LLM-enhanced Score Function Evolution for Causal Structure Learning

Zidong Wang[1] , **Fei Liu**[1] , **Qi Feng**[2] , **Qingfu Zhang**[1,*] and **Xiaoguang Gao**[2]

[1]Department of Computer Science, City University of Hong Kong, Hong Kong
[2]School of Electronic and Information, Northwestern Polytechnical University, Xi'an, China

{zidowang, qingfu.zhang}@cityu.edu.hk, fq19990906@mail.nwpu.edu.cn,
fliu36-c@my.cityu.edu.hk, cxg2012@nwpu.edu.cn.

1 Frameworks

1.1 Prompts design in L-SFE

The **Code Snippets** of the initialization under discrete dataset [1]:

```python
import numpy as np
import pandas as pd
def subscoreLLM(child, parents, Data):
    # child: an index;
    # parents: a list of index, it is [] when child is a root node;
    # Data: a m*n numpy, where m is the number of instances, and n is the numer of
    ↪  variables.

    lambda_value = 1
    var_states = np.unique(Data[:,child])
    var_cardinality = len(var_states)

    if len(parents) != 0:
        num_parents_states = np.prod([len(np.unique(Data[:,var])) for var in parents])
        state_counts = pd.crosstab(index=[Data[:,child]], columns=[Data[:,var] for var in
        ↪  parents])
    else:
        num_parents_states = 1
        _, state_counts = np.unique(Data[:, child], return_counts=True)
    sample_size = Data.shape[0]

    counts = np.asarray(state_counts)
    log_likelihoods = np.zeros_like(counts, dtype=float)

    # Compute the log-counts
    np.log(counts, out=log_likelihoods, where=counts > 0)

    # Compute the log-conditional sample size
    log_conditionals = np.sum(counts, axis=0, dtype=float)
    if isinstance(log_conditionals, np.ndarray):
        np.log(log_conditionals, out=log_conditionals, where=log_conditionals > 0)
    else:
        log_conditionals = [np.log(log_conditionals)]

    # Compute the log-likelihoods
    log_likelihoods -= log_conditionals
    log_likelihoods *= counts

    score = np.sum(log_likelihoods)
```

[1]https://github.com/pgmpy/pgmpy

```
    score -= 0.5 * lambda_value * log(sample_size) * num_parents_states * (var_cardinality
    ↪  - 1)

    return score
```

The **Code Snippets** of the initialization under continuous dataset [2]:

```
import numpy as np
def subscoreLLM(child, parents, Data):
    # child: an index;
    # parents: a list of index, it is [] when child is a root node;
    # Data: a m*n numpy, where m is the number of instances, and n is the numer of
    ↪  variables.

    m = Data.shape[0]
    lambda_value = 2
    cov = np.cov(Data.T)

    if len(parents) == 0:
        H = np.log(cov[child, child])
        score = -1 * (m * H)
        return score

    yX = cov[np.ix_([child], parents)]
    XX = cov[np.ix_(parents, parents)]
    H = np.log(cov[child, child] - yX @ np.linalg.inv(XX) @ yX.T)

    score = -1 * (m * H + np.log(m) * len(parents) * lambda_value * 0.5)
    score = score.item()
    return score
```

The **Task Assignment** of the initialization:

(i) Understand the BIC Score: The demo uses the Bayesian Information Criterion (BIC) score for the subgraph. Make sure you understand how it works.

(ii) Modify to Prevent Overfitting: Think about how to adjust the algorithm to reduce overfitting and improve the graphical accuracy of the results. You can experiment with different parameters or try a new fitting approach. Write a brief summary of your idea in one sentence and add it as a comment at the start of the code.

(iii) Implement Your Solution: Write the code to apply your changes. Ensure the function name, input, and output are the same as in the demo code. Only provide the code—no explanations needed.

The **Task Assignment** of the crossover:

(i) Read the code comments: Understand how each of the two algorithms calculates the score for the subgraph.

(ii) Review the code: Analyze the reasons behind the good or poor performance of the two algorithms.

(iii) Design a new algorithm: Think of a new algorithm that combines ideas from both, improving their performance. Summarize your idea in one sentence and add it to the beginning of the code comments.

(iv) Implement your solution: Write the code to implement your new idea. Ensure the function name, input, and output match the demo code. Provide only the code—no explanations are needed.

The **Task Assignment** of the mutation:

(i) Read the code comments: Understand how the algorithm calculates the score for the subgraph.

(ii) Review the code: Analyze why the algorithm performs well or poorly.

(iii) Improve the algorithm: Think of ways to modify the algorithm for better performance. / Think about how to adjust parameters for better performance. Summarize your idea in one sentence and add it to the beginning of the code comments.

(iv) Implement your solution: Modify the code according to your idea. Ensure that the function name, input, and output remain the same as in the demo code. Provide only the code—no explanations are needed.

[2] https://github.com/py-why/causal-learn

The **Task Assignment** of the injection:

(i) Read the code comments: Understand how the current algorithm calculates the score for the subgraph.

(ii) Design a new algorithm: Think of a new approach that is different from the demo in both structure and concept. Write a one-sentence summary of your idea and add it to the beginning of the code comments.

(iii) Implement your solution: Write the code to implement your new idea. Make sure the function name, input, and output match the demo code. Provide only the code—no explanations are needed.

1.2 Pseudocode of L-SFE

Algorithm 1 L-SFE

Input: Training datasets $\mathbb{D} = \bigcup_{i=1}^{n_p} \mathcal{D}_i$, ground-of-truth graphs $\mathbb{G} = \bigcup_{i=1}^{n_p} \mathcal{G}_i$, training method GLS$(.)$ with its default parameters Θ.
Output: An optimal score function Ψ^*.
Parameter: Mutation, crossover and injection rate p_u, p_o, p_e, the max iteration n_t of EA, the max number of algorithms n_a in population.

$\mathbf{\Psi} = \varnothing$
while $i < n_a$ **do**
  $\Psi_i = \text{LLM}(pt_{ini})$
  Ψ_i.GetFitness$(\mathbb{D}, \mathbb{G}, \text{GLS}, \Theta)$
  $\mathbf{\Psi} = \mathbf{\Psi} \cup \Psi_i$
end while
while $iter < n_t$ **do**
  for $j < n_a * (p_e + p_o + p_u)$ **do**
    Generate corresponding prompts pt according to the selected operator, randomly sample the required demo algorithms $\mathbf{\Psi}_{sub}$
    $\Psi_j = \text{LLM}(pt, \mathbf{\Psi}_{sub})$
    Ψ_j.GetFitness$(\mathbb{D}, \mathbb{G}, \text{GLS}, \Theta)$
    $\mathbf{\Psi} = \mathbf{\Psi} \cup \Psi_j$
  end for
  Resort $\mathbf{\Psi}$ in descending order according to fitness, and select the top n_a elements as the new $\mathbf{\Psi}$
end while
return the first algorithm in $\mathbf{\Psi}$ =0

