A. Proof of propositions 1 and 2

The propositions follow directly from the definitions, but we state the proofs here for completeness. For proposition 1, first note ES is a functional of the covariance matrix Σ and it is by definition identifiable. Thus, if Q is identifiable, we can also uniquely compute Q from Σ and, since $B = ES - Q$, and each of its components are identifiable, B can also be uniquely computed from Σ and it is thus identifiable. Conversely, if B is identifiable, just write $Q = ES + B$, which means Q can be uniquely determined from Σ and it is also identifiable.

Proposition 2 follows the same argument. First note that if Q is θ-identifiable then we can write $B(\theta) = ES - Q(\theta)$ which is uniquely determined by Σ and θ, giving us a bias function parameterized in terms of θ. Conversely, if there exists a function $B(\theta)$ which, by definition, gives us a unique bias in terms of θ (and the data Σ), we can write $Q(\theta) = ES + B(\theta)$. This implies Q can be uniquely determined from Σ and θ and it is thus θ-identifiable.

B. Proof and pseudocode for Theorem 1

Theorem 1 (PUSHFORWARD)**.** *Given a linear SCM with graph G, covariance matrix Σ, a set of known directed edges $\mathcal{D}$, and known bidirected edge ϵ_{xy}, let the pair $\langle G', \Sigma' \rangle$ be constructed from G and Σ as follows:*

1. *$x \leftrightarrow y$ is removed and $\sigma'_{xy} = \sigma_{xy} - \epsilon_{xy}$;*
2. *$\forall c \in Ch(x), c \neq y$, the bidirected edges $c \leftrightarrow y$ are added if they do not exist, and $\epsilon'_{cy} = \epsilon_{cy} + \lambda_{cy}\epsilon_{xy}$;*
3. *$\forall z \in De(y)$, $z \neq x$, if there is an edge on any directed path from y to z that is not in $\mathcal{D}$, then z is removed from G'. For the remaining z, $\sigma'_{xz} = \sigma_{xz} - \epsilon_{xy}\delta_{yz}$, where δ_{yz} is the sum of all directed paths from y to z;*
4. *All other parameters and covariances remain the same.*

Then if λ_{ab} is identifiable in G' it is $(\epsilon_{xy}, \mathcal{D})$-identifiable in G.

Before moving forward, we use a couple definitions from the literature, which make reasoning about paths in the graph easier:

Definition 3. *(Foygel et al., 2012) A path π from v to w is a **trek** if it has no colliding arrowheads, that is, π is of the form:*

$$v \leftarrow ... \leftarrow \leftrightarrow \rightarrow ... \rightarrow w$$
$$v \leftarrow ... \leftarrow k \rightarrow ... \rightarrow w$$
$$v \leftarrow ... \leftarrow w$$
$$v \rightarrow ... \rightarrow w$$

Algorithm 2 PF - PUSHFORWARD

function PF($G, \Sigma, \mathcal{D}, \epsilon_{xy}, x$)
  initialize $\langle G', \Sigma' \rangle \leftarrow \langle G, \Sigma \rangle$
  update $\epsilon'_{xy} \leftarrow 0$ in G' and $\sigma'_{xy} \leftarrow \sigma'_{xy} - \epsilon_{xy}$ in Σ'
  for each $c \in Ch(x)$ **do**
    update $\epsilon'_{cy} \leftarrow \epsilon'_{cy} + \lambda_{xc}\epsilon_{xy}$ in G'
  end for
  for each $z \in De(y)$ **do**
    if $Edges(\delta_{yz}) \subseteq \mathcal{D}$ **then**
      update $\sigma'_{xz} = \sigma_{xz} - \epsilon_{xy}\delta_{yz}$
    else
      remove z from G'
    end if
  end for
  return $\langle G', \Sigma' \rangle$
end function

Definition 4. *(Foygel et al., 2012) A **trek monomial** $\pi(\Lambda, \mathcal{E})$ for trek π is defined as the product of the structural parameters along the trek, multiplied by the trek's top error term covariance.*

In particular, if π does not contain a bidirected edge[9]*,*

$$\pi(\Lambda, \mathcal{E}) = \epsilon_k^2 \prod_{x \rightarrow y \in \pi} \lambda_{xy}$$

where k is the node at the "top" of the trek (it has no incoming edges). If the trek contains bidirected edge ϵ_{ab}, then

$$\pi(\Lambda, \mathcal{E}) = \epsilon_{ab} \prod_{x \rightarrow y \in \pi} \lambda_{xy}$$

Lemma 1. *(Foygel et al., 2012) The covariance between v and w, σ_{vw} can be written as the sum of the trek monomials of all treks between v and w ($\mathcal{T}_{vw}$):*

$$\sigma_{vw} = \sum_{\pi \in \mathcal{T}_{vw}} \pi(\Lambda, \mathcal{E})$$

At its core, identifiability of an edge λ in linear Gaussian SCM can be reduced to the problem of finding whether there exists a unique solution for λ in terms of covariances in the system of equations defined by the rules of path analysis (Foygel et al., 2012), and knowledge of existing directed and bidirected effects.

With this in mind, we can prove PUSHFORWARD.

Proof. Specified in the theorem is a covariance matrix Σ, a graph of the structural equations G, a set of known directed edges $\mathcal{D}$, and known bidirected edge ϵ_{xy}.

[9] Note also that we can have a trek from v to v, including a trek that takes no edges at all, which would be simply ϵ_v^2

The system of equations constraining values of structural parameters is

$$\sigma_{vw} = \sum_{\pi \in \mathcal{T}_{vw}} \pi(\Lambda, \mathcal{E}) \qquad \forall v, w \in G$$

We first look at σ_{xy}, and define a new known quantity σ'_{xy}:

$$\sigma_{xy} = \epsilon_{xy} + \sum_{\pi \in \mathcal{T}_{xy} \setminus \{\epsilon_{xy}\}} \pi(\Lambda, \mathcal{E})$$
$$\sigma'_{xy} = \sigma_{xy} - \epsilon_{xy} = \sum_{\pi \in \mathcal{T}_{xy} \setminus \{\epsilon_{xy}\}} \pi(\Lambda, \mathcal{E})$$

We also look at all descendants of y, ($z \in Z$) where the directed paths from y to z (δ_{yz}) are made entirely of known edges ($Edges(\delta_{yz}) \subseteq \mathcal{D}$). We define

$$\delta_{ab} = \frac{1}{\epsilon_a^2} \sum_{\pi \in \mathcal{T}^{\rightarrow}_{xy}} \pi(\Lambda, \mathcal{E})$$

where $\mathcal{T}^{\rightarrow}_{xy}$ represents the set of treks taking only directed edges from a to b: $a \rightarrow ... \rightarrow b$.

