{T}_{\mathcal{G}}| = |\mathcal{T}_{\mathcal{G}'}| + \sum_{i=1}^{k} |\mathcal{T}_{\mathcal{G}[A_i]}|.$$

We now apply induction to the original claim in terms of the number of vertices. The base case clearly holds, and the induction assumption is that for all DAGs with $m < n$ vertices, the number of transit components is less or equal to $\frac{m(m+1)}{2} - 1$. For the induction step, applying the induction assumption to the above equation yields the following inequality:

$$\begin{aligned}|\mathcal{T}_\mathcal{G}| &\leq \frac{(n_0+k)(n_0+k+1)}{2} - 1 + \sum_{i=1}^{k}\left(\frac{n_i(n_i+1)}{2} - 1\right) \\ &= \frac{(n_0+k)(n_0+k+1)}{2} - k - 1 + \sum_{i=1}^{k}\frac{n_i(n_i+1)}{2}.\end{aligned}$$

Note that in the case that $n_i = 1$ for all $i = 1, \ldots, n$ we have $\sum_{i=1}^{k} n_i = k$, $n_0 + k = n$ and we obtain:

$$\begin{aligned}\frac{(n_0+k)(n_0+k+1)}{2} - k - 1 + \sum_{i=1}^{k}\frac{n_i(n_i+1)}{2} &= \frac{n(n+1)}{2} - k - 1 + \sum_{i=1}^{k}\frac{1\cdot 2}{2} \\ &= \frac{n(n+1)}{2} - k - 1 + k \\ &= \frac{n(n+1)}{2} - 1.\end{aligned}$$

In the case that at least one $n_i > 1$ we obtain instead (by multiplying both sides by 2 for convenience):

$$\begin{aligned}2|\mathcal{T}_\mathcal{G}| &\leq (n_0+k)(n_0+k+1) - 2k - 2 + \sum_{i=1}^{k} n_i(n_i+1) \\ &= n_0^2 + 2kn_0 + k^2 + n_0 - k - 2 + \sum_{i=1}^{k} n_i(n_i+1) \\ &= n_0^2 + 2kn_0 + k^2 + n_0 - k - 2 + \sum_{i=1}^{k} n_i^2 + \sum_{i=1}^{k} n_i \\ &= \sum_{i=0}^{k} n_i^2 + 2kn_0 + k^2 - k - 2 + \sum_{i=0}^{k} n_i \\ &< \sum_{i=0}^{k} n_i^2 + 2n_0\sum_{i=1}^{k} n_i + k(k-1) - 2 + \sum_{i=0}^{k} n_i \\ &< \sum_{i=0}^{k} n_i^2 + 2n_0\sum_{i=1}^{k} n_i + \sum_{i\neq j, 0<i,j\leq k} n_i n_j + \sum_{i=0}^{k} n_i - 2 \\ &= \sum_{i=0}^{k} n_i^2 + \sum_{i\neq j, 0\leq i,j\leq k} n_i n_j + \sum_{i=0}^{k} n_i - 2 \\ &= \left(\sum_{i=0}^{k} n_i\right)\left(\sum_{i=0}^{k} n_i + 1\right) - 2 \\ &= n(n+1) - 2.\end{aligned}$$

The first inequality in (2a) is due to $k < \sum_{i=1}^{k} n_i$ and the last inequality follows from the fact that there are $k(k-1)$ pairs i, j such that $i \neq j$ and $0 < i, j \leq k$. The second and third terms after the second inequality can be combined by changing the summation indices, and the second to last line is just a further refactoring of the terms. Thus we have that $|\mathcal{T}_{\mathcal{G}}| \leq \frac{n(n+1)}{2} - 1$ for all $1 \leq k \leq n$ and partitions $n_0, n_1, \ldots, n_k$ such that $\sum_{i=0}^{k} n_i = |V|$.

b) Without loss of generality, assume that $Z = \emptyset$. This means that each of the k components have the same set of receivers, mainly, the empty set. Now consider the possible number of candidate pairs (V_i, V_j), the amount of unique receivers has decreased by $k-1$, because we know the empty set to be the candidate receiver set of at least k pairs (V_i, V_j). Now, we have found k transit components, but the potential number of remaining (V_i, V_j) pairs has decreased by at least k as well.

