Causal Inference (MGT-416)
Final Project

- You can work on this project in groups of 1 up to 3 students. You must mention the name of all the participants. All the students in a group will get the same grade.
- All of the programming implementations must be done using either Python or Matlab.
- Deadline: Tuesday, JUNE 13, 2023, 09:00 AM (No late submissions will be accepted.)
- The technical report cannot exceed the 5-page limit. Jupyter notebook reports do not count.
- Upload a single .zip file on Moodle, including the implementations, necessary output files, and the report as a pdf file.
- Grading policy: correct output on test benches: $65-70\%$, technical report (pdf): 25%, efficiency/creativity in implementations and/or report: $5-10\%$. NOTE: the grade for the technical report IS CONTINGENT ON the correctness of your code.

This project touches on two crucial topics in causal inference. A vast majority of the results in the causal inference literature hinges on the knowledge of the causal mechanisms, often in the form of a causal DAG. Therefore, the first part of the project focuses on recovering the causal graphical structure through data, a.k.a. causal discovery. In the second part of the project, the problem of causal effect identification is considered when the causal structure is known. The necessary material is provided for each part of the project, and then the tasks are given.

1 A Causal Discovery Algorithm

The approaches to causal discovery can be mainly categorized into score-based and constraint-based methods. While the former group focuses on maximizing a specific score function over the space of graphs, the latter relies on statistical tests, namely *conditional independence* (CI) tests, to recover the structure. For this part of the project, you will implement one of the most well-known constraint-based causal discovery methods, the Grow-Shrink algorithm [7]. It is recommended for the interested to refer to the original article for comprehensive details regarding this algorithm. Throughout this project, we assume Markov property and faithfulness hold. That is, d-separation is equivalent to conditional independence. We use $\perp\!\!\!\perp$ and $\perp$ to denote conditional independence and d-separation, respectively. We also make the assumption that there are no hidden (latent) variables.

As a notational convention, we denote variables and sets of variables by lowercase and uppercase letters, respectively. Let V be the set of variables. We represent each variable by a vertex (node) in

the causal graph, denoted by $\mathcal{G}$. Given a causal DAG $\mathcal{G}$, the undirected graph resulting from dropping the edge orientations of $\mathcal{G}$ is called the *skeleton* of $\mathcal{G}$. We will therefore utilize the terms variable and vertex interchangeably. Grow-Shrink (referred to as GS henceforth) is a two-step algorithm. The first step comprises recovering the Markov boundaries of every variable in V. The second step uses the Markov boundary information to recover the (partially oriented) causal graph up to Markov equivalence. At this stage, Meek rules can be applied to maximally orient the edges.

1.1 Step 1: recovering the Markov boundary

Recall the definition of Markov boundary from your midterm exam:

Definition 1 (Markov boundary)**.** *Let V be the set of all variables. For a variable x, a set $B \subseteq V$ is said to be a Markov blanket of x if*

$$\forall y \in V \setminus B, \quad x \perp\!\!\!\perp y \mid B. \tag{1}$$

Markov boundary of x, denoted by $Mb(x)$, is the smallest Markov blanket of x. That is, the smallest subset of variables that satisfy Eq. (1)*.*

Suppose we are interested in recovering the Markov boundary of a variable x. GS achieves this in two phases:

- Grow phase: begin with $M \leftarrow \emptyset$. While there exists $y \in V \setminus (M \cup \{x\})$ such that $x \not\perp\!\!\!\perp y \mid M$, add y to M, i.e., $M \leftarrow M \cup \{y\}$.
- Shrink phase: while there exists $y \in M$ such that $x \perp\!\!\!\perp y \mid M \setminus \{y\}$, remove y from M, i.e., $M \leftarrow M \setminus \{y\}$.

The output M of the two-phase algorithm described above is the Markov boundary of x, i.e., $Mb(x) = M$.

1.2 Step 2: recovering the Markov equivalence class

The recovery of the causal structure is greatly facilitated by the knowledge of Markov boundaries. Note that if we represent each variable by a vertex and connect each vertex to every vertex in its Markov boundary by an edge, the resulting undirected graph is exactly the moralized graph. The objective of this step is to recover the skeleton and v-structures of the causal DAG, starting from the moralized graph (or equivalently, the Markov boundary of every vertex).

For any vertex x and any $y \in Mb(x)$, GS utilizes the following simple rule to determine whether x and y are neighbors in the causal graph:

- Determine y to be a direct neighbor of x if $x \not\perp\!\!\!\perp y \mid S$ for every $S \subseteq T$, where T is the smaller (in cardinality) of the two sets $Mb(x) \setminus \{y\}$ and $Mb(y) \setminus \{x\}$.

