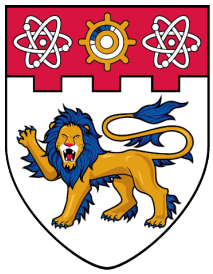
Project No: SCSE19-0810



NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

Cluster Analysis on Dynamic Graphs

Submitted by: Nguyen Ngoc Khanh
Matriculation Number: U1720451D

Supervisor: Asst Prof Ke Yiping, Kelly
Examiner: Assoc Prof Douglas Leslie Maskell

School of Computer Science and Engineering

A final year project report presented to the Nanyang Technological University in partial fulfilment of the requirements of the degree of Bachelor of Engineering

Oct 2020

Abstract

This project describes: a novel graph clustering algorithm that is an efficient extension of the Gibbs sampling under *distance dependent Chinese Restaurant Process, ddCRP* **?** for graph, a general cluster ensemble and cluster matching algorithm based on the concept of *Meta-Graph* **?**, an algorithm pipeline to tackle the dynamic graph clustering problem, intensive experiments to measure performance of the new algorithms.

Acknowledgements

I would like to express my sincere gratitude to my supervisor Asst Prof Ke Yiping, Kelly for the continuous research guidance and her valuable feedback and encouragement though-out the final year project.

Contents

List of Figures

Chapter 1

Introduction

1.1 Background

The emergence of Dynamic Graph Clustering Problem.

Network has been a powerful tool to model many complex systems across different fields such as telecommunications, computer networks, biological systems, knowledge representation, social network analysis. Such networks have many surprisingly similar properties. Hence, network science aims to study the complex networks that provides a universal framework to describe complex data, handles the emergence of big data era and contributes a huge impact to many different applications.

Under network analysis, graph clustering is a commonly used tool. For example, graph clustering helps to distribute cache servers to densely connected groups of users in a computer network that minimizes cache misses, graph clustering helps to optimize the transportation schedule by finding the inter-cluster links, graph clustering helps ads providers to target the most related ads to users. The analysis of graph clustering aims to decompose the network into multiple disjoint clusters. Clusters in network are usually defined as the densely connected groups of nodes and have a sparse connectivity between them. The study of clustering reveals the common properties of a collection of entities especially for large and complex data-sets.

Graph clustering has been studied for a long time, most of the research has been focused on the static graph problem. However, many real networks are dynamic where nodes and edges can appear and disappear over time and they are becoming more common in everyday life but the evolution of clustering property has not been considered rigorously.

It is although not a trivial task to convert a static graph clustering algorithm to a dynamic context. Since number of changes are usually much larger than number of nodes **?**, static graph clustering algorithms usually suffers from intractable computation. There is a need of developing algorithms that is capable to adapt to changes in dynamic networks.

1.2 Objectives

The study of Dynamic Graph Clustering

This project aims to improve a graph clustering algorithm and apply it in the context of dynamic graph. The work consists of studying node embedding techniques and clustering techniques, designing the dynamic graph clustering algorithm and evaluating the algorithm in both synthetic networks and real dynamic networks.

1.3 Research Significance

This study introduced two new algorithms: *ddCRP* for graphs and *MCLA* for clustering evolution detection that are applicable to detect the presence of community structure for weighted undirected networks in both static and dynamic settings. Furthermore, we constructed an efficient implementation of the *ddCRP* algorithm for graphs and analyzed its time complexity in the order of receptive field and cluster size per each Gibbs update.

Chapter 2

Related Work

A few approximation techniques for the graph clustering problem that inspired our approach will be discussed in this chapter.

2.1 Spectral Graph Clustering

A classic technique for graph clustering is spectral clustering which is introduced comprehensively in the textbook of Fan Chung **?**. The spectral methods typically using several eigen vectors corresponding to the set of smallest eigenvalues on normalized Laplacian matrix that provide an embedding of nodes satisfying the optimality on the chosen objective [1].

Recently, A. Saade et al introduced Bethe-Hessian matrix **?** for spectral clustering that is provable to perform down to the detect-ability threshold under stochastic block model **?** which is related to the *non-backtracking operator* firstly published by Florent Krzakala et al **?**. Later, another study from Lorenzo **?** extended A. Saade et al work for degree corrected stochastic block model.

Later on, which is inspired by the *non-backtracking operator*, Fei Jiang at al **?** published a novel edge embedding framework called *NOBE* which has a superior clustering performance comparing to many state of the art node embedding algorithms.

However, one disadvantage of spectral clustering is that it suffers from the high computational complexity since the algorithm is based on the estimation of eigen vectors. Nonetheless, it might be more applicable in the future with the work on randomized algorithms in numerical linear algebra **?** and the advancement in engineering **?**.

To wrap up this section, Marina Meila and Jianbo Shi **?** **?**, Pekka Orponen, Satu Elisa Schaeffer, and Vanesa Avalos Gaytan **?** have demonstrated the relationship between spectral clustering and random walk on graph. Hence, exploiting the short random walks in graph is a very promising approach to explore.

[1] Ratio cut, Normalized cut, Min-Max cut, etc

2.2 Graph Embedding

Graph embedding falls into four different categories **?**: *node embedding*, *edge embedding*, *hybrid embedding* and *whole graph embedding*. Nodes, edges or graph are typically represented by a point in Euclidean space. In this section, we will focus on node embedding and hybrid embedding only.

2.2.1 Node Embedding

In *node embedding*, each node is represented as a vector in a small d-dimensional Euclidean space. The common proximity measures are Euclidean norm or inner product.

2.2.1.1 Matrix Factorization

Matrix factorization aims to represent the graph as a matrix of neighbourhood proximity and use matrix factorization techniques to obtain the node embedding.

The most well-known technique is Laplacian Eigenmaps which factorizes the Laplacian matrix and get top-d the eigen vectors as the embedding of nodes.

The other method we wanted to discuss in this section is *Graph Factorization* **?**. The authors adopted stochastic gradient descent to approximate a maximum rank-d matrix of the adjacency with a regularization term that penalizes L2 norm.

$$f(Y, Z, \lambda) = \frac{1}{2} \sum_{(i,j) \in E} (Y_{i,j} - \langle Z_i, Z_j \rangle)^2 + \frac{\lambda}{2} \sum_i ||Z_i||^2 \tag{2.1}$$

Where $Y \in R^{|V| \times |V|}$ is the proximity matrix (adjacency matrix) and $Z \in R^{|V| \times d}$, each row vector of Z is an embedding vector of the corresponding node.

