THE UNIVERSITY OF QUEENSLAND

AUSTRALIA

Network Embedding For large Networks

by Xiaoguang Ye

School of Information Technology and Electrical Engineering,
University of Queensland.

Submitted for the degree of Master of Computer Science Management

in the division of Computer Science

November & 2019.

11 Sylvan Rd, Toowong QLD 4066

Tel. 0452 503 680

October 21st, 2019

November 1st, 2019

Prof Amin Abbosh
Acting Head of School
School of Information Technology and Electrical Engineering
The University of Queensland
St Lucia QLD 4072

Dear Professor Abbosh,

In accordance with the requirements of the Degree of Master of Computer Science Management in the School of Information Technology and Electrical Engineering, I submit the following thesis entitled

Network Embedding For Large Networks

The thesis was performed under the supervision of Hongzhi Yin. I declare that the work submitted in the thesis is my own, except as acknowledged in the text and footnotes, and that it has not previously been submitted for a degree at the University of Queensland or any other institution.

Yours sincerely

Xiaoguang Ye

Acknowledgements

Special thanks to my supervisor Dr. Hongzhi Yin, who has provided me with academic orientations and technical problems support during this entire thesis project. Not only did Dr. Hongzhi Yin enlighten me with the topic of my project, but also he pointed me to the most important academic publications that I needed to read. Dr. Hongzhi Yin had been giving me crucial academic advises, especially when I was trapped in the dire of speeding up the graph partitioning algorithm. Thanks to his academic ability and patience, I finally can finish this topic with relatively successful results.

Abstract

Network Embedding techniques have been created by computer scientists for three decades. In general, network embedding consists of a mathematical base solutions branch and a deep learning based solutions branch. The aforementioned two branches of techniques have their advantages and disadvantages. Unfortunately, they are all extremely expensive in terms of time complexity and memory costs. Hence, both mathematical base solutions and deep learning based solutions fail to solve network embedding problems on huge networks. Enlightened by the multi-level graph partitioning techniques, this thesis project comes up with a brand new solution, a combination of multi-level graph partitioning and single-level graph embedding techniques, to efficiently solve the network embedding problems for huge networks.

Contents

Chapter 1 Introduction

1.1 Networks

A network is a system of entities and their relationships, it represents real-world relational problems in the mathematical way. In mathematics and computer science, often graph is the standard method to describe a network. The same graph can have multiple representations, the most popular of which are: edge list, adjacency matrix, adjacency list. These representations have their own advantages and disadvantages, but they all have one thing in common: represent a graph without losing any information. At first glance, the lossless trait might be a good property. However, these traditional representation are extremely limited when dealing with large networks[1]. Therefore, network embedding as one of the recently develop techniques starts to show its value when network becomes increasingly large. Network embedding focuses on how to represent a graph in a latent low-dimensional way[2]. While ignoring the noises by performing dimension reduction, it can also preserve the underlying structure of the graph. Network embedding not only is a concise representation but also is a extremely useful technique when applying machine learning on huge network.

1.2 Network Embedding

On the one hand, there are existing ways to embed a graph, such as PCA[3], SVD[3], DeepWalk[4], Node2Vec[4]. But all the aforementioned algorithms are limited by the computational complexity and memory requirements of their algorithms. I may take days for them to embed a graph with 100,000 vertices and 1,000,000 edges. With the emergence of social media, networks with billions of vertices and trillions of edges becomes possible. Obviously,

traditional network embedding algorithms becomes computationally infeasible in these situations.

1.3 Graph Partitioning

On the other hand, there have been studies about how to divide a graph while maintaining most of the underlying features. All the minimum edges cutting algorithms[5] are available to achieve so. If we can divide the graph into much smaller subgraphs, then perform network embedding on the smaller scaled graphs. Consequently, we solve the original network embedding problem by divide and conquer.

1.4 Unsolved Problems

There are still several important problems to tackle even with the aforementioned two equipments at disposal.