For each such descendant of y, z, we define the quantity σ'_{xz}

$$\sigma_{xz} = \delta_{yz}\epsilon_{xy} + \sum_{\pi \in \mathcal{T}_{xy} \setminus \mathcal{T}^{\rightarrow}_{\epsilon_{xy} yz}} \pi(\Lambda, \mathcal{E})$$
$$\sigma'_{xz} = \sigma_{xz} - \delta_{yz}\epsilon_{xy} = \sum_{\pi \in \mathcal{T}_{xy} \setminus \mathcal{T}^{\rightarrow}_{\epsilon_{xy} yz}} \pi(\Lambda, \mathcal{E})$$

Here, we used $\mathcal{T}^{\rightarrow}_{\epsilon_{xy} yz}$ to represent the treks starting from ϵ_{xy}, and continuing from y to x (half-treks from x to z using ϵ_{xy}).

Finally, we define $\sigma'_{vw} = \sigma_{vw}$ for all other covariances between nodes a and b where both a and b are either non-descendants of y, or have their paths to y known.

This gives us a new system of equations in the original variables. All that remains to be shown is that an identified quantity λ'_{ab} in G' which contains a "pushed-forward" bidirected edge guarantees that the above-generated system of equations can be solved for the corresponding variable λ_{ab}.

As per the definition of G', it is identical to G, except:

1. the bidirected edge $x \leftrightarrow y$ is removed
2. $\forall c \in Ch(x)$, the edges $c \leftrightarrow y$ are added.
3. Descendants of y, z, where all edges of δ_{yz} are not known are removed

This new model G', with parameters Λ' and $\mathcal{E}'$ has system of equations:

$$\sigma''_{vw} = \sum_{\pi \in \mathcal{T}_{vw}} \pi(\Lambda', \mathcal{E}') \qquad \forall v, w \in G'$$

We compare this new system of equations to the modified equations of G.

- For all non-descendants of x or y, all covariance equations are identical (Both graphs have the same treks from non-descendants of x and y to all other nodes, and these covariances were not modified in the augmented equations).
- For all descendants of x, the modified equations for G have $\epsilon_{xy}\lambda_{xc}$ wherever G' has ϵ'_{cy} when G does not have bidirected edge $c \leftrightarrow y$. If G already includes an ϵ_{cy}, then it has $(\epsilon_{xy}\lambda_{xc} + \epsilon_{cy})$ for each ϵ'_{cy}. This can be seen by comparing the treks available in the two models. We can create a map of treks in G to treks in G'. Treks not crossing the added/removed bidirected edges are identical. All that remains are treks crossing ϵ_{xy} in G, and ϵ'_{cy} in G'. Suppose we have a trek from a to b in G' $a \leftarrow ... \leftarrow c \leftrightarrow y \rightarrow ... \rightarrow b$, crossing the bidirected edge ϵ'_{cy}. The corresponding trek in G across ϵ_{cy}, if it exists, and the trek $a \leftarrow ... \leftarrow c \rightarrow x \leftrightarrow y \rightarrow ... \rightarrow b$ both map to it. Since we have a map from treks in G to all treks in G', which differs only in the specified spot, the equations are likewise identical save for the mapping difference.
- The covariances between x and the descendants of y and y have likewise identical equations. This is because the removed treks in the modified equations are the only possibilities including ϵ_{xy}, so all variables behave as if the bidirected edge did not exist. This can also be seen by recognizing that setting $\epsilon_{xy} = 0$ would result in the same equation as removing all instances of the variable. Since the only treks from x which include ϵ_{xy} start by crossing $x \leftrightarrow y$, and continue on a directed path, removing all directed paths from y multiplied by ϵ_{xy} achieves the desired effect.

Finally, we notice that any algorithm for identifiability in this new model G' certifies that the system of equations can be uniquely solved for a given parameter, and the answer can be written in terms of Σ''.

We argue that the same parameter can be solved using the augmented equations of the original model G, replacing Σ'' with Σ' in the estimand returned from the identifiability algorithm for G'. This would be directly true if the modified equations for G were really identical to the equations for G'. However, the equations of G differ in ϵ_{xy} and ϵ_{xc} as

mentioned above. We show that this difference does not affect the solutions for any of the directed or bidirected edges except ϵ_{xc}.

Looking at the form of the structural equations, we get a sum of treks, which are themselves a product of paths. We exploit the mapping created above between treks in G to treks in G' to get:

$$\sigma''_{wv} = \sum \textit{treks not passing } \epsilon'_{cy} + \sum \textit{treks passing } \epsilon'_{cy}$$

Each trek can pass an ϵ at most once, at the top of the trek. Looking at the treks of G', we get:

$$\begin{aligned}\sum \textit{treks passing } \epsilon'_{cy} &= \delta'_{cw}\epsilon'_{cy}\delta'_{yv} + \delta'_{cv}\epsilon'_{cy}\delta'_{yw} \\ &= (\delta'_{cw} + \delta'_{cv})\epsilon'_{cy}(\delta'_{yv} + \delta'_{yw})\end{aligned}$$

The corresponding equation in G is:

$$\sum \textit{treks passing } \epsilon_{xy} = (\delta_{cw}+\delta_{cv})(\epsilon_{xy}\lambda_{xc}+\epsilon_{cy})(\delta_{yv}+\delta_{yw})$$

With the fact that the δ are all identical in both G and G' in terms of equation structure, we get:

$$\sum \textit{treks passing } \epsilon'_{cy} = (\delta_{cw} + \delta_{cv})\epsilon'_{cy}(\delta_{yv} + \delta_{yw})$$

Our goal now is to replace the $\epsilon_{xy}\lambda_{xc} + \epsilon_{cy}$ from the equation of G with a temporary variable ϵ_{Tc}, giving

$$\sum \textit{treks passing } \epsilon_{xy} = \delta_{cw}\epsilon_{Tc}\delta_{yv}$$

With this new system of equations, the temporary variable ϵ_{Tc} corresponds to ϵ'_{cy}, and the two systems are identical in their unknowns, making any solution in G' a solution for the modified G equations.

To achieve this, we need to show that the substitution is valid. The argument we are making is that if we have a system of equations:

$$\begin{aligned}K_1 &= x + (x + 5) \\ K_2 &= 3x\end{aligned}$$

we can replace $x + 5$ with y, giving:

$$\begin{aligned}K_1 &= x + y \\ K_2 &= 3x \\ y &= x + 5\end{aligned}$$

If the equations

$$\begin{aligned}K_1 &= x + y \\ K_2 &= 3x\end{aligned}$$

are sufficient for uniquely identifying the value of x, then assuming consistency of the original equations, the same solution for x is valid for the full system, including the third equation.