Algorithm 2 Get Fitness of Ψ

Input: Score function Ψ, training datasets $\mathbb{D}$, ground-of-truth graphs $\mathbb{G}$, training method GLS$(.)$ with default parameters Θ
Output: The fitness of Ψ

if Ψ is valid **then**
  while $k < n_p$ **do**
    $\mathcal{G}^{\dagger}_{\Psi,\mathcal{D}^k} = \text{GLS}(\mathcal{D}^k, \Psi, \Theta)$
  end while
  compute $\mathcal{L}(\Psi)$ based on eq.9 with $(\mathbb{G}, \bigcup_{k=1}^{n_p} \mathcal{G}^{\dagger}_{\Psi,\mathcal{D}^k})$
  return $\mathcal{L}(\Psi)$
else
  return -Inf
end if=0

1.3 Score functions learned by L-SFE

We only enumerate the best score functions L-SFE$^a_{b_*}$, $a \in a \in \{D, C\}$, $b \in \{1, \cdots, n_l\}$ learned in every repetition.

```python
import numpy as np
import pandas as pd

# This algorithm combines Bayesian priors for smoothing with a dual complexity penalty that accounts for both the linear and non-linear effects of model complexity, improving regularization and parameter estimation.\n

def subscoreLLM(child, parents, Data):
    alpha_prior = 0.75  # Prior for smoothing, balanced between the previous algorithms\n
    lambda_linear = 0.5
    lambda_non_linear = 0.5
    var_states = np.unique(Data[:, child])
    var_cardinality = len(var_states)

    if len(parents) != 0:
        num_parents_states = np.prod([len(np.unique(Data[:, var])) for var in parents])
        state_counts = pd.crosstab(index=[Data[:, child]], columns=[Data[:, var] for var in parents])
    else:
        num_parents_states = 1
        state_counts = np.unique(Data[:, child], return_counts=True)[1]

    sample_size = Data.shape[0]

    counts = np.asarray(state_counts, dtype=float) + alpha_prior
    log_likelihoods = np.zeros_like(counts, dtype=float)

    # Compute the log-counts
    np.log(counts, out=log_likelihoods, where=counts > 0)

    # Compute the log-conditional sample size
    log_conditionals = np.sum(counts, axis=0, dtype=float)
    if isinstance(log_conditionals, np.ndarray):
        np.log(log_conditionals, out=log_conditionals, where=log_conditionals > 0)
    else:
       log_conditionals = [np.log(log_conditionals)]

    # Compute the log-likelihoods
    log_likelihoods -= log_conditionals
    log_likelihoods *= counts

    score = np.sum(log_likelihoods)

    # Combined complexity penalty based on linear and non-linear terms
    linear_complexity_penalty = lambda_linear * (num_parents_states * (var_cardinality - 1)) / (sample_size + 1e-10)
    non_linear_complexity_penalty = lambda_non_linear * ((num_parents_states * (var_cardinality - 1))**2) / (sample_size + 1e-5)
    score -= (linear_complexity_penalty + non_linear_complexity_penalty)

    return score
```

Listing 1: Core idea with code implementation of L-SFE$^{D}_{1*}$

This can be succinctly expressed in mathematical form as Eq.6 in Section 4.2. It also has a more detailed form

$$\Psi(\mathcal{G}|\mathcal{D}) = \sum_{i=1}^{n}\sum_{j=1}^{q_i}\sum_{k=1}^{r_r}(N_{ijk}+\alpha)\log\frac{N_{ijk}+\alpha}{N_{ij}+\alpha*r_i} - \frac{\beta}{m+\epsilon}*(r_i-1)*q_i - \frac{\gamma}{m+\epsilon^2}*((r_i-1)*q_i)^2, \tag{1}$$

where N_{ijk} represents the number of samples in $\mathcal{D}$, and $\sum_{k=1}^{r_i} N_{ijk} = N_{ij}$ holds. q_i and r_i denote the number of states in $Pa^{\mathcal{G}_i}$ and X_i, respectively. And the parameters are set as $\alpha = 0.75, \beta = 0.5, \gamma = 0.5, \epsilon = 1e^{-5}$ in $\mathrm{L-SFE}^{\mathrm{D}}_1$. According to Theorem 1, L-SFE$^{D}_{1*}$ is still consistent but not equivalence Here, we briefly sketch the proof.

Proof. First, consider the consistency. As Dirichlet prior parameter is very small, the influence of it becomes negligible when $m \to \infty$, and the MAP converges to the MLE. Therefore, Equation 1 can be rewritten as

$$\begin{aligned}\Psi(\mathcal{G}|\mathcal{D}) =& m * \sum_{i=1}^{n} I_{\hat{P}}(X_i; Pa_i^{\mathcal{G}}) - m * \sum_{i=1}^{n} H_{\hat{P}}(X_i) \\ &- \frac{1}{2m}(\beta|\mathcal{G}|_d + \gamma|\mathcal{G}|_d{}^2),\end{aligned} \tag{2}$$

where $I(.)$ and $H(.)$ represent the multi-information and entropy, respectively. Therefore, $\forall \mathcal{G} \in \boldsymbol{\mathcal{G}}$, the score difference between $\mathcal{G}$ and $\mathcal{G}^*$ is given by

$$\begin{aligned}&\Psi(\mathcal{G}|\mathcal{D}) - \Psi(\mathcal{G}^*|\mathcal{D}) = m * \Delta^l - \tfrac{1}{m} * \Delta^p \\ &\Delta^l = \sum_{i=1}^{n} I_{\hat{P}}(X_i; Pa_i^{\mathcal{G}}) - \sum_{i=1}^{n} I_{\hat{P}}(X_i; Pa_i^{\mathcal{G}^*}) \\ &\Delta^p = \tfrac{1}{2}(\beta(|\mathcal{G}|_d - |\mathcal{G}^*|_d) + \gamma(|\mathcal{G}|_d{}^2 - |\mathcal{G}|_d{}^2)).\end{aligned} \tag{3}$$

If $\mathcal{G}$ admit more independence relations than the $\mathcal{G}^*$, then

$$\sum_{i=1}^{n} I_{P^*}(X_i; Pa_i^{\mathcal{G}}) < \sum_{i=1}^{n} I_{P^*}(X_i; Pa_i^{\mathcal{G}^*}) \tag{4}$$

holds, with $m \to \infty, \hat{P} \to P^*$, which indicate that $\Delta^l < 0$. As m grows faster than $\frac{1}{m}$, thus, $\Psi(\mathcal{G}|\mathcal{D}) < \Psi(\mathcal{G}^*|\mathcal{D})$ holds. If $\mathcal{G}$ admit less independence relations than the $\mathcal{G}^*$, then

$$\sum_{i=1}^{n} I_{P^*}(X_i; Pa_i^{\mathcal{G}}) - \sum_{i=1}^{n} I_{P^*}(X_i; Pa_i^{\mathcal{G}^*}) \approx 0, \tag{5}$$

which implies $\Delta^l \to 0$, but as $|\mathcal{G}|_d > |\mathcal{G}^*|_d$, it follows that $\Delta^p > 0$. Consequently, $\Psi(\mathcal{G}|\mathcal{D}) < \Psi(\mathcal{G}^*|\mathcal{D})$ holds. Therefore, Equation 1 remains consistent.