- 1st: Although graph partitioning algorithms are available, but to achieve minimum edges cutting is non-trivial in terms of computational complexity and memory complexity. Quite contrary, to achieve minimum edges cutting is no easier than graph embedding problem itself, in terms of computational complexity and memory requirements.

- 2nd: Sub-graph embedding will be mapped to different embedding spaces, even if we can partition the graph with relatively cheap cost. In other words, the network embedding techniques will lose their accuracies and fail to represent the underlying network even if the graph partitioning algorithms are relatively cheap and becomes computationally feasible.

- 3rd: Currently, there hasn't been many studies focused on distributed network embedding. Therefore, we don't know if the result is somewhat acceptable in terms of accuracies. More

importantly, there are no existing frameworks to integrate multiple network embedding algorithms to perform embeddings on large scale networks.

- 4th: The new embedding algorithm must have four desirable traits from a network embedding algorithm. First, it must represent the original network well. Second, adaptability. Meaning embedding algorithm must avoid repeat learning process when network evolves. Third, embedding algorithm must scale well when network becomes huge. Fourth, network embedding algorithm must generates low dimensionality representation. Currently, the existing embedding algorithms fail to achieve scalability while maintaining accuracy.

1.5 Hypothesis and Scope

This thesis project will take the existing network embedding algorithms as the known knowledge to further build a distributed network embedding algorithm. This thesis project will also take the current graph partitioning algorithms as the basic building block to build a multi-level graph partitioning algorithm.

- 1st: Experiment several major network embedding algorithms with different scales of input and test their result and performance, to form the input performance reference for further distributed network embedding system.

- 2nd: Build a multi-level graph partitioning algorithm to fast partitioning a given graph. This algorithm comes up with a hierarchical graph partitioning method to make minimum edges cutting algorithms becomes possible on huge networks.

- 3rd: Apply graph embedding on sub-graphs, then map different embeddings into the same embedding space. To map embeddings from their individual embedding space to the same one requires a multi-level perceptron to learn the transformation matrixes.

- 4th: To build a multi-level scalable network embedding framework which is agnostic to the underlying network embedding algorithms and take them as plugins.

- 5th: Use co-author dataset to benchmark the result by performing link predictions. There are two reasons why I choose co-author dataset. First, co-author dataset is large enough with more than four hundred thousand vertices and more than one million edges. Second, co-author dataset is sufficiently different than a randomly generate network. For example, the top 20% of the vertices occupy more than 80% of the edges. With these two properties, I can safely assume that it's a natural dataset that most of the link prediction benchmarks will work.

Chapter 2 Background / Literature Review

2.1 Network Embedding Algorithms

Network embedding algorithm is all about how to accurately learn a latent low-dimensional presentation of the original graph. As Figure 1 demonstrates, network embedding algorithms are

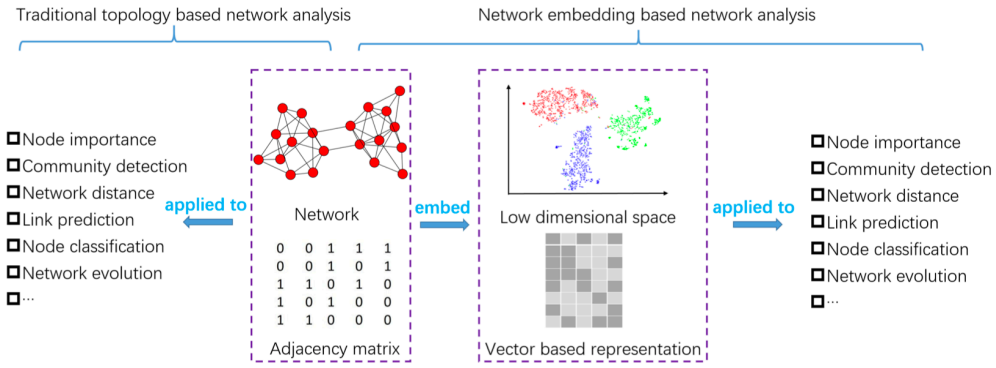

FIGURE1: PURPOSE OF NETWORK EMBEDDING AND APPLICATIONS

essentially a set of dimensionality reduction techniques that has huge amount of applications. There are two main branches of network embedding techniques, including mathematics based methods and machine learning based methods.