This is exactly the situation we have for our trek equations. We define $\epsilon_{Tc} = \epsilon_{xy}\lambda_{xc}+\epsilon_{cy}$, and if the system of equations excluding the equation constraining ϵ_{Tc} has a solution for a given parameter, the same solution will be valid for the full system.

Lastly, we notice that ϵ_{cy} can be obtained from the solutions to $\epsilon_{Tc}, \epsilon_{xy}, \lambda_{xc}$ using the same equation.

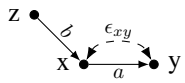
This completes the proof. □

C. Identification, sensitivity analysis and Gröbner bases

Gröbner bases are a symbolic method of computer algebra used to solve systems of polynomial equations. García-Puente et al. (2010) have shown that the identification (ID) problem in linear SCMs can be reduced to solving a system of polynomial equations and how Gröbner bases provide a complete solution.

In this section we will take a practical approach of showing how to set up the ID problem so it can be solved with Gröbner bases. We also show how to extend this to include sensitivity parameters, solving the problem of θ-identification. Our approach is based on García-Puente et al. (2010). For a basic understanding of Gröbner bases, please refer to Cox et al. (1992).

Gröbner bases can be seen as an algorithm to do variable elimination in complex polynomial equations. Let us illustrate the variable elimination approach in the simple instrumental variable graph:



We can write the (normalized) covariance equations induced by the graph as follows:

$$\begin{aligned}\sigma_{xy} &= a + \epsilon_{xy} \\ \sigma_{zy} &= b \times a \\ \sigma_{zx} &= b\end{aligned}$$

Given these equations, the goal is to solve for a in terms of the covariances of Σ only. Normally, one would approach

this directly, by simply eliminating one variable at a time. For example, after eliminating b, we get:

$$\sigma_{xy} = a + \epsilon_{xy}$$
$$\sigma_{zy} = \sigma_{zx} \times a$$

Next, we would eliminate ϵ_{xy}, by putting it in terms of a:

$$\epsilon_{xy} = \sigma_{xy} - a$$

Then, we have a final equation just in terms a and the Σ. This equation can be solved for a, and depending on how many values of a satisfy the constraint, it give us our identification result (here, only $a = \frac{\sigma_{zy}}{\sigma_{zx}}$ is valid).

Gröbner bases perform an equivalent operation—they successively eliminate variables from the system of equations. In this situation, we want to eliminate ϵ_{xy} and b, leaving only a and the covariances. In SAGE (The Sage Developers, 2018), this reduces to the following code:

```
R.<a,b,epsilon_xy,sigma_zx,sigma_zy,sigma_xy>
                                = PolynomialRing(QQ)
Ideal(
    sigma_xy - (a+epsilon_xy),
    sigma_zy - (b*a),
    sigma_zx - (b)
).elimination_ideal([epsilon_xy,b]).groebner_basis()
```

If the result is a first degree polynomial in a, there is a single solution.

The extension of this method to the θ-identification problem entailed by sensitivity analysis is straightforward. As sensitivity parameters are treated like known variables, we simply do not eliminate them. In the above example, if we were to treat ϵ_{xy} as a sensitivity parameter, our code would be:

```
R.<a,b,epsilon_xy,sigma_zx,sigma_zy,sigma_xy>
                                = PolynomialRing(QQ)
Ideal(
    sigma_xy - (a+epsilon_xy),
    sigma_zy - (b*a),
    sigma_zx - (b)
).elimination_ideal([b]).groebner_basis()
```

with an identical interpretation: if the resulting polynomials in a, Σ and ϵ_{xy} are linear in a, we conclude that knowing the givens is sufficient to identify a.

Unfortunately, despite the completeness of this approach, Gröbner bases are doubly-exponential in the number of variables, and in this case *each edge* corresponds to a variable (Bardet, 2002). This limits the practical solvable graph size to 4 or 5 nodes (Foygel et al., 2012; García-Puente et al., 2010). Our own experiments hit upon the same limitation, with attempted computations on 5-node graphs sometimes taking several days for identifying single edges, despite using an optimized representation of the equations (Foygel et al., 2012).

D. Detailed description of computational experiments

In this section, we provide a detailed description of our computational experiments, including pseudocode and additional tests. Our computational experiments have two main goals.

First, they aim to empirically verify the generality of our constrained identification algorithm CID, by comparing our results to the ground truth obtained via computer algebra.

Second, note that CID has three separate components:

1. The QID algorithm (Chen et al., 2017), which we use both for the identification of directed edges, and for incorporating constraints on directed edges that can be used as sensitivity parameters;
2. The graphical manipulations performed by PUSHFORWARD, which we use to incorporate constraints on bidirected edges; and,
3. The order in which to perform the graphical manipulation of PUSHFORWARD. In CID we chose to perform a topological ordering as described in Algorithm 1.

Thus, our computational experiments also aim to disentangle the contributions of each of those components to our results.

Solving all 3 and 4-Node sensitivity queries

Our computational experiments rely on the ability to find ground-truth answers to the question of whether a target coefficient λ_{ab} is θ-identifiable in a given graph G (this is defined to be a *query*). As explained in Section C of this supplementary material, these ground truth answers can be obtained with algebraic methods, more precisely using Gröbner bases (García-Puente et al., 2010).

For 3-node models we have 50 connected graphs with 720 possible queries; for 4-node models, we have 3,745 connected graphs and 1,059,156 possible queries. Note that, for 5-node models, we have 1,016,317 connected graphs and 11,615,669,904 possible queries. As mentioned in Section C, ground-truth computations using computer algebra can take hours (or sometimes days) for a *single* 5-node graph, redering an exhaustive study of sensitivity queries in 5-node models impractical.

We have thus performed an exhaustive computation of the ground truth answer of all possible queries in 3 and 4 node models via computer algebra using SAGE (The Sage Developers, 2018). These results give us a list stating for every graph G, every edge λ_{ab}, and *all possible subsets* of directed and bidirected edges used as sensitivity parameters θ, whether λ_{ab} can be uniquely computed from Σ and θ.

Our main interest lies on those queries that can be identified only when $\theta \neq \emptyset$ (we call this a sensitivity query)—in other words, we do not consider those edges that can be identified from Σ alone, since in these cases the parameter is identifiable and a sensitivity analysis would not be needed. The ground truth numbers of all θ-identifiable queries only when $\theta \neq \emptyset$ are 320 for 3-node models and 578,858 for 4-node models.

Our exhaustive computations also allow us to see how many sensitivity queries can be solved using *only subsets of directed edges* or *only subsets of bidirected edges* as sensitivity parameters. The decomposition then becomes the following. For 3-node models, there are 19 sensitivity queries that can be solved using only subsets of directed edges as sensitivity parameters, 109 using only subsets of bidirected edges, and, as before, 320 total queries which are solvable using an arbitrary combination of both. For 4-node models, these numbers increase to 15,740, 52,016 and 578,858 respectively. These numbers reveal that incorporating constraints on bidirected edges is an essential step for deriving sensitivity curves.