Assuming that all remaining pairs (V_i, V_j) fall into the first or the previous case, the claim immediately follows. If not, we can repeat the above argument until this is the case, because the amount of new transit components found each time cannot exceed the amount that the number of undiscovered transit components decreases by. The case for $W = \emptyset$ is identical.

3. Let $\mathcal{G} = (V, E)$ consists of single directed path $v_1 \to \cdots \to v_n$. It is easy to see that any transit component of $\mathcal{G}$ also takes the form of a directed path $v_a \to \cdots \to v_b$, $v_a, v_b \in V$. There are $n-1$ possible ways to choose the length k of such a path (the path of full length n is not a transit component), and for each length k, there are $n-k+1$ ways to choose the starting vertex of the path. Thus we obtain the following:

$$\begin{aligned}
|\mathcal{T}_{\mathcal{G}}| &= \sum_{i=1}^{n-1}(n-i+1) \\
&= -1 + \sum_{i=1}^{n}(n-i+1) \\
&= n^2 + n - 1 - \sum_{i=1}^{n} i \\
&= n(n+1) - 1 - \frac{n(n+1)}{2} \\
&= \frac{n(n+1)}{2} - 1.
\end{aligned}$$

■

Lemma 23 FindTrClust *always terminates for valid inputs* $\mathcal{G}$ *and* $\mathcal{T}_{\mathcal{G}|R}$.

Proof Because there is a finite number of restricted transit components considered in the loop on line 4 of FindTrComp, it remains to show that ExpandClust always terminates for valid inputs T, $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{G}$. First, we note that in all following recursive calls to ExpandClust, the number of elements in the set $\mathcal{B}'$ decreases, and eventually the set becomes empty. Thus any single branch of the recursion will eventually reach the returning line 7. Further, because there are only a finite number of elements considered in the loop on line 3 of ExpandClust, there can only be a finite number of branches in the recursion. Thus ExpandClust always terminates. ■

Theorem 24 (Soundness of FindTrClust) *Let* $\mathcal{G} = (V, E)$ *be a DAG,* $R \subseteq V$*, and* $\mathcal{A} =$ FindTrClust$(\mathcal{G}, \mathcal{T}_{\mathcal{G}|R})$*, then* $\mathcal{A} \subseteq \Upsilon_{\mathcal{G}|R}$.

Proof New members to the output set $\mathcal{A}$ are added from the output of the subroutine ExpandClust. This subroutine performs recursive unions of transit clusters and transit components while ensuring that the conditions of Corollary 10 hold, which guarantee that the union is a transit cluster. Thus, only transit clusters are ever added to the set $\mathcal{A}$. ■

Theorem 25 (Completeness of FindTrClust) *Let $\mathcal{G} = (V, E)$ be a DAG, $R \subseteq V$, and $\mathcal{A} =$ FindTrClust$(\mathcal{G}, \mathcal{T}_{\mathcal{G}|R})$ then $\Upsilon_{\mathcal{G}|R} \subseteq \mathcal{A}$.*

Proof By Theorem 16, any transit cluster can be constructed iteratively from transit components. It is easy to see that Corollary 10 specifies the only valid union for a transit component and a transit cluster that are disjoint and not connected in the subgraph induced by their union. It follows that all possible distinct unions of transit components are considered in the recursive calls to ExpandClust launched by FindTrClust on line 6. Following this, if a union $S \cup T$ of transit components S and T is valid according to Corollary 10 on line 5 of ExpandClust, a new recursive call to ExpandClust is launched on line 6, which will now consider all possible unions of the transit cluster $S \cup T$, and the remaining transit components of $\mathcal{T}_{\mathcal{G}|R}$. Eventually, all possible unions are considered, and thus by Theorem 16, all transit clusters are included in the set $\mathcal{A}$ that is returned by FindTrClust on line 7. ■