Given the Markov boundaries already recovered in the first step, this criterion fully recovers the skeleton of the causal graph. It is noteworthy that to recover the causal graph up to Markov equivalence class, which is the objective of this project, you will need to extract the v-structures as well. As you have seen in the lectures, v-structures can be identified by the following rule:

- Let (a, b, c) be a triplet of vertices forming the skeleton $a - b - c$, i.e., a and b are neighbors, b and c are neighbors, and there is no edge between a and c. Let S be an arbitrary separating set for a and c, i.e., $a \perp\!\!\!\perp c \mid S$. The triplet $a - b - c$ is a v-structure if and only if $b \notin S$.

1.3 Step 3: Applying Meek orientation rules

Meek presented a *complete* set of orientation rules [8]. These rules are depicted in Figure 1. Starting from a graph where only v-structures are oriented, these rules help us maximally orient the edges. The procedure would be to apply these rules iteratively until no further edges can be oriented by any of these rules:

- While a new edge can be oriented, apply rules R1 through R4.

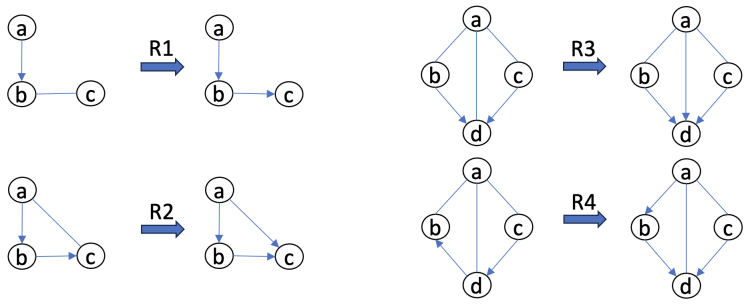


Figure 1: Meek orientation rules.

Meek orientation rules described (refer to Figure 1):

R1. If $a \to b$ and $b - c$ while a and c are not adjacent, then orient $b \to c$.

R2. If $a \to b$, $b \to c$ and $a - c$, then orient $a \to c$.

R3. If $a - b$, $a - c$, $a - d$, $b \to d$ and $c \to d$ while b and c are not adjacent, then orient $a \to d$.

R4. If $a - b$, $a - c$, $a - d$, $d \to b$ and $c \to d$ while b and c are not adjacent, then orient $a \to b$.

1.4 On conditional independence testing

Throughout, n and p denote the number of samples and the number of variables, respectively. Conditional independence (CI) tests are, in essence, statistical hypothesis tests where the null hypothesis is conditional independence. For this project, we are using a specific test which is designed for multivariate Gaussian variables. This test is driven by the rationale that multivariate Gaussian random variables are independent if and only if their correlation coefficient is zero. Due to sample noise, the correlation of two independent variables might be a very small (but nonzero) number. In practice, we threshold the correlation coefficient (after applying Fisher's Z transform [3]) to decide for conditional independence. You are provided with a function called *ci_test*(), which performs the conditional independence test of $x \perp\!\!\!\perp y \mid Z$ for you. This function receives as input the n by d matrix of data, the indices of the variables x, y and the set Z, and a constant α which controls the thresholding. That is, α determines the threshold of correlation at which we no longer accept the null hypothesis (of independence).

1.5 Implementation guidelines

Your first task is to tune the threshold parameter α of the CI tests. To this end, you are provided a dataset $\mathbf{D}_1$ generated from a structural causal model corresponding to the graph of Figure 2a.

1. Implement a module (function) that receives as input a DAG $\mathcal{G}$ (a networkx.DiGraph object in case you are using python), two vertices x and y, and a set of vertices Z, and outputs as a binary decision whether x and y are d-separated given Z in $\mathcal{G}$.

2. Compare the output of various CI tests using the dataset $\mathbf{D}_1$ and the corresponding d-separations in the graph of Figure 2a.

3. Choose a proper parameter α such that the CI test results match their corresponding d-separation as well as possible.

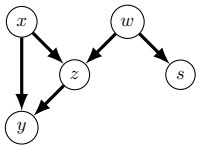
After you have tuned the parameter α, you will use that value of the parameter for the rest of the tasks. You are free to choose how you tune this parameter (as long as your approach is properly justified.) For instance, cross-validation might be an idea.

In the sequel, you are asked to implement the GS method:

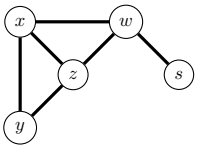
4. Implement a module (function) based on the GS algorithm described above that receives a dataset, and outputs an undirected graph (a networkx.Graph object in python) where each variable is connected to every node in its Markov boundary. That is, the output of your function must be the moralized graph. Use the provided *ci_test* function as a subroutine. As an example, if the data is according to the DAG of Figure 2a, then your function must output the undirected graph of Figure 2b.

5. Implement the Step 2 of GS algorithm. That is, a function that receives as input the moralized graph and the dataset and returns a partially oriented graph (networkx.DiGraph object) where the v-structures are oriented, and the rest of the edges of the skeleton are undirected (see Figure 2c for instance.)

