Graph Generation

For training and testing

Table Of Contents

What is a reference model?

“Traditionally, the likelihood of an observation occurring by chance has been referred to as a null model.”

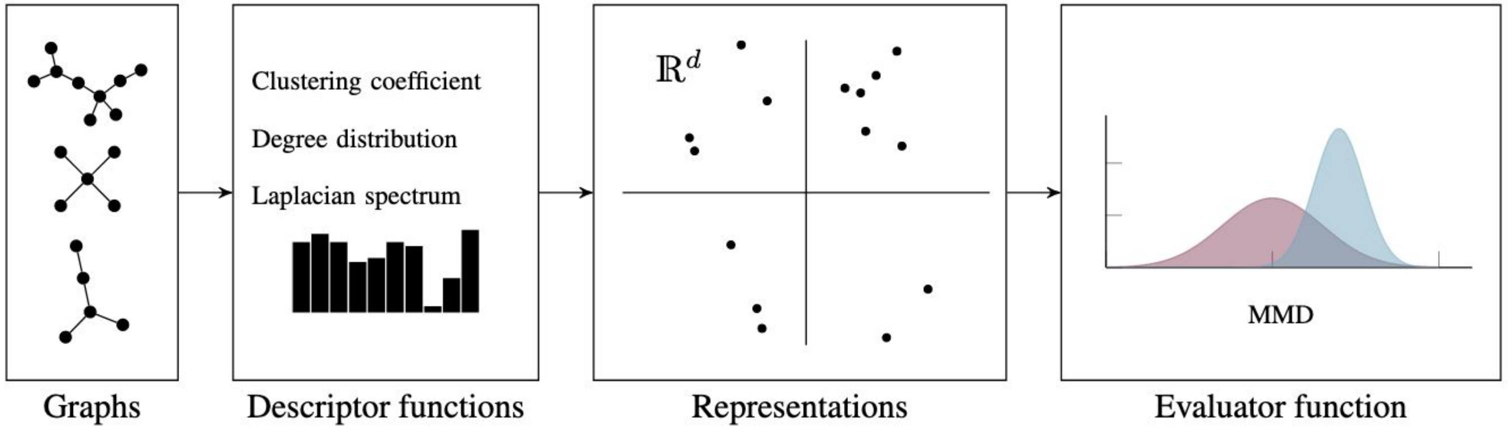
Properties of metrics

- Expressiveness
    - Should increase monotonically the further graphs are apart
- Robustness
    - Robust to small perturbations
- Efficiency
    - Fast to compute

Hobson, Elizabeth A., et al. "A guide to choosing and implementing reference models for social network analysis." Biological Reviews 96.6 (2021): 2716-2734

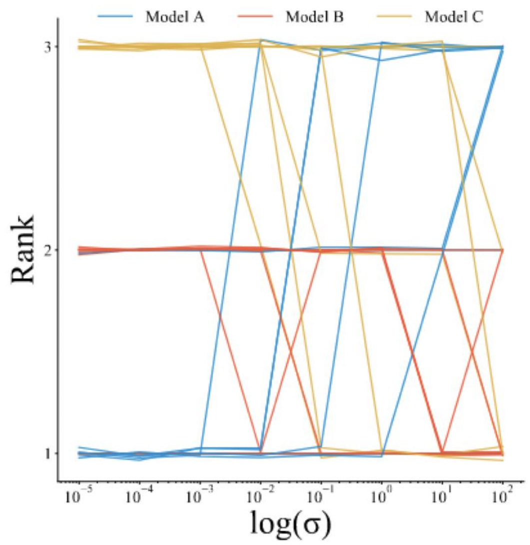
Maximum Mean Discrepancy (MMD)

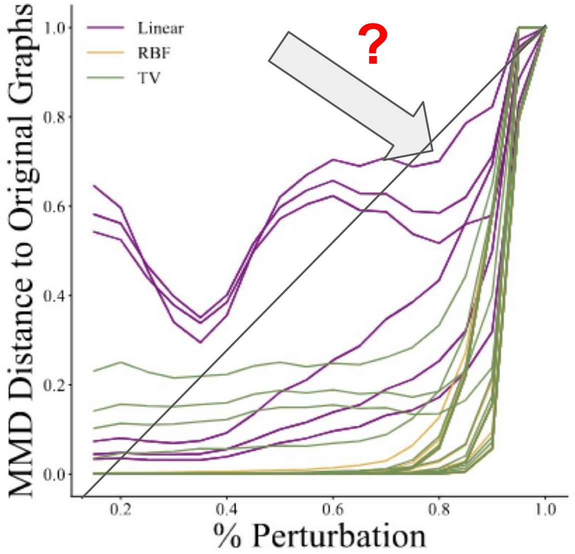
- Problem: Distance of graphs is NP-Hard
- Solution: Map to R^d and then compute distance
- MMD compares two distributions using kernel functions



Maximum Mean Discrepancy - **Problem**

Choosing the right descriptor and kernel function.