Theorem 26 FindTrClust *outputs all restricted transit clusters of a DAG $\mathcal{G} = (V, E)$ with respect to $R \subseteq V$ with $O\left(|V|^5\right)$ polynomial delay and a $O\left(|V| + |E|\right)$ initialization delay.*

Proof Recalling Theorem 22, we note that the number of transit components grows as $O(|V|^2)$. We must consider four cases: 1) the initial preprocessing step that enables the application of Corollary 10 in $O(|V|)$ time, 2) finding the initial cluster, 3) finding any following cluster, and 4) termination after finding the last cluster. Let $k = \frac{|V|(|V|+1)}{2} - 1$ denote the maximum potential number of transit components of $\mathcal{G}$.

1. To apply Corollary 10 in linear time, we must first determine the parents and children of each vertex, thus requiring the traversal of the entire graph, which takes $O(|V| + |E|)$ time. We assume that the input $|\mathcal{T}_{\mathcal{G}|R}|$ is encoded in such a way, that the sets $\mathrm{Re}(T)$ and $\mathrm{Em}(T)$ can be obtained in constant time for any transit component (i.e., the transit components "know" their own sets of receivers and emitters). Once sets of parents and children for each vertex are obtained, it is easy to see that we can construct any parent set $\mathrm{Pa}^*(\mathrm{Re}(T))$ or child set $\mathrm{Ch}^*(\mathrm{Em}(T))$ in $O(|V|)$ time. Test for equality between sets is also an $O(|V|)$ operation.

2. Before the first valid union is found, only unions of transit components are considered. There are at most k possible options for the set T and $k-1$ options for the set S. In the worst case, none of the possible pairs produces a valid union on line 5, leading to $k(k-1)/2$ operations, because only distinct unions are considered. With the preprocessing step, Corollary 10 can be applied in $O(|V|)$ time (assuming a dynamic programming approach that keeps track of the emitters, receivers and their parents and children for valid unions $S \cup T$).

3. Suppose that the sets S and T have produced a valid transit cluster in a recursive call to ExpandClust. In the worst case, this cluster was found in the first iteration of every recursive call that preceded it, it is at the greatest depth of the recursion, i.e., the set $\mathcal{B}'$ would be empty in the next call, and the next cluster is not found until the recursion exists back to the very top level and reaches its final iteration. Because the number of iterations is reduced by one at each step of the recursion, the total number of remaining iterations is $(k-1) + (k-2) + ... + 1 = (k-1)k/2$. Applying Corollary 10 is the same as in the previous case.

4. This scenario is almost identical to the previous. Here, the worst case is different only in the aspect that instead of finding the next cluster, the algorithm terminates, leading to the same number of remaining iterations $(k-1)k/2$.

In summary, the initialization of case (1) can be carried out in $O(|V|+|E|)$ time. In cases (2), (3), and (4) there are at most $(k-1)k/2$ operations, each of which can be carried out in $O(|V|)$ time, which gives us

$$O(|V|(k-1)k/2) = O\left(|V|\left[\frac{|V|(|V|-1)}{2}-1\right]\frac{|V|(|V|-1)}{2}\right) = O(|V|^5).$$

■

Theorem 32 *Let $\mathcal{G}=(V\cup U,E)$ be a DAG of a causal model $\mathcal{M}$ and let A and B be disjoint subsets of V. Let $T=\{t_1,\ldots,t_n\}$ be a restricted transit cluster with respect to $(V\cup U)\setminus(A\cup B)$ in $\mathcal{G}$, and let $\mathcal{G}'=(V'\cup U',E')$ be the induced graph of the cluster with vertex $t'\in V'$ as the representative of T. If T is plain or congested, then $p(x_A\,|\,do(x_B))$ is identifiable from $p(x_V)$ in $\mathcal{G}$ precisely when it is identifiable from $p(x_{V'})$ in $\mathcal{G}'$.*

