SUPPLEMENTARY MATERIAL

A Σ-CG UNDER MARGINALISATION AND CONDITIONING

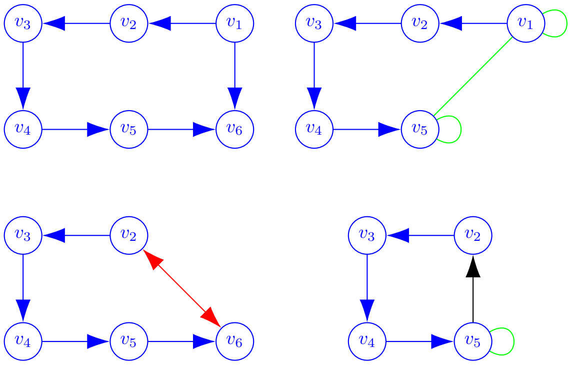

Figure 5: A directed *acyclic* graph G (top left) as the σ-connection graph (σ-CG) with the σ-equivalence classes $\{\{v_1\}, \cdots, \{v_6\}\}$. We have the σ-separation: $\{v_3\} \perp\!\!\!\perp^{\sigma}_{G} \{v_5\} \,|\, \{v_2, v_4, v_6\}$. The combination of marginalizing out v_1 and conditioning on v_6 introduces cycles. Without keeping track of aboves σ-equivalence classes we would not get the corresponding σ-separation: $\{v_3\} \perp\!\!\!\perp^{\sigma}_{G^{\{v_1\}}_{\{v_6\}}} \{v_5\} \,|\, \{v_2, v_4\}$ in the bottom right σ-CG.

Theorem A.1 (σ-Separation under Marginalisation)**.** *Let G be a σ-CG with set of nodes V and $W, X, Y, Z \subseteq V$ subsets with $W = \{w\}$ and $w \notin X \cup Y \cup Z$. Then we have the equivalence:*

$$X \underset{G}{\overset{\sigma}{\perp\!\!\!\perp}} Y \,|\, Z \iff X \underset{G^W}{\overset{\sigma}{\perp\!\!\!\perp}} Y \,|\, Z.$$

Proof. If $\pi = x \cdots y$ is a Z-σ-open path in G then every occurrence of w in π is as a non-collider. If we have $\cdots v \longrightarrow w \multimap \cdots$ in π and $v \notin Z$ then marginalising out w keeps π^W Z-σ-open in G^W. If $v \in Z$ then $v \in \sigma(w)$ by the Z-σ-openess. Since $\sigma(w) \in \mathcal{L}(G)$ is a loop we find elements $v_i \in \sigma(w)$, $r \geq 0$, $i = 1, \dots r$, and a path

$$\cdots v \longleftarrow v_1 \longleftarrow \cdots \longleftarrow v_r \longleftarrow w \multimap \cdots .$$

We do the same replacement on the right hand side of w if necessary. Then marginalising this path w.r.t. W gives a Z-σ-open path in G^W. This shows:

$$X \underset{G^W}{\overset{\sigma}{\perp\!\!\!\perp}} Y \,|\, Z \implies X \underset{G}{\overset{\sigma}{\perp\!\!\!\perp}} Y \,|\, Z.$$

Now let $x \cdots y$ be a Z-σ-open path in G^W. Then every edge lifts to a subpath in G where w only occurs as a non-collider. If a path $\cdots v \longleftarrow z \cdots$ in G^W with $z \in Z \cap \sigma(v)$ in G comes from $\cdots v \longleftarrow w \longleftarrow z \cdots$ or $v \longleftrightarrow w \longrule w \longleftarrow z$ then we can, again since $\sigma(v) \in \mathcal{L}(G)$ is a loop, find nodes $v_i \in \sigma(v)$, $r \geq 0$, $i = 1, \dots r$, and a path in G of the form:

$$\cdots v \longrightarrow v_1 \longrightarrow \cdots \longrightarrow v_r \longrightarrow z \cdots$$

which is in any case Z-σ-open in G (whether v or z are colliders or not). So we can construct a Z-σ-open path in G and we get:

$$X \underset{G}{\overset{\sigma}{\perp\!\!\!\perp}} Y \,|\, Z \implies X \underset{G^W}{\overset{\sigma}{\perp\!\!\!\perp}} Y \,|\, Z.$$

□

Theorem A.2 (σ-Separation under Conditioning)**.** *Let G be a σ-CG with set of nodes V and $C, X, Y, Z \subseteq V$ subsets with $C = \{c\}$ and $c \notin X \cup Y \cup Z$. Then we have the equivalence:*

$$X \underset{G}{\overset{\sigma}{\perp\!\!\!\perp}} Y \,|\, Z \cup C \iff X \underset{G_C}{\overset{\sigma}{\perp\!\!\!\perp}} Y \,|\, Z.$$

Proof. Let $C = \{c\}$. Let $\pi = x \cdots y$ be a $(Z \cup C)$-σ-open path in G with a minimal number of arrowheads pointing to nodes in C. Then at c (if c occurs) there are no undirected edges. So we have the cases:

1. fork: $\cdots v_1 \longleftarrow c \longrightarrow v_2 \cdots$ in G with $\sigma(v_1) = \sigma(c) = \sigma(v_2)$. Then $\cdots v_1 \longleftrightarrow v_2 \cdots$ is in G_C with $\sigma(v_1) = \sigma(v_2) = \sigma(c)$. So the triple situation for v_1, v_2 stays the same in G_C.

2. right chain: $\cdots v_1 \leftrightarrows c \longrightarrow v_2 \cdots$ in G with $\sigma(c) = \sigma(v_2)$. Then $\cdots v_1 \leftrightarrows v_2 \dots$ is in G_C with $\sigma(v_2) = \sigma(c)$. So the triple situation for v_1, v_2 stays the same in G_C.

3. left chain: similar to right chain.

4. collider: $\cdots v_1 \longleftrightarrow c \longleftrightarrow v_2 \cdots$ in G. Then $\cdots v_1 \longleftrightarrow v_2 \cdots$ is in G_C. So the triple situation for v_1, v_2 stays the same in G_C.

5. collider: $\cdots v_1 \longrightarrow c \longleftarrow v_2 \cdots$ in G with $v_1, v_2 \notin Z$. Then $\cdots v_1 \longrule v_2 \cdots$ is in G_C with v_1, v_2 non-collider. So it is Z-open at v_1, v_2.