Next, we provide a counterexample to prove that it is not equivalent. Consider two I-equivalent DAGs: $\mathcal{G}_1 : X \to Y$ and $\mathcal{G}_2 : Y \to X$, and the dataset $\mathcal{D} = \{(x^0, y^1), (x^0, y^1), (x^1, y^1), (x^1, y^0)\}$, then:

$$\begin{aligned}&\Psi(\mathcal{G}_1|\mathcal{D}) = \Psi(X, \varnothing|\mathcal{D}) + \Psi(Y, X|\mathcal{D}) \\ &= \sum_{t=0,1} (N_{X=x^t} + \alpha) \log \tfrac{(N_{X=x^t} + \alpha)}{(N_X + |X| * \alpha)} \\ &+ \sum_{k=0,1} \sum_{t=0,1} (N_{Y=y^t|X=x^k} + \alpha) \log \tfrac{(N_{Y=y^t|X=x^k} + \alpha)}{(N_{X=x^k} + |Y| * \alpha)} \\ &-\tfrac{1}{4} \approx -3.9 \\ &\Psi(\mathcal{G}_2|\mathcal{D}) = \Psi(Y, \varnothing|\mathcal{D}) + \Psi(X, Y|\mathcal{D}) \approx -3.6.\end{aligned} \tag{6}$$

Obviously, Equation 1 is not equivalent. □

```python
import numpy as np
import pandas as pd

# This new algorithm combines Bayesian approach with an optimized Dirichlet prior and a
↪  dynamic complexity penalty based on the sample size.\n

def subscoreLLM(child, parents, Data):
    alpha = 0.7  # Optimized Dirichlet prior parameter for enhanced performance
    var_states = np.unique(Data[:, child])
    var_cardinality = len(var_states)

    if len(parents) != 0:
        parent_states = [len(np.unique(Data[:, var])) for var in parents]
        num_parents_states = np.prod(parent_states)
        state_counts = pd.crosstab(index=[Data[:, child]], columns=[Data[:, var] for var in
        ↪  parents])
    else:
        num_parents_states = 1
        _, state_counts = np.unique(Data[:, child], return_counts=True)
```

```
    sample_size = Data.shape[0]

    counts = np.asarray(state_counts) + alpha  # Apply Dirichlet prior
    log_likelihoods = np.zeros_like(counts, dtype=float)

    # Compute the log-counts
    np.log(counts, out=log_likelihoods)

    # Compute the log-conditional sample size
    log_conditionals = np.sum(counts, axis=0, dtype=float)
    if isinstance(log_conditionals, np.ndarray):
        np.log(log_conditionals, out=log_conditionals)
    else:
        log_conditionals = [np.log(log_conditionals)]

    # Compute the log-likelihoods
    log_likelihoods -= log_conditionals
    log_likelihoods *= counts
    score = np.sum(log_likelihoods)

    # Calculate the number of parameters estimated and apply a dynamic complexity penalty
    num_parameters = (var_cardinality - 1) * num_parents_states
    complexity_penalty = (0.6 * alpha * num_parameters) / max(1, sample_size)  # Dynamic
    ↪  penalty based on sample size
    score -= complexity_penalty

    return score
```

Listing 2: Core idea with code implementation of L-SFE$_{2*}^{D}$

It can be expressed in the mathematical form as

$$\Psi(\mathcal{G}|\mathcal{D}) = \sum_{i=1}^{n}\sum_{j=1}^{q_i}\sum_{k=1}^{r_r}(N_{ijk}+\alpha)\log\frac{N_{ijk}+\alpha}{N_{ij}+\alpha*r_i} - \frac{0.6*\alpha}{\max(1,\mathrm{m})}*(r_i-1)*q_i. \tag{7}$$

Equation 7 utilizes a single parameter $\alpha = 0.7$ to control the priors and penalty term. Since Eq.7 still employs the $Dirichlet(\alpha,\ldots,,\alpha)$, , similar to Eq.1, the score remains non-equivalent. Moreover, as the penalty for structural complexity uses $1/m$, the score maintains consistency and is decomposable. The proof follows a similar approach to that of L-SFE$_{1*}^{D}$.

```
import numpy as np
import pandas as pd

# This new score function employs a Non-parametric Bayesian approach using Dirichlet
↪  Process prior to measure the fit, allowing for flexibility in model complexity based on
↪  the data observed.
def subscoreLLM(child, parents, Data):
    gamma = 1  # Hyperparameter for Dirichlet Process prior
    sample_size = Data.shape[0]

    var_states = np.unique(Data[:, child])
    var_cardinality = len(var_states)

    if len(parents) != 0:
        state_counts = pd.crosstab(index=[Data[:, child]], columns=[Data[:, var] for var in
        ↪  parents]).values
        num_parents_states = np.prod([len(np.unique(Data[:, var])) for var in parents])
    else:
        state_counts = np.unique(Data[:, child], return_counts=True)[1]
        num_parents_states = 1

    counts = np.asarray(state_counts)
    counts += gamma  # Applying Dirichlet Process smoothing\n
```

```
    log_likelihoods = np.zeros_like(counts, dtype=float)
    np.log(counts, out=log_likelihoods)

    log_conditionals = np.sum(counts, axis=0, dtype=float)
    log_conditionals = np.log(log_conditionals + gamma)  # Applying Dirichlet Process prior
    ↪  for conditioning\n

    log_likelihoods -= log_conditionals
    log_likelihoods *= counts

    score = np.sum(log_likelihoods)

    # Introduce a complexity term based on the number of unique configurations observed
    complexity_penalty = (num_parents_states ** 1.5) / sample_size
    score -= complexity_penalty * (var_cardinality - 1)

    return score
```

Listing 3: Core idea with code implementation of L-SFE$^{D}_{3_*}$

It can be expressed in the mathematical form as

$$\Psi(\mathcal{G}|\mathcal{D}) = \sum_{i=1}^{n}\sum_{j=1}^{q_i}\sum_{k=1}^{r_r}(N_{ijk}+\gamma)\log\frac{N_{ijk}+\gamma}{N_{ijk}+\gamma*r_i} - \frac{1}{m}(r_i-1)*{q_i}^{1.5}, \tag{8}$$

where $\gamma = 1$. Eq.8 amplifies the graphical complexity from q_i to ${q_i}^{1}.5$. Despite this transformation, the structure remains decomposable, consistent, and non-equivalent, similar to the behavior described in Eq.7.

```
    import numpy as np

# This function constructs an enhanced scoring mechanism that synergizes predictive
↪  accuracy with a refined complexity penalty,
# # optimizing the balance between explained variance and model simplicity for better
↪  performance.
# # Adjusted penalty and variance calculations improve robustness in model selection.

def subscoreLLM(child, parents, Data):
    m = Data.shape[0]
    lambda_value = 1.7  # Enhanced lambda for stronger regularization on complexity
    cov = np.cov(Data.T)
    if len(parents) == 0:
        score = -1 * m * np.log(cov[child, child])
        return score
    yX = cov[np.ix_([child], parents)]
    XX = cov[np.ix_(parents, parents)]
    if np.linalg.det(XX) == 0:
        return np.inf
    predicted_variance = yX @ np.linalg.inv(XX) @ yX.T
    residual_var = cov[child, child] - predicted_variance
    explained_variance_ratio = predicted_variance / cov[child, child]
    H = np.log(residual_var + 1e-10)  # Adding a small constant to avoid log(0)
    # Combined penalty for complexity, considering both number of parameters and explained
    ↪  variance
    penalty = (len(parents) + 1) * (np.log(m) + lambda_value * np.log(len(parents) + 1) *
    ↪  (1 - explained_variance_ratio))
    score = -1 * (m * H + penalty)
    score = score.item()
    return score
```