Proof

Plain transit cluster: We show that neither T in $\mathcal{G}$ nor t in $\mathcal{G}'$ can be a part of a hedge for $p(x_A\,|\,\mathrm{do}(x_B))$ and that clustering cannot affect hedges outside T. A hedge consists two c-components that fulfill certain conditions. Importantly, one of these c-components shares some vertices with set B. Condition $\mathrm{Pa}^*(\mathrm{Re}(T))\cup\mathrm{Em}(T)\subseteq V$ in Definition 30 guarantees that there is no variable in U with children both in T and $V\setminus T$. Consequently, there cannot be a c-component that contains members from both T and B. Similarly in $\mathcal{G}'$, there is no variable in U' with children both in $\{t\}$ and $V\setminus\{t\}$.

It remains to show that clustering does not change existing c-components. Assume that $v_i,v_j\in V\setminus T$ belong to same c-component and are therefore connected by a path specified in Definition 29. Condition $\mathrm{Pa}^*(\mathrm{Re}(T))\cup\mathrm{Em}(T)\subseteq V$ guarantees that such a path cannot contain members T and cannot be affected by clustering of T. We conclude that clustering a plain transit cluster does not change the identifiability properties of $p(x_A\,|\,\mathrm{do}(x_B))$.

Congested transit cluster: We need to show that there exists a hedge in $\mathcal{G}$ if and only if there exists a hedge in $\mathcal{G}'$. More precisely, we have to prove that c-components and their root sets fulfill the requirements of hedges in $\mathcal{G}$ and $\mathcal{G}'$

We first show T is a subset of the vertex set C of a c-component in $\mathcal{G}$ if and only if $\{t\}$ is a subset of the vertex set C' of a c-component in $\mathcal{G}'$. If $C=T$ or $C'=\{t\}$ we may use similar reasoning as in the first part of the proof to conclude that neither T in $\mathcal{G}$ nor t in $\mathcal{G}'$ can be a part of a hedge for $p(x_A\,|\,\mathrm{do}(x_B))$. Assume next $C\setminus T\neq\emptyset$. As $\mathrm{Em}(T)\subseteq V$ by Definition 31, there must exist $u_i\in U\setminus T$ such u_i has one child in T and one child in $V\setminus T$. Consequently, in $\mathcal{G}'$, there is an edge $u_i\to t$ and $t\in C'$. The vertices outside T are unaffected and $C\setminus T=C'\setminus\{t\}$. Similarly, if we assume $C'\setminus\{t\}\neq\emptyset$, there is an edge $u_i\to t$ and $t\in C'$. By Definition 31, all vertices of T belong to the same c-component whose vertex set must be C.

We have to also show that clustering cannot change the properties that a c-component is required to have in order to be a hedge. Consider a hedge in $\mathcal{G}$ and let Q denote its root set (sink). Suppose that there exists $t_i\in T\cap Q$. If there exists in $\mathcal{G}$ a directed path from B to t_i in the larger c-forest of the hedge then such a path will also exists in $\mathcal{G}'$ from B to t because t_i must be on a path from $\mathrm{Re}(T)$ to $\mathrm{Em}(T)$. Any path from B to $Q\setminus T$ remain unchanged by the clustering, thus a corresponding hedge can be constructed in G' with the root set $(Q\setminus T)\cup\{t\}$).