How to generate a reference model?

Permutation

- observations are *swapped* without replacement

Resampling

- observations are *sampled* from the observed data with replacement

Distribution-sampling

- observations are *drawn* from a fixed distribution

Generative models

- synthetic data or networks are *constructed* from stochastic rules

Hobson, Elizabeth A., et al. "A guide to choosing and implementing reference models for social network analysis." Biological Reviews 96.6 (2021): 2716-2734

Problem Definition

Given

Given a set of graphs {G1, G2, …, GN}, what is the probability of a graph G?

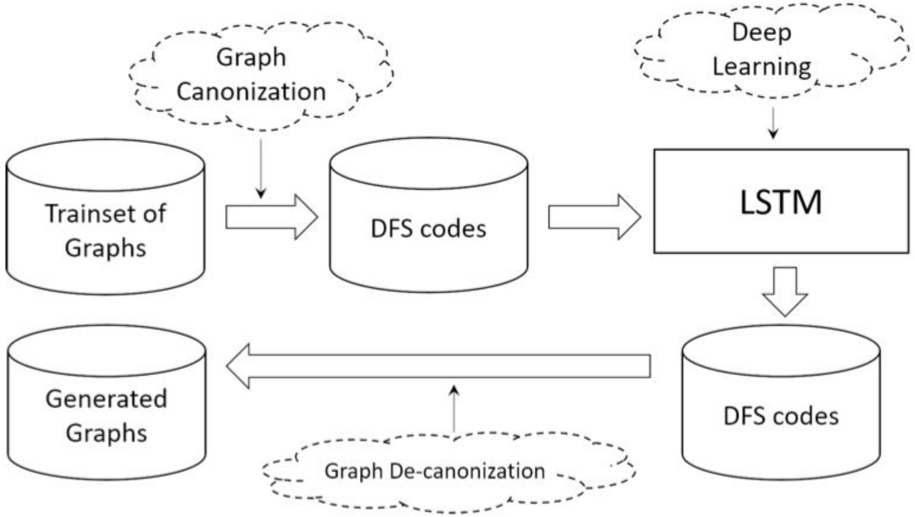
Method

To find a solution of the problem, we build graph generative models.