Listing 4: Core idea with code implementation of L $-$ SFE$^{C}_{3}$

For the continuous dataset, experimental evaluation reveals that only L-SFE$^{C}_{3_*}$ outperforms the original BIC under various settings. Therefore, we focus the analysis on L-SFE$^{C}_{3_*}$ here. It can be expressed in the mathematical form as

$$\begin{aligned}\Psi(Pa_i^{\mathcal{G}} \to X_i|\mathcal{D}) = &- m * \log(\Sigma_{i,i} - \Sigma_{i,Pa_i^{\mathcal{G}}} \cdot \Sigma^{-1}_{Pa_i^{\mathcal{G}},Pa_i^{\mathcal{G}}} \cdot \Sigma^{\mathrm{T}}_{i,Pa_i^{\mathcal{G}}} + \varepsilon) \\ &-(\left|Pa_i^{\mathcal{G}}\right| + 1) * (\log(m) + \lambda \log(\left|Pa_i^{\mathcal{G}}\right| + 1) \cdot (1 - R^2)) \\ R^2 = &\frac{\Sigma_{i,Pa_i^{\mathcal{G}}} \cdot \Sigma^{-1}_{Pa_i^{\mathcal{G}},Pa_i^{\mathcal{G}}} \cdot \Sigma^{\mathrm{T}}_{i.Pa_i^{\mathcal{G}}}}{\Sigma_{i,i}},\end{aligned} \tag{9}$$

where Σ is a covariance matrix, and $\lambda = 1.7, \varepsilon = 1e^{-10}$. Eq.9 computes a regularized log-likelihood score for $Pa_i^{\mathcal{G}} \to X_i$ in a linear model. It incorporates a fitting term based on the residual variance, which quantifies the unexplained variance of the X_i after accounting for the $Pa_i^{\mathcal{G}}$, as well as a penalty term that accounts for model complexity—specifically, the number of parents and the variance they explain. Additionally, a regularization term is introduced to mitigate the risk of overfitting. Since Eq.9 modifies only the parameters in penalty without introducing a higher-order penalty than $log(m)$, it remains decomposable and consistent.

2 Experiments

2.1 Settings

The detailed parameters for L-SFE are listed in Table 1. The comparison metrics involve the structural Hamming distance (SHD), the F1 score for adjacency accuracy(F1^{adj}), and the F1 score for arrow precision (F1^{arr}). And they are defined as follows

$$\begin{aligned}\mathrm{SHD} =& \mathrm{FN}^{\mathrm{arr}} + \mathrm{FP}^{\mathrm{arr}} \\ \mathrm{F1}^{\mathrm{arr}} =& 2\frac{\mathrm{P}^{\mathrm{arr}} \cdot \mathrm{R}^{\mathrm{arr}}}{\mathrm{P}^{\mathrm{arr}} + \mathrm{R}^{\mathrm{arr}}}, \mathrm{P}^{\mathrm{arr}} = \frac{\mathrm{TP}^{\mathrm{arr}}}{\mathrm{TP}^{\mathrm{arr}} + \mathrm{FP}^{\mathrm{arr}}}, \mathrm{R}^{\mathrm{arr}} = \frac{\mathrm{TP}^{\mathrm{arr}}}{\mathrm{TP}^{\mathrm{arr}} + \mathrm{FN}^{\mathrm{arr}}} \\ \mathrm{F1}^{\mathrm{adj}} =& 2\frac{\mathrm{P}^{\mathrm{adj}} \cdot \mathrm{R}^{\mathrm{adj}}}{\mathrm{P}^{\mathrm{adj}} + \mathrm{R}^{\mathrm{adj}}}, \mathrm{P}^{\mathrm{adj}} = \frac{\mathrm{TP}^{\mathrm{adj}}}{\mathrm{TP}^{\mathrm{adj}} + \mathrm{FP}^{\mathrm{adj}}}, \mathrm{R}^{\mathrm{arr}} = \frac{\mathrm{TP}^{\mathrm{adj}}}{\mathrm{TP}^{\mathrm{adj}} + \mathrm{FN}^{\mathrm{adj}}},\end{aligned} \tag{10}$$

where:

- $\mathrm{TP}^{\mathrm{adj/arr}}$ represents the number of true edges/arcs present in both learned $\mathcal{G}$ and ground-of-truth $\mathcal{G}^*$.
- $\mathrm{FP}^{\mathrm{adj/arr}}$ represents the number of true edges/arcs present in learned $\mathcal{G}$ but absent in ground-of-truth $\mathcal{G}^*$.
- $\mathrm{TN}^{\mathrm{adj/arr}}$ represents the number of true edges/arcs absent in learned $\mathcal{G}$ but present in ground-of-truth $\mathcal{G}^*$.
- $\mathrm{FN}^{\mathrm{adj/arr}}$ represents the number of true edges/arcs absent in both learned $\mathcal{G}$ and ground-of-truth $\mathcal{G}^*$.

The baseline algorithms include

- **PC**: A classical constraint-based CSL algorithm, utilizing its stable version in this study.
- **HC**: A classical greedy local search method applied in DAG space.
- **BOSS**: A greedy local search algorithm in permutation space, employing the grow-shrink tree (GST) method to compute the score.
- **GRaSP**: A permutation-based CSL method under weaker causal assumptions.
- **fGES**: An enhanced version of the GES algorithm, designed for learning large-scale causal structures.
- **DAGMA**: A continuous CSL algorithm that enforces acyclic constraints based on the log-determinant (log-det) function.
- **LiNGAM**: A method for identifying linear, non-Gaussian causal relationships between variables using Independent Component Analysis (ICA).

BIC is used as the scoring metric, also with $\lambda = 1$ for discrete datasets and $\lambda = 2$ for continuous datasets. For independence testing, the chi-square test with $\alpha = 0.01$ is applied to discrete datasets, while the Fisher-Z test with $\alpha = 0.01$ to continuous datasets. All parameters, except those for HC, follow the default settings in PGMPY, while HC adheres to the default settings in PyTetrad. Our code is available on here (https://github.com/wzd2502/L-SFE).