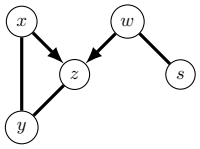
6. Implement the Meek orientation rules, and put all these together as a function that receives data as input and outputs a maximally oriented graph. As an example, Figure 2d illustrates the graph of Figure 2c after applying Meek rules.



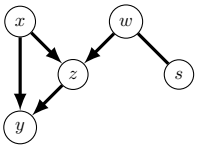
(a) Ground truth DAG $\mathcal{G}$



(b) Step 1: Moralized graph



(c) Step 2: Skeleton and v-structures



(d) Step 3: Meek orientation rules

Figure 2: An example of applying GS causal discovery algorithm.

What to include in your report:

- Describe how you tuned the parameter α, and justify your method of tuning.
- Markov boundary is (theoretically) symmetric. That is, if $x \in Mb(y)$ then $y \in Mb(x)$. Might it happen in practice that the Markov boundaries are not symmetric? Explain how you would draw the moralized graph in such cases.
- CI test results are used to orient v-structures. However, due to sample noise, the result of certain CI tests might be wrong. In such cases, the v-structures might be in conflict with each other. Consider, for instance the skeleton $a - b - c - d$, and suppose the CI test results indicate $a \perp\!\!\!\perp c \mid \{d\}$ and $b \perp\!\!\!\perp d$. The former independence implies that $a \rightarrow b \leftarrow c$ is a v-structure, imposing the direction $b \leftarrow c$. However, the latter independence implies $b \rightarrow c \leftarrow d$ is a v-structure, imposing the direction $b \rightarrow c$. In practice, such conflicts must be resolved. What is your idea (what have you done) for resolving such conflicts? Note that you may refer to data for further evidence.
- If the output of a constraint-based causal discovery method depends on the order in which the CI tests are performed, we say that algorithm is order-dependent. Do you think GS is order-dependent? For instance, if the columns of data were shuffled, would you get exactly the same output? If yes, justify. If no, what can be done to make it order-independent?
- Explain your ideas to implement Meek's orientation rules efficiently. Also, applying Meek rules can result in conflicting directions due to sample noise, exactly as in the example of conflicting v-structures above. What have you done to resolve such conflicts in your code?
- Include the output of steps 1, 2 and 3 of GS (tasks 4,5, and 6, respectively) separately for both datasets $\mathbf{D}_1$ and $\mathbf{D}_2$.
- Among GS and PC, which one would you choose to rely on as a causal discovery method? What is your criterion?

2 Experimental Design for Causal Effect Identification

The three rules of Pearl's do calculus equipped with basic probability manipulations are *complete* for causal effect identification [6, 9]. This is to say, given a causal graph $\mathcal{G}$ and a causal query of interest q, the query q is *identifiable* w.r.t. $\mathcal{G}$ if and only if q can be expressed as a functional of the observational distribution through a series of applications of do calculus rules. However, in many cases, the query q is not *identifiable*; that is, it cannot be expressed as a functional of the observational distribution. The simplest example of a non-identifiable query is $P(y|do(x))$ w.r.t. the graph of Figure 3a, where u is a hidden variable highlighted with dashed lines.

(a) latent DAG (b) ADMG representation

Figure 3: Two representations of a causal graph; (a) shows a latent DAG, and (b) is the corresponding ADMG. $P(y|do(x))$ is known to be not identifiable in this graph.

Throughout this project, we utilize the *acyclic directed mixed graph* (ADMG) representation of the causal graph. In this representation, the existence of a hidden confounder between two variables is shown by a bidirected edge between those variables. As an example, the hidden confounder u in the latent DAG model of Figure 3a is represented as a bidirected edge between x and y in Figure3b, highlighted in blue. Every pair (a, b) of vertices of an ADMG may have a directed edge, a bidirected edge, or both between them. Note that only observable variables are present in the ADMG representation.

Shpitser and Pearl [9] derived a *complete* graphical criterion to decide whether a given causal query q is identifiable w.r.t. a graph. They showed that a query q is not identifiable if and only if a specific graphical structure, called a *hedge* exists in the graph w.r.t. that query. We encourage the interested reader again to study the original paper pertaining to the *hedge criterion* [9], briefly discussed in Section 2.1 below.

When a given query q is said to be not identifiable, it is meant that q is not identifiable from the observational distribution. However, performing further interventions can help identify q, using the newly gathered interventional data. As a simple example again, if we intervened upon x in the graph of Figure 3, we would obtain the distribution $P(y|do(x))$, directly identifying the desired query. These interventions, when thought about from a graphical perspective, correspond to eliminating the hedge structures that prevent identification. We shall clarify this point in the sequel. It is noteworthy that different interventions in the system may impose heterogeneous costs. Our objective in this project is to identify the query of interest by conducting the intervention with the minimum cost. This problem was the focus of [1], where hardness results were shown. Before describing the hedge criterion, we set a formal framework for ADMGs.