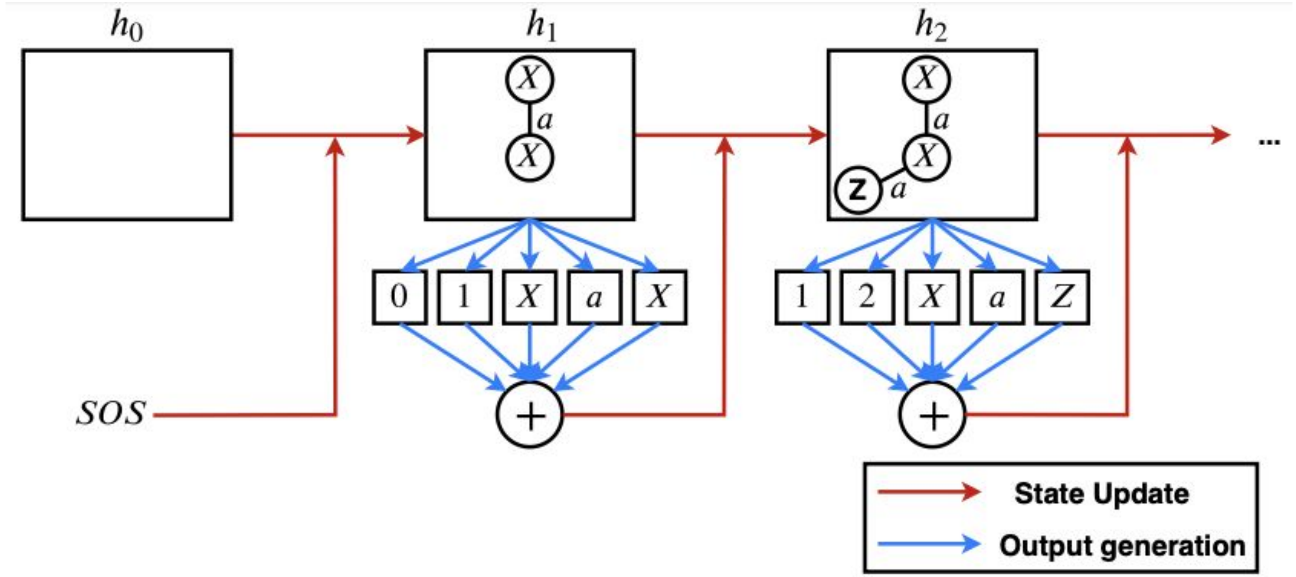
Edge-driven GraphGen

- Autoregressive algorithm
- Sequence labels are created from a bijective function
- Labels isomorph graphs with the same label
- p(Graph) is equal to p(Sequence Label)
- Predicts edge between old nodes or edge to a new node at a time
- Uses LSTM to predict the probability of a sequence element given the already generated sequence elements
- Embeds generated information in feature vector for next sequence item
- Decode generated sequence

1. Graph to Label (Sequence of descriptions of nodes)



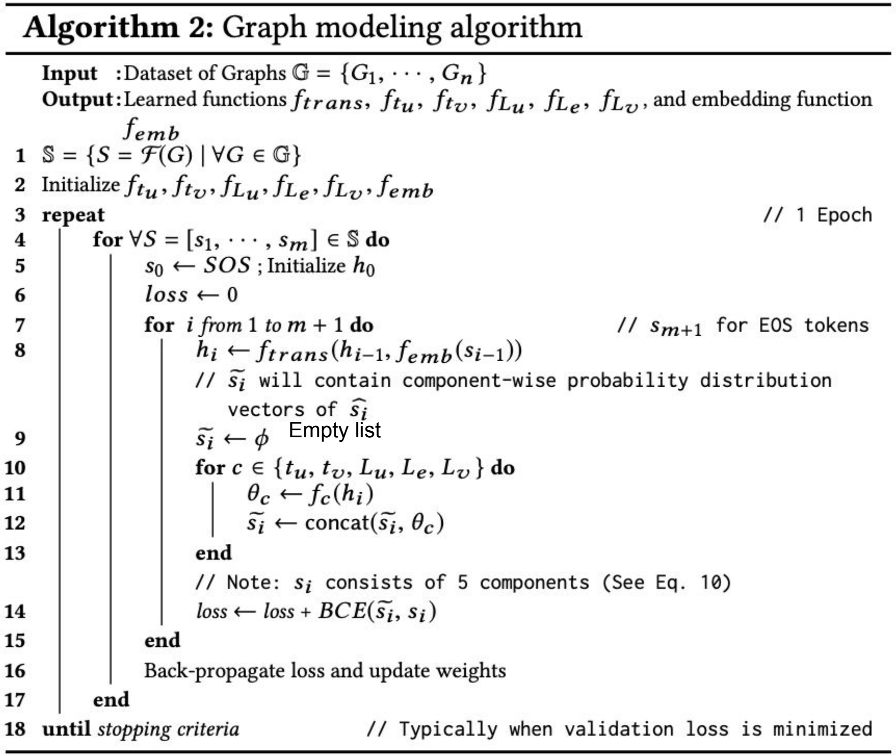
LSTM Definition



$$h_i = f_{trans}(h_{i-1}, f_{emb}(s_{i-1}))$$
$$t_u \sim_M \ \theta_{t_u} = f_{t_u}(h_i)$$
$$t_v \sim_M \ \theta_{t_v} = f_{t_v}(h_i)$$
$$L_u \sim_M \ \theta_{L_u} = f_{L_u}(h_i)$$
$$L_e \sim_M \ \theta_{L_e} = f_{L_e}(h_i)$$
$$L_v \sim_M \ \theta_{L_u} = f_{L_v}(h_i)$$
$$s_i = \text{concat}(t_u, t_v, L_u, L_e, L_v)$$