2.2.1.2 Random-walk-based and SkipGram

Random-walk-based method treats the network as a collection of node walks and the algorithm operates on this collection.

Bryan Perozz et al adopted *SkipGram* **?** from natural language processing and proposed *Deepwalk* **?**. The authors observed a power-law property in the distribution of nodes in short random walks and the distribution of words in documents. Then, they performed random-walk on graph data then used these short walks to train the SkipGram objective.

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq i\leq +c, i\neq 0} log(P(w_{t+i}|w_t)) \tag{2.2}$$

$$P(w_c|w_t) = \frac{exp(v_c^{'T}v_t)}{\sum_{w_i\in W} exp(v_i^{'T}v_t)} \tag{2.3}$$

Where w_i denotes the i-th word in a sentence, T denotes the length of that sentences, c denotes the context size. $P(w_c|w)$ is the probability of a context word given a word. $v_c^{'}$ and v_t are the corresponding context embedding vector and the word embedding vector.

In summary, *SkipGram* maximizes the log-likelihood of the occurrences of context words given a word and *Deepwalk* replaces word by node in a binary edge network. Their contribution motivates many subsequent studies on the random walks.

Aditya Grover and June Leskovec (*node2vec*) extended *Deepwalk* by providing two tunable parameters p and q guiding the random walk process that balances between depth-first-search and breath-first-search corresponding to homophily as opposed to structural equivalence. Their random walk process was described as: Consider a walk just traversed from a to b, let the next node in the walk be c, then the transition probability is as below:

$$P((...,a,b,c)|p,q,(...,a,b)) \propto \begin{cases} \frac{1}{p} & \text{if } d_{a,c}=0 \\ 1 & \text{if } d_{a,c}=1 \\ \frac{1}{q} & \text{if } d_{a,c}=2 \end{cases} \tag{2.4}$$

Where $d_{a,c}$ is the shortest distance from node a to node c. By setting $1/p \to 0$, the random walk becomes first-order non-backtracking which is described in **?**. By setting $1/q \to 0$, the random walk tends to traverse to the direct neighbours of node a. By setting $1/q \to +\infty$, the random walk tends to traverse as far as possible.

In *LINE* **?**, Jian Tang et al proposed two optimization objectives called first-order proximity and second-order proximity. First-order proximity models the presence of an edge (v_i, v_j) by $p_1(v_i, v_j)$:

$$p_1(v_i,v_j) = \frac{1}{1+exp(-u_i^T u_j)} \tag{2.5}$$

In order to produce a good node representation, they minimized the KL-divergence between the empirical distribution edge weights $w_{i,j}$ and the first-order proximity $p_1(v_i, v_j)$ which resulted to:

$$O_1 = -\sum_{(i,j)\in E} w_{i,j} p_1(v_i, v_j) \tag{2.6}$$

Second-order proximity models the prestige of a node λ_i by $p_2(\cdot|v_i)$. Each node plays two role: *vertex* and *context* to over vertices. They defined the probability of *context* v_j over *vertex* v_j as:

$$p_2(v_j|v_i) = \frac{exp(u_j^{'T} u_i)}{\sum_{k=1}^{|V|} exp(u_k^{'T} u_i)} \tag{2.7}$$

$$p_2(\cdot|v_i) = \prod_{v_j \in V} p_2(v_j|v_i) \tag{2.8}$$

Similar to first-order proximity, they minimized the KL-divergence between the empirical distribution prestige of a node λ_i and the second proximity $p_2(\cdot|v_i)$. They chose the prestige of a node by its degree so that the objective was reduced to:

$$O_2 = -\sum_{directed(i,j)\in E} w_{i,j} p_2(v_i|v_j) \tag{2.9}$$

Finally, in order to manage the weighted graph, they proposed an edge sampling scheme where probability is proportional to the edge weight.

It also is worth to mention the two techniques to approximate the full softmax (equation **??**) has been used in *SkipGram* model: *hierarchical softmax* **?** and *negative sampling* **? ?**. *Hierarchical softmax* handles the calculation of full softmax by a binary tree that reduces the time complexity of the calculation on the probability of vertex given context from $O(V)$ to $O(logV)$. *Negative sampling* on the other hand, approximate the probability of vertex given context by sampling the negative edges according to a noise distribution $P_n(v)$. The *negative sampling* objective is used to replace all $P(w_c|w)$ in the *SkipGram* model.

$$log\sigma(v_c^{'T} v) + \sum_{i=1}^{K} \mathbf{E}_{v_i \sim P_n(v)}[log\sigma(-v_i^{'T} v)] \tag{2.10}$$

2.2.1.3 Neural Network

The study of neural network on graph has emerged in recent years. Many proposed solution to graph problems are inspired by Convolutional Neural Network (CNN).

In *Graph Convolutional Network* (*GCN*) **?**, each layer weighted aggregates a node neighbourhood embedding vector [2], then filtered by a linear layer. The propagation rule is described as below:

$$H^{(l+1)} = \sigma(\widetilde{D}^{-1/2}\widetilde{A}\widetilde{D}^{-1/2}H^{(l)}W^{(l)} + b^{(l)}) \tag{2.11}$$

Where $\widetilde{A} = A + I$, A is the adjacency. $\widetilde{D}_{ij} = \sum_j \widetilde{A}_{ij}$. $H^{(l)} \in R^{|V| \times d^{(l)}}$ is the representation at layer (l), $d^{(l)}$ is the dimension at layer (l). $W^{(l)}$ and $b^{(l)}$ are weight and bias terms at layer (l). $\sigma(\cdot)$ is the activation.

Later on, Difan Zou et al **?** improved the calculation on *GCN* by making use of selective sampling for each layer. *LADIES*. In node classification task, stochastic gradient descent only use a subset of nodes to train the model. Hence, in order to save the calculation step, they used a sampled kernel matrix which only has the corresponding neighbours that are related to the set of labeled nodes.

In *Gaussian-Induced Convolution*, Jiatao Jiang **?** proposed a novel method that utilizes both first-order statistics (mean) and second-order statistics (variance). The two presented types of layer are Convolution: Edge-Induced GMM and Coarsening: vertex-induced GMM. In edge induced GMM, inspired by the work on *Fisher vector* **?**, each vertex embedding is calculated based on the derivative of probability density of the receptive field w.r.t a gaussian mixture. It exploits the local information w.r.t to a node using the concept of receptive field in graph. In vertex induced GMM, vertices are partitioned using EM algorithm on GMM. It reduced the computational complexity by looking to the graph in a more global view as merging the nodes into clusters. It is important to highlight that, in vertex partitioning using GMM, the kernel trick has been used in order to reduce the main calculation of the algorithm.