Comparing QID and CID to ground-truth answers

Once we have obtained ground-truth answers to all queries in 3 and 4-node models, we run both the QID as well as the CID algorithm for each of those queries and check whether they can correctly decide whether θ is an admissible set of sensitivity parameters for λ_{ab} in G (and thus able to provide a sensitivity curve). This comparison gives us the numbers we have presented in the main text in Table 1.

Alternative ordering methods for PUSHFORWARD

In the main text, the CID algorithm applies PUSHFORWARD in a topological ordering for processing multiple bidirected edges. The method does not perform all possible graphical manipulations, and as such, a valid concern is that it might be less capable than a more general search. Another interesting question is to check whether simpler methods would perform as well as the current CID implementation. To tackle these questions, we tested additional ordering methods for handling multiple bidirected edges.

For simplicity of exposition, the CID algorithm in the main text has the ordering method embedded in the pseudocode itself. For the purposes of this section, however, it is conceptually easier to create a meta algorithm that repeats the following process: (i) first it creates a collection of valid modified graphs $\mathcal{G}$ applying PUSHFORWARD according to some ordering method; then, (ii) it applies an identification algorithm to each of those modified graphs. This is given in Algorithm 3, which we call CID*.

In Algorithm 3, the argument PFORDER represents a function that takes as inputs a graph G, a covariance matrix Σ, a set of known bidirected edges $\mathcal{B}$ and a set of known directed edges $\mathcal{D}$. It then returns a *collection* $\mathcal{G}$ of valid modified models $\langle G', \Sigma' \rangle$ by iteratively applying PUSHFORWARD following a particular ordering method (for example, topological ordering). The argument IDMETHOD refers to an identification method for directed edges (for instance, QID). It is a function that takes as inputs a graph G, a covariance matrix Σ and a set of known directed edges $\mathcal{D}$ and it returns the new set of known directed edges.

Algorithm 3 Meta constrained ID algorithm.

function CID*($G, \Sigma, \mathcal{B}, \mathcal{D}$, PFORDER, IDMETHOD)
  repeat
    $\mathcal{G} \leftarrow$ PFORDER($G, \Sigma, \mathcal{B}, \mathcal{D}$)
    for $\langle G', \Sigma' \rangle \in \mathcal{G}$ **do**
      $\mathcal{D} \leftarrow \mathcal{D} \cup$ IDMETHOD($G', \Sigma', \mathcal{D}$)
    end for
  until all directed edges have been identified or no edge has been identified in the last iteration
  return $\mathcal{D}$
end function

We can now create different functions for different ordering methods. For instance, the function PFT described in Algorithm 7 applies PUSHFORWARD in topological ordering (as embedded in Algorithm 1 of the main text) and returns all valid modified graphs. We now define three additional ordering methods.

- PFO described in Algorithm 5. This function pushes forward each bidirected edge only once, considering the original graph. This method is the simplest application of PUSHFORWARD, and serves as a base of comparison to assess the gains of more elaborate methods.
- PFS described in Algorithm 6. This function tries to apply PUSHFORWARD once to all subsets of bidirected edges connected to each end node. This procedure has exponential computational complexity.
- PFR described in Algorithm 8. This function recursively tries every possible combination of applying PUSHFORWARD for each bidirected edge connected to the same end node (it tries each subset once, and of those that can be pushed forward again, tries each subset, and so on). This procedure has doubly exponential computational complexity.

All these function return a collection $\mathcal{G}$ of valid modified graphs, and can be used as the PFORDER argument in the CID* function. Of these methdos, PFR is arguably the most important for comparison with our current implementation of topological ordering. The results are shown in the

			3 NODES			4 NODES		
	PF order	ID Alg. directed edges	*Directed*	*Bidirected*	*Both*	*Directed*	*Bidirected*	*Both*
	none	QID	19	-	68	14,952	-	170,304
	PFO	QID	19	101	304	14,952	43,526	505,076
	PFS	QID	19	105	308	14,952	46,630	517,036
	PFR	QID	19	109	320	14,952	50,708	555,758
(CID)	**PFT**	**QID**	19	109	320	14,952	50,708	555,758
	none	Complete	19	-	68	15,740	-	177,216
	PFO	Complete	19	101	304	15,740	44,680	524,846
	PFS	Complete	19	105	308	15,740	47,962	538,332
	PFR	Complete	19	109	320	15,740	51,992	578,758
	PFT	Complete	19	109	320	15,740	51,992	578,758
	GROUND TRUTH		19	109	320	15,740	52,016	578,858

Table 2: Number of θ-identifiable queries (only when $\theta \neq \emptyset$) per type of sensitivity parameters θ, using different ordering methods for PUSHFORWARD and different ID algorithm for the directed edges. Ground Truth is computed using Gröbner bases. The first column defines the ordering method of PUSHFORWARD used for incorporating constraints on bidirected edges—this is passed as the argument PFORDER in the general function CID*. The second column refers to the identification algorithm used for directed edges—this is passed as the argument IDMETHOD in the general function CID*. "Complete" means we used Gröbner bases to simulate a complete ID algorithm for *directed edges* running inside CID*. Note the first row corresponds to QID and the boldfaced row corresponds to CID as presented in the main text applying PUSHFORWARD in topological ordering. These two rows are the ones presented in Table 1 of the main text. A pseudocode for computing these numbers is given in Algorithm 4.

first half of Table 2, which compares CID* using the same ID method for directed edges (QID) but different ordering methods for applying PUSHFORWARD. Our preferred version, which was presented in the main text as CID, corresponds to the boldfaced row with ordering method PFT and ID method QID. As we can see, topological ordering performs as well as the brute-force recursive search of all subsets performed by PFR, which has doubly exponential computational complexity.

Disentangling PF and QID

Finally, the incompleteness of CID can stem from two sources: limitations of the graphical manipulations performed by PUSHFORWARD or the incompleteness of the identification algorithm for directed edges, QID. Separating the two can help guide efforts for future research. To achieve that, we used algebraic methods to simulate how CID would have performed if it had access to a complete identification algorithm for directed edges instead of QID.

More precisely, we use Gröbner bases as our ID algorithm for directed edges (IDMETHOD) in CID*, where, just like QID, Gröbner bases only have access to constraints on bidirected edges via the graphical manipulation performed by PUSHFORWARD. That is, Gröbner bases is dealing with the problem as if it were a "vanilla" identification problem, not explicitly knowing that the bidirected edge is fixed. The results can be seen in the second half of Table 2. The last row indicates, for instance, that incorporating constraints on bidirected edges using PUSHFORWARD in topological order, in combination with a complete identification algorithm for directed edges, would have identified over 99.99% of 4-node sensitivity queries.