6. collider: $\cdots \rightleftarrows v_1 \longrightarrow c \leftrightarrows v_2 \cdots$ in G with $v_1 \in Z \cap \sigma(c)$. Since $\sigma(c) \in \mathcal{L}(G)$ is a loop there is a path in G with $w_i \in \sigma(c)$, $r \geq 0$, $i = 1, \dots, r$, of the form:

$$\cdots \rightleftarrows v_1 \longleftarrow w_1 \dashleftarrow w_r \longleftarrow c \leftrightarrows v_2 \cdots ,$$

which then is $(Z \cup C)$-σ-open. So the path

$$\cdots \overset{\longleftrightarrow}{\longleftarrow} v_1 \longleftarrow w_1 \longleftarrow \cdots w_r \overset{\longleftarrow}{\longleftrightarrow} v_2 \cdots$$

is then Z-σ-open in G_C.

7. collider: $\cdots v_1 \longrightarrow c \longleftrightarrow v_2 \cdots$ in G with $v_1 \notin Z$. Then $\cdots v_1 \longrightarrow v_2 \cdots$ is in G_C and Z-open.
8. collider: as before with v_1 and v_2 swapped. Same arguments.

These cover all cases and we have shown:

$$X \underset{G_C}{\overset{\sigma}{\perp\!\!\!\perp}} Y \,|\, Z \implies X \underset{G}{\overset{\sigma}{\perp\!\!\!\perp}} Y \,|\, Z \cup C.$$

Now let $x \cdots y$ be a Z-open path in G_C. Then the rules for conditioning lift every edge in G_C to an edge or triple in G, where the triple situation for c is C-σ-open and where the triple situation for the endnodes stays the same. So it is clearly $(Z \cup C)$-open in G. This shows:

$$X \underset{G}{\overset{\sigma}{\perp\!\!\!\perp}} Y \,|\, Z \cup C \implies X \underset{G_C}{\overset{\sigma}{\perp\!\!\!\perp}} Y \,|\, Z.$$

□

B THE Σ-SEPARATION CRITERION FOR MSCMS

The trick to prove the σ-separation criterion is to transform the σ-connection graph G of an mSCM, which has no undirected edges and can be seen as a directed mixed graph (DMG), into an acyclic directed mixed graph (ADMG) that encodes the same conditional independencies in terms of the well known d-separation. This also shows that every σ-separation-equivalence-class contains an acyclic graph (if one only looks at the observational distributions). Caution: the constructed ADMG is not well-behaved under marginalisation or interventions. We will refer to the d-separation criterion as the *directed global Markov property (dGMP)* and to the σ-separation criterion as the *generalized directed global Markov property (gdGMP)* in the following.

Lemma B.1. *Let $G = (V, E, H)$ be an acyclic directed mixed graph (ADMG) and $(X_v)_{v \in V}$ be random variables that satisfy the dGMP w.r.t. G. Let E_w be a random variable independent of $(X_v)_{v \in V}$ and X_w be another random variable, $w \notin V$, given by a functional relation:*

$$X_w = f\big((X_v)_{v \in P}, E_w\big),$$

where $P \subseteq V$ is a subset of nodes. Let $G' = (V', E', H')$ be the ADMG with set of nodes $V' := V \cup \{w\}$, set of edges $E' := E \cup \{v \longrightarrow w | v \in P\}$ and set of bidirected edges $H' := H$. Then $G \subseteq G'$ is an ancestral sub-ADMG and $(X_v)_{v \in V'}$ satisfies the dGMP w.r.t. G'.

Proof. Since w is a childless node in G' clearly G' is acyclic, $\mathrm{Pa}^{G'}(w) = P$ and $G \subseteq G'$ is an ancestral sub-ADMG. So there exists a topological order $<$ for G' such that w is the last element. Since for an ADMG the directed global Markov property (*dGMP*) is equivalent to the ordered local Markov property (*oLMP*) w.r.t. any topological order (see [12,28]) we only need to check the local independence:

$$\{w\} \underset{\mathbb{P}}{\perp\!\!\!\perp} A \setminus \{w\} \,|\, \partial_{A^{\mathrm{mor}}}(w)$$

for every ancestral $A \subseteq G'$ with $w \in A$. Since $\partial_{A^{\mathrm{mor}}}(w) = \mathrm{Pa}^{G'}(w) = P$ and $A \setminus \{w\} \subseteq V$ the statement follows directly from the implication:

$$\begin{aligned} &E_w \underset{\mathbb{P}}{\perp\!\!\!\perp} (X_v)_{v \in V} \implies \\ &f\big((X_v)_{v \in P}, E_w\big) \underset{\mathbb{P}}{\perp\!\!\!\perp} (X_v)_{v \in A \setminus \{w\}} \,|(X_v)_{v \in P}. \end{aligned}$$

□

Theorem B.2. *Let $G = (V, E, H)$ be a directed mixed graph (DMG) and $\mathcal{S}(G)$ the set of its strongly connected components. Assume that we have:*

1. *random variables $(X_v)_{v \in V}$,*
2. *random variables $(E_v)_{v \in V}$ that jointly satisfy the* dGMP *w.r.t. the bidirected graph $(V, \emptyset, H)$, i.e. for every $W, Y \subseteq V$ we have the implication:*

$$W \underset{(V,\emptyset,H)}{\overset{d}{\perp\!\!\!\perp}} Y \implies (E_v)_{v \in W} \underset{\mathbb{P}}{\perp\!\!\!\perp} (E_v)_{v \in Y},$$

3. *a tuple of functions $(g_S)_{S \in \mathcal{S}(G)}$ indexed by the strongly connected components S of G,*

such that we have the following equations for $S \in \mathcal{S}(G)$:

$$(X_v)_{v \in S} = g_S\big((X_w)_{w \in \mathrm{Pa}^G(S) \setminus S}, (E_w)_{w \in S}\big).$$

Then $(X_v)_{v \in V}$ satisfy the general directed global Markov property (gdGMP) *w.r.t. the DMG G, i.e. for every three subsets $W, Y, Z \subseteq V$ we have the implication:*

$$W \underset{G}{\overset{\sigma}{\perp\!\!\!\perp}} Y \,|\, Z \implies (X_v)_{v \in W} \underset{\mathbb{P}}{\perp\!\!\!\perp} (X_v)_{v \in Y} \,|(X_v)_{v \in Z}.$$

Proof. By assumption we have that $(E_v)_{v \in V}$ satisfies the *dGMP* w.r.t. the ADMG $(V, \emptyset, H)$. By lemma B.1 we can inductively add:

$$X_v = g_{S,v}\big((X_w)_{\mathrm{Pa}^G(S) \setminus S}, (E_w)_{w \in S}\big)$$

for $v \in V$ where $S = \mathrm{Sc}^G(v)$. We then finally get an ADMG G' with nodes $(E_v)_{v\in V}$ and $(X_v)_{v\in V}$ that satisfy the *dGMP* w.r.t. this G'. This implies that for $W, Y, Z \subseteq V$ we have:

$$W \underset{G'}{\overset{d}{\perp\!\!\!\perp}} Y \mid Z \implies (X_v)_{v\in W} \underset{\mathbb{P}}{\perp\!\!\!\perp} (X_v)_{v\in Y} \,|(X_v)_{v\in Z}.$$