2.2 Results

Figure 1 compares the stability of L-SFE in discrete datasets sampled from SF graphs with BDeu, BD, and BIC. The results clearly show that L-SFE$^{D}_{1_5}$ in Repeat1, L-SFE$^{D}_{2_1}$ and L-SFE$^{D}_{2_5}$ in Repeat2, L-SFE$^{D}_{3_1}$ and L-SFE$^{D}_{3_2}$ in Repeat3 deliver the best performance on both F1^{arr} and F1^{adj}.

Parameter	Meaning	Value
n	the number of variables	30 (for training); $\{10, 20, 30, 40, 50\}$ (for testing)
m	the number of samples	$\{1000 * k \| k = 1, \cdots, 10\}$ (for training); 5000 (for testing)
n_p	the number of datasets	10
p_u	mutation rate	0.2
p_o	crossover rate	0.2
p_e	injection rate	0.2
n_t	max iterations	5
n_a	max algs in the population	5
n_l	repeats of L-SFE	3
λ	penalty parameters for BIC	1 (for discrete data), 2 (for continuous data)
/	edge prob in EndoRenyi graph	0.3
/	degree of per nodes	random in $[2, 6]$
/	categories for per node in discrete dataset	random in $[2, 5]$ when $n < 40$ and 3 when $n \geq 40$
/	the weights of SEM in continuous dataset	$U(-1, 1)$
/	the exponential noise	$Exp(U(-1, 1))$
/	the gumbel noise	$Gumbel(0, U(-1, 1))$
/	the gaussian noise	$\mathcal{N}(0, U(-1, 1))$
/	training method	HC with the length of tabu list is 100
/	temperature of LLM	default

Table 1: Parameter settings in L-SFE.

Figures 2 and 3 separately compare the F1 scores obtained by L-SFE and BIC-gaussian on continuous data under the linear exponential SEM and the linear gumbel SEM. In both cases, the score functions learned by the LLM consistently demonstrate a clear advantage over the BIC-gaussian method. Furthermore, in Repeat2, nearly all five score functions achieve identical F1^{arr} and F1^{adj} scores, suggesting that the EA may get stuck in a local optimum, making it difficult to escape.

Figure 4 evaluates the generality of L-SFE on discrete datasets sampled from SF graphs. $\text{L-SFE}^{D}_{1_*}$ and $\text{L-SFE}^{D}_{2_*}$ demonstrate similar improvements in F1^{arr} and F1^{adj} when using GLS in the EC space for $n \geq 30$, outperforming BDeu, which performs worse despite incorporating Dirichlet priors. In the permutation space, $\text{L-SFE}^{D}_{1_*}$ continues to exhibit the best performance, with all three LLM-learned score functions surpassing human-designed alternatives. Notably, BDeu struggles to learn valid structures due to its high time complexity when $n \geq 30$.

Figures 5 and 6 evaluate the generality of L-SFE on continuous datasets under the linear exponential and linear gumbel SEMs. In the EC space, the performance gap between LLM-generated and human-designed score functions has narrowed, with all L-SFE^{C} showing a slight advantage over BIC-gaussian. However, in the permutation space, $\text{L-SFE}^{C}_{3_*}$ fails to identify the DAG when $n \geq 30$ under exponential noise and $n \geq 40$ under gumbel noise. $\text{L-SFE}^{C}_{2_*}$ demonstrates greater robustness, but its F1 scores are relatively low compared to BIC-gaussian. $\text{L-SFE}^{C}_{1_*}$ achieves a slight advantage over BIC-gaussian under both noise settings. The expensive computational cost of all score functions is primarily due to the GST in BOSS, which involves large matrix operations, especially for dimensions greater than 40. Therefore, further efforts are needed to optimize GLS in the permutation space.

Tables 2 to 9 present a comprehensive comparison of the F1^{acc} and F1^{adj} scores for the L-SFE across the $\boldsymbol{\mathcal{G}}$, $\boldsymbol{\mathcal{E}}$, and $\boldsymbol{\mathcal{O}}$ spaces under various settings. Notably, L-SFE with GLS achieve the highest F1^{arr} and F1^{adj} scores in 16/20 conditions. For discrete datasets, L-SFE performes best in the EC space. This can be attributed to the efficiency GLS for CPDAG compared with DAG. Additionally, the single-edge operations are more conducive to discovering high-quality structures than the permutation space search, which tends to modify multiple edges simultaneously. Conversely, for continuous datasets, the performance of HC and GES significantly decline, and BOSS with L-SFE^{C}_{*}, exhibites improved accuracy. This finding highlights the effectiveness of score functions generated by LLM in contrast to traditional human-designed score functions.

Using the discrete dataset as example, Figures 7 and 8 illustrate the performance of different algorithms as the dataset size varies. The performance of all algorithms improves with an increasing number of observations. LFE^{D}_{*} with GLS outperforms other baseline algorithms. Among the evaluated methods, the combination of GES and LLM generated score functions achieves the highest F1 and the lowest SHD across nearly all test sets.

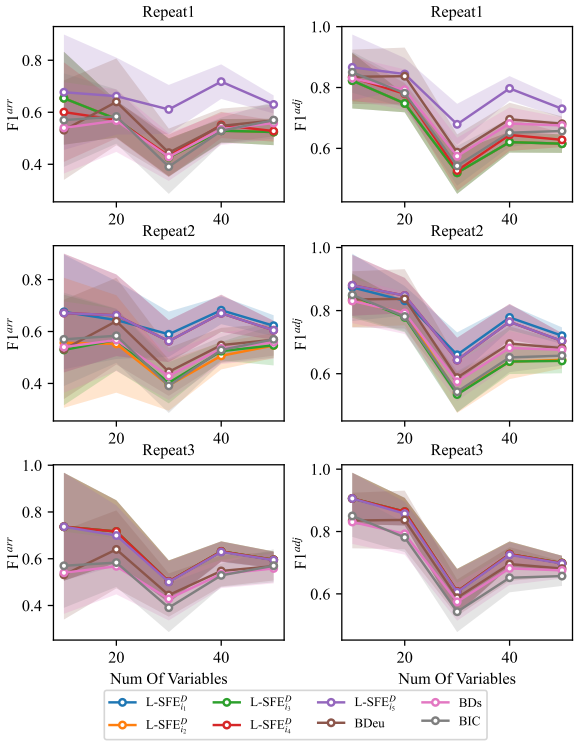

Figure 1: Stability analysis: L-SFE against human-designed scores on discrete datasets sampled from SF graphs.