Multi-head LSTM

GraphGen



Algorithm 2: Graph modeling algorithm

```
   Input  :Dataset of Graphs $\mathbb{G} = \{G_1, \cdots, G_n\}$
   Output :Learned functions $f_{trans}, f_{t_u}, f_{t_v}, f_{L_u}, f_{L_e}, f_{L_v}$, and embedding function
           $f_{emb}$
1  $\mathbb{S} = \{S = \mathcal{F}(G) \mid \forall G \in \mathbb{G}\}$
2  Initialize $f_{t_u}, f_{t_v}, f_{L_u}, f_{L_e}, f_{L_v}, f_{emb}$
3  repeat                                                        // 1 Epoch
4      for $\forall S = [s_1, \cdots, s_m] \in \mathbb{S}$ do
5          $s_0 \leftarrow SOS$ ; Initialize $h_0$
6          $loss \leftarrow 0$
7          for $i$ from 1 to $m + 1$ do                  // $s_{m+1}$ for EOS tokens
8              $h_i \leftarrow f_{trans}(h_{i-1}, f_{emb}(s_{i-1}))$
               // $\widetilde{s_i}$ will contain component-wise probability distribution
                  vectors of $\widehat{s_i}$
9              $\widetilde{s_i} \leftarrow \phi$   Empty list
10             for $c \in \{t_u, t_v, L_u, L_e, L_v\}$ do
11                 $\theta_c \leftarrow f_c(h_i)$
12                 $\widetilde{s_i} \leftarrow \text{concat}(\widetilde{s_i}, \theta_c)$
13             end
               // Note: $s_i$ consists of 5 components (See Eq. 10)
14             $loss \leftarrow loss + BCE(\widetilde{s_i}, s_i)$
15         end
16         Back-propagate loss and update weights
17     end
18 until stopping criteria          // Typically when validation loss is minimized
```

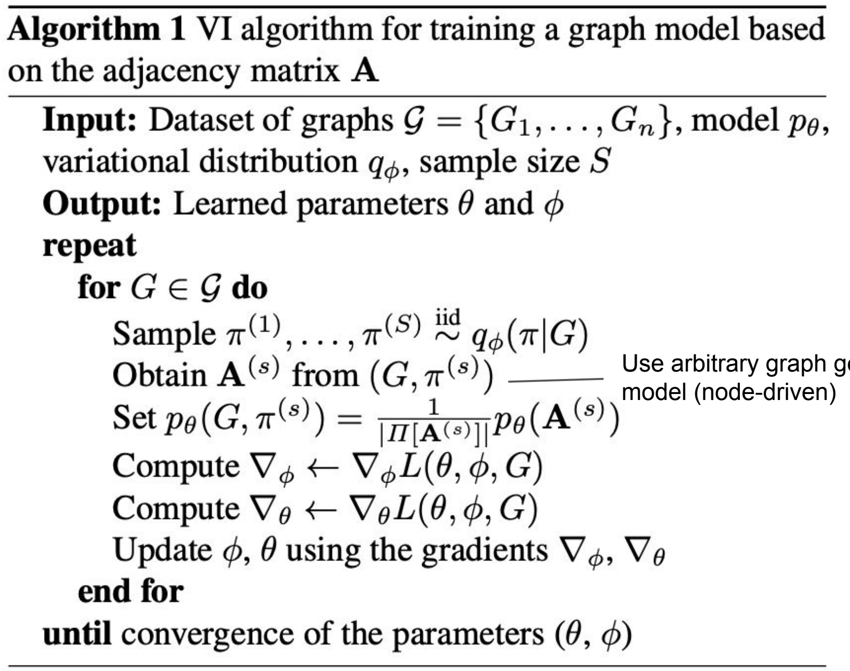
$$BCE(\widetilde{s_i}, s_i) = -\sum_{c} \left(s_i[c]^T \log \widetilde{s_i}[c] + (1 - s_i[c])^T \log(1 - \widetilde{s_i}[c]) \right)$$

GraphGen and Failure Propagation Graph

1. Sample failure graphs.
2. For each graph, compute sequence encoding.
3. Train LSTM or any other sequence model.
4. Sample graph using autoregressive method.

Edge-driven VI Algorithm for training a graph model

- Graphs have automorphisms
- Number of permutations of nodes that build the same graph equals number of automorphism
- Account for different order of generation during model optimization
- Use color algorithm to find lower bound of orbits (nodes that are equal to an automorphism)
- Sample automorphisms



Algorithm 1 VI algorithm for training a graph model based on the adjacency matrix $\mathbf{A}$

```
**Input:** Dataset of graphs $\mathcal{G} = \{G_1, \dots, G_n\}$, model $p_\theta$, variational distribution $q_\phi$, sample size $S$
**Output:** Learned parameters $\theta$ and $\phi$
**repeat**
  **for** $G \in \mathcal{G}$ **do**
    Sample $\pi^{(1)}, \dots, \pi^{(S)} \overset{\text{iid}}{\sim} q_\phi(\pi|G)$
    Obtain $\mathbf{A}^{(s)}$ from $(G, \pi^{(s)})$
    Set $p_\theta(G, \pi^{(s)}) = \frac{1}{|\Pi[\mathbf{A}^{(s)}]|} p_\theta(\mathbf{A}^{(s)})$
    Compute $\nabla_\phi \leftarrow \nabla_\phi L(\theta, \phi, G)$
    Compute $\nabla_\theta \leftarrow \nabla_\theta L(\theta, \phi, G)$
    Update $\phi$, $\theta$ using the gradients $\nabla_\phi$, $\nabla_\theta$
  **end for**
**until** convergence of the parameters $(\theta, \phi)$
```

Use arbitrary graph generating model (node-driven)

Problem of most graph models

The common goal is to generate a maximum likelihood estimation of a graph G under a model. **But**, we don't know the order of **nodes** of the Graph, due to possible **automorphism**. Thus, we need to evaluate **all** different orderings.

$$p(G) = \sum_{\pi} p(G, \pi).$$

Problem: **Very** expensive.