It is thus left to show that we also have the implication:

$$W \underset{G}{\overset{\sigma}{\perp\!\!\!\perp}} Y \mid Z \implies W \underset{G'}{\overset{d}{\perp\!\!\!\perp}} Y \mid Z.$$

For this it is enough to show that every Z-d-open path π' from W to Y in G' lifts to a Z-σ-open path π from W to Y in G. The construction is straightforward. For details see [12]. □

Corollary B.3. *The observed variables $(X_v)_{v\in V}$ of any* mSCM $M = (G^+, \mathcal{X}, \mathbb{P}, g)$, $G^+ = (U\dot{\cup}V, E^+)$, *satisfy the σ-separation criterion w.r.t. the induced σ-connection graph (σ-CG) G.*

Proof. For $v \in V$ we put $E_v := (E_u)_{\substack{u\in U\\ v\in \mathrm{Ch}^{G^+}(u)}}$. The $(E_v)_{v\in V}$ then entail the conditional independence relations implied by d-separation of the bidirected graph $(V, \emptyset, H)$. Furthermore, for $S \in \mathcal{S}(G)$ we have equations:

$$X_S = g_S(X_{\mathrm{Pa}^G(S)\setminus S}, E_S).$$

The claim then directly follows from B.2. □

As a motivation for future work on selection bias we state the following direct corollary.

Corollary B.4 (mSCM with context)**.** *Let $M = (G^+, \mathcal{X}, \mathbb{P}, g)$ be a* mSCM *with $G^+ = (U\dot{\cup}V, E^+)$ and $C \subseteq V$ a subset. Let $G_C = (G^+)^U_C$ be the induced σ-CG of M conditioned on C. Then the observed variables $(X_v)_{v\in V\setminus C}$ satisfy the σ-separation criterion w.r.t. G_C and w.r.t. the regular conditional probability distribution $\mathbb{P}^{|X_C=x_C}$ given $X_C = x_C$ (for $\mathbb{P}^{X_C}$-almost-all values $x_C \in \mathcal{X}_C$): For all subsets $W, Y, Z \subseteq V \setminus C$ we have the implication:*

$$W \underset{G_C}{\overset{\sigma}{\perp\!\!\!\perp}} Y \mid Z \implies X_W \underset{\mathbb{P}^{|X_C=x_C}}{\perp\!\!\!\perp} X_Y \mid X_Z.$$

Proof. The lhs is equivalent to $W \perp\!\!\!\perp^{\sigma}_{G} Y \mid Z \cup C$ (see Theorem 2.20, Theorem A.2, resp.) and this implies $X_W \perp\!\!\!\perp_{\mathbb{P}} X_Y \mid X_Z, X_C$ (see Theorem 2.14, Corollary B.3, resp.), which implies the claim on the rhs (for $\mathbb{P}^{X_C}$-almost-all values $x_C \in \mathcal{X}_C$). □

The last corollary can be used as a starting point for conditional independence constraint-based causal discovery in the presence of (unknown) *selection bias* given by the unknown context C and $x_C \in \mathcal{X}_C$ (in addition to non-linear functional relations, cycles and latent confounders etc.).

C NEURAL NETWORKS AS MSCMS

For constructing causal mechanisms we could use any parametric or non-parametric family of functions. Since we want to stay as general as possible and also make use of the practical advantages of parametric models we represent/approximate the structural functions $g_{\{v\}}$, $v \in V$ by *universal approximators*. A well known class of universal approximators are *neural networks* (see e.g. [16]). A neural network is a function that is constructed from several compositions of linear maps and a fixed one-dimensional *activation* function h. A sufficient condition to have the universal approximation property is if one assumes h be continuous, non-polynomial and piecewise differentiable. A further advantage of neural networks is that the *hidden units* (given by composition of functions $z \mapsto h(w^T z + b)$) can be interpreted as intermediate variables of an extended structural causal model. This means that by modelling the hidden units of every $g_{\{v\}}$ explicitely as a node in an extended graph we can restrict—for the analysis purposes here—to this extended setting, where now the functions $g_{\{v\}}$ (the index $\{v\}$ refers to the trivial loop) are of the form:

$$\begin{aligned} & g_{\{v\}}(x_{\mathrm{Pa}^{G^+}(v)\setminus\{v\}}) \\ =\ & h\left(\textstyle\sum_{k\in \mathrm{Pa}^{G^+}(v)\setminus\{v\}} A_{v,k}\cdot x_k + b_v\right), \end{aligned}$$

with weights $A_{v,k}$ and biases b_v.

Further note that introducing or marginalizing intermediate variables will not change the outcome of the σ-separation criterion defined in Definition 2.10, Theorem 2.14, and Theorem 2.20 (also see [12]). So also this part is compatible with our theory.

Theorem C.1. *The conditions for the contractiveness of the iterations scheme from subsection 4.1 are satisfied if the following three points hold:*

1. $\sup_z |h'(z)| \leq C$ *with* $0 < C < \infty$*, and*
2. $A_{v,k} := 0$ *for* $k \notin \mathrm{Pa}^{G^+}(v) \setminus \{v\}$*, and*
3. $||(A_{v,k})_{v,k\in S}|| < \frac{1}{C}$ *for every* non-trivial *loop* $S \subseteq G$*, where* $||\cdot||$ *can be one of the matrix norms:* $||\cdot||_p$*,* $p \geq 1$*, or* $||\cdot||_\infty$*.*

In this case the functions $(g_{\{v\}})_{v\in V}$ will constitute a well-defined mSCM.

Note that we can put $C = 1$ for popular activation functions $h(z)$ like $\tanh(z)$, $\mathrm{ReLU}(z) = \max(0, z)$, $\sigma(z) =$

$\frac{1}{1+\exp(-z)}$, LeakyRelu, SoftPlus$(z) = \ln(1+e^z)$, etc.. Further note that by using one of these activation functions $h(z)$ and $||\cdot|| = ||\cdot||_\infty$ all the conditions are satisfied if we choose the $A_{v,k}$ such that for all $v \in V$:

$$\begin{array}{c} \sum_{k \in \mathrm{Pa}^{G^+}(v)\setminus\{v\}} |A_{v,k}| < 1 \\ \text{and} \quad A_{v,k} := 0 \quad \text{for} \quad k \notin \mathrm{Pa}^{G^+}(v) \setminus \{v\}. \end{array}$$

Furthermore, we can then iterate the whole system for given error value x_U and initialization $x_V^{(0)}$:

$$\begin{aligned} x_V^{(t+1)} &:= (g_{\{v\}})_{v\in V}(x_V^{(t)}, x_U) \\ &= h\left(A_{V^+} \cdot \binom{x_V^{(t)}}{x_U} + b_V\right) \end{aligned}$$

and reach a unique fixed point x_V. This analyis also holds if we have the error variables outside of the activation function as additive noise.