Definition 2 (ADMG)**.** *An ADMG is a tuple $\mathcal{G} = (V, E^d, E^b)$, where V denotes the set of vertices, $E^d \subseteq V \times V$ is the set of directed edges (ordered pairs of vertices), and $E^b \subseteq \binom{V}{2}$ denotes the set of bidirected edges (unordered pair of vertices). The directed edges, E^d, do not form any cycles.*

For instance, in the ADMG of Figure 3b, E^d and E^b are highlighted in solid black and dashed blue, respectively. Given an ADMG $\mathcal{G} = (V, E^d, E^b)$ and a set of vertices $X \subseteq V$, the (vertex-

induced) subgraph of $\mathcal{G}$ over X, denoted by $\mathcal{G}_{[X]}$, is defined as

$$\mathcal{G}_{[X]} = (X, E^d_{[X]}, E^b_{[X]}), \quad \text{where} \quad E^d_{[X]} = E^d \cap (X \times X), \quad E^b_{[X]} = E^b \cap \binom{X}{2}.$$

Definition 3 (c-component)**.** *Given an ADMG $\mathcal{G}$, we say a subgraph $\mathcal{G}_{[X]}$ is a confounded component (c-component in short) if for every pair $(x_1, x_2) \in X$, there is a path from x_1 to x_2 that goes only through bidirected edges in $\mathcal{G}_{[X]}$. In other words, the bidirected edges of $\mathcal{G}_{[X]}$ form a connected graph.*

For instance, in the ADMG of Figure 4, $\mathcal{G}_{[\{v_2,s_2\}]}$ is a c-component, whereas $\mathcal{G}_{[\{v_2,s_1\}]}$ is not one. In this Figure, $\mathcal{G}_{[\{v_2,s_2\}]}$ and $\mathcal{G}_{[\{v_3,s_1,t,v_1\}]}$ are two maximal c-components. It is straightforward to see that a graph is always partitioned into its maximal c-components.

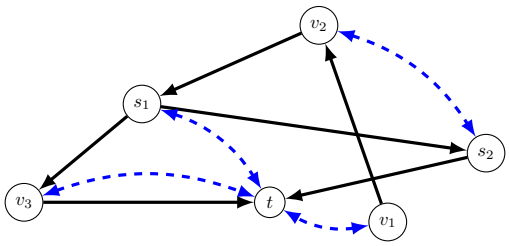


Figure 4: Example ADMG.

Let $\mathcal{G}$ be an ADMG with vertices V. Note that any arbitrary causal query $P(Y|do(X))$ for any disjoint sets of variables X and Y can be written as

$$P(Y|do(X)) = \sum_{S \setminus Y} P(S|do(V \setminus S)),$$

where S is the set of ancestors of Y after deleting X from the graph. Therefore, throughout this project, we only consider queries of the form $P(S|do(V \setminus S))$. For ease of notation, we will denote $P(S|do(V \setminus S))$ by $Q[S]$, following [10].

2.1 Hedge criterion

We describe a simplified version of the hedge criterion, which is provably[1] equivalent to the original definition in [9].

Definition 4 (hedge)**.** *Let $\mathcal{G}$ and S be an ADMG and a subset of its vertices, respectively. Suppose $\mathcal{G}_{[S]}$ is a c-component. A subset F is a hedge formed for $Q[S]$ in $\mathcal{G}$ if $S \subsetneq F$, F is the set of ancestors of S in $\mathcal{G}_{[F]}$, and $\mathcal{G}_{[F]}$ is a c-component.*

As an example, consider $S = \{t, v_3\}$ and $F = \{t, v_3, s_1\}$ in Figure 4. By definition, $\mathcal{G}_{[S]}$ and $\mathcal{G}_{[F]}$ are both c-components. Moreover, every vertex in F is an ancestor of at least one vertex in S in the graph $\mathcal{G}_{[F]}$. Therefore, F is a hedge formed for $Q[S]$. Hedges are the structures preventing a query from identifiability. This is clarified through the following theorem.

[1] See Appendix A of [1] for the proof.

Theorem 1 ([9]). *Let $\mathcal{G}$ and S be an ADMG and a subset of its vertices, respectively. Let $S_1, \ldots, S_k$ be the partitioning of $\mathcal{G}_{[S]}$ to its maximal c-components. The query $Q[S]$ is identifiable if and only if there is no hedge formed for any of the sets $S_1, \ldots, S_k$.*

We emphasize that even in the case there is a hedge formed for only one of S_is, then $Q[S]$ is not identifiable. Revisiting the example of Figure 4, suppose $S = \{t, v_3, s_2, v_2\}$. The maximal c-components of $\mathcal{G}_{[S]}$ in this case are $S_1 = \{t, v_3\}$ and $S_2 = \{s_2, v_2\}$. You can verify that there are no hedges formed for S_2. However, as discussed above, $F = \{t, v_3, s_1\}$ is a hedge formed for S_1. Therefore, $Q[S]$ is not identifiable.