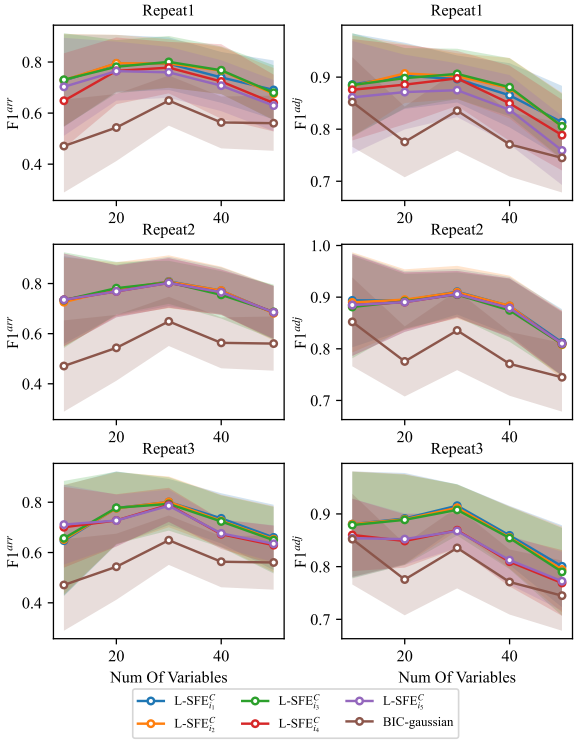

Figure 2: Stability analysis: L-SFE against human-designed scores on continuous datasets sampled from linear exp models.

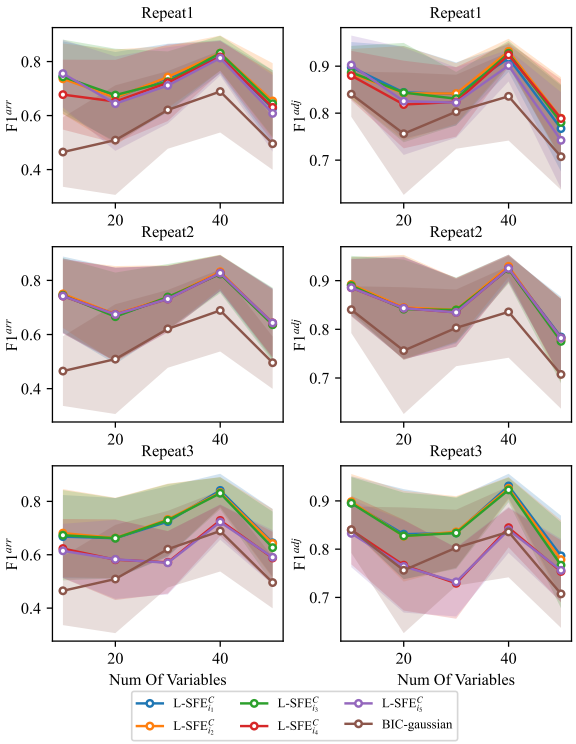

Figure 3: Stability analysis: L-SFE against human-designed scores on continuous datasets sampled from linear gumbel models.

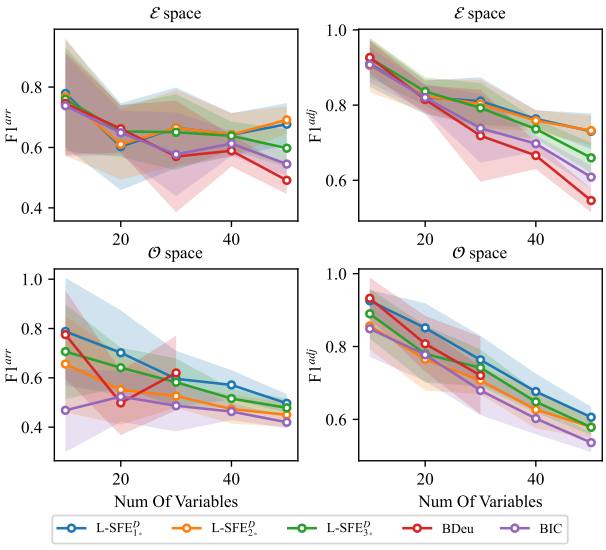

Figure 4: Generality analysis: L-SFE against human-designed scores on discrete datasets sampled from SF graphs.

	ER-10	ER-20	ER-30	ER-40	ER-50
PC	0.827±0.104	0.725±0.093	/	0.401±0.026	0.306±0.02
HC	0.793±0.137	0.74±0.148	0.698±0.061	0.559±0.059	0.555±0.049
BOSS	**0.852±0.129**	0.832±0.05	0.764±0.082	0.607±0.059	0.565±0.028
GRaSP	0.812±0.097	0.778±0.102	0.748±0.061	0.603±0.074	0.555±0.026
fGES	0.835±0.089	0.796±0.087	0.759±0.059	0.612±0.074	0.532±0.035
L-SFE$^D_*(\mathcal{G})$	0.762±0.16	0.826±0.092	0.858±0.045	0.735±0.094	0.705±0.044
L-SFE$^D_*(\mathcal{E})$	0.843±0.143	**0.846±0.078**	**0.888±0.058**	**0.819±0.041**	**0.795±0.013**
L-SFE$^D_*(\mathcal{O})$	0.835±0.153	0.775±0.096	0.713±0.069	0.588±0.05	/

Table 2: F1^{arr} comparison of L-SFE in different space against baseline methods on discrete dataset sampled from ER graphs.

	ER-10	ER-20	ER-30	ER-40	ER-50
PC	0.976±0.029	0.945±0.039	/	0.695±0.024	0.566±0.019
HC	0.978±0.036	0.906±0.052	0.835±0.04	0.693±0.042	0.641±0.028
BOSS	0.977±0.028	0.925±0.026	0.831±0.053	0.675±0.057	0.598±0.022
GRaSP	**0.982±0.027**	0.909±0.023	0.821±0.05	0.67±0.06	0.596±0.021
fGES	**0.982±0.027**	0.917±0.032	0.832±0.04	0.676±0.068	0.587±0.022
L-SFE$^D_*(\mathcal{G})$	0.924±0.033	0.946±0.06	0.917±0.024	0.812±0.066	0.763±0.031
L-SFE$^D_*(\mathcal{E})$	0.925±0.059	**0.961±0.036**	**0.937±0.027**	**0.876±0.038**	**0.811±0.011**
L-SFE$^D_*(\mathcal{O})$	0.938±0.045	0.878±0.083	0.798±0.042	0.686±0.036	/

Table 3: F1^{adj} comparison of L-SFE in different space against baseline methods on discrete dataset sampled from ER graphs.

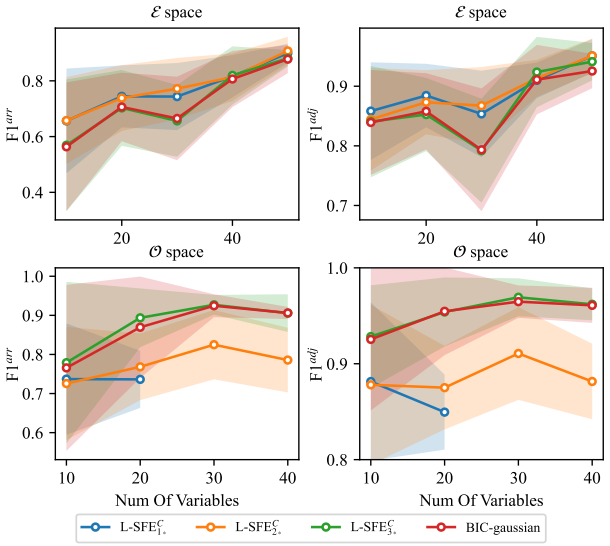

Figure 5: Generality analysis: L-SFE against human-designed scores on continuous datasets sampled from linear exp models.