2.1.1 Mathematical Embedding Methods

All the mathematics based embedding algorithms are based on linear algebra and orthogonal components decomposition. Although some of the more advanced mathematical embedding algorithms can somewhat capture non-linear relationship. However, the way they achieve non-linear relationship analysis is based on skewed graph sampling, not based on actual non-linear analysis methods.

2.1.1.1 Principal Component Analysis

The PCA (called principal component analysis) was invented to reduce the dimensionality of large dataset. PCA calculates a set of major orthogonal elements, where all major components are linear combinations of the main variables[6]. Ideally, the number of these elements should be smaller than the original size. After the main component is calculated, any raw data points can be projected into their specified dimensional space[6]. Essentially, PCA is a dimension reduction method, by projecting data points on to the most important orthogonal Eigen principle

1: **procedure** PCA
2: Compute dot product matrix: $\mathbf{X}^T\mathbf{X} = \sum_{i=1}^{N}(\mathbf{x}_i - \boldsymbol{\mu})^T(\mathbf{x}_i - \boldsymbol{\mu})$
3: Eigenanalysis: $\mathbf{X}^T\mathbf{X} = \mathbf{V}\boldsymbol{\Lambda}\mathbf{V}^T$
4: Compute eigenvectors: $\mathbf{U} = \mathbf{X}\mathbf{V}\boldsymbol{\Lambda}^{-\frac{1}{2}}$
5: Keep specific number of first components: $\mathbf{U_d} = [\mathbf{u}_1, \ldots, \mathbf{u}_d]$
6: Compute d features: $\mathbf{Y} = \mathbf{U_d}^T\mathbf{X}$

FIGURE2: PCA

components. Figure 2 demonstrate the pseudocode for PCA, where X is the original data point and Y is the projected data with dimension d(d << Dim(X)).

2.1.1.2 Singular Value Decomposition

The technique of singular value decomposition(SVD) has a long history. It started out in the social sciences with intelligence testing. Early intelligence researchers noted that tests given to measure different aspects of intelligence, such as verbal and spatial, were often closely correlated[2]. SVD is very similar to PCA, which was originally invented to perform dimension reduction on high-dimensional data, especially for computer graphic[6]. Compare to SVD, PCA takes use of Eigen vectors and Eigen values to achieve dimension reduction. SVD on the other hand, takes use of singular vector. Then project each data node on the singular vector and try to minimize the sum of squared perpendicular distances to the vector[6]. Essentially, SVD is a dimension reduction method, by projecting data points on to the most important orthogonal singular components.

2.1.1.3 Linear Discriminant Analysis

As explained above, PCA first finds the principal components, then second maximize the data variance without taking into account the class labels. Compare to PCA, Linear Discriminant Analysis (LDA) computes the linear directions maximize the separation between multiple classes[6]. In other words, LDA maximize the variance on the principal components by taking classification into account. This can be an important feature, as we preprocess the raw data, we may want to perform node classification simply by the degrees and label them as authorities, medias and non-important participant. Figure 3 illustrate the pseudocode for LDA.