2.2 Objective of the project

We discussed above that $Q[S]$ in the graph of Figure 4 is not identifiable, where $S = \{t, v_3, s_2, v_2\}$. However, intervening upon certain vertices can make this query identifiable. For instance, after intervention on s_1, the resulting graph after surgery is depicted as Figure 5 below. It is straightforward to see that there are no hedges left for S_1 or S_2 anymore, and therefore, $Q[S]$ is identifiable.

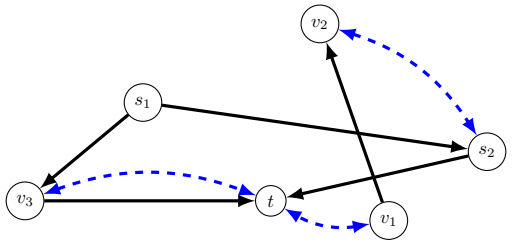


Figure 5: ADMG of Fig. 4 after intervening upon s_1.

As mentioned earlier, intervention on different variables may impose different costs on the practitioner. We model these costs through a pre-defined cost function $C : V \to \mathbb{R}^{\geq 0}$. Note that intervention costs cannot be negative. Our objective in this project is to find a set of *minimum cost*, intervening upon which makes the causal query of interest identifiable. We make the assumption that the cost of intervening on a set of variables is the sum of the costs of each variable. With slight abuse of notation, we use the function $C(\cdot)$ for sets of variables as well:

$$C(X) = \sum_{x \in X} C(x), \quad \forall X \subseteq V.$$

Consider for instance the toy example of Figure 6 below. Suppose we are interested in identifying the query $Q[S]$, where $S = \{y\}$. Clearly, $F = \{y, x, z\}$ forms a hedge for $Q[S]$ in this graph. However, intervention upon either z or x eliminates this hedge. Based on the costs, if $C(z) < C(x)$, then the optimal intervention to identify $Q[S]$ would be to intervene on z, and otherwise on x. Note that $\{x, z\}$ would never be optimal, as its cost is always larger than the singleton interventions.

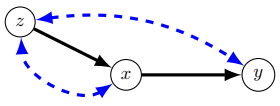


Figure 6: A toy example.

Formally speaking, given an ADMG $\mathcal{G}$ and a subset S of its nodes, the objective of this project is to find the intervention set with minimum cost, such that after intervening on that set, no hedge remains for $Q[S]$:

$$X_S^* \in \underset{X \in \mathbf{ID}(S)}{\arg\min} \sum\nolimits_{x \in X} C(x), \tag{2}$$

where $\mathbf{ID}(S)$ is the set of interventions after which there are no hedges left for $Q[S]$. As we shall shortly see, this problem is, in essence, a minimum hitting set problem. We first formally define the weighted minimum hitting set (WMHS) problem:

Definition 5 (WMHS)**.** *Let $V = \{v_1, ..., v_n\}$ be a set of objects along with a weight function $\omega : V \to \mathbb{R}^{\geq 0}$. Given a set family of subsets of V such as $\mathbf{F} = \{F_1, ..., F_k\}$, $F_i \subseteq V$, $1 \leq i \leq k$, a hitting set for $\mathbf{F}$ is a subset $A \subseteq V$ such A hits all the sets in $\mathbf{F}$, i.e., for any $1 \leq i \leq k$, $A \cap F_i \neq \emptyset$. The objective of the weighted minimum hitting set problem is to find a set A^* among all hitting sets that minimizes $\sum_{a \in A} \omega(a)$.*

The following lemma makes this project much simpler.

Lemma 1 ([1])**.** *Let $\mathcal{G}$ be an ADMG with vertex set V, along with a cost function $C : V \to \mathbb{R}^{\geq 0}$. Let S be a subset of V such that $S_1, \ldots, S_k$ is the partitioning of $\mathcal{G}_{[S]}$ into its maximal c-components. Let $\mathbf{F}_i = \{F_{i,1}, \ldots, F_{i,m_i}\}$ be the set of all hedges formed for $Q[S_i]$ in $\mathcal{G}$, for every $1 \leq i \leq k$. Then X_S^* is a solution to Equation (2) if and only if it is a solution to the MWHS problem for the sets $\{(F_{1,1} \setminus S_1), \ldots, (F_{1,m_1} \setminus S_1), (F_{2,1} \setminus S_2), \ldots, (F_{k,m_k} \setminus S_k)\}$, with the weight function $\omega(\cdot) := \mathbf{C}(\cdot)$.*

The intuition behind this lemma is straightforward: the hedges are the structures that prevent a query from being identified. It suffices therefore to intervene upon at least one vertex from each hedge to make the query identifiable. The minimum set that satisfies this property then corresponds to the minimum hitting set problem described above. In view of Lemma 1, the objective is clear: we will first identify the hedges formed for a given query $Q[S]$ and then solve the corresponding minimum hitting set problem.