Proof. For a non-trivial loop $S \subseteq G$ we want to show that for every value $x_{\mathrm{Pa}^{G^+}(S)\setminus S}$ and initialization $x_S^{(0)}$ the iteration (using vector and matrix notations):

$$\begin{aligned} & x_S^{(t+1)} := \\ & h\left(A_{\mathrm{Pa}^{G^+}(S)\cup S} \cdot (x_S^{(t)}, x_{\mathrm{Pa}^{G^+}(S)\setminus S})^T + b_S\right) \end{aligned}$$

converges to a unique point x_S (for $t \to \infty$) under the three stated assumptions in the text.

For applying Banach's fixed point theorem we need to show that for every value $x_{\mathrm{Pa}^{G^+}(S)\setminus S}$ we have a bounded partial Jacobian (S a *non-trivial* loop):

$$\sup_{x_S} ||J_S(x_{\mathrm{Pa}^{G^+}(S)\cup S})|| \leq L(x_{\mathrm{Pa}^{G^+}(S)\setminus S}) < 1$$

where $L(x_{\mathrm{Pa}^{G^+}(S)\setminus S})$ is a constant smaller than 1 and $||\cdot||$ is a suitable matrix norm. In our case we have:

$$\begin{aligned} & J_S(x_{\mathrm{Pa}^{G^+}(S)\cup S}) \\ := & \left(\frac{\partial g_{\{v\}}}{\partial x_k}\right)_{v,k\in S} (x_{\mathrm{Pa}^{G^+}(S)\cup S}) \\ = & \nabla_{x_S} h\left(A_{\mathrm{Pa}^{G^+}(S)\cup S} \cdot \begin{pmatrix} x_S \\ x_{\mathrm{Pa}^{G^+}(S)\setminus S} \end{pmatrix} + b_S\right) \\ = & \Big(h'\left(\textstyle\sum_{j\in \mathrm{Pa}^{G^+}(v)\setminus\{v\}} A_{v,j}\cdot x_j + b_v\right) \\ & \quad \cdot A_{v,k} \cdot \mathbb{1}_{k\in \mathrm{Pa}^{G^+}(v)\setminus\{v\}}\Big)_{v,k\in S} \\ = & \operatorname{diag}(h') \cdot (A_S \odot \mathbb{1}_S)\,. \end{aligned}$$

Here $\operatorname{diag}(h')$ refers to the diagonal matrix with the corresponding values of h' and $\mathbb{1}_S$ is the adjacency matrix as indicated on the line above.

If $|h'(z)| \leq C < \infty$ and $||\cdot||$ is either $||\cdot||_p$, $p \geq 1$, or $||\cdot||_\infty$ then $||\operatorname{diag}(h')|| \leq C$. If, furthermore, $||A_S \odot \mathbb{1}_S|| < \frac{1}{C}$ then we get:

$$\begin{aligned} ||J_S|| &\leq ||\operatorname{diag}(h')|| \cdot ||A_S \odot \mathbb{1}_S|| \\ &\leq C \cdot ||A_S \odot \mathbb{1}_S|| =: L \\ &< C \cdot \tfrac{1}{C} \\ &= 1. \end{aligned}$$

Note that we can represent $A \odot \mathbb{1}$ in a single matrix A if we put $A_{v,k} := 0$ whenever $k \notin \mathrm{Pa}^{G^+}(v) \setminus \{v\}$. From the above then follows that the map of the iteration scheme becomes contractive and the series thus converges to a unique fixed point x_S. g_S can then be defined via:

$$g_S(x_{\mathrm{Pa}^{G^+}(S)\setminus S}) := x_S.$$

The system $(g_S)_{S\in\mathcal{L}(G)}$ is also compatible. Indeed, the convergence shows that the above element $(x_{\mathrm{Pa}^{G^+}(S)\setminus S}, x_S)$ simultaneously solves the system $x_v = g_{\{v\}}(x_{\mathrm{Pa}^{G^+}(v)\setminus\{v\}})$, $v \in S$. So for a loop $S' \subseteq S$ the corresponding components $(x_{\mathrm{Pa}^{G^+}(S')\setminus S'}, x_{S'})$ simultaneously solves the system $x_v = g_{\{v\}}(x_{\mathrm{Pa}^{G^+}(v)\setminus\{v\}})$, $v \in S'$. Since also the solution for the loop S' is unique we get:

$$g_{S'}(x_{\mathrm{Pa}^{G^+}(S')\setminus S'}) = x_{S'},$$

which shows the compatibility. The measurability of this map follows from a measurable choice theorem (see [2]) as explained in [3]. □

If we want to uniformly sample weights for the parent nodes one can use the following:

Remark C.2 (See [1])**.** *To uniformly sample from the d-dimensional L_p-ball $B_p^d := \{x \in \mathbb{R}^d : ||x||_p \leq 1\}$ we can sample i.i.d. $y_1, \ldots, y_d \sim p(t) = \frac{1}{2\Gamma(1+\frac{1}{p})} e^{-|t|^p}$, $t \in \mathbb{R}$ and $z \sim p(s) = e^{-s}$, $s \geq 0$. Then $x = \frac{(y_1,\ldots,y_d)^T}{\left(\sum_{j=1}^d |y_j|^p + z\right)^{1/p}}$ is uniformly sampled from B_p^d.*

D MORE DETAILS ON THE ALGORITHM

D.1 SCORING FEATURES

In order to score features, which can be defined as Boolean functions of the causal graph G, we define a modified loss function

$$\begin{aligned} & \mathcal{L}(G, S, f) := \\ & \sum_{(w_j, y_j, Z_j, I_j, \lambda_j)\in S} \lambda_j (\mathbb{1}_{\lambda_j > 0} - \mathbb{1}_{w_j \perp\!\!\!\perp^{\sigma}_{G_{\mathrm{do}(I_j)}} y_j \mid Z_j} \mathbb{1}_{f(G)}) \end{aligned} \tag{4}$$

[21] proposed to score the confidence of a feature with

$$\begin{aligned} C(S, f) := & \min_{G\in\mathbb{G}(V)} \mathcal{L}(G, S, \neg f) \\ & - \min_{G\in\mathbb{G}(V)} \mathcal{L}(G, S, f). \end{aligned} \tag{5}$$

They showed that this scoring method is sound for oracle inputs.