1: **procedure** LDA
2: Find eigenvectors of $\mathbf{S_w}$ that correspond to non-zero eigenvalues (usually $N - C$), i.e. $\mathbf{U} = [\mathbf{u}_1, \ldots, \mathbf{u}_{N-C}]$ by performing eigen-analysis to $(\mathbf{I} - \mathbf{M})\mathbf{X}^T\mathbf{X}(\mathbf{I} - \mathbf{M}) = \mathbf{V_w}\mathbf{\Lambda}_{\boldsymbol{w}}{\mathbf{V_w}}^T$ and computing $\mathbf{U} = \mathbf{X}(\mathbf{I} - \mathbf{M})\mathbf{V_w}{\mathbf{\Lambda}_{\boldsymbol{w}}}^{-1}$ (performing whitening on $\mathbf{S}_w$).
3: Project the data as $\tilde{\mathbf{X}}_\mathbf{b} = \mathbf{U}^T\mathbf{X}\mathbf{M}$.
4: Perform PCA on $\tilde{\mathbf{X}}_\mathbf{b}$ to find $\mathbf{Q}$ (i.e., compute the eigenanalysis of $\tilde{\mathbf{X}}_\mathbf{b}\tilde{\mathbf{X}}_\mathbf{b}^T = \mathbf{Q}\mathbf{\Lambda}_b\mathbf{Q}^T$).
5: The total transform is $\mathbf{W} = \mathbf{UQ}$

FIGURE3: LDA

2.1.1.4 Multi-Dimensional Scaling

Multi-dimensional scaling(MDS) projects a M-dimensional data points to a K-dimensional space, where that the distance between different data point in the original space is best preserved in the projection space[6]. In other words, MDS best preserves the pair-wise distance when projecting data from a high dimensional space to a low dimensional space. MDS focuses more on a global level topology by minimizes an objective function called stress function. The goal of an MDS analysis is to find a spatial configuration of objects when all that is known is some measure of their general similarity[7]. The spatial configuration need provide a level of insight into how to evaluate the stimuli in a small number of dimensions. Once the proximities are derived the data collection is concluded, and the MDS solution has to be determined using a computer program[7]. Classical MDS algorithms typically involve four major steps. 1st: Set up the matrix of squared proximities. 2nd: Apply the double centering by matrix transformation. 3rd: Extract the m largest positive eigenvalues λ1 . . . λm of B and the corresponding m eigenvectors e1 . . . em[8]. 4th: A m-dimensional spatial configuration of the n objects is derived from the coordinate matrix X = EmΛ1/2 m , where Em is the matrix of m eigenvectors and Λm is the diagonal matrix of m eigenvalues of B[9].

2.1.1.5 Isomap

Isomap(isometric mapping) is a mathematics based non-linear dimensionality reduction technique[10]. Isometric mapping method is developed based on the spectral theory which best preserve the geodesic distances[10] when projecting high dimensional data into a much lower dimensional space. In other words, Isomap is a technique developed based on MDS with the means to preserve geodesic distances[10] within neighborhood of a the network vertex. Isomap has three major steps. 1st: Construct A Graph G' from original G with only neighborhood information. Define the graph G on the entire dataset by connecting random data points x and y, if the distance between x and y is smaller than the given epsilon. 2nd: Use Floyd-Warshall algorithm to compute the all source shortest paths between pairs within only five edges away(5 is the perimeter of the neighborhood). 3rd: Construct d-Dimensional space(d << dim(x), where original data is mapped into. Very similar to PCA, now use Eigen principle components and

Eigen values to perform decomposition on the newly constructed network G' that only contains neighborhood information.

2.1.1.5 Locally-Linear Embedding

Locally-Linear Embedding(LLE) was introduced at about the same time as Isomap[11]. It has several benefits over Isomap, with a quicker optimization strategy to take advantage of sparse adjacency matrix algorithms[11], and eventually out-performs Isomap in many problems. Therefore, its a very appropriate choice for network embedding. Similar to Isomap, to better preserve the neighborhood information, LLE starts with finding a subset of the nearest neighbors from the given graph for each vertex. Then, LLE learns a vector of weights as the linear factors for each vertex that best fits the vertex value as a form of linear combination, relative to all its computed neighbors. Lastly, LLE takes use of an Eigen Principle Component optimization technique to find a suitable low-dimensional embedding space for all the vertex, such that each vertex is still represented with the same linear combination of its neighbors[11]. In other words, LLE finds an appropriate low-dimensional embedding space for the entire neighborhood.

2.1.2 Deep Learning Methods

Most of the Deep Learning algorithms has two major components. Random walk the graph from each node to form new data vectors. Then use new data vectors to train a Convoluted Neural Networks(CNN). Finally, feed the original data points into the trained model to generate the final embeddings.