For ease of presentation, let us consider the case where $\mathcal{G}_{[S]}$ is a c-component, that is, the partitioning of $\mathcal{G}_{[S]}$ to its maximal c-components comprises only one part. Algorithm 1 is presented to deal with the problem of minimum cost intervention problem in this setting. It uses a subroutine HHULL$(\cdot)$, which returns the maximal hedge formed for $Q[S]$ in the remaining graph. The subroutine HHULL(.) forms this maximal hedge by iteratively constructing the maximal c-component containing S (namely, F_1) and then keeping the subgraph pertaining to the ancestors of S in $\mathcal{G}_{[F_1]}$. If there is a hedge formed for $Q[S]$ in $\mathcal{G}$, then HHULL$(S, \mathcal{G})$ returns the maximal hedge formed for $Q[S]$. Otherwise, it returns the set S itself. If the returned set is S itself, no more interventions are needed, as the query is already identifiable (see lines 4-5 of Alg. 1.) Otherwise, the returned set is a hedge.

The main idea of algorithm 1 is then to use the subroutine HHULL(.) to discover a subset of the hedges formed for $Q[S]$ in $\mathcal{G}$ denoted by $\mathbf{F}$, such that the minimum hitting set solution for $\mathbf{F}$ is exactly the solution to the original min-cost intervention problem. It constructs $\mathbf{F}$ iteratively. To this end, within the inner loop (lines 7-13), it selects a vertex a in $H \setminus S$ with the minimum cost and removes a from H (resolves the hedge H). If this hedge elimination makes $Q[S]$ identifiable (i.e., HHULL$(S, \mathcal{G}_{[H \setminus \{a\}]}) = S$), it updates $\mathbf{F}$ (the set of already discovered hedges) in line 10 by adding H to it. Otherwise, it updates H by HHULL$(S, \mathcal{G}_{H \setminus \{a\}})$ in line 13. The reason for updating $\mathbf{F}$ only when $Q[S]$ becomes identifiable is that the hedge discovered in the last step of the inner loop H is a subset of all the hedges discovered earlier. Therefore, hitting (eliminating) H also hits all its super-sets.

Algorithm 1 Exact algorithm for minimum-cost intervention$(S, \mathcal{G})$, where $\mathcal{G}_{[S]}$ is a c-component.

```
1: function MinCostIntervention(S, G, C(·))
2:     F ← ∅
3:     H ← Hhull(S, G)
4:     if H = S then
5:         return ∅
6:     while True do
7:         while True do
8:             a ← arg min_{a∈H\S} C(a)
9:             if Hhull(S, G_{[H\{a}]}) = S then
10:                F ← F ∪ {H}
11:                break
12:            else
13:                H ← Hhull(S, G_{[H\{a}]})
14:        A ← solve min hitting set for {F \ S | F ∈ F}
15:        if Hhull(S, G_{[V\A]}) = S then
16:            return A
17:        H ← Hhull(S, G_{[V\A]})
```

```
1: function Hhull(S, G)
2:     Initialize F ← set of vertices of G
3:     while True do
4:         F_1 ← maximal c-component in G_{[F]} that contains S
5:         F_2 ← ancestors of S in G_{[F_1]}
6:         if F_2 ≠ F then
7:             F ← F_2
8:         else
9:             return F
```

At the end of the inner loop, it solves a minimum hitting set problem for the constructed $\mathbf{F}$ to find A in line 14. If intervening on A suffices to identify $Q[S]$, then A is returned (line 16) as the optimal intervention set. Otherwise, it updates H and repeats the outer loop by going back to line 6 to discover new hedges formed for $Q[S]$. It can be shown that the output of algorithm 1 is the optimal solution to Eq. (2).

2.3 A heuristic algorithm

The problem of minimum-cost intervention is NP-hard [1]. It comes with no surprise that Algorithm 1 has an exponential worst-case running time. It is not practical to run Algorithm 1 on large graphs. Herein, we present a fast heuristic algorithm to deal with large graphs. Note that this algorithm may be sub-optimal. We need a definition first:

Definition 6. *(Minimum weight vertex cut) Let $\mathcal{H}$ be a (un)directed graph over vertices V, with a weight function $\omega : V \to \mathbb{R}^{\geq 0}$. For two non-adjacent vertices $x, y \in V$, a subset $A \subset V \setminus \{x, y\}$ is said to be a vertex cut for $x - y$, if there is no (un)directed path that connects x to y in $\mathcal{H}_{[V\setminus A]}$. The objective of minimum weight vertex cut problem is to identify a vertex cut for $x - y$ that minimizes $\sum_{a\in A} \omega(a)$.*

Min-weight vertex cut problem can be solved in polynomial time by, for instance, casting it as a max-flow problem and then using algorithms such as Ford-Fulkerson, Edmonds-Karp, or push-relabel algorithm [4, 2, 5].

