A Proof of Theorem 3.1

Let $G_1 = (V_1, E_1, o_1)$ and $G_2 = (V_2, E_2, o_2)$ be two labelled graphs, then G_1 and G_2 are isomorphic (i.e.G_1 and G_2 represent the same computation structure.) if and only if their canonical form are identical, i.e. $C(G_1) = C(G_2)$. Note that it means equality between the canonical forms, not isomorphism. Let $C(G_1) = (V_1^C, E_1^C, o_1^C)$ and $C(G_2) = (V_2^C, E_2^C, o_2^C)$, then there exists bijections $\pi_1 : V_1 \rightarrow V_1^C$ and $\pi_2 : V_2 \rightarrow V_2^C$, and we use $\pi_1^{-1} : V_1^C \rightarrow V_1$ and $\pi_2^{-1} : V_2^C \rightarrow V_2$ to denote their inverse functions. Hence, we have $C(G_1) = C(G_2)$ if and only if (1) $o_1^C(i) = o_2^C(i)$ for $\forall i$; and (2) $(i, j) \in E_1^C \Leftrightarrow (i, j) \in E_2^C$ for $\forall i, j$.

Next, we will prove Theorem 3.1 by equivelantly showing that the sequence $(o(\pi^{-1}(1)), p_{\pi^{-1}(1)}), (o(\pi^{-1}(2)), p_{\pi^{-1}(2)}), ..., (o(\pi^{-1}(n)), p_{\pi^{-1}(n)})$ can guarantee the distinctness of canonical forms $C(G)$. For the notation convenience, let function $f(\pi(j), \{\pi(i), (i, j) \in E\}) = \textit{Combine}(\pi(j), \textit{Agg}(\pi(i), (i, j) \in E))$ be the composition of function Agg and Combine, then it is injective if and only if both Agg and Combine are injective. Forthermore, we use $\textit{Seq}_1$ to denote the sequence $(o_1(\pi_1^{-1}(1)), p_{\pi_1^{-1}(1)}), (o_1(\pi_1^{-1}(2)), p_{\pi_1^{-1}(2)}), ..., (o_1(\pi_1^{-1}(n)), p_{\pi_1^{-1}(n)})$, and $\textit{Seq}_2$ to denote the sequence $(oo_2(\pi_2^{-1}(1)), p_{\pi_2^{-1}(1)}), (o_2(\pi_2^{-1}(2)), p_{\pi_2^{-1}(2)}), ..., (o_2(\pi_2^{-1}(n)), p_{\pi_2^{-1}(n)})$.

So far, we know $C(G_1) \neq C(G_2) \Leftrightarrow$ there (1) exist i such that $o_1^C(i) \neq o_2^C(i)$, or (2) exist i, j such that $(i, j) \in E_1^C$ but $(i, j) \notin E_2^C$ (equivalently, $(i, j) \notin E_1^C$ but $(i, j) \in E_2^C$).

Now, let's prove $C(G_1) \neq C(G_2) \Rightarrow \textit{Seq}_1 \neq \textit{Seq}_2$.

- **(1) For the first case**, π_1, π_2 are the bijections that map G_1 and G_2 to their canonical forms, then we have:

$$\begin{aligned} o_1(\pi_1^{-1}(i)) &= o_1^C(\pi_1(\pi_1^{-1}(i))) \\ &= o_1^C(i) \\ o_2(\pi_2^{-1}(i)) &= o_2^C(\pi_2(\pi_2^{-1}(i))) \\ &= o_2^C(i) \end{aligned}$$

  Since $o_1^C(i) \neq o_2^C(i)$, then we get $o_1(\pi_1^{-1}(i)) \neq o_2(\pi_2^{-1}(i))$. indicating that $\textit{Seq}_1 \neq \textit{Seq}_2$.
- **(2) For the second case**, according to the definition of canonical form, we know that $(\pi_1^{-1}(i), \pi_1^{-1}(j)) \in E_1 \Leftrightarrow (i, j) \in E_1^C$ (similarly,$(\pi_2^{-1}(i), \pi_2^{-1}(j)) \in E_2 \Leftrightarrow (i, j) \in E_2^C$). As such, we get:

$$\begin{aligned} p_{\pi_1^{-1}(j)} &= f(\pi_1(\pi_1^{-1}(j)), \{\pi_1(\pi_1^{-1}(s)), (\pi_1^{-1}(s), \pi_1^{-1}(j)) \in E_1\}) \\ &= f(j, \{s, (s, j) \in E_1^C\}) \\ p_{\pi_2^{-1}(j)} &= f(\pi_2(\pi_2^{-1}(j)), \{\pi_2(\pi_2^{-1}(s)), (\pi_2^{-1}(s), \pi_2^{-1}(j)) \in E_1\}) \\ &= f(j, \{s, (s, j) \in E_2^C\}) \end{aligned}$$