2.1.2.1 DeepWalk

DeepWalk[12] was initially created as the very first network embedding algorithm by taking advantage of techniques from the deep learning methods. DeepWalk bridges the gap between network embeddings[12] and natural language embeddings by viewing vertices as characters or words and randomly generated short paths as words or sentences. As illustrated in Figure 4, DeepWalk uses on demand random walk to generate sentences and Skip-gram, a Natural

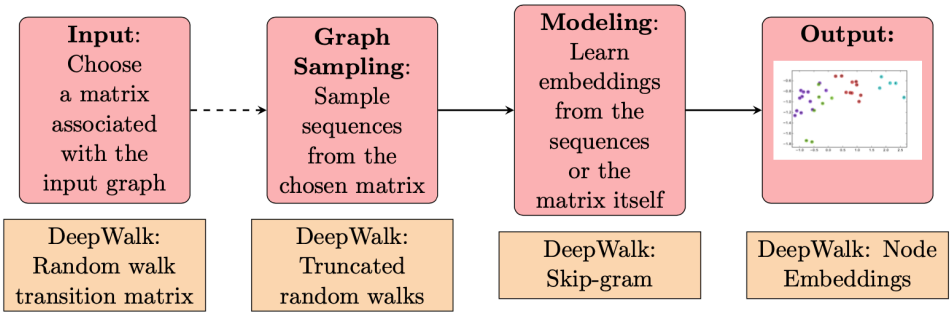

FIGURE4: DEEPWALK'S PROCEDURE

Language Processing model, as the CNN to learning the embeddings. First of all, the combination of random walk and Skip-gram makes DeepWalk an online algorithm[12]. Second, both random walks generating and Skip-gram learning models are relatively efficient and easy to parallelize.

2.1.2.2 Node2Vec

Node2Vec[13] is enlightened by DeepWalk where the only major difference is in the way they generate the random walks. The easiest technique to bias the random walks would be to sample the next vertex on the walk based on the weights of edges. Yet this method suffers from two

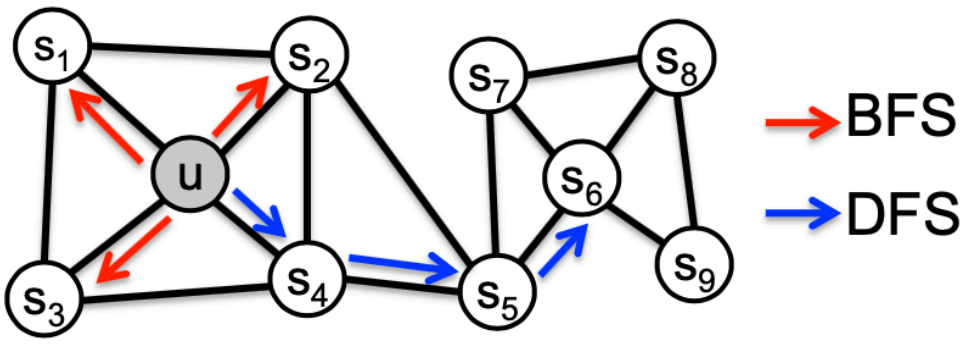

FIGURE5: BFS AND DFS WALKS

major drawbacks. First, weighted edge bias does not provide the algorithm description for the underlying network structure and guide the traverse system to explore different kinds of network neighborhoods. Second, weighted edge bias does not work on the unweighted graphs. Unlike the traditional DFS and BFS, demonstrated in Figure 5, which are extreme sampling models suited for homophily and architecturally identical respectively. Node2Vec's random walk procedure supports the networks in the real-world which are commonly a combination of architecturally identical and homophily. As illustrate in Figure 6, Node2Vec defines a 2nd order random walk with two parameters p and q which guide the walk. Intuitively, p and q controls probability of the random walk procedure to walk out of the neighborhood. To be more specific, with the bias p and q, Node2Vec's random walk procedure simulates a combination between the traditional DFS and BFS. Therefore, the Node2Vec's random walk works well on an affinity of homophily and structural equivalent structure.