How is this commonly handled?

Most models use a **specialy-designed**, or from **DFS** or **BFS** derived order.

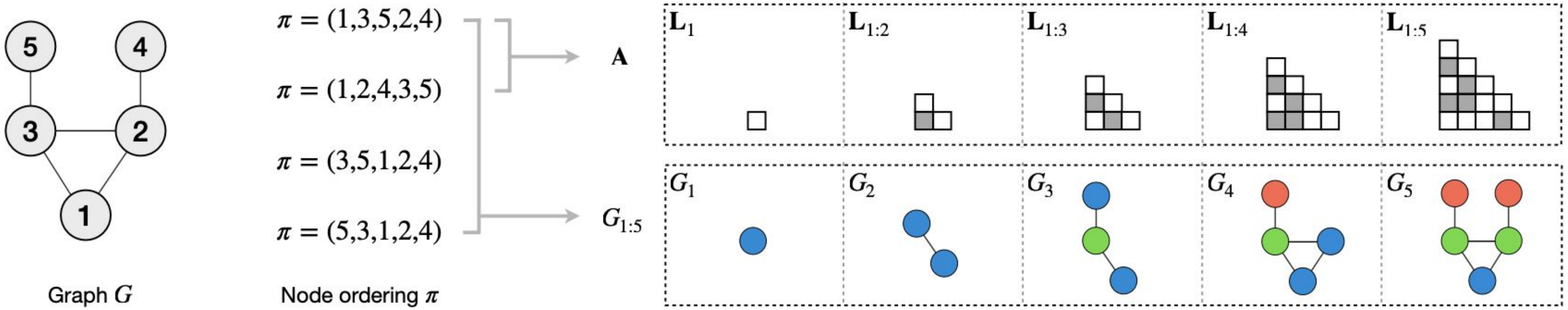
Solution

We just need to use the node ordering of the given graph in training.

Problem:

Which is the correct **node ordering** of the graph? We cannot derive a node ordering due to **automorphisms**.

Example of orderings



Solution

Therefore, we need to make the order a **random variable.**

Let Π(A) be the set of orderings that give the same adjacency matrix A.

Let $G_{1:n}$ be a generated Graph defined by the generation sequence (G_1,...,G_n).

Let $\Pi(G_{1:n})$ be the set of orderings that give the same graph, which is defined by the sequence.

$$p(\pi|\mathbf{A}) = \frac{1}{|\,\Pi[\mathbf{A}]\,|}$$

Node-driven generation

$$p(\pi|G_{1:n}) = \frac{1}{|\,\Pi[G_{1:n}]\,|}$$

Edge-driven generation

Joint Probability

$$p(G, \pi) = \frac{1}{\mid \mathit{\Pi}[\mathbf{A}] \mid} p(\mathbf{A}) = \frac{1}{\mid \mathit{\Pi}[G_{1:n}] \mid} p(G_{1:n})$$

Obtaining number of orderings

$|\Pi(A)|$ is equal to the number of automorphisms in a graph. This is well studied and can be solved in *exp(O(sqrt(n log n)))*.

$|\Pi(G_{1:n})|$ is related to orbits (r) in a graph. The orbit of a node is the set of nodes that are equal by an automorphism.

$$|\,\Pi[G_{1:n}]\,| = \prod_{t=1}^{n} |\,r(G_t, \pi_t)\,|$$

Identifiying orbits

To estimate the number of orbits, we use a heuristic which gives us a **lower bound** on the number of orbits. We use a coloring algorithm that applies the **same color to all nodes in an orbit**, but does not always color different orbits differently.

$$\beta(G_{1:n}) \triangleq \prod_{t=1}^{n} |r_{\mathrm{CR}}(G_t, \pi_t)| \geq |\ \Pi[G_{1:n}]\ |$$

$$\widehat{p}(G,\pi) \triangleq \frac{1}{\beta(G_{1:n})} p(G_{1:n}) \leq p(G,\pi).$$

$$\sum_{\pi} \widehat{p}(G,\pi) \approx p(G)$$

Estimate graph probability

We use a variational distribution $q_\phi(\pi|G)$ as an estimation.

