Latent Dependency Forest Models

**Basics**

Let \( x = (X_1, X_2, \ldots, X_n) \) be a set of random variables and \( x = (x_1, x_2, \ldots, x_n) \) be an assignment to the set of random variables. Given an assignment \( x \), we construct a complete directed graph \( G_{x} = (V_x, E_x) \) such that:

\[ V_x = \{x_1, x_2, \ldots, x_n\} \]

\[ E_x = \{(x_i, x_j) | i \neq j, \ 0 \leq i < n, \ 1 \leq j \leq n\} \]

We obtain a single dependency tree structure that is a directed spanning tree of the graph \( G_{x} \) rooted at \( x_0 \), where the weight of the graph is denoted as \( w_{ij} \). We denote this tree by \( T = (V_T, E_T) \), where \( V_T = V_x, E_T \subseteq E_x \). We can compute the strength of a spanning tree \( T = (V_T, E_T) \) as the product of the edge weights:

\[ w(T) = \prod_{(i,j) \in E_T} w_{ij} \]

**Weight of the assignment** is the sum over the weights of all possible dependency trees for \( x \).

**LDFM**, a generative model based on the above framework.

An assignment \( x = (x_1, x_2, \ldots, x_n) \) is generated recursively in a top-down manner.

**Firstly**, we generate a dependency tree with \( n + 1 \) nodes uniformly at random. We label the root node as \( x_0 \).

**Then**, starting from the root node, we recursively traverse the tree in pre-order; at each non-root node, we generate a \((\text{variable}, \text{value})\), pair conditioned on the \((\text{variable}, \text{value})\) pair of its parent node. The probability of generating an assignment \( x \) is:

\[ p(x) = \frac{\text{beta}!}{\text{beta}^{(n+1)}} \times Z \]

**Matrix Tree Theorem (MTT)**

Let \( G \) be a graph with nodes \( V = \{x_1, x_2, \ldots, x_n\} \) and edges \( E \). Define (Laplacian) matrix \( Q \) as \( (n+1) \times (n+1) \) matrix indexed from \( 0 \) to \( n \). For all \( i \) and \( j \):

\[ Q_{ij} = \left\{ \begin{array}{ll} w_{ij} & \text{if } i = j \text{ and } V_j \text{ is a dummy root node.} \\ -w_{ij} & \text{if } i \neq j \text{ and } (x_i, x_j) \in E \end{array} \right. \]

If the \( i \)-th row and column are removed from \( Q \) to produce the matrix \( Q' \), then the sum of the weights of all the directed spanning trees rooted at node \( i \) is equal to the determinant of \( Q' \). Here, \( Z = \det(Q') \)

\[ p(x) = Z^{-1} \]

**Properties**

- **Properties**
  - **Independence Property**
  - **Consistency Property**
  - **Constrained Dependencies**
  - **Direct Acyclic Graphs**
  - **Latent Variables**

**Learning**

- **Key Ideas**
  - **Avoid the difficult structure learning problem by assuming a complete LDFM structure**
  - **Rely on parameter learning to specify the weights of all the dependencies**

- **Objective Function**

**Inference**

- **Probabilistic Inference**
- **Gibbs Sampling**
- **Tree Sampling**

**Experiments**

- **9 benchmark BNs from the bnlearn repository**
- **We sampled two training sets of 5000 and 500 instances, one validation set of 1000 instances, and one testing set of 1000 instances**
- **All the random variables are discrete**
- **Comparisons by their accuracy in query answering on the test data, which is to compute the conditional log likelihood (CLL) \( p(q | e) \) where \( q \) and \( e \) are the values that \( Q \) and \( E \) in the test data sample.**
- **Due to sample sparsity of sampling algorithms, we report the maximum of \( \text{CLL} \) and \( \text{CMML} \) \( (\sum_{x \in X} \log p(x | e) = e) \) values.**
- **We empirically showed that LDFMs are competitive with existing probabilistic models.**

**Table 1**: The maximum of CLL and CMML normalized by the number of query variables. The bold numbers mark the best performances. g-LDFM denotes the results of LDFM by using Gibbs sampling and t-LDFM denotes the results of LDFM by using tree-augmented sampling.