This suggests that: (i) the main bottleneck of the current implementation of CID is QID itself; (ii) PUSHFORWARD with topological ordering is an efficient procedure for dealing with bidirected edges that can reap the benefits of improved identification algorithms.

E. The missed cases

As discussed in the previous section, PUSHFORWARD in topological order, in combination with a complete identification algorithm for *directed edges*, would have identified over 99.99% of all 4-node sensitivity queries. In this section we briefly discuss some of the missed cases, which may provide guidance for further improvements of the CID algorithm. We also provide all the missed cases for those interested in exploring them further (Tables 3 and 4).

When iterating over modified graphs, the CID algorithm feeds its next iteration *only* identification results for *direct effects* (single coefficients), not of path specific effects or total effects (sums of products of coefficients), which may nevertheless be identified. Figure 8 shows two simple examples that illustrates how not exploiting the knowledge of

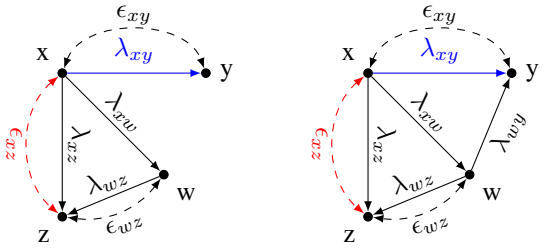


Figure 8: Examples of missed cases using PUSHFORWARD with a complete identification algorithm of *directed edges*. In both examples, λ_{xy} is ϵ_{zx}-identifiable, but the algorithm fails due to lack of exploitation of identified total effects. In example 8a, it turns out a simple marginalization of w suffices for the ϵ_{zx}-identification of λ_{xy} using the current implementation of the CID algorithm. However, marginalization alone is often not enough, as shown in example 8b.

known total effects can result in a failure of identification.

Let us start with Figure 8a. In this example, our task is to find a sensitivity curve for λ_{xy} in terms of ϵ_{zx}. First note that z is not a valid instrument for λ_{xy} since it is a descendant of x. However, pushing forward ϵ_{zx} allows us to identify the *total effect* of x on z. This, in turn, permits the creation of the auxiliary variable $z^* = z-(\lambda_{xz}+\lambda_{xw}\lambda_{wz})x$ which is now a valid instrument for λ_{xy}. In the example of Figure 8a, it turns out a simpler solution would also suffice—marginalizing w. Note the marginalized DAG results in a simple three node model which can be solved by the current implementation of CID. Nevertheless, marginalization by itself may not always be sufficient, as a simple variation of this very example shows (Figure 8b).

In sum, θ-identification in these cases require systematically exploiting known total effects (for instance, creating AVs subtracting out total effects) or known path-specific effects, a task which still does not have a satisfactory solution in the literature. A final interesting (and challenging) example in which CID failed to find the sensitivity curve is shown in Fig .9.

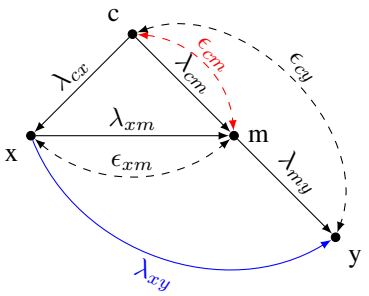


Figure 9: An interesting missed case example. Here λ_{xy} is ϵ_{cm}-identifiable. All examples can be found in Tables 3 and 4.

F. Utility of descendants

Here we show that not pruning descendants of variable y can be useful for identification. An example is given in Fig. 10.

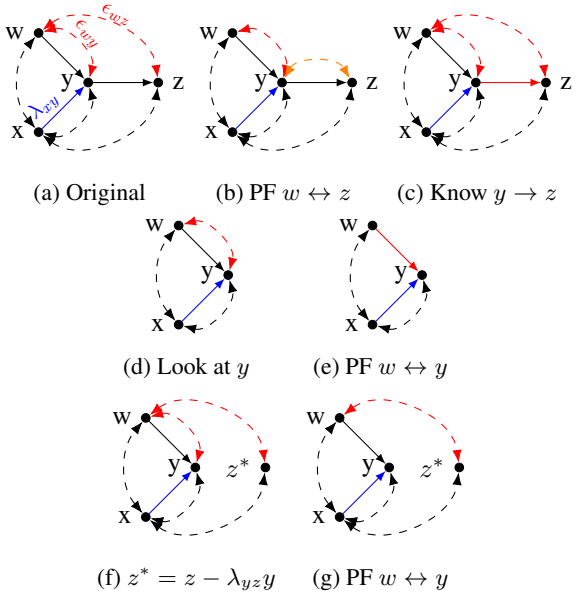


Figure 10: Take the graph in 10a. The red bidirected edges $(\epsilon_{wy}, \epsilon_{wz})$ are assumed to be known, and the target quantity is λ_{xy} (blue). First, we can use knowledge of $w \leftrightarrow z$ to use w as an instrument for $y \to z$ (10b,10c). This knowledge, however, does not help in solving for $x \to y$ (10d), even when pushing forward $w \leftrightarrow y$ (10e). The issue is that we pruned z when pushing forward $w \leftrightarrow y$, since it is a descendant of y. However, note we can create an AV z^*, which behaves as if it were not a descendant of y (10f). It turns out that z^* is an instrument for $x \to y$ conditioned on w in 10g, solving the problem!

Algorithm 4 Pseudocode for checking performance of CID* with different PUSHFORWARD orders and different ID algorithm for directed edges. In the code, IDENTIFYDIRECTEDEDGES and ISIDENTIFIED are computed using computer algebra (Gröbner bases), and give the ground-truth values.

initialize Total $\leftarrow 0$
initialize PFtotal $\leftarrow 0$
$\mathcal{S} \leftarrow$ set of all possible connected DAGs, with all combinations of directed and bidirected edges.
for each graph $\langle G, \Sigma \rangle \in \mathcal{S}$ **do**
  IDedges $\leftarrow$ IDENTIFYDIRECTEDEDGES($\mathcal{G}$)
  for all $(x \to y) \in \mathcal{G}$ **where** $(x \to y) \notin$ IDedges **do**
    $\mathcal{SPS} \leftarrow$ All subsets of directed and bidirected edges of G which do not contain $(x \to y)$
    for each set $\langle \mathcal{D}, \mathcal{B} \rangle \in \mathcal{SPS}$ **do**
      if ISIDENTIFIED($\mathcal{G}, x \to y, \mathcal{D}, \mathcal{B}$) **then**
        Total $\leftarrow$ Total $+ 1$
        if $(x \to y) \in$ CID*($G, \Sigma, \mathcal{D}, \mathcal{B}$, PFORDER, IDMETHOD) **then**
          PFTotal $\leftarrow$ PFTotal $+ 1$
        end if
      end if
    end for
  end for
end for
return $\frac{PFtotal}{Total}$