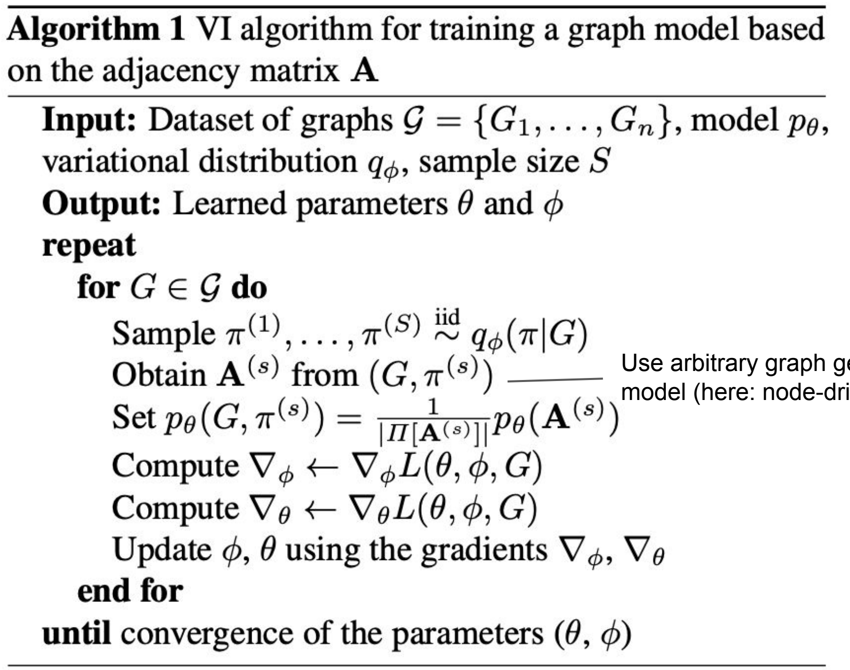
To train a model, we define the loss:

$$L(\theta, \phi, G) = \mathbb{E}_{q_\phi(\pi|G)}\left[\log p_\theta(G, \pi) - \log q_\phi(\pi|G)\right]$$

It gives a lower bound:

$$L(\theta, \phi, G) \leq \log p_\theta(G)$$

VI Algorithm



Algorithm 1 VI algorithm for training a graph model based on the adjacency matrix $\mathbf{A}$

```
**Input:** Dataset of graphs $\mathcal{G} = \{G_1, \ldots, G_n\}$, model $p_\theta$, variational distribution $q_\phi$, sample size $S$
**Output:** Learned parameters $\theta$ and $\phi$
**repeat**
    **for** $G \in \mathcal{G}$ **do**
        Sample $\pi^{(1)}, \ldots, \pi^{(S)} \overset{\text{iid}}{\sim} q_\phi(\pi|G)$
        Obtain $\mathbf{A}^{(s)}$ from $(G, \pi^{(s)})$
        Set $p_\theta(G, \pi^{(s)}) = \frac{1}{|\Pi[\mathbf{A}^{(s)}]|} p_\theta(\mathbf{A}^{(s)})$
        Compute $\nabla_\phi \leftarrow \nabla_\phi L(\theta, \phi, G)$
        Compute $\nabla_\theta \leftarrow \nabla_\theta L(\theta, \phi, G)$
        Update $\phi$, $\theta$ using the gradients $\nabla_\phi$, $\nabla_\theta$
    **end for**
**until** convergence of the parameters $(\theta, \phi)$
```

Use arbitrary graph generating model (here: node-driven)

STATIC

Node-driven generation - **Priority Attachment**

- **Idea:** Build priority queue R of nodes based on distance function D to generate network topologies (no prior hyperparameters needed)
- **Intuition:** Which next node v_j should a node v_i be connected to? (each node has a local ranking of target nodes by their importance)

```
Algorithm 1 Priority Rank generative network model
Require: $D : V \times V \to \mathbb{R}$, distance function
Require: $V = v_1, \ldots, v_n$, a set of vertices
Require: $k$, the number of edges each vertex creates
1: for $i = 1$ to $n$ do
2:     compute the ranking $R_i$ using the distance function $D$
3:     for $j = 1$ to $k$ do
4:         $t \leftarrow sample(\langle 1, \ldots, n \rangle, P)$        /* random selectio
5:         $v_t \leftarrow R_i[t]$                                       /* priority attachm
6:         add edge $(v_i, v_t)$
7:     end for
8: end for
```

probability P of picking node is inversely proportional to ranking:

$$P(i) = \frac{1}{\sum_{k=1}^{n-1} \frac{1}{k}} \frac{1}{i} = \frac{1}{H_{n-1} i}$$