Theorem D.1. *For any feature f, the confidence score $C(S,f)$ of (5) is sound for oracle inputs with infinite weights. In other words, $C(S,f) = \infty$ if f is identifiable from the inputs, $C(S,f) = -\infty$ if $\neg f$ is identifiable from the inputs, and $C(S,f) = 0$ if f is unidentifiable from the inputs.*

Furthermore, they showed that the scoring method is asymptotically consistent under a consistency condition on the statistical independence test.

Theorem D.2. *Assume that the weights are asymptotically consistent, meaning that*

$$\log p_N - \log \alpha_N \overset{P}{\to} \begin{cases} -\infty & H_1 \\ +\infty & H_0, \end{cases} \tag{6}$$

as the number of samples $N \to \infty$, where the null hypothesis H_0 is independence and the alternative hypothesis H_1 is dependence. Then for any feature f, the confidence score $C(S,f)$ of (5) is asymptotically consistent, i.e., $C(S,f) \to \infty$ in probability if f is identifiably true, $C(S,f) \to -\infty$ in probability if f is identifiably false, and $C(S,f) \to 0$ in probability otherwise.

By using the scoring method of [21] as explained above, our algorithm inherits these desirable properties.

D.2 ENCODING IN ANSWER SET PROGRAMMING

In order to test whether a causal graph G entails a certain independence, we create a computation graph of σ-connection graphs. A computation graph of σ-connection graphs is a DAG with σ-connection graphs as nodes, and directed edges that correspond with the operations of conditioning and marginalisation. The “source node” of an encoding DAG is an (intervened) causal graph. The “sink” nodes are σ-connection graphs that consist of only two variables (because all other variables have been conditioned or marginalised out) that can be reached from the source node by applying a sequence of conditioning and marginalisation operations. Testing a σ-separation statement in the intervened causal graph reduces to testing for adjacency in the corresponding sink node.

Since interventions and conditioning do not commute, one has to take care to employ these operations in the right ordering. We define the computation graph in such a way that intervention operations are performed first, followed by marginalisations, and finally conditioning operations. At each stage, we always remove the node with the highest possible label first, which means that our computation graph is actually a computation tree.

Below we provide the source code of the essential part of the algorithm, using the ASP syntax for `clingo 4`. It is based upon the source code provided by [19]. The differences to [19], i.e. of σ-separation vs. d-separation, are indicated with “*(sigma)*” in the comments, i.e. at lines 100, 128, 138. Note that the main difference between the encoding of d-separation and σ-separation is that in the non-collider case (see definition 2.9) we need to check in which strongly connected component $\sigma(v)$ the non-collider node lies in comparison to its adjacent nodes. This boils down to checking ancestral relations. Since the σ-structure is inherited in a trivial fashion during the marginalisation and conditioning operations, it only needs to be found once (namely in the original σ-CG induced by the mSCM).