Despite the success of neural network, neural network on graph is typically suitable for supervised learning tasks. In order to solve the unsupervised tasks where no label is provided, an autoencoder often takes place with careful designed regularization constraints on the hidden embedding which typically requires the expert experience.

[2] the weight depends on the respective kernel

2.2.2 Hybrid Embedding

Hybrid embedding discussed in this section is the embedding of nodes and clusters together.

The two hybrid embedding techniques is being discussed in this section are ComE and ComE+ which both are the work of Sandro Cavallar et al.

In *ComE* **?**, the node embedding (LINE **?**) and community embedding are jointly optimized together. Community embedding objective used in *ComE* was the likelihood of a gaussian mixture.

$$O_3 = -\frac{\beta}{K}\sum_{i=1}^{|V|} log \sum_{k=1}^{K} \pi_{ik}\mathcal{N}(\phi_i|\psi_k, \Sigma_k) \tag{2.12}$$

It is noteworthy that, maximizing community embedding log likelihood of GMM was relaxed using log concavity. The objective at equation **??** was replaced by:

$$O_3^{'} = -\frac{\beta}{K}\sum_{i=1}^{|V|}\sum_{k=1}^{K} log\pi_{ik}\mathcal{N}(\phi_i|\psi_k, \Sigma_k) \tag{2.13}$$

In *ComE+* **?**, they extended the work of *ComE* by employing an infinite number of clusters model *IGM* **?** using stick-breaking process. Then, they adopted the variational inference approach **?** to cast the maximum posterior problem of detecting community into a minimization of the KL divergence between variational distribution and the posterior.

Hybrid embedding usually has their advantages on their predefined problem. Hence, they are not applicable to produce a meaningful embedding for the general problem. **?**

2.3 Dynamic Graph Clustering

With regard to Dynamic Graph Clustering, Giulio Rossetti et al **?** published a comprehensive survey on techniques have been used. They classified operations in dynamic graph clustering evolution are as follow:

- Birth (New): First appearance of a cluster.
- Death: The vanishing of a cluster.
- Merge (Join): Two or more clusters merge into one.
- Split: One cluster splits into two or more components.
- Resurgence: A community vanishes for a short period and comes back.
- Grow: A cluster accepts nodes.
- Contraction: A cluster rejects nodes.
- Continue: A cluster remains unchanged.

2.4 Relation to our work

These presented methods have inspired our approach in different ways. For example, spectral clustering has settled a very clear intuition to the problem, the concept of receptive field in neural network brought an idea of the selective customers to calculate the probability (will be discussed in Method), the idea of jointly optimizing two objectives in *ComE* has inspired the idea of the greedy clustering algorithm using both information from node embedding and graph structure, the idea of infinite clusters in *ComE+* has inspired us about a random process that does not limit the number of clusters that is more suitable in to dynamic set up. Finally, a concept in network science, preferential attachment where a newly created node is more probable to connect to a node with a higher degree which is very similar to Chinese Restaurant Process where a new customer is more probable to connect to a table with more customers already sit guided us to the distance dependent Chinese Restaurant Process which is the main clustering algorithm in this project.

Chapter 3

Method

3.1 Preliminaries

3.1.1 Node Embedding

Definition 1 (Node Embedding)**.** *Given a weighted undirected graph* $G = (V, E, w)$ *where* V *is the set of vertices,* $E \subseteq V \times V$ *is the set of edges, and* $w : V \times V \mapsto R^+$ *is the edge weight. A node embedding of dimension* d *is a function* $u : V \mapsto R^d$ *that maps each vertex to a vector.*

Deepwalk **?** was empirically provable to produce a meaningful representation of nodes for unweighted directed graph. The detailed algorithm is as follow:

Algorithm 1 Deepwalk

Input:
$G = (V, E, w)$: Undirected weighted graph
d: Embedding dimension
c: Context size
γ Walks per vertex
l: Walk length

Output:
$X \in R^{|V| \times d}$: Node embedding

```
Initialization: Sample X
for i = 1 to γ do
    O = Shuffle(V)
    for v ∈ O do
        walk = RandomWalk(G, v, l)
        SkipGram(X, walk, c)
    end for
end for
```

Algorithm 2 SkipGram

Input:
X: node embedding
$walk$: walk, an array of vertices
c: Context size

for $v_i \in walk$ **do** ▷ v_i is the i-th item in the array
  for $v_j \in walk[i-c, i+c]$ **do** ▷ array[a, b] is the subarray
    $J(X) = -log(Pr(u'_j|u_i))$
    $X = X - \alpha \frac{\partial J}{\partial X}$
  end for
end for

Algorithm 3 RandomWalk

Input:
$G = (V, E, w)$: Undirected weighted graph
v_1: Start vertex
l: Walk length

Output:
$walk$: walk

$walk = [v_1]$
while $walk.length() < l$ **do**
  $v = Choose(G, walk[-1])$ ▷ $array[-1]$ is the last item of the array
  $walk.push\_back(v)$
end while

Choose function returns a uniformly random out-going neighbour of the input node. Formally,

$$ChooseProbability(v_j|v_i) \propto \begin{cases} 1 & \text{if } (v_i, v_j) \in E \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

Probability of context j given vertex i is given by **?**:

$$Pr(u'_j|u_i) = \frac{exp(u_j'^T u_i)}{\sum_{k=1}^{|V|} exp(u_k'^T u_i)} \tag{3.2}$$

Where u'_i and u_i are the embedding of vertex v_i when it is treated as "context" and "vertex" respectively.

3.1.2 Community Detection

Definition 2 (Community Detection)**.** *Given a graph $G = (V, E)$ and a node embedding $X \in R^{n \times d}$. Define a partition P of V is a set of disjoint non-empty subsets of V such that union of P elements is V.*

$P = \{P_k\}_{k=1}^{K}$ where

- $P_k \neq \emptyset$
- $\cup_{k=1}^{K} P_k = V$
- $P_i \cap P_j = \emptyset$ *for all* $P_i \neq P_j$

A community detection algorithm returns a partition of vertices.