name	formulation
random distance	$D(v_i, v_j) \sim N(\mu, \sigma)$
degree distance	$D(v_i, v_j) = \frac{1}{C_D(v_j)+\varepsilon}$
betweenness distance	$D(v_i, v_j) = \frac{1}{C_B(v_j)+\varepsilon}$
closeness distance	$D(v_i, v_j) = \frac{1}{C_C(v_j)+\varepsilon}$
page rank distance	$D(v_i, v_j) = \frac{1}{C_P(v_j)+\varepsilon}$
euclidean 1-D distance	$D(v_i, v_j) = \left\lvert a_i^1 - a_j^1 \right\rvert$
euclidean 2-D distance	$D(v_i, v_j) = \sqrt{(a_i^1 - a_j^1)^2 + (a_i^2 - a_j^2)^2}$
cosine distance	$D(v_i, v_j) = 1 - \frac{v_i \circ v_j}{\lVert v_i \rVert \cdot \lVert v_j \rVert}$
aggregate distance	$D(v_i, v_j) = \sum_{k=1}^{m} w_k D(a_i^k, a_j^k)$
linear regression distance	$D(v_i, v_j) = W_{ij}\beta, W_{ij} = (a_i^1, \ldots, a_i^p, a_j^1, \ldots, a_j^p, \varepsilon)$
naive bayes classifier distance	$D(v_i, v_j) = \frac{P(C=1 \mid W_{ij})}{P(C=0 \mid W_{ij})+\varepsilon}$

Table 2. Distance functions

Morzy, M., Kajdanowicz, T., Kazienko, P. *et al.* "Priority Attachment: a Comprehensive Mechanism for Generating Networks."

STATIC

Node-driven generation - **Priority Attachment**

- **Example:**

$$D(v_i, v_j) = \begin{cases} |v_i[age] - v_j[age]|, & \text{if } v_i[sex] = v_j[sex] \\ |v_i[age] - v_j[age]| + 10, & \text{otherwise} \end{cases}$$

	name	age	sex
v_1	Alice	30	female
v_2	Bob	40	male
v_3	Cecil	25	male
v_4	Diana	20	female
v_5	Eve	35	female

Alice				**Bob**				**Cecil**			
rank	name	dist	prob	rank	name	dist	prob	rank	name	dist	prob
1	Eve	5	48%	1	Cecil	15	39%	1	Alice	15	31%
2	Diane	10	24%	1	Eve	15	39%	1	Bob	15	31%
3	Cecil	15	16%	3	Alice	20	13%	1	Diane	15	31%
4	Bob	20	12%	4	Diane	30	9%	4	Eve	20	7%

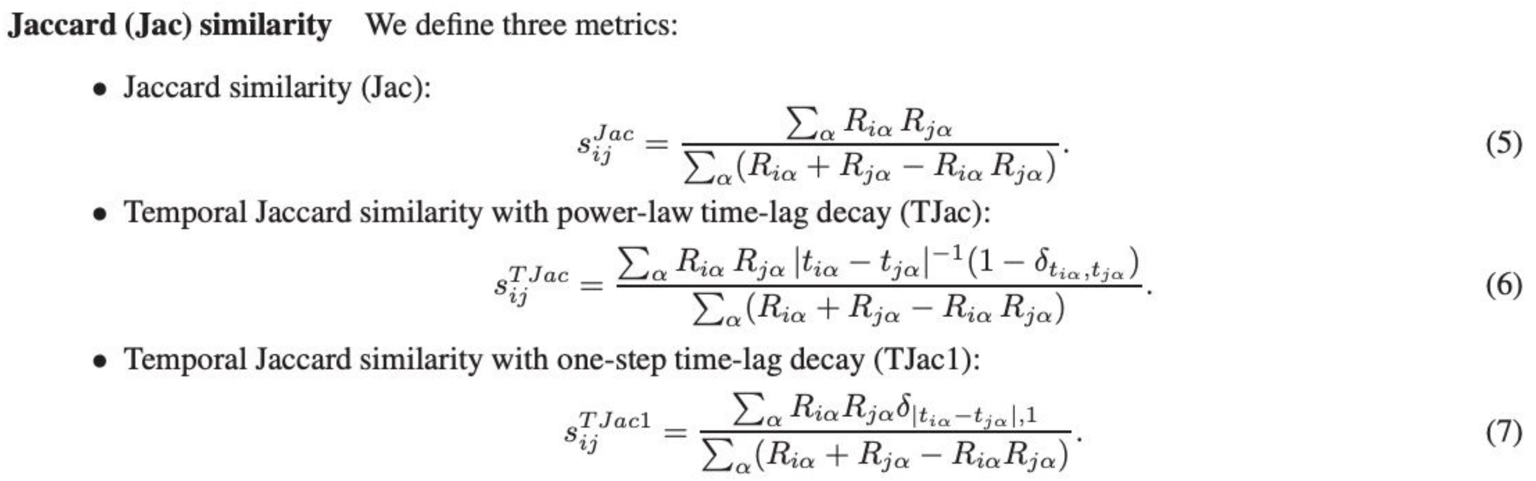
- **Limitations:**
  - finding suitable distance function is more or less brute-force (*idea:* use neural network)
  - amount of randomness is limited (*idea:* modify D or P to include more randomness)