Heuristic algorithm: For a given graph $\mathcal{G}$ and a subset $S \subseteq V$, this algorithm builds an undirected graph $\mathcal{H}$ with the vertex set $H \cup \{x, y\}$, where $H = \textsc{Hull}(S, \mathcal{G})$, and x and y are two auxiliary vertices. For any pair of vertices $\{v_1, v_2\} \in H$, if v_1 and v_2 are connected with a bidirected edge in $\mathcal{G}$, they will be connected in $\mathcal{H}$. Vertex x is connected to all the vertices in $\mathbf{pa}(S) \cap H$, and y is connected to all vertices in S. The output of the algorithm is the minimum weight vertex cut for $x - y$, with the weight function $\omega(\cdot) := \mathbf{C}(\cdot)$. Importantly, we impose that the cost of intervention on any variable inside S is infinity (to assert that intervention on S is not allowed). Intervening on the output set of this algorithm will identify $Q[S]$, although this set is not necessarily minimum-cost.

2.4 Implementation guidelines

You will implement Algorithm 1 and the heuristic algorithm in a modular fashion and compare these two with each other. You can refer to Appendix A for a toy example of how these algorithms work.

1. Implement the function $\textsc{Hull}(S, \mathcal{G})$ according to the provided pseudo-code. Given that $\mathcal{G}_{[S]}$ comprises one c-component, this function returns the maximal hedge formed for $Q[S]$ in $\mathcal{G}$ (and S itself if there is no hedge). Since the input graph is an ADMG, it is recommended to represent this ADMG with two graphs, one directed (representing E^d, the directed edges among vertices) and one undirected (representing E^b, the bidirected edges among vertices).

2. Implement a function that receives a set family (set of sets) and returns the weighted minimum hitting set of these sets. This problem itself is NP-hard. You are not expected to do anything creative. Even a brute-force algorithm testing every subset works.

3. Combine your subroutines and implement Algorithm 1. This algorithm receives an ADMG $\mathcal{G}$ (you can again utilize the idea of inputting two separate graphs for representing directed and bidirected edges), a subset of its vertices S, and an array of costs C representing the cost of intervention on each vertex. The output must be the minimum-cost intervention that eliminates every hedge formed for $Q[S]$. If there is no such hedge, it outputs the empty set.

4. Implement the heuristic algorithm described above. Let A be the output of this algorithm for a certain input[2]. Do a sanity check by testing whether intervention on A does eliminate every hedge.

5. Both of the algorithms work when $\mathcal{G}_{[S]}$ consists of only one maximal c-component. Modify these algorithms properly so that they can handle cases where $\mathcal{G}_{[S]}$ consists of several maximal c-components.

6. (bonus) There are several approximation algorithms to solve the minimum hitting set problem (the subroutine of line 14 in Algorithm 1). One such method is using linear programs, described here. Implement the approximation algorithm of your choice (it doesn't have to be through linear programming, there are quite a few techniques out there!) Have a version of Algorithm 1 that uses this approximation method in line 14.

[2]Note that you need a module to solve a minimum-weight vertex cut problem. A typical strategy to tackle this problem is to reduce it to the minimum-weight edge cut problem, and then you can apply efficient max-flow/min-cut algorithms.

What to include in your report:

- Justify why HHULL(.) returns the maximal hedge.
- Explain how you modified your algorithm in task 5 to handle the cases where $\mathcal{G}_{[S]}$ has multiple c-components.
- Briefly describe your algorithm for solving the minimum-weight vertex cut problem for the heuristic algorithm.
- We are interested in comparing the exact algorithm (Alg. 1) to the heuristic algorithm in terms of run time and optimality of output. Generate random ADMGs according to the model of your choice over a range of graph sizes (number of vertices), and draw two plots: The first compares the run time of these two algorithms, and the second compares how far the output of these was from the optimal solution. Use your own metrics for comparison. The design of the tests is completely up to you. Note that Algorithm 1 in general cannot be run for very large and dense graphs, so you need to keep your graphs relatively small (or relatively sparse, as you wish.) Explain why you chose this model for generating random ADMGs, include the details of your parameters, and report the result of your comparison.
- Include one graph where the heuristic algorithm would return a sub-optimal solution. Do you have any suggestions to improve this algorithm?
- (bonus) What does your approximation algorithm in task 6 guarantee in terms of optimality? Does this guarantee apply to Algorithm 1 when used with this version of minimum hitting set solver? Include this algorithm as well in your time-optimality comparison of the previous part.