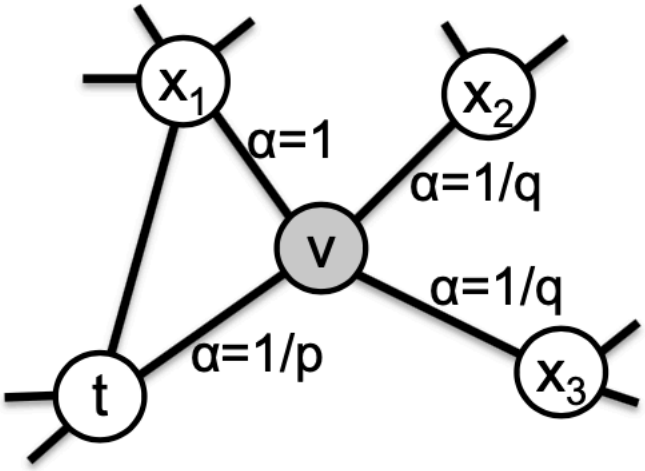

FIGURE6: NODE2VEC RANDOM WALK PROCEDURE

2.1.2 Deep Learning Algorithms Comparison

There are many favors in deep learning algorithms based on DeepWalk. Such as the aforementioned Node2Vec, or the very popular LINE[14], or SDNE[15] which adopts a different

CNN compare to the others' learning engine Skip-gram. The following table compares the major deep learning algorithms including their corresponding features and their characteristics.

Algorithm	Context Vertices	Embedding Learning Algorithm
DeepWalk[12]	Random Walks	Skip-gram with Softmax
Node2Vec[13]	Biased Random Walks	Skip-gram with Hierarchical Softmax
GraRep[16]	K-hop Neighbors	Matrix Factorization
LINE[14]	1-hop and 2-hop Neighbors	Skip-gram with Negative Sampling
SDNE[15]	1-hop and 2-hop Neighbors	Deep Autoencoder
GraphAttention[16]	K-hop Neighbors	Graph-Likelihood
Walklets[17]	K-hop Neighbors	Skip-gram with Hierarchical Softmax
DNGR[18]	Random Walks	Stacked Denoising Autoencoder

- LINE[14] adopts a BFS approach for creating context vertices: only vertices that are at most two hops away from a given vertex are regarded as its neighboring vertices[14]. Additionally, LINE utilizes negative sampling strategy to optimize the Skip-gram learning procedure, compare to the pure Softmax method in DeepWalk.
- GraRep[16] exploits vertices cooccurrence information at different scales by raising the graph adjacency matrix to different powers[16]. Then, Singular value decomposition (SVD) is utilized to the dominance of the adjacency matrix to get low-dimensional description of vertices.

2.1.3 Mathematics vs Deep Learning Algorithms

Most of mathematics based embedding algorithms suffer from two major problems. First, mathematics based embedding algorithms are hard to parallelize. The failed to scale up with the increasing size of the problem. Further more, they all failed to capture the non-linear relationship between the embeddings numbers, because they assume that this is a linear relationship between

the values projected on the orthogonal components. Even Isomap and Locally-Linear Embedding claim that can capture the non-linear relationship, they achieved so only by biased sampling strategy. Therefore, they can only capture the linear relationship with bias towards local neighborhood in the network. But deep down they still failed to find the non-linear features globally. On the contrary, most of the deep learning algorithms can scale well when problem gets big. And the deep learning algorithms can capture the non-linear features due to the innate ability of CNN. The following table listed the comparisons between the major mathematics based embedding algorithms and deep learning algorithms.