Morzy, M., Kajdanowicz, T., Kazienko, P. *et al.* “Priority Attachment: a Comprehensive Mechanism for Generating Networks.”

DYNAMIC

Node-driven generation - **Priority Attachment**

- Compute similarity metric between nodes given an adoption time
- Use temporal similarity metrics as distance function to add dynamic aspect for priority attachment



Jaccard (Jac) similarity We define three metrics:

- Jaccard similarity (Jac):

$$s_{ij}^{Jac} = \frac{\sum_\alpha R_{i\alpha} R_{j\alpha}}{\sum_\alpha (R_{i\alpha} + R_{j\alpha} - R_{i\alpha} R_{j\alpha})}. \qquad (5)$$

- Temporal Jaccard similarity with power-law time-lag decay (TJac):

$$s_{ij}^{TJac} = \frac{\sum_\alpha R_{i\alpha} R_{j\alpha} \, |t_{i\alpha} - t_{j\alpha}|^{-1} (1 - \delta_{t_{i\alpha}, t_{j\alpha}})}{\sum_\alpha (R_{i\alpha} + R_{j\alpha} - R_{i\alpha} R_{j\alpha})}. \qquad (6)$$

- Temporal Jaccard similarity with one-step time-lag decay (TJac1):

$$s_{ij}^{TJac1} = \frac{\sum_\alpha R_{i\alpha} R_{j\alpha} \delta_{|t_{i\alpha} - t_{j\alpha}|, 1}}{\sum_\alpha (R_{i\alpha} + R_{j\alpha} - R_{i\alpha} R_{j\alpha})}. \qquad (7)$$

Liao, Hao, et al. "Temporal similarity metrics for latent network reconstruction: The role of time-lag decay."

DYNAMIC

Node-driven generation - **Connectivity of Components**

- **Persistent connected component (PCC)** p of graph G=(V,E) is a set of k vertices in V that are connected in the graph for l consecutive time steps
- **Idea:** compute largest, in terms of size and length, PCC in a dynamic graph
    - A maximal PCC is a PCC such that its vertex set is not included in a bigger vertex set connected on the same time steps *(condition 1)*, and the same vertex set is not connected on the previous time step *(condition 2)* or on the next time step *(condition 3)*
- **PICCNIC algorithm** to compute maximal PCCs in polynomial time
- Not a real model to generate graphs, but more analysis tool
    - *idea:*
    - combine with priority attachment to use as distance function (+ MMD)

Vernet, Mathilde, Yoann Pigne, and Eric Sanlaville. "A study of connectivity on dynamic graphs: computing persistent connected components."

Node-driven generation - **Markov Process Modelling**

- **Idea:** Nodes fixed, edges appear and disappear by making transitions from present to absent or vice versa with fixed rates per unit time
  - rates can differ from one node pair to another, depending on latent properties of the nodes
  - by choosing this dependence, we can model various kinds of dynamic network structure
- **Approach:**
  - **λ** - rate of edge appearance, **μ** - rate of edge disappearance (both in continuous time)
  - **p_1(t)** - edge exists at time t, **p_0(t)** - no edge exists at time t

$$p_1(t+\mathrm{d}t) = p_1(t) + \lambda p_0(t)\,\mathrm{d}t - \mu p_1(t)\,\mathrm{d}t,$$
$$p_0(t+\mathrm{d}t) = p_0(t) - \lambda p_0(t)\,\mathrm{d}t + \mu p_1(t)\,\mathrm{d}t,$$

→ derive per-snapshot probabilities for (dis-)appearance of edges

- **Extension:** more advanced modelling to replace markov processes

Zhang, Xiao, Cristopher Moore, and Mark EJ Newman. "Random graph models for dynamic networks."

Discussion

How to map failure propagation to graph?

- Fully connected graph by omitting not affected nodes
  - Edge driven graph generation
- Partial connected graph by capturing every node
  - Node driven graph generation