We used the state-of-the-art ASP solver `clingo 4` [13] in our experiments to run the ASP program.

```
%%%%%%%%%% COMPUTATION GRAPH %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% preliminaries
node(0..nrnodes-1).
set(0..2**nrnodes-1).
ismember(M,Z) :- set(M), node(Z), M & (2**Z) != 0.

% first intervene, then marginalize, then condition;
%   always remove the node with the highest possible label first
intervene(0,Jsub,Z,J,0)   :- node(Z), set(J), set(Jsub),
                             ismember(J,Z), not ismember(Jsub,Z),
                             (Jsub + 2**Z) == J, 2**Z > Jsub.
marginalize(0,J,Msub,Z,M) :- node(Z), set(M), set(J), set(Msub),
                             ismember(M,Z), not ismember(Msub,Z),
                             (Msub + 2**Z) == M, 2**Z > Msub.
condition(Csub,Z,C,J,M)   :- node(Z), set(M), set(C), set(J), set(Csub),
                             (C & M) == 0, ismember(C,Z), not ismember(Csub,Z),
                             (Csub + 2**Z) == C, 2**Z > Csub.

% guess (generate all possible directed mixed graphs)
{ edge(X,Y) } :- node(X), node(Y), X != Y.
{ conf(X,Y) } :- node(X), node(Y), X < Y.

% source node in computation graph
th(X,Y,0,0,0) :- edge(X,Y).
hh(X,Y,0,0,0) :- conf(X,Y).

% ancestral relations after perfect (surgical) intervention on J
ancestor(X,Y,J) :- th(X,Y,0,J,0), X!=Y, node(X), node(Y), set(J).
ancestor(X,Y,J) :- ancestor(X,Z,J), ancestor(Z,Y,J),
                   X!=Y, X!=Z, Y!=Z,
                   node(X), node(Y), node(Z), set(J).


%%%%%%%%%% INTERVENTION %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

tt(X,Y,C,J,M) :- tt(X,Y,C,Jsub,M),
                 X <= Y,
                 not ismember(C,X), not ismember(C,Y),
                 not ismember(M,X), not ismember(M,Y),
                 node(X),node(Y),
                 intervene(C,Jsub,Z,J,M).

th(X,Y,C,J,M) :- th(X,Y,C,Jsub,M),
                 X != Y,
                 not ismember(J,Y),
                 not ismember(C,X), not ismember(C,Y),
                 not ismember(M,X), not ismember(M,Y),
                 node(X),node(Y),
                 intervene(C,Jsub,Z,J,M).

hh(X,Y,C,J,M) :- hh(X,Y,C,Jsub,M),
                 X < Y,
                 not ismember(J,Y), not ismember(J,X),
                 not ismember(C,X), not ismember(C,Y),
                 not ismember(M,X), not ismember(M,Y),
                 node(X),node(Y),
                 intervene(C,Jsub,Z,J,M).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%%%%%%%%%% CONDITIONING %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% X---Y => X---Y
tt(X,Y,C,J,M) :- tt(X,Y,Csub,J,M),
                 X <= Y,
                 not ismember(C,X), not ismember(C,Y),
                 not ismember(M,X), not ismember(M,Y),
                 node(X),node(Y),
                 condition(Csub,Z,C,J,M).

%% X-->Z<--Y => X---Y
tt(X,Y,C,J,M) :- th(X,Z,Csub,J,M),th(Y,Z,Csub,J,M),
                 X <= Y,
                 not ismember(C,X), not ismember(C,Y),
                 not ismember(M,X), not ismember(M,Y),
                 node(X),node(Y),
                 condition(Csub,Z,C,J,M).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%% X-->Y => X-->Y
th(X,Y,C,J,M) :- th(X,Y,Csub,J,M),
                 X != Y,
                 not ismember(C,X), not ismember(C,Y),
                 not ismember(M,X), not ismember(M,Y),
                 node(X),node(Y),
                 condition(Csub,Z,C,J,M).

%% X-->Z<->Y => X-->Y
th(X,Y,C,J,M) :- th(X,Z,Csub,J,M),
                 { hh(Z,Y,Csub,J,M); hh(Y,Z,Csub,J,M) } >= 1,
                 X != Y,
                 not ismember(C,X), not ismember(C,Y),
                 not ismember(M,X), not ismember(M,Y),
                 node(X),node(Y),
                 condition(Csub,Z,C,J,M).

%% X-->Z-->Y (anc of Z) => X-->Y (sigma)
th(X,Y,C,J,M):- th(X,Z,Csub,J,M), th(Z,Y,Csub,J,M),
                 ancestor(Y,Z,J),
                 X != Y,
                 not ismember(C,X), not ismember(C,Y),
                 not ismember(M,X), not ismember(M,Y),
                 node(X),node(Y),node(Z),
                 condition(Csub,Z,C,J,M).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% X<->Y => X<->Y
hh(X,Y,C,J,M) :- hh(X,Y,Csub,J,M),
                 X < Y,
                 not ismember(C,X), not ismember(C,Y),
                 not ismember(M,X), not ismember(M,Y),
                 node(X),node(Y),
                 condition(Csub,Z,C,J,M).

%% X<->Z<->Y => X<->Y
hh(X,Y,C,J,M) :- { hh(Z,X,Csub,J,M); hh(X,Z,Csub,J,M) } >= 1,
                 { hh(Z,Y,Csub,J,M); hh(Y,Z,Csub,J,M) } >= 1,
                 X < Y,
                 not ismember(C,X), not ismember(C,Y),
                 not ismember(M,X), not ismember(M,Y),
                 node(X),node(Y),
                 condition(Csub,Z,C,J,M).

%% X<->Z-->Y (anc of Z)  => X<->Y (sigma)
hh(X,Y,C,J,M) :- { hh(Z,X,Csub,J,M); hh(X,Z,Csub,J,M) } >= 1,
                 th(Z,Y,Csub,J,M),
                 ancestor(Y,Z,J),
                 X < Y,
                 not ismember(C,X), not ismember(C,Y),
                 not ismember(M,X), not ismember(M,Y),
                 node(X),node(Y),node(Z),
                 condition(Csub,Z,C,J,M).

%% (anc of Z) X<--Z-->Y (anc of Z) => X<->Y (sigma)
hh(X,Y,C,J,M) :- th(Z,X,Csub,J,M), th(Z,Y,Csub,J,M),
                 ancestor(X,Z,J),
                 ancestor(Y,Z,J),
                 X < Y,
                 not ismember(C,X), not ismember(C,Y),
                 not ismember(M,X), not ismember(M,Y),
                 node(X),node(Y),node(Z),
                 condition(Csub,Z,C,J,M).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%%%%%%%%%% MARGINALIZATION %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% X---Y => X---Y