Algorithm	Embedding Method	Scalable	Non-Linear Feature
PCA[6]	Principle Component Analysis	No	No
SVD[2]	Singular Value Decomposition	No	No
MDS[7]	Biased Sampling + Principle Component Analysis	No	Partially
Isomap[10]	Floyd-Warshall shortest path + Principle Component Analysis	No	Partially
DeepWalk[12]	Random Walks + Skip-gram with Softmax	Yes	Yes
Node2Vec[13]	Biased Random Walks + Skip-gram with Hierarchical Softmax + Negative Sampling	Yes	Yes
DNGR[18]	Random Walks + Stacked Denoising Autoencoder	Yes	Yes

2.2 Graph Partitioning Algorithms

In mathematics, a graph partitioning algorithm's job is to divide a network into a set of smaller graphs by partitioning its vertices into mutually exclusive associations in an ideal situation. The

edges from the original network that connect the subgraphs will create edges in the partitioned graph. In theory, if the amount of connecting edges is tiny compared to the original network, then the newly partitioned graph stands a great change for analyzing the original network, because the newly partitioned graph represents the original network well. Therefore, to find the minimal edges cutting partition is the key for graph partitioning algorithms. Because network partitioning is a difficult problem, possible solutions are all based on heuristics. Among them, there are two obvious types of algorithms, namely local vs global.

2.2.1 Local Optimal Algorithms

2.2.1.1 Kernighan–Lin algorithm Algorithm

The purpose of the Kernighan–Lin algorithm[19] is to find a partition of original graph G into two disjoint subsets G' and G'' of roughly the same size. While minimizing the summation of the

```
function Kernighan-Lin(G(V, E))
    determine a balanced initial partition of the
    nodes into sets A and B
    do
        A1 := A; B1 := B
        compute D values for all a in A1 and b in B1
        for i := 1 to |V|/2 do
            find a[i] from A1 and b[i] from B1, such
            that
            g[i] = D[a[i]] + D[b[i]] − 2 * c[a[i]][b[i]]
            is maximal
            move a[i] to B1 and b[i] to A1
            remove a[i] and b[i] from further
            consideration in this pass
            update D values for the elements of
            A1 = A1/a[i] and B1 = B1/b[i]
        end
        find k which maximizes g_max, the sum of
        g[1], . . . , g[k]
        if g_max > 0 then
            Exchange a[1], a[2], . . . , a[k] with
            b[1], b[2], . . . , b[k]
        end
    until g_max ≤ 0
return: G(V,E)
```

FIGURE7: KERNIGHAN–LIN ALGORITHM

weights of the edges that connect G' and G''. The Kernighan–Lin algorithm greedily improves a certain partition. For each round of running, the Kernighan–Lin algorithm randomly samples a pair of vertices from one partition, move one vertex to the other partition, then check if the newly computed summation of the weights cross edges is smaller than the previous summation. If so, breaking the paired vertices into two partitions improves the network partition result. After matching the vertices, it then performs a subset of the pairs chosen to have the best overall effect on the solution quality T[19]. Figure 7 demonstrate the pseudocode for the Kernighan-Lin algorithm. The runtime of Kernighan–Lin algorithm is bound by O(n^3 log n).

2.2.1.2 Fiduccia-Mattheyses Algorithm

Fiduccia-Mattheyses algorithm[20], like the Kernighan–Lin algorithm, is a greedy algorithm focused on getting minimal edge cutting based on local optimal. Unlike the Kernighan–Lin algorithm, one vertex is moved this time instead of of two. Therefore, Fiduccia-Mattheyses algorithm is applicable to partitions of unequal size or the presence of initially fixed cells[20].

2.2.2 Global Optimal Algorithms

2.2.2.1 Spectral Bisection Algorithm

Spectral Bisection Algorithm[21] starts with letting each vertex of the network to be assigned either 1 or -1. For representational purposes, the value assigned to vertex i be denoted by Vi. This creates a bisection of the graph in which the vertices having value 1 form one subdomain and those with value -1 form the other[21]. Then the edge cutting can be expressed as $Ec = 1/4 \sum_i (Vi - Vj)$. Now network bisection problem becomes a problem of minimizing equation $Ec = 1/4 \sum_i (Vi - Vj)$, where i and j are the vertices in the given network. To find the eigenvector corresponding to the second smallest eigenvalue, the Lanczos algorithm can be employed[21]. Therefore, each round of the problem can be solved within time limit of n^3,

where n is the size of the input vertices. And if the Lanczos algorithm converges in m rounds, we can assume that the Spectral Bisection Algorithm will run to finish within O(n^3*m).