Algorithm 5 Push forward each bidirected edge once.

function PFO($G, \Sigma, \mathcal{B}, \mathcal{D}$)
  let $\mathcal{B}_y$ represent subset of $\mathcal{B}$ where all edges have y as end point ($B_y = \{(x \leftrightarrow y) \in \mathcal{B}, \forall x\}$)
  $\mathcal{G} \leftarrow \{(G, \Sigma)\}$
  for each node $y \in G$ **do**
    for bidirected edge $\epsilon_{xy} \in \mathcal{B}_y$ **do**
      if $x \notin$ DE(y) **or** $\delta_{yx} \in \mathcal{D}$ **then**
        add PF($G, \Sigma, \mathcal{D}, \epsilon_{xy}, x$) **to** $\mathcal{G}$
      end if
    end for
  end for
  return $\mathcal{G}$
end function

Algorithm 6 Push forward all subsets once.

function PFS($G, \Sigma, \mathcal{B}, \mathcal{D}$)
  let $\mathcal{B}_y$ represent subset of $\mathcal{B}$ where all edges have y as end point ($B_y = \{(x \leftrightarrow y) \in \mathcal{B}, \forall x\}$)
  $\mathcal{G} \leftarrow \{(G, \Sigma)\}$
  for each node y **do**
    for each $B'_y \subseteq B_y$ **do**
      $\langle G', \Sigma' \rangle \leftarrow \langle G, \Sigma \rangle$
      for each $\epsilon_{xy} \in B'_y$ **do**
        if $x \notin$ DE(y) **or** $\delta_{yx} \in \mathcal{D}$ **then**
          $\langle G', \Sigma' \rangle \leftarrow$PF($G', \Sigma', \mathcal{D}, \epsilon_{xy}, x$)
          for all $z \in$ CH(x) **do**
            if $\lambda_{xz} \notin \mathcal{D}$ **then**
              remove ϵ_{zy} from $\mathcal{B}'_y$ if it was not yet processed.
            end if
          end for
        end if
      end for
      add $\langle G', \Sigma' \rangle$ **to** $\mathcal{G}$
    end for
  end for
  return $\mathcal{G}$
end function

Algorithm 7 Push forward in topological order.

function PFT($G, \Sigma, \mathcal{B}, \mathcal{D}$)
  let $\mathcal{B}_y$ represent subset of $\mathcal{B}$ where all edges have y as end point ($B_y = \{(x \leftrightarrow y) \in \mathcal{B}, \forall x\}$)
  $\mathcal{G} \leftarrow \{(G, \Sigma)\}$
  for each node y **do**
    $\langle G', \Sigma' \rangle \leftarrow \langle G, \Sigma \rangle$
    for each $\epsilon_{xy} \in B_y$ in topological order on x **do**
      if $x \notin$ DE(y) **or** $\delta_{yx} \in \mathcal{D}$ **then**
        $\langle G', \Sigma' \rangle \leftarrow$PF($G', \Sigma', \mathcal{D}, \epsilon_{xy}, x$)
        add $\langle G', \Sigma' \rangle$ **to** $\mathcal{G}$
        for all $z \in$ CH(x) **do**
          if $\lambda_{xz} \notin \mathcal{D}$ **then**
            remove ϵ_{zy} from $\mathcal{B}_y$ if it was not yet processed.
          else
            add ϵ_{zy} to $\mathcal{B}_y$
          end if
        end for
      end if
    end for
  end for
  return $\mathcal{G} \cup$ PFO($G, \Sigma, \mathcal{B}, \mathcal{D}$)
end function

Algorithm 8 Push forward all subsets recursively.

function PFR($G, \Sigma, \mathcal{B}, \mathcal{D}$)
  let $\mathcal{B}_y$ represent subset of $\mathcal{B}$ where all edges have y as end point ($B_y = \{(x \leftrightarrow y) \in \mathcal{B}, \forall x\}$)
  initialize $\mathcal{G} \leftarrow \{(G, \Sigma, \emptyset)\}$
  for each node y **do**
    $PushSets \leftarrow \{\langle G, \Sigma, B'_y \rangle$ for all $B'_y \subseteq B_y\}$
    while $PushSets$ **not empty do**
      pop $\langle G', \Sigma', B'_y \rangle$ from $PushSets$
      $PushAgain \leftarrow \{\}$
      for each $\epsilon_{xy} \in B'_y$ **do**
        if $x \notin \text{DE}(y)$ **or** $\delta_{yx} \in \mathcal{D}$ **then**
          $\langle G', \Sigma' \rangle \leftarrow$PF($G', \Sigma', \mathcal{D}, \epsilon_{xy}, x$)
          for all $z \in \text{CH}(x)$ **do**
            if $\lambda_{xz} \notin \mathcal{D}$ **then**
              remove ϵ_{zy} from $\mathcal{B}'_y$ if it was not yet processed.
            else
              add ϵ_{zy} **to** $PushAgain$
            end if
          end for
        end if
      end for
      add $\langle G', \Sigma' \rangle$ **to** $\mathcal{G}$
      for all$B''_y \subseteq PushAgain$ **do**
        add $\langle G', \Sigma', B''_y \rangle$ **to** $PushSets$
      end for
    end while
  end for
  return $\mathcal{G}$
end function