Consider next a hedge in $\mathcal{G}'$ that contains t. It follows that vertex t must have at least one parent in $\mathcal{G}$. If t is in the root set Q, we may construct a corresponding hedge in $\mathcal{G}$ by choosing a receiver as the root set. If t is not in the root set Q, the hedge must have a path $v_i \to t \to v_j$. This path can be replaced in $\mathcal{G}$ by a path $v_i \to r \to \ldots \to e \to v_j$, where $r \in \mathrm{Re}(T)$ and $e \in \mathrm{Em}(T)$. Combining the conclusions above, it follows that set T is a part of a hedge in $\mathcal{G}$ if and only if vertex t is a part of a hedge in $\mathcal{G}'$.

It remains to show that clustering cannot affect hedges outside T. Assume that $v_i, v_j \in V \setminus T$ belong to same c-component and are therefore connected by a path specified in Definition 29. Condition $\mathrm{Em}(T) \subseteq V$ together with the fact that members of $\mathrm{Re}(T)$ must be observed by Definition 27 guarantees that such a path cannot contain members T and cannot be affected by clustering of T. We conclude that clustering a congested transit cluster does not change the identifiability properties of $p(x_A \,|\, \mathrm{do}(x_B))$.

■

Theorem 33 *Let X_V be a vector of observed random variables, X_U a vector of unobserved random variables, and $\mathcal{G}' = (V' \cup U', E')$ the causal diagram of a causal model $\mathcal{M}'$ where vertex $t \in V'$ represents set $T = \{t_1, \ldots, t_n, u_1, \ldots, u_m\}$ in $\mathcal{G}'$, $t_1, \ldots, t_n \in V$, $u_1, \ldots, u_m \in U$, and $v \in V' \setminus \{t\}$ implies $v \in V$. Let $\mathcal{G} = (V \cup U, E)$ be a DAG obtained from $\mathcal{G}'$ by applying a series of peripheral extensions to vertex t such a way that $\mathcal{G}$ is a causal diagram. If T is a plain or congested transit cluster in $\mathcal{G}$, the following holds for disjoint subsets A and B of V' such that $T \cap (A \cup B) = \emptyset$.*

1. *Causal effect $p(x_A \,|\, do(x_B))$ is identifiable from $p(x_V)$ in $\mathcal{G}$ exactly when it is identifiable from $p(x_{V'})$ in $\mathcal{G}'$.*
2. *If $g(p(x_{V'}))$ is an identifying functional for $p(x_A \,|\, do(x_B))$ in $\mathcal{G}'$, it is also an identifying functional for $p(x_A \,|\, do(x_B))$ in $\mathcal{G}$.*

Proof First we note that peripheral extension is guaranteed to produce a causal diagram if all vertices $t_1, \ldots, t_n$ corresponding observed variables are added first and vertices $u_1, \ldots, u_n$ corresponding unobserved background variables are then added using only operations 1 and 4 with the restriction that vertices $u_1, \ldots, u_m$ cannot not have parents.

The first claim follows directly from Theorem 32.

For the second claim, we show the do-calculus derivation for obtaining $g(p(x_{V'}))$ in $\mathcal{G}'$ is valid in $\mathcal{G}$ when the relevant sets are replaced by clustering equivalent sets.

We show that the d-separation $V_1' \perp\!\!\!\perp V_2' \,|\, V_3'$ in $\mathcal{G}'$ implies $V_1 \perp\!\!\!\perp V_2 \,|\, V_3$ in $\mathcal{G}$ where V_1', V_2', V_3' are the clustering equivalent sets of $V_1, V_2, V_3 \subset V$, respectively. Consider a path from V_1' to V_2' in $\mathcal{G}'$ that is blocked on the condition of V_3'. Now there are four options that arise from the definition of d-separation: 1) The path does not contain t. 2) The path contains t but t does not block the path. 3) The path contains a chain or a fork where the middle vertex t belongs to V_3'. 4) The path contains a collider t and $\mathrm{De}_{\mathcal{G}'}(t) \cap V_3' = \emptyset$.