Different from the traditional *Chinese Restaurant Process* (*CRP*), in the work of David M. Blei and Peter I. Frazier on *distance dependent Chinese Restaurant Process*, *ddCRP* **?** generalized the sitting arrangement of customers by a unweighted directed graph. Each vertex is corresponding to a customer, each weakly connected component is corresponding to a table. In this model, each vertex either links to another vertex or links to itself. Let z be the vector represent the linking process such that $z[i]$ is the customer that customer i links to. Then, *ddCRP* defines the conditional probability as follow **?**:

$$P(z[i] = j \mid z[0 : i-1], \alpha) \propto \begin{cases} \alpha & \text{if } j = i \\ f(i, j) & \text{if } j \neq i \end{cases} \tag{3.3}$$

Where α is the concentration parameter and $f(i, j)$ is defined as the decay function. The decay function controls the probability of two customers sitting in the same table. If the decay function equals 1 for every pair of customers, the model is equivalent to the traditional *CRP*.

Let x be the observations. In Gibbs sampling, each latent variable is sampled given all other latent variables being observed. The Gibbs sampling probability is as thus **?**:

$$P(z[i] = j|(z - z[i]), \alpha, x, G_0) \propto P(x|(z - z[i] + j), G_0)P(z[i] = j|(z - z[i]), \alpha) \tag{3.4}$$

Where $(z - z[i])$ denotes the customer assignment without $z[i]$, $(z - z[i] + j)$ denotes the customer assignment where $z[i]$ is replaced by j. G_0 denotes the prior distribution on x.

The first term is the likelihood of data given the new customer assignment $(z - z[i] + j)$ and prior G_0. The second term is *ddCRP* process probability (equation **??**). The authors of **?** factorized the first term into a product of each table likelihood.

$$P(x|(z - z[i] + j), G_0) = \prod_{t \in T} P(x_t|G_0) \tag{3.5}$$

Where T is the set of tables. This leads to an efficient Gibbs sampling scheme:

$$P(z[i] = j \mid (z{-}z[i]), \alpha, x, G_0) \propto \begin{cases} \alpha & \text{if } j = i \\ f(i,j) & \text{if } j \neq i \text{ and no table join} \\ f(i,j)\frac{P(x_{t_{ij}}|G_0)}{P(x_{t_i}|G_0)P(x_{t_j}|G)} & \text{if } j \neq i \text{ and table join} \end{cases} \tag{3.6}$$

Where t_i and t_j is the table of customer i and j, t_{ij} is the join table.

3.1.3 Cluster Ensemble

Definition 3 (Meta-Graph **?**). *Given a set of U of n elements, a clustering result is defined as a family of clusters/subsets of U: $C \subseteq \mathcal{P}(U)$. Each meta-node is defined as an element c of C. Construct meta-edge by the Jaccard similarity (intersection over union): $w_{i,j} = \frac{|c_i \cap c_j|}{|c_i \cup c_j|}$*

Alexander Strehl and Joydeep Ghosh **?** introduced an algorithm to ensemble clustering results namely MCLA algorithm based on the concept of the *Meta-Graph*. The algorithm consists of four major steps.

1. Construct the *Meta-Graph* from clustering.
2. Partition the *Meta-Graph* into K-balance meta-clusters.
3. Construct the probability of a given node belongs to a meta-cluster by averaging all clusters.
4. For each node, decide its corresponding meta-cluster based on the calculated probability.

3.1.4 Data marginal likelihood

Assuming data points distribute according to a Gaussian, choosing the Normal-inverse-Wishart (NIW) prior, the data marginal likelihood is hence calculated according to Kevin P. Murphy work **?** (page 21).

$$P(\mu, \Sigma|D) = NIW(\mu, \Sigma|m_N, k_N, v_N, S_N) \tag{3.7}$$

$$k_N = k_0 + N \tag{3.8}$$

$$v_N = v_0 + N \tag{3.9}$$

$$m_N = \frac{k_0 m_0 + N\overline{x}}{k_N} \tag{3.10}$$

$$S_N = S_0 + S + k_0 m_0 m_0^T - k_N m_N m_N^T \tag{3.11}$$

$$P(D) = \frac{1}{\pi^{Nd/2}} \frac{\Gamma_d(v_N/2)}{\Gamma_d(v_0/2)} \frac{|S_0|^{v_0/2}}{|S_N|^{v_N/2}} (\frac{k_0}{k_N})^{d/2} \tag{3.12}$$

Where $S \triangleq \sum_{i=1}^{N} x_i x_i^T$ as the uncentered sum-of-squares matrix.

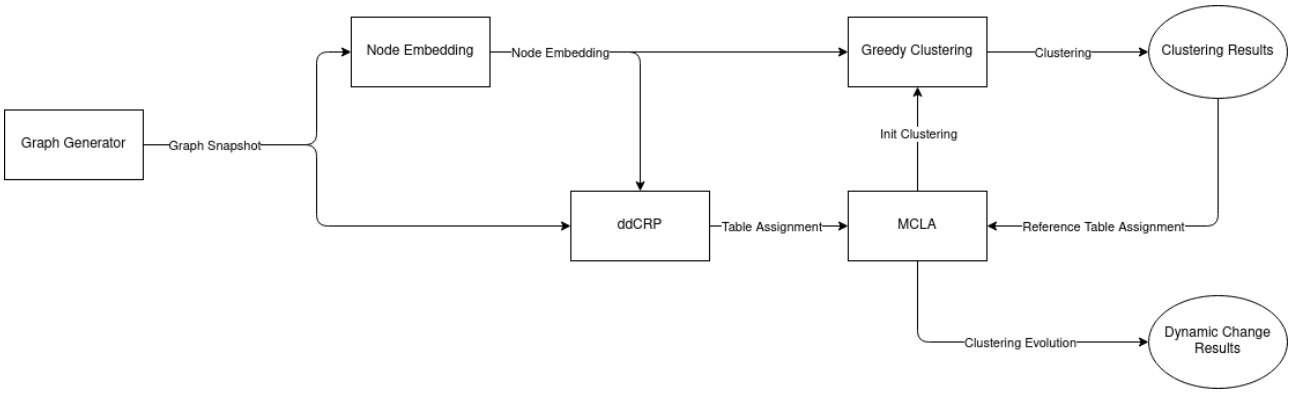


Figure 3.1: Algorithm Pipeline

3.2 Implementation

Our method consists of a node embedding algorithm (*Deepwalk*), a greedy clustering algorithm (*K-means*), the modified version of *ddCRP* and the modified version of MCLA. The whole pipeline is described in Figure **??**.