  Then, since $(i, j) \in E_1^C$ but $(i, j) \notin E_2^C$, we have $\{s, (s, j) \in E_1^C\} \neq \{s, (s, j) \in E_2^C\}$. Since function f is injective, then we have $p_{\pi_1^{-1}(j)} \neq p_{\pi_2^{-1}(j)}$. Hence, $\textit{Seq}_1 \neq \textit{Seq}_2$

In the end, let's prove the other direction, i.e. $\textit{Seq}_1 \neq \textit{Seq}_2 \Rightarrow C(G_1) \neq C(G_2)$. When $\textit{Seq}_1 \neq \textit{Seq}_2$, there must (1) exist i such that $o_1(\pi_1^{-1}(i)) \neq o_2(\pi_2^{-1}(i))$, or (2) exist j such that $p_{\pi_1^{-1}(j)} \neq p_{\pi_2^{-1}(j)}$.

- **(1) For the first case**, as previous analysis, we have

$$\begin{aligned} o_1^C(i) &= o_1(\pi_1^{-1}(i)) \\ o_2^C(i) &= o_2(\pi_2^{-1}(i)) \end{aligned}$$

  Hence, we can get $o_1^C(i) \neq o_2^C(i)$, which indicates $C(G_1) \neq C(G_2)$.
- **(2) For the second case**, according to previous analysis, we know that:

$$\begin{aligned} p_{\pi_1^{-1}(j)} &= f(j, \{s, (s, j) \in E_1^C\}) \\ p_{\pi_2^{-1}(j)} &= f(j, \{s, (s, j) \in E_2^C\}) \end{aligned}$$

Since f is injective, $p_{\pi_1^{-1}(j)} \neq p_{\pi_2^{-1}(j)}$ implies that $\{s, (s,j) \in E_1^C\} \neq \{s, (s,j) \in E_2^C\}$, which indicates that there exists i such that $(i,j) \in E_1^C$ but $(i,j) \notin E_2^C$ (or $(i,j) \notin E_1^C$ but $(i,j) \in E_2^C$). Henceforth, we get $C(G_1) \neq C(G_2)$.

B Mask Matrix

Here we provide two potential ways to get the mask matrix in PACE. Following the same notation as the main paper, we use $C(G) = (V^C, E^C, o^C)$ to denote the canonical form of the input DAG G.

DFS Algorithm This algorithm takes the canonical form $C(G)$ as input and performs the DFS algorithm on the graph to explore all the nodes of the graph. Before we start the deep first search, we traverse all edges in E^C to find the direct-successors of each node i, and then put them in a set $S(i)$. Then, for each node i, we perform the DFS to get a dependent set *D*(i, and we have $M_{i,j} =$ *False* if and only if $j \in$ *D*(i).

Algorithm 1 DFS Algorithm

```
 1: Initialization: $D(i) = \{\}$; $Visited = [False\ for\ i \in V^C]$; a source (start) node $i$, $T = [i]$ (T is a
    stack).
 2: $Visited[i] = True$
 3: while $|T| > 0$ do
 4:     $j = T[-1]$
 5:     delete $j$ from $T$
 6:     for $k$ in $S(j)$ do
 7:         if $Visited[k] = Flase$ then
 8:             put $k$ in $D(i)$
 9:             $Visited[k] = True$
10:             put $k$ in $T$
11:         end if
12:     end for
13: end while
```

Floyd Algorithm The Floyd algorithm is originally proposed to for finding shortest paths in directed weighted graphs. Here, we initialize the edge weights to be 1, and implement the Floyd algorithm to find the distance *dist*(i,j) (i.e. length of the shortest directed path) between each node pair i, j in $C(G)$. Then we have $M_{i,j} =$ *False* if and only if *dist*$(i,j) > 0$.