1) The same path exists also in $\mathcal{G}$ and is blocked. 2) The definition of a transit cluster guarantees that the corresponding path exists in $\mathcal{G}$. The path is blocked by the same vertex in both $\mathcal{G}$ and $\mathcal{G}'$. 3) In the case of a chain $v_1 \to t \to v_2$, vertex v_1 is a parent of a receiver and v_2 is a child of an emitter. In $\mathcal{G}$, we have a corresponding directed path $v_1 \to r \to \ldots \to e \to v_2$ where $r \in T$ is a receiver and $e \in T$ is an emitter. In the case of a fork $v_1 \leftarrow t \to v_2$, both v_1 and v_2 are children of an emitter. In $\mathcal{G}$, we have a corresponding fork $v_1 \leftarrow e \to v_2$ where $e \in T$ is an emitter. In both cases, the path is blocked because $T \subseteq V_3$. 4) Collider $v_1 \to t \leftarrow v_2$ implies that t is a receiver in $\mathcal{G}'$. In $\mathcal{G}$, we have a corresponding collider $v_1 \to r \leftarrow v_2$ where $r \in T$ is receiver. Assumption $t \notin V_3'$ implies $T \cap V_3 = \emptyset$. For the descendants it holds $\mathrm{De}_{\mathcal{G}'}(t) = De_{\mathcal{G}}(r) \setminus T$. We conclude that the path is blocked in $\mathcal{G}$.

The rules of do-calculus operate in graphs obtained from $\mathcal{G}'$ by removing some edges. There are three possibilities when each edge removed is considered: i) the edge does not involve t, ii) the edge (v,t) is an incoming edge of t, and iii) the edge (t,v) is an outgoing edge of t. In case i) the removed edge exists also in $\mathcal{G}$ and can be removed there as well. In case ii), all edges from v to $\mathrm{Re}(T)$ are removed in $\mathcal{G}$. This breaks all undirected paths that correspond a path containing edge (v,t) in $\mathcal{G}'$. In case iii), all edges from $\mathrm{Em}(T)$ to v are removed in $\mathcal{G}$. This breaks all undirected paths that correspond a path containing edge (t,v) in $\mathcal{G}'$. Together with the d-separation properties proved above, this guarantees that the do-calculus derivation applicable in $\mathcal{G}'$ is also applicable in $\mathcal{G}$ and $g(p(x'_V))$ is an identifying functional for $p(x_A \mid \mathrm{do}(x_B))$ in $\mathcal{G}$. ■

Appendix B. Hedges and Causal Effect Identifiability

We introduce c-trees, c-forests and hedges as defined by Shpitser and Pearl (2006) using our notation. We begin by defining maximal c-components.

Definition 34 (Maximal c-component) *Let $\mathcal{G} = (V \cup U, E)$ be a causal diagram. Then a subgraph $\mathcal{G}' = (V', E')$ of $\mathcal{G}$ is a maximal c-component if it is a c-component and if $\mathcal{G}^* \subseteq \mathcal{G}'$ for all c-components $\mathcal{G}^* = (V^* \cup U^*, E^*)$ such that $V' \cap V^* = \emptyset$.*

Next, we define c-trees which characterize direct effects.

Definition 35 (c-tree) *Let $\mathcal{G}$ be a causal diagram such that it has only one maximal c-component, and such that every observed node has most one child. If there exists a node y such that $\mathcal{G}[An(y)] = \mathcal{G}$ then $\mathcal{G}$ is a y-*rooted c-tree*.*

C-forest is the generalization of a c-tree where the root y is extended to a *root set*, i.e., the set of nodes $\{v \in V \mid \mathrm{De}^*(v)_\mathcal{G} = \emptyset\}$ for a causal diagram $\mathcal{G} = (V \cup U, E)$.

Definition 36 (c-forest) *Let $\mathcal{G} = (V \cup U, E)$ be a causal diagram and let Y be its root set. If $\mathcal{G}$ is c-component and every observed node has at most one child, then $\mathcal{G}$ is a Y-*rooted c-forest*.*

A special pair of c-forests can be used to characterize general causal effects of the form $p(x_A \mid \mathrm{do}(x_B))$.