In each iteration, Graph Generator generates graph snapshots. The graph snapshots are then passed to Node Embedding algorithm and *ddCRP*. *ddCRP* then takes the results from Node Embedding to generate a series of table assignments which are the high probability region. The *MCLA* algorithm takes the table assignments from *ddCRP* together with the old clustering result generates a initial clustering. Greedy Clustering algorithm takes the node embedding and the generated initial clustering then produces the final clustering result. At the same time, *MCLA* algorithm produces the clustering evolution by comparing the old clustering result to the new clustering result.

3.2.1 Node Embedding

In the node embedding algorithm, we chose *Deepwalk* as the function converting a graph snapshot into a d-dimensional representation. In order to handle the weighted graph, we modified the random-walk probability into:

$$ChooseProbability(v_j|v_i) \propto \begin{cases} w(v_i, v_j) & \text{if } (v_i, v_j) \in E \\ 0 & \text{otherwise} \end{cases} \tag{3.13}$$

Where $w(v_i, v_j)$ is the weight of edge (v_i, v_j). This is equivalent to consider an edge of weight w as w edges.

3.2.2 *ddCRP*

Inspired by the work of David M. Blei and Peter I. Frazier, we extended the Gibbs sampling algorithm on *ddCRP* for graph clustering problem. In the context of graph, we relaxed the Gibbs sampling scheme by limiting the non-zero probabilities only to the vertex in the receptive field. Define h-order receptive field by the matrix $A_h^* = (A+I)^h$ where A is the adjacency. The Gibbs sampling scheme is modified to:

$$P^*(z[i] = j \mid (z - z[i]), \alpha, x, G_0) = r(i,j)P(z[i] = j \mid (z - z[i]), \alpha, x, G_0) \quad (3.14)$$

Where $r(i,j)$ is:

$$r(i,j) = \begin{cases} 1 & \text{if } A_h^*(i,j) \neq 0 \\ 0 & \text{if } A_h^*(i,j) = 0 \end{cases} \quad (3.15)$$

We then defined the decay function $f(i,j)$ as:

$$f(i,j) = exp(-s(u_i - u_j)^2) \quad (3.16)$$

Where $s \in R^+$ (scale or precision) is a tune-able hyper-parameter which controls the cluster sizes. This formulation can be seen as assuming each vertex is a Gaussian with precision matrix $2sI$, the decay value is hence proportional to the likelihood of the other vertex given the Gaussian.

We assume each cluster distributes according to a Gaussian and choose NIW as the prior. The prior parameters are initialized according to the book **?** (sec 4.6.3.2).

The calculation of $P^*(z[i] = j \mid (z - z[i]), \alpha, x, G_0)$ breaks down into. From **??**, calculate $r(i,j)$ and calculate $P(z[i] = j \mid (z - z[i]), \alpha, x, G_0)$ according to **??**. In **??**, calculate the data likelihood term (the first term) with the assumption that cluster points distribute according to a Gaussian (sec Data marginal likelihood).

The algorithm is hence described as below:

Algorithm 4 Node clustering using Gibbs sampling on ddCRP

Input:
$X \in R^{|V| \times d}$: node embedding
$G = (V, E, w)$: Undirected weighted graph
s: decay scale
T: Number of iterations

Output:
Table assignment at every iteration

Data Structures:
G_C: customer graph: an unweighted directed graph
G_T: customer-table graph: a bipartite graph between customer nodes and table nodes. Each table is associated with a table likelihood calculated based on equation **??**

```
Initialization: table assignment
for t = 1 to T do
    for i = 1 to |V| do
        Unlink vertex i with its parent in G_C, Update G_T.
        Calculate P*(z[i] = j | (z − z[i]), α, x, G_0)
        Random sampling a vertex j according to P*
        Link vertex i to chosen j in G_C, Update G_T.
    end for
    Record the table assignment at iteration t
end for
```

It is notably that the calculation of $P^*(z[i] = j \mid (z - z[i]), \alpha, x, G_0)$ is reduced from $O(|V|)$ to $O(d^h_{max})$ where d_{max} is the maximum degree of the graph. Additionally, all calculations can be performed in log-scale that totally avoids the problem of floating-point overflow.

The asymptotic complexity is described as follow. In each iteration, the algorithm performs $n = |V|$ updates. Each update consist of (1) Unlink the source node: $O(log(neighboursize))$, update the table assignment, this consists of (1.1) find the new weakly connected component correspond to the source node after the unlink, since the graph has at most one out-going edge for each node, the time complexity for depth-first search is $O(componentsize)$, (1.2) verify whether the old parent was included in the new connected component: $O(log(componentsize))$, in the worst case, split. (1.3) If split, update the table assignment. this step consists of (1.3.1) find the new weakly connected component correspond to the old parent of the source node: $O(componentsize)$. (1.3.2) update the bipartite graph, (1.3.2.1) link customers to new table: $O(componentsize)$, (1.3.2.2) update table likelihood: $O(tablelikelihood)$. (2) For each target node in the receptive field, this multiplies the complexity by $receptivefieldsize$, (2.1) calculate the transition probability, (2.1.1) look up for decay value: $O(1)$, (2.1.2)

Check if there is a table join: $O(1)$, in the worst case, there is a table join, calculate the joint table likelihood: $O(componentsize)$. (2.2) random sampling: $O(receptivefieldsize)$, (2.3) Link the source node: $O(log(neighbourhoodsize))$ to the target node, update the table assignment, this consists of (2.3.1) check whether there is a table join: $O(1)$, in the worst case, there is a table join, (2.3.2) update the bipartite graph (2.3.2.1) link customers to new table: $O(componentsize)$, (2.3.2.2) update table likelihood: $O(tablelikelihood)$. The total time complexity for each update is:

$$O(tablelikelihood + receptivefieldsize \times componentsize) \tag{3.17}$$

Given fixed dimension d, the time complexity for *tablelikelihood* is in the order of $O(componentsize)$ (we assumed the floating-point calculation takes a constant time). $receptivefieldsize$ is in the order of d_{max}^h where d_{max} is the maximum degree of the network. *componentsize* which is typically chosen before hand. The update time complexity is hence:

$$O(d_{max}^h \times c) \tag{3.18}$$

3.2.3 *MCLA*

Since the trajectory of MCMC algorithm lies on the high probability region, we adopted a cluster ensemble method to combine the results from Gibbs sampling to obtain an average of clustering results. Inspired by the concept of *Meta-Graph*, we designed an clustering ensemble algorithm that is capable to produce informative responses from clustering evolution.