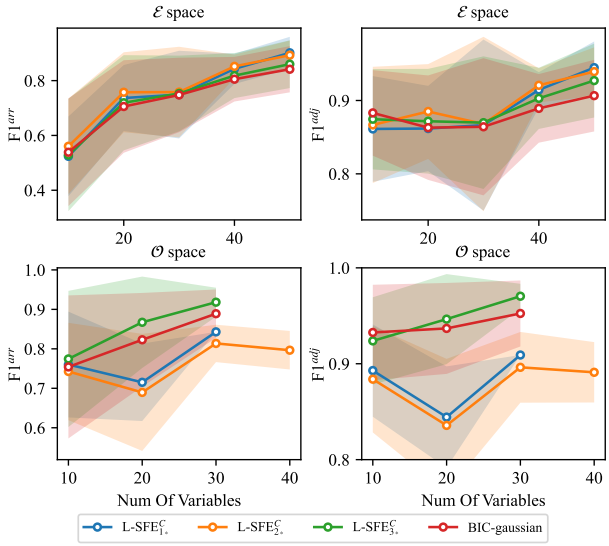

Figure 6: Generality analysis: L-SFE against human-designed scores on continuous datasets sampled from linear gumbel models.

In Table 10, we examine the performance of various LLMs for the L-SFE. We employ Random graphs with 30 nodes for training and evaluate the generated score functions on both ER and SF graphs. Notably, inference-increment LLMs, such as DeepSeek-r1 and GPT-o1, terminate during early iterations, likely due to their prolonged slow-thinking mechanisms exceeding the time constraints imposed by L-SFE. Among all models, GPT-4o achieves the highest performance. While GPT-4o mini demonstrates proficiency in learning score functions for smaller networks, its efficacy diminishes as network scale increases, underperforming relative to other LLMs. To validate the hypothesis that LLMs can indeed identify superior score functions, we intentionally selected the lowest-performing variant, GPT-4o mini, to conduct the primary experiments.

	SF-10	SF-20	SF-30	SF-40	SF-50
PC	0.554±0.077	0.503±0.096	0.445±0.054	0.279±0.032	0.272±0.029
HC	0.528±0.113	0.6±0.112	0.608±0.079	0.398±0.046	0.376±0.04
BOSS	0.614±0.172	0.547±0.097	0.596±0.063	0.342±0.024	0.329±0.021
GRaSP	0.663±0.162	0.589±0.082	0.597±0.103	0.344±0.021	0.332±0.015
fGES	0.535±0.134	0.556±0.085	0.587±0.033	0.340±0.035	0.411±0.166
L-SFE$^D_*(\mathcal{G})$	0.773±0.12	0.72±0.15	**0.714±0.102**	0.514±0.058	0.508±0.019
L-SFE$^D_*(\mathcal{E})$	0.751±0.17	0.702±0.109	0.677±0.106	**0.546±0.072**	**0.514±0.029**
L-SFE$^D_*(\mathcal{O})$	**0.781±0.166**	**0.772±0.12**	0.64±0.134	0.457±0.05	0.448±0.029

Table 4: F1^{arr} comparison of L-SFE in different space against baseline methods on discrete dataset sampled from SF graphs.

	SF-10	SF-20	SF-30	SF-40	SF-50
PC	0.922±0.031	**0.89±0.015**	0.785±0.008	0.431±0.03	0.419±0.011
HC	0.859±0.049	0.808±0.032	0.787±0.044	0.494±0.031	0.458±0.029
BOSS	0.877±0.062	0.803±0.027	0.734±0.046	0.415±0.024	0.402±0.015
GRaSP	0.896±0.053	0.807±0.025	0.732±0.057	0.41±0.014	0.396±0.015
fGES	0.84±0.047	0.781±0.022	0.718±0.037	0.340±0.035	0.405±0.014
L-SFE$^D_*(\mathcal{G})$	0.902±0.05	0.884±0.041	**0.875±0.03**	0.603±0.036	0.552±0.012
L-SFE$^D_*(\mathcal{E})$	0.921±0.053	0.884±0.042	0.841±0.045	**0.622±0.032**	**0.564±0.024**
L-SFE$^D_*(\mathcal{O})$	**0.942±0.032**	0.883±0.048	0.8±0.073	0.564±0.023	0.526±0.023

Table 5: F1^{adj} comparison of L-SFE in different space against baseline methods on discrete dataset sampled from SF graphs.

	Exp-10	Exp-20	Exp-30	Exp-40	Exp-50
PC	0.454±0.184	0.57±0.096	0.548±0.059	0.63±0.062	0.689±0.06
FGES	0.577±0.162	0.602±0.137	0.669±0.188	0.771±0.114	0.872±0.043
BOSS	0.779±0.181	0.774±0.173	0.848±0.165	0.877±0.051	0.929±0.019
DAGMA	0.427±0.155	0.326±0.066	0.416±0.079	0.532±0.077	0.515±0.065
LiNGAM	**0.949±0.032**	**0.921±0.037**	0.926±0.026	0.919±0.029	0.926±0.011
L-SFE$^C_*(\mathcal{G})$	0.603±0.143	0.73±0.164	0.539±0.105	0.776±0.055	0.852±0.025
L-SFE$^C_*(\mathcal{E})$	0.592±0.148	0.592±0.147	0.685±0.167	0.779±0.115	0.873±0.046
L-SFE$^C_*(\mathcal{O})$	0.71±0.242	0.92±0.059	**0.93±0.057**	**0.934±0.032**	**0.939±0.008**

Table 6: F1^{arr} comparison of L-SFE in different space against baseline methods on continuous dataset sampled from linear exp models.

	Exp-10	Exp-20	Exp-30	Exp-40	Exp-50
PC	0.773±0.044	0.828±0.042	0.808±0.047	0.852±0.027	0.928±0.016
FGES	0.863±0.048	0.797±0.09	0.807±0.106	0.871±0.08	0.942±0.03
BOSS	0.93±0.063	0.898±0.082	0.919±0.071	0.949±0.021	0.97±0.012
DAGMA	0.804±0.06	0.731±0.043	0.743±0.037	0.765±0.036	0.766±0.03
LiNGAM	**0.949±0.032**	0.921±0.037	0.928±0.025	0.919±0.029	0.926±0.011
L-SFE$^C_*(\mathcal{G})$	0.851±0.05	0.83±0.118	0.721±0.056	0.88±0.021	0.933±0.019
L-SFE$^C_*(\mathcal{E})$	0.872±0.046	0.789±0.086	0.804±0.103	0.862±0.076	0.937±0.026
L-SFE$^C_*(\mathcal{O})$	0.895±0.095	**0.963±0.031**	**0.954±0.034**	**0.963±0.023**	**0.973±0.005**

Table 7: F1^{adj} comparison of L-SFE in different space against baseline methods on continuous dataset sampled from linear exp models.