Definition 37 (Hedge) *Let $\mathcal{G} = (V \cup U, E)$ be a causal diagram, and let $A, B \subset V$ be disjoint subsets. If there exists two R-rooted c-forests $\mathcal{F} = (V_\mathcal{F} \cup U_\mathcal{F}, E_\mathcal{F})$ and $\mathcal{F}' = (V_{\mathcal{F}'} \cup U_{\mathcal{F}'}, E_{\mathcal{F}'})$ such that $V_\mathcal{F} \cap B \neq \emptyset$, $V_{\mathcal{F}'} \cap B = \emptyset$, $\mathcal{F}' \subseteq \mathcal{F}$ and $R \subset An_{\mathcal{G}[\overline{B}]}(A)$, then $\mathcal{F}$ and $\mathcal{F}'$ form a* hedge *for $p(x_A \mid do(x_B))$ in $\mathcal{G}$.*

Hedges completely characterize the identifiability of causal effects from the joint distribution over the observed variables of the causal model.

Theorem 38 (Hedge criterion, Corollary 3 of (Shpitser and Pearl, 2006)) *$p(x_A \mid do(x_B))$ is identifiable from $p(x_V)$ in $\mathcal{G}$ if and only if there does not exist a hedge for $p(x_{A'} \mid do(x_{B'}))$ in $\mathcal{G}$, for any $A' \subseteq A$ and $B' \subseteq B$.*

References

T. Anand, A. Ribeiro, J. Tian, and E. Bareinboim. Effect identification in causal diagrams with clustered variables. Technical report, 2021. Columbia CausalAI Laboratory, Technical Report (R-77).

D. Entner and P. O. Hoyer. Estimating a causal order among groups of variables in linear models. In *International Conference on Artificial Neural Networks*, pages 84–91. Springer, 2012.

S. Greenland, J. Pearl, and J. Robins. Causal diagrams for epidemiologic research. *Epidemiology*, 10(1):37–48, 1999.

F. J. He, J. Li, and G. A. MacGregor. Effect of longer term modest salt reduction on blood pressure: Cochrane systematic review and meta-analysis of randomised trials. *BMJ*, 346:f1325, 2013.

J. Helske, S. Tikka, and J. Karvanen. Estimation of causal effects with small data in the presence of trapdoor variables. *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, 184 (3):1030–1051, 2021.

G. W. Imbens. Potential outcome and directed acyclic graph approaches to causality: Relevance for empirical practice in economics. *Journal of Economic Literature*, 58(4):1129–1179, 2020.

J. Karvanen, S. Tikka, and A. Hyttinen. Do-search: A tool for causal inference and study design with multiple data sources. *Epidemiology*, 32(1):111–119, 2020.

E. M. Kornaropoulos and I. G. Tollis. DAGView: An approach for visualizing large graphs. In W. Didimo and M. Patrignani, editors, *Graph Drawing*, pages 499–510, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

J. J. R. Lee and I. Shpitser. Identification methods with arbitrary interventional distributions as inputs. *arXiv preprint arXiv:2004.01157*, 2020.

S. Lee, J. Correa, and E. Bareinboim. General identifiability with arbitrary surrogate experiments. In *Proceedings of the 35th Conference on Uncertainty in Artificial Intelligence*, 2019.

A. Magrini, S. D. Blasi, and F. M. Stefanini. A conditional linear Gaussian network to assess the impact of several agronomic settings on the quality of Tuscan Sangiovese grapes. *Biometrical Letters*, 54(1):25–42, 2017.

F. D. Malliaros and M. Vazirgiannis. Clustering and community detection in directed networks: A survey. *Physics reports*, 533(4):95–142, 2013.