tt(X,Y,C,J,M) :- tt(X,Y,C,J,Msub),
                 X <= Y,
                 not ismember(C,X), not ismember(C,Y),
                 not ismember(M,X), not ismember(M,Y),
                 node(X),node(Y),
                 marginalize(C,J,Msub,Z,M).

%% X-->Z---Y => X---Y
tt(X,Y,C,J,M) :- th(X,Z,C,J,Msub),
                 { tt(Z,Y,C,J,Msub); tt(Y,Z,C,J,Msub) } >= 1,
                 X <= Y,
                 not ismember(C,X), not ismember(C,Y),
                 not ismember(M,X), not ismember(M,Y),
                 node(X),node(Y),
                 marginalize(C,J,Msub,Z,M).

%% X---Z<--Y => X---Y
tt(X,Y,C,J,M) :- { tt(X,Z,C,J,Msub); tt(Z,X,C,J,Msub) } >= 1,
                 th(Y,Z,C,J,Msub),
                 X <= Y,
                 not ismember(C,X), not ismember(C,Y),
                 not ismember(M,X), not ismember(M,Y),
                 node(X),node(Y),
                 marginalize(C,J,Msub,Z,M).

%% X---Z---Y => X---Y
tt(X,Y,C,J,M) :- { tt(X,Z,C,J,Msub); tt(Z,X,C,J,Msub) } >= 1,
                 { tt(Z,Y,C,J,Msub); tt(Y,Z,C,J,Msub) } >= 1,
                 X <= Y,
                 not ismember(C,X), not ismember(C,Y),
                 not ismember(M,X), not ismember(M,Y),
                 node(X),node(Y),
                 marginalize(C,J,Msub,Z,M).

%% X-->Z---Z<--Y => X---Y
tt(X,Y,C,J,M) :- th(X,Z,C,J,Msub),th(Y,Z,C,J,Msub),tt(Z,Z,C,J,Msub),
                 X <= Y,
                 not ismember(C,X), not ismember(C,Y),
                 not ismember(M,X), not ismember(M,Y),
                 node(X),node(Y),
                 marginalize(C,J,Msub,Z,M).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% X-->Y => X-->Y
th(X,Y,C,J,M) :- th(X,Y,C,J,Msub),
                 X != Y,
                 not ismember(C,X), not ismember(C,Y),
                 not ismember(M,X), not ismember(M,Y),
                 node(X),node(Y),
                 marginalize(C,J,Msub,Z,M).

%% X-->Z-->Y => X-->Y
th(X,Y,C,J,M) :- th(X,Z,C,J,Msub),th(Z,Y,C,J,Msub),
                 X != Y,
                 not ismember(C,X), not ismember(C,Y),
                 not ismember(M,X), not ismember(M,Y),
                 node(X),node(Y),
                 marginalize(C,J,Msub,Z,M).

%% X---Z<->Y => X-->Y
th(X,Y,C,J,M) :- { tt(X,Z,C,J,Msub); tt(Z,X,C,J,Msub) } >= 1,
                 { hh(Z,Y,C,J,Msub); hh(Y,Z,C,J,Msub) } >= 1,
                 X != Y,
                 not ismember(C,X), not ismember(C,Y),
                 not ismember(M,X), not ismember(M,Y),
                 node(X),node(Y),
                 marginalize(C,J,Msub,Z,M).

%% X---Z-->Y => X-->Y
th(X,Y,C,J,M) :- { tt(X,Z,C,J,Msub); tt(Z,X,C,J,Msub) } >= 1,
                 th(Z,Y,C,J,Msub),
                 X != Y,
                 not ismember(C,X), not ismember(C,Y),
                 not ismember(M,X), not ismember(M,Y),
                 node(X),node(Y),
                 marginalize(C,J,Msub,Z,M).

%% X-->Z---Z<->Y => X-->Y
th(X,Y,C,J,M) :- th(X,Z,C,J,Msub),
                 tt(Z,Z,C,J,Msub),
                 { hh(Z,Y,C,J,Msub); hh(Y,Z,C,J,Msub) } >= 1,
                 X != Y,
                 not ismember(C,X), not ismember(C,Y),
                 not ismember(M,X), not ismember(M,Y),
                 node(X),node(Y),
                 marginalize(C,J,Msub,Z,M).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%% X<->Y => X<->Y
hh(X,Y,C,J,M) :- hh(X,Y,C,J,Msub),
                 X < Y,
                 not ismember(C,X), not ismember(C,Y),
                 not ismember(M,X), not ismember(M,Y),
                 node(X),node(Y),
                 marginalize(C,J,Msub,Z,M).

%% X<->Z-->Y => X<->Y
hh(X,Y,C,J,M) :- { hh(X,Z,C,J,Msub); hh(Z,X,C,J,Msub) } >= 1,
                 th(Z,Y,C,J,Msub),
                 X < Y,
                 not ismember(C,X), not ismember(C,Y),
                 not ismember(M,X), not ismember(M,Y),
                 node(X),node(Y),
                 marginalize(C,J,Msub,Z,M).

%% X<--Z-->Y => X<->Y
hh(X,Y,C,J,M) :- th(Z,X,C,J,Msub),
                 th(Z,Y,C,J,Msub),
                 X < Y,
                 not ismember(C,X), not ismember(C,Y),
                 not ismember(M,X), not ismember(M,Y),
                 node(X),node(Y),
                 marginalize(C,J,Msub,Z,M).

%% X<--Z<->Y => X<->Y
hh(X,Y,C,J,M) :- th(Z,X,C,J,Msub),
                 { hh(Y,Z,C,J,Msub); hh(Z,Y,C,J,Msub) } >= 1,
                 X < Y,
                 not ismember(C,X), not ismember(C,Y),
                 not ismember(M,X), not ismember(M,Y),
                 node(X),node(Y),
                 marginalize(C,J,Msub,Z,M).

%% X<->Z---Z<->Y => X<->Y
hh(X,Y,C,J,M) :- { hh(X,Z,C,J,Msub); hh(Z,X,C,J,Msub) } >= 1,
                 { hh(Y,Z,C,J,Msub); hh(Z,Y,C,J,Msub) } >= 1,
                 tt(Z,Z,C,J,Msub),
                 X < Y,
                 not ismember(C,X), not ismember(C,Y),
                 not ismember(M,X), not ismember(M,Y),
                 node(X),node(Y),
                 marginalize(C,J,Msub,Z,M).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%%%%%%%%% LOSS FUNCTION %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

fail(X,Y,C,J,M,W) :- th(X,Y,C,J,M), indep(X,Y,C,J,M,W), X<Y.
fail(X,Y,C,J,M,W) :- th(Y,X,C,J,M), indep(X,Y,C,J,M,W), X<Y.
fail(X,Y,C,J,M,W) :- hh(X,Y,C,J,M), indep(X,Y,C,J,M,W), X<Y.
fail(X,Y,C,J,M,W) :- tt(X,Y,C,J,M), indep(X,Y,C,J,M,W), X<Y.

fail(X,Y,C,J,M,W) :- not th(X,Y,C,J,M),
                     not th(Y,X,C,J,M),
                     not hh(X,Y,C,J,M),
                     not tt(X,Y,C,J,M),
                     dep(X,Y,C,J,M,W), X<Y.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%%%%%%%%% OPTIMIZATION PROBLEM %%%%%%%%%%%%%%%%%%%%%%%%%%%%%

#minimize{W,X,Y,C,J,M:fail(X,Y,C,J,M,W) }.
```