	Graph	Target Quantity	Sensitivity Parameters
1	1→2 1→3 1→4 2→4 1↔3 1↔4 2↔4	1→3	1↔4
2	1→2 1→3 1→4 2→4 1↔3 1↔4 2↔4	1→3	1↔4 1→2
3	1→2 1→3 1→4 2→4 1↔3 1↔4 2↔4 3↔4	1→3	1↔4 3↔4
4	1→2 1→3 1→4 2→4 1↔3 1↔4 2↔4 3↔4	1→3	1↔4 3↔4 1→2
5	1→2 1→3 1→4 2→4 3→4 1↔3 1↔4 2↔4	1→3	1↔4 3→4
6	1→2 1→3 1→4 2→4 3→4 1↔3 1↔4 2↔4	1→3	1↔4 1→2 3→4
7	1→2 1→3 1→4 2→4 3→4 1↔3 1↔4 2↔4 3↔4	1→3	1↔4 3↔4 3→4
8	1→2 1→3 1→4 2→4 3→4 1↔3 1↔4 2↔4 3↔4	1→3	1↔4 3↔4 1→2 3→4
9	1→2 1→3 2→3 1→4 2→4 1↔3 1↔4 2↔4	1→3	1↔4
10	1→2 1→3 2→3 1→4 2→4 1↔3 1↔4 2↔4	1→3	1↔4 2→3
11	1→2 1→3 2→3 1→4 2→4 1↔3 1↔4 2↔4	1→3	1↔4 1→2
12	1→2 1→3 2→3 1→4 2→4 1↔3 1↔4 2↔4	1→3	1↔4 1→2 2→3
13	1→2 1→3 2→3 1→4 2→4 1↔3 1↔4 2↔4 3↔4	1→3	1↔4 3↔4
14	1→2 1→3 2→3 1→4 2→4 1↔3 1↔4 2↔4 3↔4	1→3	1↔4 3↔4 2→3
15	1→2 1→3 2→3 1→4 2→4 1↔3 1↔4 2↔4 3↔4	1→3	1↔4 3↔4 1→2
16	1→2 1→3 2→3 1→4 2→4 1↔3 1↔4 2↔4 3↔4	1→3	1↔4 3↔4 1→2 2→3
17	1→2 1→3 2→3 1→4 2→4 3→4 1↔3 1↔4 2↔4	1→3	1↔4 3→4
18	1→2 1→3 2→3 1→4 2→4 3→4 1↔3 1↔4 2↔4	1→3	1↔4 3→4 1→2
19	1→2 1→3 2→3 1→4 2→4 3→4 1↔3 1↔4 2↔4	1→3	1↔4 2→3 3→4
20	1→2 1→3 2→3 1→4 2→4 3→4 1↔3 1↔4 2↔4	1→3	1↔4 2→3 3→4 1→2
21	1→2 1→3 2→3 1→4 2→4 3→4 1↔3 1↔4 2↔4 3↔4	1→3	1↔4 3↔4 3→4
22	1→2 1→3 2→3 1→4 2→4 3→4 1↔3 1↔4 2↔4 3↔4	1→3	1↔4 3↔4 3→4 1→2
23	1→2 1→3 2→3 1→4 2→4 3→4 1↔3 1↔4 2↔4 3↔4	1→3	1↔4 3↔4 2→3 3→4
24	1→2 1→3 2→3 1→4 2→4 3→4 1↔3 1↔4 2↔4 3↔4	1→3	1↔4 3↔4 2→3 3→4 1→2
25	1→2 1→3 2→3 2→4 3→4 1↔3 2↔3 1↔4	2→4	1↔3
26	1→2 1→3 2→3 2→4 3→4 1↔3 2↔3 1↔4	2→4	1↔3 1→2
27	1→2 1→3 2→3 2→4 3→4 1↔3 2↔3 1↔4	3→4	1↔3
28	1→2 1→3 2→3 2→4 3→4 1↔3 2↔3 1↔4	3→4	1↔3 1→2
29	1→2 1→3 2→3 2→4 3→4 1↔3 2↔3 1↔4 3↔4	2→4	1↔3 3↔4
30	1→2 1→3 2→3 2→4 3→4 1↔3 2↔3 1↔4 3↔4	2→4	1↔3 3↔4 1→2
31	1→2 1→3 2→3 2→4 3→4 1↔3 2↔3 1↔4 3↔4	3→4	1↔3 3↔4
32	1→2 1→3 2→3 2→4 3→4 1↔3 2↔3 1↔4 3↔4	3→4	1↔3 3↔4 1→2
33	1→2 1→3 2→3 1→4 1↔3 2↔3 1↔4	1→4	1↔3
34	1→2 1→3 2→3 1→4 1↔3 2↔3 1↔4	1→4	1↔3 1→2
35	1→2 1→3 2→3 1→4 1↔3 2↔3 1↔4 3↔4	1→4	1↔3 3↔4
36	1→2 1→3 2→3 1→4 1↔3 2↔3 1↔4 3↔4	1→4	1↔3 3↔4 1→2
37	1→2 1→3 2→3 1→4 3→4 1↔3 2↔3 1↔4	1→4	1↔3
38	1→2 1→3 2→3 1→4 3→4 1↔3 2↔3 1↔4	1→4	1↔3 3→4
39	1→2 1→3 2→3 1→4 3→4 1↔3 2↔3 1↔4	1→4	1↔3 1→2
40	1→2 1→3 2→3 1→4 3→4 1↔3 2↔3 1↔4	1→4	1↔3 1→2 3→4
41	1→2 1→3 2→3 1→4 3→4 1↔3 2↔3 1↔4 3↔4	1→4	1↔3 3↔4
42	1→2 1→3 2→3 1→4 3→4 1↔3 2↔3 1↔4 3↔4	1→4	1↔3 3↔4 3→4
43	1→2 1→3 2→3 1→4 3→4 1↔3 2↔3 1↔4 3↔4	1→4	1↔3 3↔4 1→2
44	1→2 1→3 2→3 1→4 3→4 1↔3 2↔3 1↔4 3↔4	1→4	1↔3 3↔4 1→2 3→4
45	1→2 1→3 2→3 1→4 2→4 1↔3 2↔3 1↔4	1→4	1↔3
46	1→2 1→3 2→3 1→4 2→4 1↔3 2↔3 1↔4	1→4	1↔3 2→4
47	1→2 1→3 2→3 1→4 2→4 1↔3 2↔3 1↔4	1→4	1↔3 1→2
48	1→2 1→3 2→3 1→4 2→4 1↔3 2↔3 1↔4	1→4	1↔3 1→2 2→4
49	1→2 1→3 2→3 1→4 2→4 1↔3 2↔3 1↔4 3↔4	1→4	1↔3 3↔4
50	1→2 1→3 2→3 1→4 2→4 1↔3 2↔3 1↔4 3↔4	1→4	1↔3 3↔4 2→4

Table 3: Missed sensitivity queries of PUSHFORWARD in topological order, in combination with a complete identification algorithm for *directed edges*. Part 1.