Algorithm 2 Floyd Algorithm

```
    Initialization: $dist(i,j) = 1$ if $(i,j) \in E^C$ else 0
 2: for i $\in V^C$ do
        for j $\in V^C$ do
 4:         for k $\in V^C$ do
                if $dist(j,k) > dist(j,i) + dist(i,k)$ then
 6:                 $dist(j,k) = dist(j,i) + dist(i,k)$
                end if
 8:         end for
        end for
10: end for
```

C Multi-head Self-Attention Mechanism

Here we introduce the multi-head (masked) self-attention attention mechanism in the Transformer encoder blocks of PACE. For notation convenience, we use H_k to denote the output representation of the kth Transformer encoder block, and use H_0 to denote the input (i.e. the representation of the sequence generated by dag2seq) to the first Transformer encoder block. Furthermore, we denote the number of heads in the self-attention mechanism as h, and the embedding dimension (of each item in the sequence) as d. Then the Transformer encoder blocks update representation H_k as following.

$$H_k^j = \mathit{softmax}(\frac{Q_k^j(K_k^j)^T}{d})V_k^j \quad \mathit{for}\ j = 1, 2, ...h \tag{5}$$

$$H_{k+1} = \mathit{feed\text{-}forward}(\|_{j=1}^h H_k^j) \tag{6}$$

where $Q_k^j = H_k W_{k,q}^j$, $K_k^j = H_k W_{k,k}^j$, $V_k^j = H_k W_{k,v}^j$ are the query matrix, key matrix, value matrix, respectively (i.e. $W_{k,q}^j, W_{k,k}^j, W_{k,v}^j$ are trainable parameter matrices); $\|$ represents the concatenation operation; Feed-forward is a one-layer MLP. When we introduce the mask operation into the Transformer encoder block. let M be the mask matrix from the Floyd algorithm or the BFS algorithm, then we use following equation to replace equation 5 in the Transformer encoder block.

$$H_k^j = \mathit{softmax}(\frac{Q_k^j(K_k^j)^T + -\infty * M}{d})V_k^j \quad \mathit{for}\ j = 1, 2, ...h \tag{7}$$

D More Details about PACE in the VAE Architecture

In the section, we describe the decoder of PACE-VAE. Figure 4 illustrates the overall architecture. In the main paper, we have introduced how PACE maps input DAGs to the latent space, here we focus on the decoder of PACE-VAE.

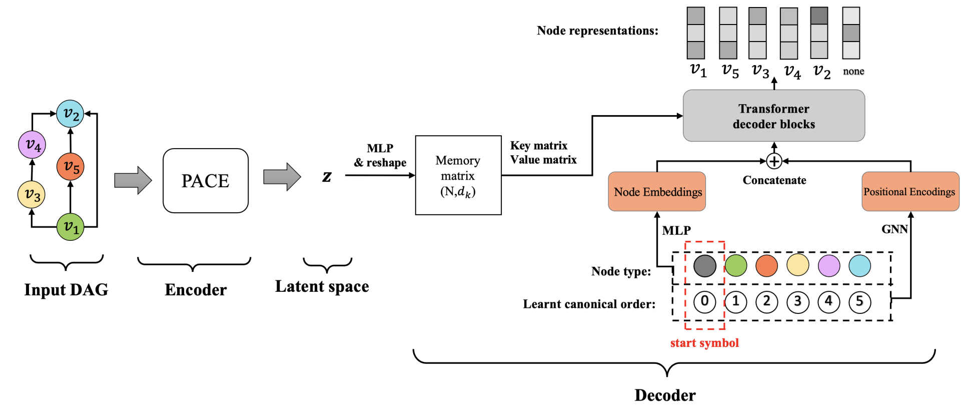

Figure 4: The illustration of PACE in the VAE architecture (PACE-VAE)