	Gum-10	Gum-20	Gum-30	Gum-40	Gum-50
PC	0.545±0.15	0.651±0.095	0.665±0.064	0.659±0.093	0.633±0.042
FGES	0.591±0.225	0.754±0.129	0.782±0.148	0.803±0.116	0.793±0.079
BOSS	0.846±0.212	0.93±0.04	0.884±0.06	0.895±0.076	**0.883±0.027**
DAGMA	0.246±0.087	0.372±0.075	0.435±0.096	0.497±0.079	0.504±0.066
LiNGAM	0.872±0.079	0.871±0.039	0.844±0.036	0.862±0.028	0.846±0.036
L-SFE$^C_*(\mathcal{G})$	0.626±0.195	0.746±0.143	0.704±0.155	0.706±0.128	0.775±0.094
L-SFE$^C_*(\mathcal{E})$	0.641±0.238	0.787±0.14	0.815±0.131	0.86±0.12	0.825±0.094
L-SFE$^C_*(\mathcal{O})$	**0.979±0.02**	**0.961±0.018**	**0.933±0.018**	**0.926±0.032**	/

Table 8: F1^{arr} comparison of L-SFE in different space against baseline methods on continuous dataset sampled from linear gumbel models.

	Gum-10	Gum-20	Gum-30	Gum-40	Gum-50
PC	0.796±0.062	0.861±0.034	0.86±0.037	0.913±0.028	0.858±0.027
FGES	0.825±0.083	0.864±0.072	0.897±0.068	0.899±0.083	0.908±0.059
BOSS	0.925±0.086	0.964±0.013	0.951±0.037	0.952±0.049	**0.968±0.016**
DAGMA	0.74±0.064	0.733±0.038	0.758±0.039	0.764±0.04	0.78±0.036
LiNGAM	0.872±0.079	0.873±0.036	0.845±0.033	0.862±0.028	0.846±0.036
L-SFE$^C_*(\mathcal{G})$	0.82±0.09	0.856±0.1	0.85±0.08	0.84±0.073	0.884±0.053
L-SFE$^C_*(\mathcal{E})$	0.825±0.099	0.883±0.076	0.898±0.067	0.921±0.076	0.913±0.051
L-SFE$^C_*(\mathcal{O})$	**0.979±0.02**	**0.969±0.014**	**0.967±0.015**	**0.967±0.017**	/

Table 9: F1^{adj} comparison of L-SFE in different space against baseline methods on continuous dataset sampled from linear gumbel models.

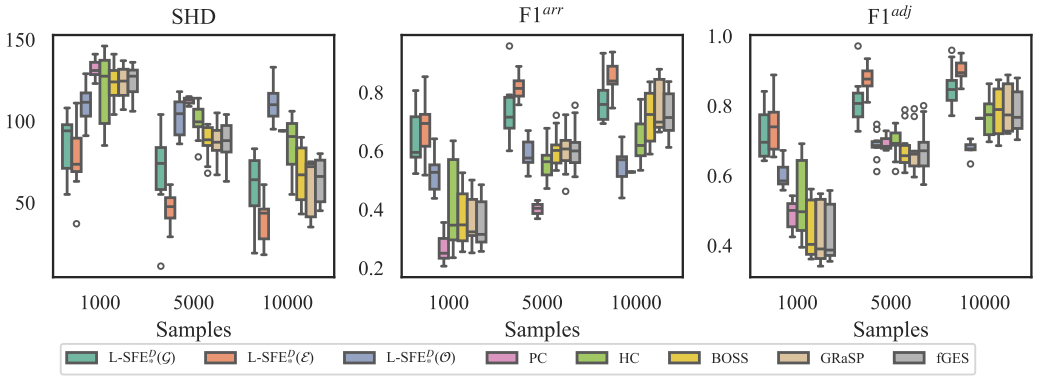

Figure 7: L-SFE + GLS in $\mathcal{G}$, $\mathcal{E}$ and $\mathcal{O}$ space against baseline methods under different size of datasets sampled from ER graphs.

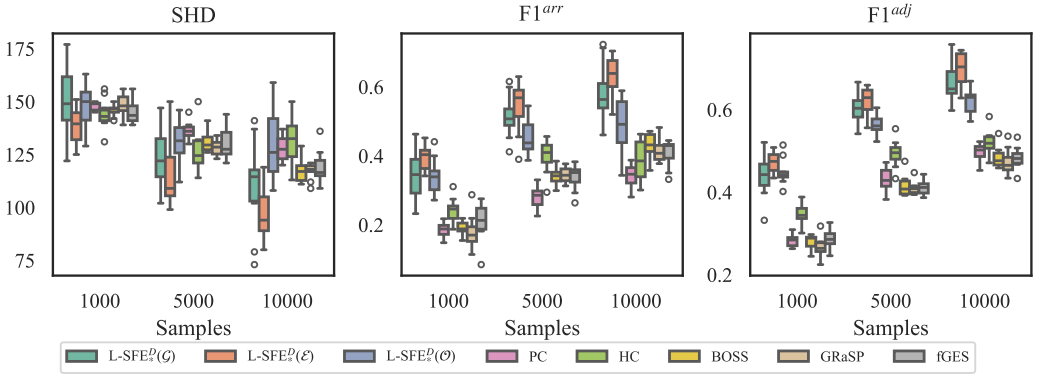

Figure 8: L-SFE + GLS in $\mathcal{G}$, $\mathcal{E}$ and $\mathcal{O}$ space against baseline methods under different size of datasets sampled from SF graphs.

Table 10: SHD comparison of different LLMs on discrete dataset.

Networks	Deepseek-R1	Deepseek-V3	GPT-4o	GPT-4o-mini	GPT-o1	Qwen-Turbo
ER-10	2.5±1.962	4.6±2.653	1.9±1.920	**0.9±1.135**	2.7±2.648	1.9±1.920
ER-20	14.0±5.118	14.6±8.743	**8.0±6.148**	9.1±3.534	17.2±11.591	12.8±3.893
ER-30	34.0±9.581	30.4±9.667	**17.6±6.406**	27.0±8.775	35.0±12.450	26.2±9.652
ER-40	69.0±7.0	60.7±15.613	**42.7±18.809**	62.7±12.689	72.7±12.710	57.0±9.602
ER-50	148.4±4.477	143.4±13.749	**92.4±10.983**	150.0±6.066	152.0±15.046	139.3±12.248
SF-10	3.7±4.405	7.9±5.300	3.2±3.995	**2.8±2.181**	4.3±4.267	3.9±5.147
SF-20	31.0±7.576	27.9±11.458	31.6±10.781	**30.0±8.544**	31.3±7.376	31.9±8.324
SF-30	63.7±9.602	59.7±11.832	**45.1±13.065**	62.1±11.335	62.0±10.537	57.6±7.902
SF-40	83.2±6.225	76.4±13.177	**39.5±8.675**	83.2±5.250	83.4±10.219	77.3±10.364
SF-50	142.9±8.561	127.6±14.015	**82.4±17.042**	140.4±12.901	142.7±14.947	127.0±13.387