	Graph	Target Quantity	Sensitivity Parameters
51	1→2 1→3 2→3 1→4 2→4 1↔3 2↔3 1↔4 3↔4	1→4	1↔3 3↔4 1→2
52	1→2 1→3 2→3 1→4 2→4 1↔3 2↔3 1↔4 3↔4	1→4	1↔3 3↔4 1→2 2→4
53	1→2 1→3 2→3 1→4 2→4 3→4 1↔3 2↔3 1↔4	1→4	1↔3 3→4
54	1→2 1→3 2→3 1→4 2→4 3→4 1↔3 2↔3 1↔4	1→4	1↔3 3→4 1→2
55	1→2 1→3 2→3 1→4 2→4 3→4 1↔3 2↔3 1↔4	1→4	1↔3 2→4
56	1→2 1→3 2→3 1→4 2→4 3→4 1↔3 2↔3 1↔4	1→4	1↔3 1→2 2→4
57	1→2 1→3 2→3 1→4 2→4 3→4 1↔3 2↔3 1↔4	1→4	1↔3 3→4 2→4
58	1→2 1→3 2→3 1→4 2→4 3→4 1↔3 2↔3 1↔4	1→4	1↔3 3→4 1→2 2→4
59	1→2 1→3 2→3 1→4 2→4 3→4 1↔3 2↔3 1↔4	1→4	1↔3 1↔4
60	1→2 1→3 2→3 1→4 2→4 3→4 1↔3 2↔3 1↔4	1→4	1↔3 1↔4 1→2
61	1→2 1→3 2→3 1→4 2→4 3→4 1↔3 2↔3 1↔4	2→4	1↔3 1→4
62	1→2 1→3 2→3 1→4 2→4 3→4 1↔3 2↔3 1↔4	2→4	1↔3 1→2 1→4
63	1→2 1→3 2→3 1→4 2→4 3→4 1↔3 2↔3 1↔4	2→4	1↔3 1↔4
64	1→2 1→3 2→3 1→4 2→4 3→4 1↔3 2↔3 1↔4	2→4	1↔3 1↔4 1→2
65	1→2 1→3 2→3 1→4 2→4 3→4 1↔3 2↔3 1↔4	3→4	1↔3 1→4
66	1→2 1→3 2→3 1→4 2→4 3→4 1↔3 2↔3 1↔4	3→4	1↔3 1→2 1→4
67	1→2 1→3 2→3 1→4 2→4 3→4 1↔3 2↔3 1↔4	3→4	1↔3 1↔4
68	1→2 1→3 2→3 1→4 2→4 3→4 1↔3 2↔3 1↔4	3→4	1↔3 1↔4 1→2
69	1→2 1→3 2→3 1→4 2→4 3→4 1↔3 2↔3 1↔4 3↔4	1→4	1↔3 3↔4 3→4
70	1→2 1→3 2→3 1→4 2→4 3→4 1↔3 2↔3 1↔4 3↔4	1→4	1↔3 3↔4 3→4 1→2
71	1→2 1→3 2→3 1→4 2→4 3→4 1↔3 2↔3 1↔4 3↔4	1→4	1↔3 3↔4 2→4
72	1→2 1→3 2→3 1→4 2→4 3→4 1↔3 2↔3 1↔4 3↔4	1→4	1↔3 3↔4 1→2 2→4
73	1→2 1→3 2→3 1→4 2→4 3→4 1↔3 2↔3 1↔4 3↔4	1→4	1↔3 3↔4 3→4 2→4
74	1→2 1→3 2→3 1→4 2→4 3→4 1↔3 2↔3 1↔4 3↔4	1→4	1↔3 3↔4 3→4 1→2 2→4
75	1→2 1→3 2→3 1→4 2→4 3→4 1↔3 2↔3 1↔4 3↔4	1→4	1↔3 1↔4 3↔4
76	1→2 1→3 2→3 1→4 2→4 3→4 1↔3 2↔3 1↔4 3↔4	1→4	1↔3 1↔4 3↔4 1→2
77	1→2 1→3 2→3 1→4 2→4 3→4 1↔3 2↔3 1↔4 3↔4	2→4	1↔3 3↔4 1→4
78	1→2 1→3 2→3 1→4 2→4 3→4 1↔3 2↔3 1↔4 3↔4	2→4	1↔3 3↔4 1→2 1→4
79	1→2 1→3 2→3 1→4 2→4 3→4 1↔3 2↔3 1↔4 3↔4	2→4	1↔3 1↔4 3↔4
80	1→2 1→3 2→3 1→4 2→4 3→4 1↔3 2↔3 1↔4 3↔4	2→4	1↔3 1↔4 3↔4 1→2
81	1→2 1→3 2→3 1→4 2→4 3→4 1↔3 2↔3 1↔4 3↔4	3→4	1↔3 3↔4 1→4
82	1→2 1→3 2→3 1→4 2→4 3→4 1↔3 2↔3 1↔4 3↔4	3→4	1↔3 3↔4 1→2 1→4
83	1→2 1→3 2→3 1→4 2→4 3→4 1↔3 2↔3 1↔4 3↔4	3→4	1↔3 1↔4 3↔4
84	1→2 1→3 2→3 1→4 2→4 3→4 1↔3 2↔3 1↔4 3↔4	3→4	1↔3 1↔4 3↔4 1→2
85	1→2 2→3 2→4 3→4 1↔2 1↔4 3↔4	1→2	1↔4
86	1→2 2→3 2→4 3→4 1↔2 1↔4 3↔4	1→2	1↔4 2→3
87	1→2 2→3 2→4 3→4 1↔2 1↔4 2↔4 3↔4	1→2	1↔4 2↔4
88	1→2 2→3 2→4 3→4 1↔2 1↔4 2↔4 3↔4	1→2	1↔4 2↔4 2→3
89	1→2 2→3 1→4 2→4 3→4 1↔2 1↔4 3↔4	1→2	1↔4 1→4
90	1→2 2→3 1→4 2→4 3→4 1↔2 1↔4 3↔4	1→2	1↔4 2→3 1→4
91	1→2 2→3 1→4 2→4 3→4 1↔2 1↔4 2↔4 3↔4	1→2	1↔4 2↔4 1→4
92	1→2 2→3 1→4 2→4 3→4 1↔2 1↔4 2↔4 3↔4	1→2	1↔4 2↔4 2→3 1→4
93	1→2 1→3 1→4 3→4 1↔2 1↔4 3↔4	1→2	1↔4
94	1→2 1→3 1→4 3→4 1↔2 1↔4 3↔4	1→2	1↔4 1→3
95	1→2 1→3 1→4 3→4 1↔2 1↔4 2↔4 3↔4	1→2	1↔4 2↔4
96	1→2 1→3 1→4 3→4 1↔2 1↔4 2↔4 3↔4	1→2	1↔4 2↔4 1→3
97	1→2 1→3 1→4 2→4 3→4 1↔2 1↔4 3↔4	1→2	1↔4 2→4
98	1→2 1→3 1→4 2→4 3→4 1↔2 1↔4 3↔4	1→2	1↔4 1→3 2→4
99	1→2 1→3 1→4 2→4 3→4 1↔2 1↔4 2↔4 3↔4	1→2	1↔4 2↔4 2→4
100	1→2 1→3 1→4 2→4 3→4 1↔2 1↔4 2↔4 3↔4	1→2	1↔4 2↔4 1→3 2→4

Table 4: Missed sensitivity queries of PUSHFORWARD in topological order, in combination with a complete identification algorithm for *directed edges*. Part 2.