Similar to PACE, the decoder is constructed upon the Transformer decoder block. Each Transformer decoder block consists of a masked multi-head self-attention layer (i.e. Euqtion 7), a multi-head attention layer (i.e. Equation 5 except that the key matrix and value matrix are computed from points z in the latent space), and a feed-forward layer (i.e. Equation 6). The decoder takes a MLP as the embedding layer to generate node type embeddings as PACE. In analogous to the dag2seq framework in PACE, the decoder also uses a GNN to generate the positional encoding based on the learnt canonical order of nodes. Then the node embeddings and positional encodings are concatenated and then fed into multiple consecutive Transformer decoder blocks to predict the node representations, which is used to predict the node types and the existence of edges. In analogous to the standard Transformer decoder, the decoder performs the shift right trick (i.e. the ith outputed node representation corresponds to the $i+1$th node in the sequence) and adds a start symbol node (i.e. the black node in Figure 4) at the beginning of the node sequence. Specifically, the canonical label of the start symbol node is different from any possible canonical label in the dag2seq framework to distinguish it's position. For instance, DAG in the searching space contains at most N nodes, then the canonical order of the start symbol node can be 0 or $N+1$. Let o_i denote the output representation of node i in the sequence, then it is used to predict the type of node $i+1$ in the sequence through a MLP. Similarly, for any $j < i$, we use another MLP, which takes the concatenation of o_j and o_i as input, to predict the existence of an directed edge from node $j+1$ to node $i+1$ in the sequence. Note that the canonical order can be generated from the topological sort by breaking ties using canonicalization tools, such as Nauty. Thus, for each node i in the sequence, any dependent node j of this node must be arranged in a prior position in the sequence (i.e. $j < i$). In the end, based on these predictions (node representations), we can perform the teacher forcing to train the VAE.

E Visualization of Detected Optimal architectures on NA and BN

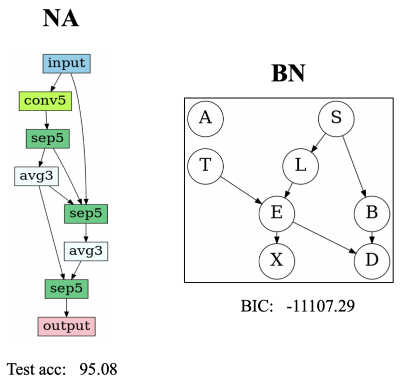

Figure 5: Best architectures on NA and BN detected by PACE.

In this section, we visualize the optimal architectures detected by Bayesian optimization (over the latent DAG encoding space generated by PACE) on datasets NA and BN. Figure 5 illustrates our results. On dataset BN, we find that the detected optimal Bayesian network structure is almost the same as the ground truth (Figure 2 of (Lauritzen & Spiegelhalter, 1988)). In the ground truth, there is another directed edge from node A (visit to Asia ?) to node T (Tuberculosis).

F Reconstruction accuracy and generation performance comparison

Table 5: Recon. accuracy, valid prior, uniqueness, novelty and overall (ave) performance %

	NA					BN				
Methods	Accuracy ↑	Valid ↑	Unique ↑	Novel ↑	**Overall ↑**	Accuracy ↑	Valid ↑	Unique ↑	Novel ↑	**Overall ↑**
PACE	99.97	98.16	57.77	100.00	**88.98**	99.99	99.96	45.10	98.50	**85.88**
DAGNN	99.97	99.98	37.36	100.00	84.33	99.96	99.89	37.61	98.16	83.91
D-VAE	99.96	100.00	37.26	100.00	84.31	99.94	98.84	38.98	98.01	83.94
S-VAE	99.98	100.00	37.03	99.99	84.25	99.99	100.00	35.51	99.70	83.80
GraphRNN	99.85	99.84	29.77	100.00	82.37	96.71	100.00	27.30	98.57	80.65
GCN	5.42	99.37	41.48	100.00	61.57	99.07	99.89	30.53	98.26	81.94

Models parameterized with neural networks contribute to the inductive biases of the deep generative models (Zhang et al., 2016; Keskar et al., 2017), thus the quality of the DAG encoder can be characterized by the reconstruction accuracy (Accuracy) and the generation performance (i.e. the proportions of valid/ unique/ novel architectures in generated DAGs.) of the corresponding VAE.

The reconstruction accuracy, prior validity, uniqueness and novelty are calculated in the same way as Zhang et al. (2019). Empirical results are presented in Table 5, and we take the average of these four measurements to characterize the overall performance of the deep generative model (i.e. VAE), which also measures the quality of the DAG encoder. We find that PACE performs similarly well in reconstruction accuracy, prior validity and novelty with D-VAE, DAGNN and S-VAE, while significantly improving the uniqueness. Hence, PACE achieves the best overall performance and generates more diverse DAG architectures.