E EXPERIMENTAL RESULTS

Here we provide additional visualisations of the results of our experiments, for which no space was left in the main paper.

Figure 6 shows ROC curves and PR curves for detecting directed edges (i.e., direct causal relations) and for detecting latent confounders in the causal graph. Results are shown for the purely observational setting ("0 interventions") and for a combination of observational and interventional data ("1–5 interventions") where the targets of the stochastic surgical interventions are single variables chosen randomly, without replacement. Clearly, making use of interventional data is beneficial for causal discovery.

Figure 7 shows similar curves, now for 5 interventions only, but for different encodings: σ-separation (this work), d-separation (allowing for cycles, [19]) and d-separation (acyclic, [19]). Interestingly, the differences between σ-separation and d-separation turn out to be quite small in our simulation setting. The difference is largest for the detection of confounders. On the other hand, the difference between assuming acyclicity and allowing for cycles is much more pronounced, and is also significant for the detection of direct causal relations.

We expect that when going to larger graphs with more variables and with nested loops, the differences between σ-separation and d-separation should increase. However, due to computational restrictions we were not able to perform sufficiently many experiments in this regime to gather enough empirical support for that hypothesis and leave this for future research.

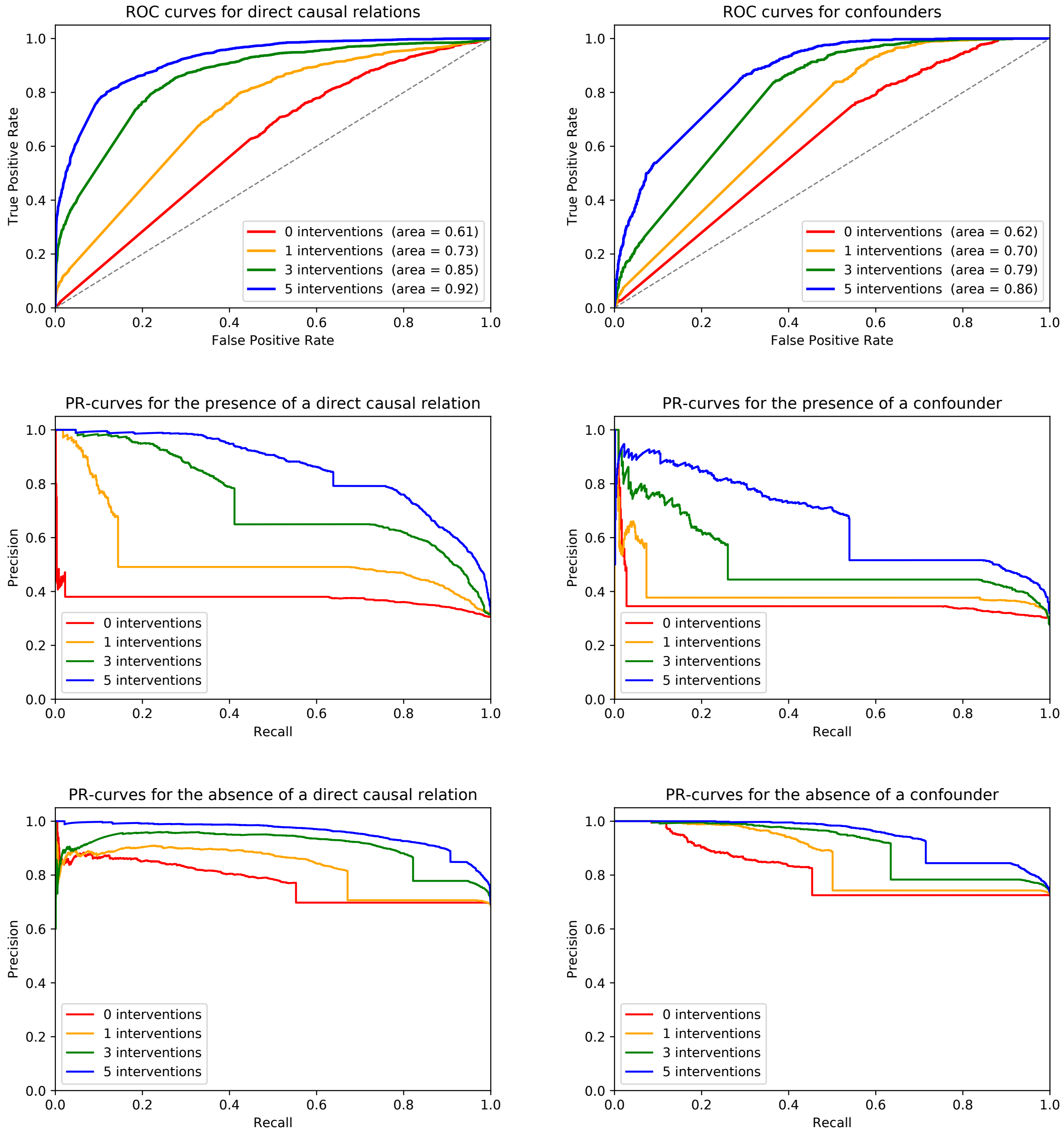

Figure 6: ROC curves (top) and PR curves (center, bottom) for directed edges (left) and confounders (right), for different numbers of single-variable interventions. All results shown here use σ-separation.

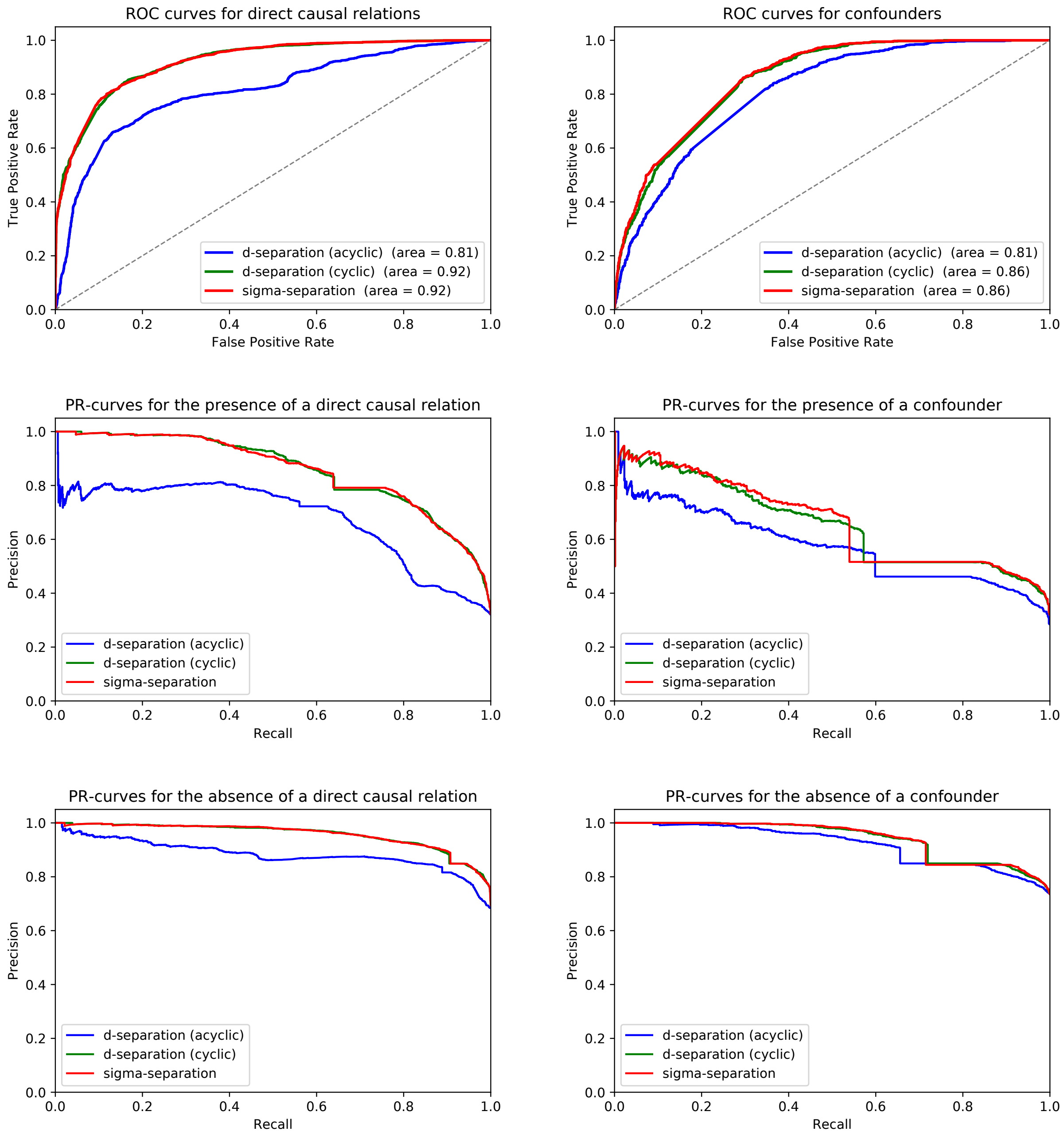

Figure 7: ROC curves (top) and PR curves (center, bottom) for directed edges (left) and confounders (right), for different encodings. All results shown here use observational and 5 interventional data sets.