The algorithm is described as below:

Algorithm 5 MCLA

Input:

$C_0 \subseteq \mathcal{P}(U)$: reference clustering, a family of subsets of U

$C \subseteq \mathcal{P}(U)$: new clustering, a family of subsets of U

Output:

P: a partition of U

Response of new clusters, joined clusters.

Initialization: construct meta-graph according to definition **??**

Filtering: Filter out all disconnected components

Clustering: Partition the meta-graph.

Voting: For each node, decide the corresponding meta-cluster.

Response: A meta-cluster is a new cluster if there is no reference cluster belongs to it. A meta-cluster is a join cluster if there is more than one reference clusters belong to it.

Chapter 4

Experiment

Our initialization for table assignment was chosen such at each customer is at a separate table at the beginning. Louvain method **?** was used in MCLA algorithm due to its low time complexity.

4.1 Synthetic Networks

4.1.1 Setup

In this section, the performance of proposed method is demonstrated using synthetic network as described below:

Algorithm 6 Power-Law clustering generator

Input:
γ: gamma constant
K: number of clusters

Output:
$size$: relative sizes of clusters distributed according to density $f(x) \propto x^{-\gamma}$

$size = \{\frac{k-0.5}{K}\}_{k=1}^{K}$
$size = size^{\frac{1}{1-\gamma}}$ ▷ Element-wise operation
$size = \frac{size}{size.sum()}$ ▷ Element-wise operation

Stochastic Block Model is used to generate the networks with power-law cluster sizes, average degree and intra-cluster edge probability over inter-cluster edge probability $p_{in}/p_{out} = 10$.

We performed grid search over the set of parameters as follow: Number of vertices $|V| \in \{500, 1000, 1500, 2000\}$, average degree $avgdeg \in \{10, 20, 30, 40, 50\}$, scale parameter $s \in \{1000, 2000, ..., 29000\}$

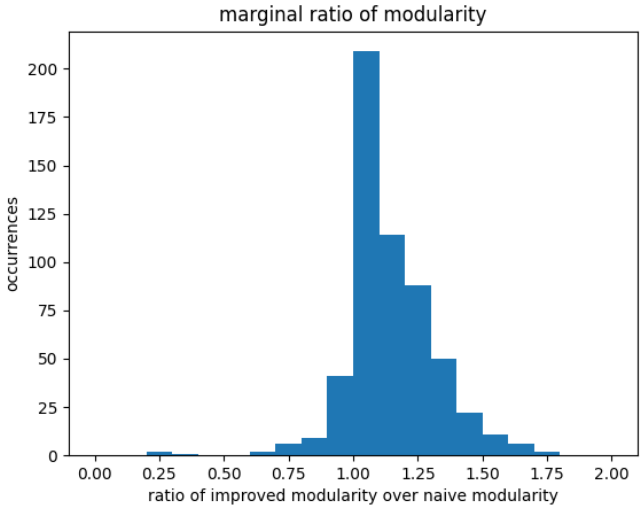


Figure 4.1: Distribution of ratio $\frac{\text{modularity(ddcrp-mcla)}}{\text{modularity(kmeans++)}}$

The same node embedding from *Deepwalk* was used for different scale parameter. We fixed these parameters: $\gamma = 2.5$, embedding dimension $d = 50$, walks per vertex $\gamma = 2|E|/|V|$, context size $c = 5$, walk length $l = 3c$, receptive field hop $h = 1$, *Deepwalk* epochs = 10 and *ddCRP* epochs = 10 for each run. We took only the last 5 iterations from *ddCRP* as the set of stable states.

We compared two versions: initialized Kmeans using ddCRP and MCLA **ddcrp-mcla** and initialized Kmeans "k-means++" **kmeans++** with 10 times of initialization which is builtin within sklearn library.

4.1.2 Results

4.1.2.1 Modularity

Modularity of each algorithm was evaluated for each setup. Figure **??** represent the distribution of the ratio $\frac{\text{modularity(ddcrp-mcla)}}{\text{modularity(kmeans++)}}$. The figure reflects the superior performance of **ddcrp-mcla** initialization for kmeans as compare to **kmeans++**. On average, **ddcrp-mcla** performs better than **kmeans++** by 14.1% with a standard deviation of 17.5%.

Figure **??**, **??**, **??**, **??**, and **??** show that the modularity obtained by **ddcrp-mcla** always better than **kmeans++** on all different predicted number of clusters even if the estimation diverges from the true number of clusters (50).

The ratio $\frac{\text{modularity(ddcrp-mcla)}}{\text{modularity(kmeans++)}}$ depends on its predicted number of clusters as shown in figure **??**. The ratio appears to achieve its best performance at the predicted number of clusters between 50 and 100 where the actual number of

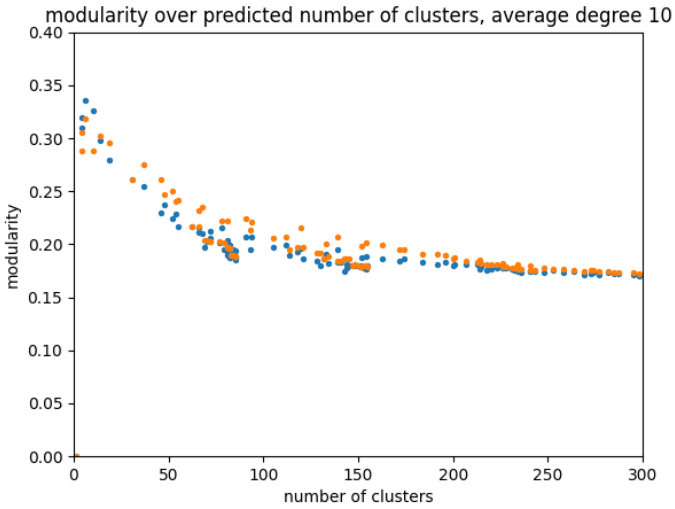


Figure 4.2: Modularity w.r.t predicted number of clusters for average degree 10 (**ddcrp-mcla**: orange, **kmeans++**: blue)