S. Nisimov, Y. Gurwicz, R. Y. Rohekar, and G. Novik. Improving efficiency and accuracy of causal discovery using a hierarchical wrapper. In *The 37th Conference on Uncertainty in Artificial Intelligence, Workshop on Tractable Probabilistic Modeling*, 2021.

P. Parviainen and S. Kaski. Learning structures of Bayesian networks for variable groups. *International Journal of Approximate Reasoning*, 88:110–127, 2017.

J. Pearl. Comment: graphical models, causality and intervention. *Statistical Science*, 8(3):266–269, 1993.

J. Pearl. Causal diagrams for empirical research. *Biometrika*, 82(4):669–688, 1995.

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

H. Purchase. Which aesthetic has the greatest effect on human understanding? In G. DiBattista, editor, *Graph Drawing*, pages 248–261, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2022. URL `https://www.R-project.org/`.

P. R. Rosenbaum and D. B. Rubin. The central role of the propensity score in observational studies for causal effects. *Biometrika*, 70(1):41–55, 1983.

D. Rubin. Estimating causal effects of treatments in randomized and nonrandomized studies. *Journal of Educational Psychology*, 66:688–701, 1974.

I. Shpitser and J. Pearl. Identification of joint interventional distributions in recursive semi-Markovian causal models. In *Proceedings of the 21st National Conference on Artificial Intelligence*, volume 2, pages 1219–1226. AAAI Press, 2006.

G. Skorstad. Clustered causal ordering. In *Proceedings of 4th International Workshop on Qualitative Physics*, pages 290–299, 1990.

P. Spirtes, C. Glymour, and R. Scheines. Constructing Bayesian networks models of gene expression networks from microarray data. In *Proceedings of the Atlantic Symposium on Computational Biology*, 2000.

P. Spirtes, C. Glymour, and R. Scheines. *Causation, Prediction, and Search*. MIT Press, 2nd edition, 2001.

P. W. G. Tennant, E. J. Murray, K. F. Arnold, L. Berrie, M. P. Fox, S. C. Gadd, W. J. Harrison, C. Keeble, L. R. Ranker, J. Textor, G. D. Tomova, M. S. Gilthorpe, and G. T. H. Ellison. Use of directed acyclic graphs (DAGs) to identify confounders in applied health research: review and recommendations. *International Journal of Epidemiology*, 50(2):620–632, 4 2021.

S. Tikka and J. Karvanen. Identifying causal effects with the R package causaleffect. *Journal of Statistical Software, Articles*, 76(12):1–30, 2017a.

S. Tikka and J. Karvanen. Simplifying probabilistic expressions in causal inference. *The Journal of Machine Learning Research*, 18(1):1203–1232, 2017b.

S. Tikka and J. Karvanen. Enhancing identification of causal effects by pruning. *The Journal of Machine Learning Research*, 18(194):1–23, 2018.

S. Tikka, A. Hyttinen, and J. Karvanen. Identifying causal effects via context-specific independence relations. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

S. Tikka, A. Hyttinen, and J. Karvanen. Causal effect identification from multiple incomplete data sources: A general search-based approach. *Journal of Statistical Software*, 99(5), 2021.

R. Tu, C. Zhang, P. Ackermann, K. Mohan, H. Kjellström, C. Glymour, and K. Zhang. Causal discovery in the presence of missing data. In *Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics*, 2019.

T. Verma. Graphical aspects of causal models. *Technical Report R-191, UCLA*, 1993.

K. Zhang, J. Zhang, B. Huang, B. Schölkopf, and C. Glymour. On the identifiability and estimation of functional causal models in the presence of outcome-dependent selection. In *Proceedings of the 32rd Conference on Uncertainty in Artificial Intelligence*, 2016.

K. Zhang, M. Gong, J. Ramsey, K. Batmanghelich, P. Spirtes, and C. Glymour. Causal discovery in the presence of measurement error: Identifiability conditions. In *UAI 2017 Workshop on Causality: Learning, Inference, and Decision-Making*, 2017.