References

[1] Sina Akbari, Jalal Etesami, and Negar Kiyavash. Minimum cost intervention design for causal effect identification. In *International Conference on Machine Learning*, pages 258–289. PMLR, 2022.

[2] Jack Edmonds and Richard M Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM (JACM)*, 19(2):248–264, 1972.

[3] Ronald A Fisher. Frequency distribution of the values of the correlation coefficient in samples from an indefinitely large population. *Biometrika*, 10(4):507–521, 1915.

[4] Lester Randolph Ford and Delbert R Fulkerson. Maximal flow through a network. *Canadian journal of Mathematics*, 8:399–404, 1956.

[5] Andrew V Goldberg and Robert E Tarjan. A new approach to the maximum-flow problem. *Journal of the ACM (JACM)*, 35(4):921–940, 1988.

[6] Yimin Huang and Marco Valtorta. Pearl's calculus of intervention is complete. In *Proceedings of the Twenty-Second Conference on Uncertainty in Artificial Intelligence*, pages 217–224, 2006.

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

[8] Christopher Meek. Causal inference and causal explanation with background knowledge. In *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*, pages 403–410, 1995.

[9] Ilya Shpitser and Judea Pearl. Identification of joint interventional distributions in recursive semi-markovian causal models. In *Proceedings of the National Conference on Artificial Intelligence*, volume 21, page 1219, 2006.

[10] Jin Tian and Judea Pearl. On the testable implications of causal models with hidden variables. In *Proceedings of the eighteenth conference on Uncertainty in Artificial Intelligence*, volume 1, pages 519–527, 2002.

A Toy example

Consider the graph $\mathcal{G}$ of Figure 7, and suppose $S = \{s\}$. Also let the costs of intervention on z, w, and t be $2, 2$ and 3, respectively.

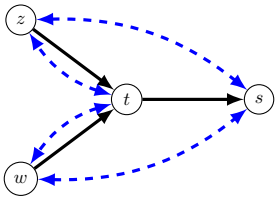


Figure 7: A toy example for the algorithms.

Algorithm 1. It can be verified that $\textsc{Hull}(S, \mathcal{G}) = \{s, t, w, z\}$. Line 8 of Alg. 1 chooses one of z or w to intervene upon first. Without loss of generality, let it be w. After intervention on w, the new $\textsc{Hull}(S, \mathcal{G}_{V\setminus\{w\}})$ is $H = \{s, t, z\}$. This time, $a = z$ in line 6 of the algorithm is chosen. After intervention on z as well, in the remaining graph, $\mathcal{G}_{V\setminus\{w,z\}}$, there is no hedge left. As a result, the hedge $H = \{s, t, z\}$ is added to $\mathbf{F}$. Then in line 14 of the algorithm, the minimum cost hitting set is found, which is $\{z\}$ (when there is only one set, the minimum hitting set is the minimum cost element of that set, which is z here). For the next loop of the algorithm, we start with a graph where z is already intervened upon. $H = \{s, t, w\}$, and in this loop, $a = w$ is chosen in line 6. After this intervention, $Q[S]$ becomes identifiable, and therefore the hedge $H = \{s, t, w\}$ is also added to $\mathbf{F}$. Now the set of discovered hedges is $\mathbf{F} = \{\{s, t, z\}, \{s, t, w\}\}$. This time, solving the weighted minimum hitting set for $\{F \setminus S | F \in \mathbf{F}\} = \{\{t, z\}, \{t, w\}\}$ returns $\{t\}$. Note that we need to choose either t, or z and w. Choosing t has cost 3, whereas z and w impose a cost of 4. So $\{t\}$ is the optimal solution. Finally, intervening on x alone makes $Q[S]$ identifiable, and the algorithm therefore terminates by returning $\{t\}$ as the optimal solution in line 16.

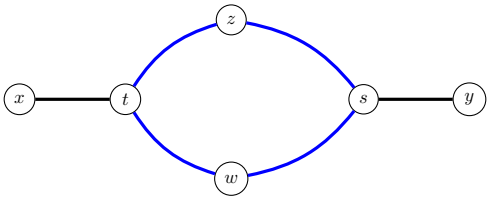


Figure 8: Corresponding undirected graph $\mathcal{H}$.

Heuristic algorithm. The corresponding undirected graph $\mathcal{H}$ is shown in Figure 8. The heuristic algorithm returns the minimum vertex cut between the pair $x - y$. Note that $\{t\}$ and $\{z, w\}$ are the two possible vertex cuts (and of course the combination $\{t, z, w\}$), as deleting any of these would make x and y disconnected. Considering the costs, the algorithm would return t, which happens to coincide with the true optimal solution.