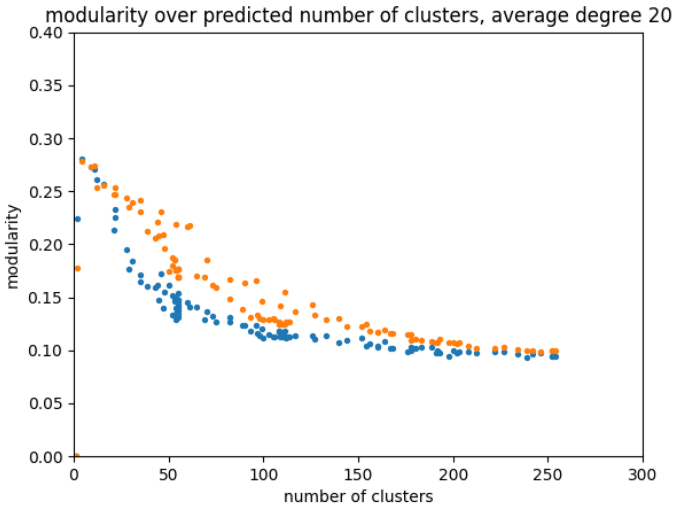


Figure 4.3: Modularity w.r.t predicted number of clusters for average degree 20 (**ddcrp-mcla**: orange, **kmeans++**: blue)

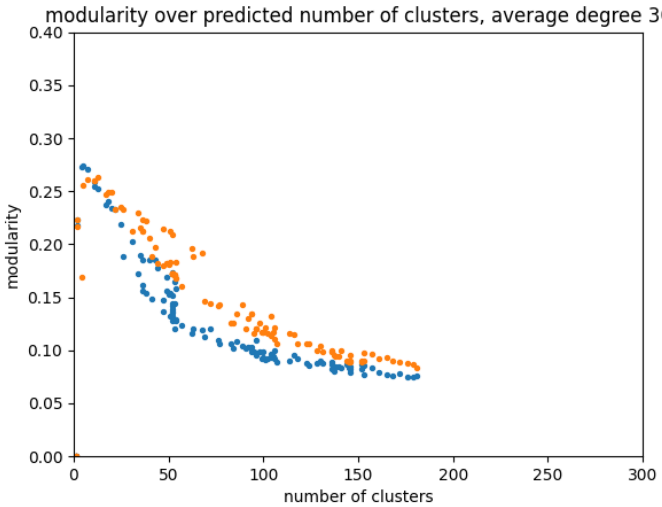


Figure 4.4: Modularity w.r.t predicted number of clusters for average degree 30 (**ddcrp-mcla**: orange, **kmeans++**: blue)

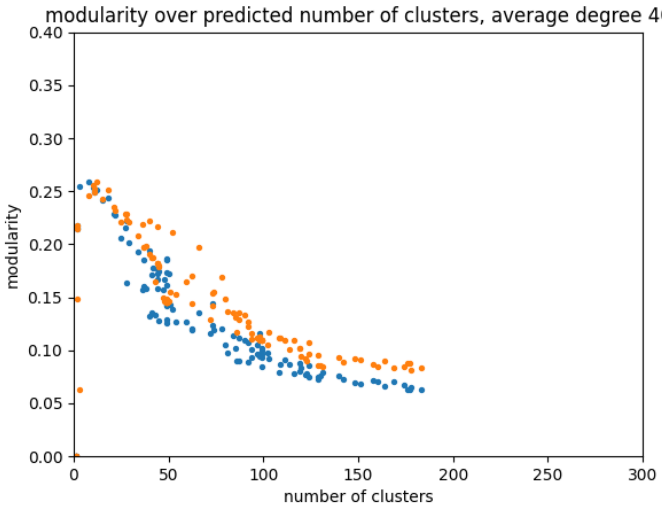


Figure 4.5: Modularity w.r.t predicted number of clusters for average degree 40 (**ddcrp-mcla**: orange, **kmeans++**: blue)

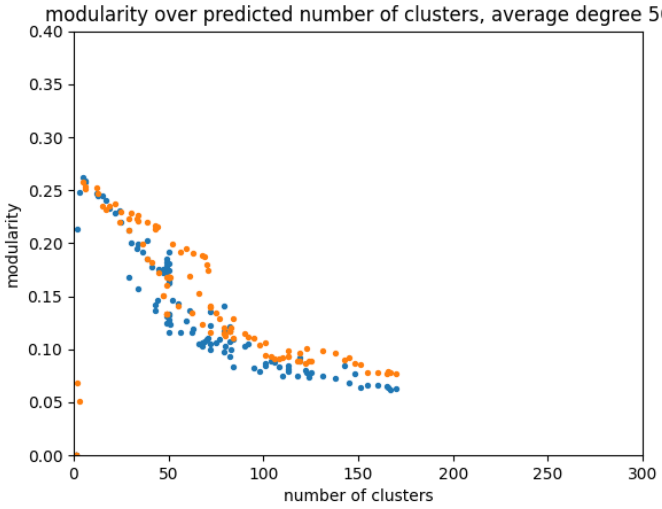


Figure 4.6: Modularity w.r.t predicted number of clusters for average degree 50 (**ddcrp-mcla**: orange, **kmeans++**: blue)

clusters is 50.

4.2 Real Networks

4.2.1 Setup

In this experiment, we used the dataset at **?**. The temporal network consists of 986 nodes and 332334 temporal edges in the time span of 803 days. We firstly sorted the temporal edges by its timestamp. Then we splitted the set of temporal edges into 1000 equal folds where each fold has roughly 332 edges. We set the window size of 10 folds. After each iteration, we slided the window by 1 fold. We then took the two edge timestamps of the window and filtered all the temporal edges between these timestamps as the set of edges for the graph snapshot.

Similar to the previous experiment, we set embedding dimension $d = 50$, walks per vertex $\gamma = 2|E|/|V|$, context size $c = 5$, walk length $l = 3c$, *Deepwalk* epochs $= 10$ and *ddCRP* epochs $= 10$ for each run. We took only the last 5 iterations from *ddCRP* as the set of stable states.

We performed grid search over the set of parameters as follow. Receptive field hop $h \in \{1, 2\}$, scale parameter $s \in \{1000, 2000, ..., 8000\}$

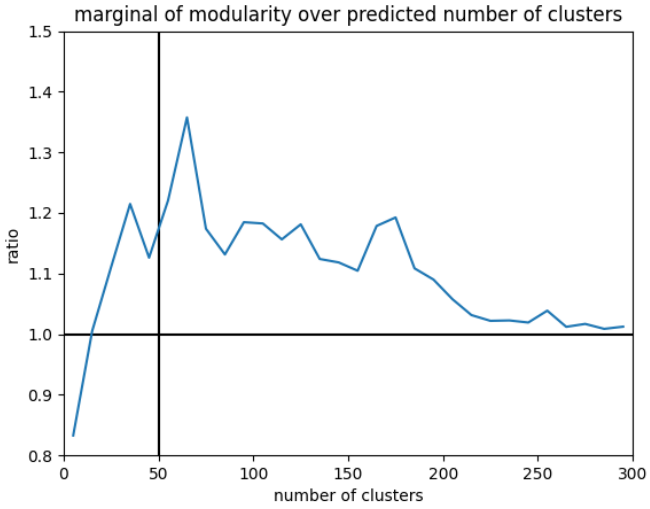


Figure 4.7: ratio $\frac{\text{modularity(ddcrp-mcla)}}{\text{modularity(kmeans++)}}$ over predicted number of clusters

4.2.2 Results

4.2.2.1 Modularity

Figure **??** shows the ratio between $\frac{\text{modularity(ddcrp-mcla)}}{\text{modularity(kmeans++)}}$ over different predicted number of clusters. If we limit the predicted number of clusters to be greater than 60, the **ddcrp-mcla** method performs better than **kmeans++** by 4.4% with a standard deviation of 5.6%.

4.2.2.2 Clustering Evolution

Our *MCLA* algorithm is capable to produce some useful response from the evolution of clustering. Noted that, these results are completely automated. Textbox **??**.

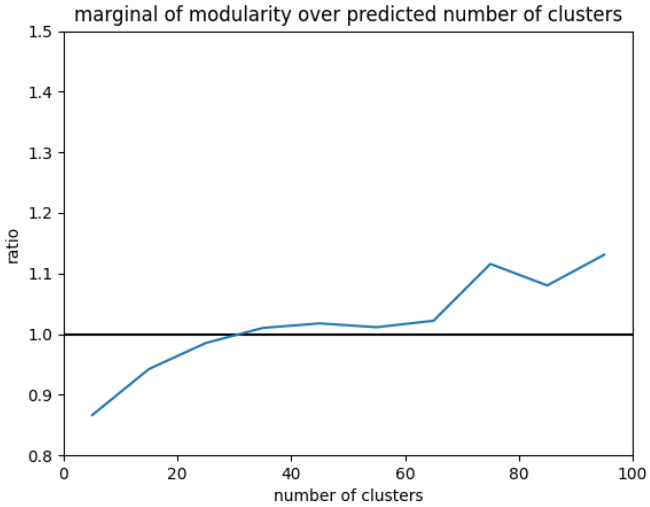


Figure 4.8: ratio $\frac{\text{modularity(ddcrp-mcla)}}{\text{modularity(kmeans++)}}$ over predicted number of clusters

```
message type: join
9 nodes remained
3 nodes joined
0 nodes left
message type: old
2 nodes remained
0 nodes joined
0 nodes left
message type: old
0 nodes remained
25 nodes joined
7 nodes left
message type: old
3 nodes remained
0 nodes joined
0 nodes left
message type: join
418 nodes remained
0 nodes joined
153 nodes left
...
```

Chapter 5

Conclusion and Future Work

The experiment shows a superior performance on modularity quality metric to the new algorithm as compare to the naive clustering algorithm in the regime where number of clusters are large as it exploits the neighbourhood information rather than the node embedding only. Additionally, the new algorithm introduces a reasonable time complexity on the matching between new graph snapshot clustering and the reference clustering.

For future work, there should be modification on *ddCRP* to adopt different cluster size distributions. Many other decay functions are possible to be experimented for better performance. The *MCLA* matching scheme is still in its initial version, there is not any quality metric to measure its matching performance. The matching performance is still greatly dependent on the performance of the clustering algorithm.

Chapter 6

Appendix

Proof of **??**. :

$$P(A|BCD) = \frac{P(ABCD)}{P(BCD)}$$

$$P(ABCD) = P(C|ABD)P(ABD)$$

Assume that A and D are independent, B and D are independent.

$$P(ABD) = P(AB)P(D) = P(A|B)P(B)P(D)$$

$$P(A|BCD) = \frac{P(C|ABD)P(A|B)P(B)P(D)}{P(BCD)}$$

Let A be $z[i] = j$, B be $((z - z[i]), \alpha)$, x be C and G_0 be D. We have:

$$P(z[i] = j|(z - z[i]), \alpha, x, G_0) \propto P(x|(z - z[i] + j), \alpha, G_0)P(z[i] = j|(z - z[i]), \alpha) \tag{6.1}$$

We have x and α are independent,

$$P(x|(z - z[i] + j), \alpha, G_0) = P(x|(z - z[i] + j), G_0) \tag{6.2}$$

QED

□

Proof of Algorithm **??**. :

Let $X \sim \mathcal{U}(0, 1)$

For convenience, we treat $P[x \in [a, b]] = -P[x \in [b, a]]$, $P[x \in [a, b]]$ is positive when $a < b$

Define probability density of X as $d_X(x = x_0) = \lim_{\Delta x \to 0} \frac{P[x \in [x_0, x_0 + \Delta x]]}{\Delta x}$

Let $Y = X^\delta$, we have:

$d_Y(y = y_0) = \lim_{\Delta y \to 0} \frac{P[y \in [y_0, y_0 + \Delta y]]}{\Delta y}$

Let $y_1 = x_1^\delta$, hence $y_1 + \Delta y = (x_1 + \Delta x)^\delta = x_1^\delta + \delta x_1^{\delta - 1}\Delta x + \sum_{i=2}^{+\infty} \frac{f^{(i)}(x_1)\Delta x^i}{i!}$

Or $\Delta y = \delta x_1^{\delta-1}\Delta x + \sum_{i=2}^{+\infty} \frac{f^{(i)}(x1)\Delta x^i}{i!}$
$d_Y(y = y_1) = \lim_{\Delta y \to 0} \frac{P[y \in [y_1, y_1+\Delta y]]}{\Delta y} = \lim_{\Delta y \to 0} \frac{P[x \in [x_1, x_1+\Delta x]]}{\Delta y} = \lim_{\Delta y \to 0} \frac{\Delta x}{\Delta y}$
$d_Y(y = y_0) = \lim_{\Delta x \to 0} \frac{1}{\delta x_1^{\delta-1} + \sum_{i=2}^{+\infty} \frac{f^{(i)}(x_1)\Delta x^{i-1}}{i!}} = \frac{1}{\delta} x_1^{1-\delta}$
$d_Y(y = y_0) = \frac{1}{\delta} x_1^{1-\delta} = \frac{1}{\delta}(y_1^{1/\delta})^{1-\delta} = \frac{1}{\delta} y_1^{(1-\delta)/\delta}$
Choose $\delta = \frac{1}{1-\gamma}$, so $d_Y(y = y_0) \propto y^{-\gamma}$
QED

□