2.2.2.2 Spectral Partitioning Algorithm

The Spectral Partitioning Algorithm[21], like the previously mentioned Spectral Bisection Algorithm, adopts the eigenvector decomposition as the fundamental solver. Unlike the Spectral Bisection Algorithm, the spectral partitioning algorithm initially label the vertices into N categories where N is a given number that is greater than 2[22]. Therefore, Spectral Partitioning Algorithm has the similar accuracy and performance compared to the Spectral Bisection Algorithm.

2.3 Performance Problems

2.3.1 Network Embedding Algorithms

As I discussed previously, all the listed network embedding algorithms suffers from two performance bottle-necks. First, the fastest mathematical based network embedding algorithms will be bounded by O(n^3*logn) where n is the number vertices in the input network. To add insult into injury, all the mathematical based network embedding algorithms are not adaptable, meaning we have to recalculate the embedding even if just one edge is added. In other words, the embedding can not evolve with the network. Second, although the deep learning based algorithms has the similar time complexity compared to the mathematics based embedding algorithms, however, all the the deep learning based embedding algorithms suffered from high memory usage due to stochastic nature of generating walks. Moreover, the neural network model is quite expensive to train in terms of time and memory costs.

2.3.2 Graph Partitioning Algorithms

The graph partitioning algorithms are all bounded by O(n^3*big constant) where n is the size of the input vertices, in terms of time complexity. Therefore, the graph partitioning algorithms are not much faster than the network embedding algorithms. This poses a huge problem to the initial idea, where speed-up lays within the belief that running network embedding algorithm on the much smaller graph than the original one is manageable. The belief is the reason why we tried to partition the original graph into many smaller gub-graphs. From there, graph embedding becomes possible. With the fact that partition the graph is no easier than embedding the graph, the project needs to figure out a way where graph partitioning can be more efficient.

Chapter 3 Methodology And Execution

3.1 Multi-Level Graph Partition

A multi-level graph partitioning method operates by applying single-level graph partitioning methods one or more steps. Every stage decreases the volume of the graph by merging vertices and edges, running graph partitioning algorithm on a smaller scaled graph, then refines the already partitioned graph back to the original graph. Here the single-level graph partitioning algorithm can be any previously introduced methods, such as the Fiduccia-Mattheyses Algorithm, the Kernighan–Lin Algorithm or the Spectral Partitioning Algorithm. An ample range of graph partitioning algorithms can be implemented in the multi-level graph partitioning system. In a lot of cases, a multi-level graph partitioning method can achieve both fast performance times and high accuracy.

3.1.1 Graph Coarsening

Graph coarsening is the stage where the vertices and edges merges into each other, resulting original graph shrink into a smaller graph. The purpose of this stage is to merge the original graph into much smaller graph while keep the underlying structure of the original graph intact[23]. The main concern at this graph coarsening stage is to consolidate the rules on when to perform merge operation. To better define this procedure, I need to define the important notions

and concepts in the following table.

Symbol	Definition
Gi	The already coarsened graph after i stages of coarsening process
Vi	The set of vertices in Gi
Ei	The set of edges in Gi
Ai	Adjacency matrix of Gi
Di	Degree matrix of Gi
d	Dimensionality of the embedding
m	The total level of graph coarsening
P(*)	The base graph partitioning algorithm applicable on Gi
ϵ**i**	Graph partitioning result on Gi
Mi, i+1	Matching matrix from Gi to Gi+1

3.1.1.1 Structural Equivalent Matching(SEM)

Two vertices u and v in the given unweighted graph G. Vertices u and v are structurally equivalent if and only if they are connected to the exactly same set of neighboring nodes[23].

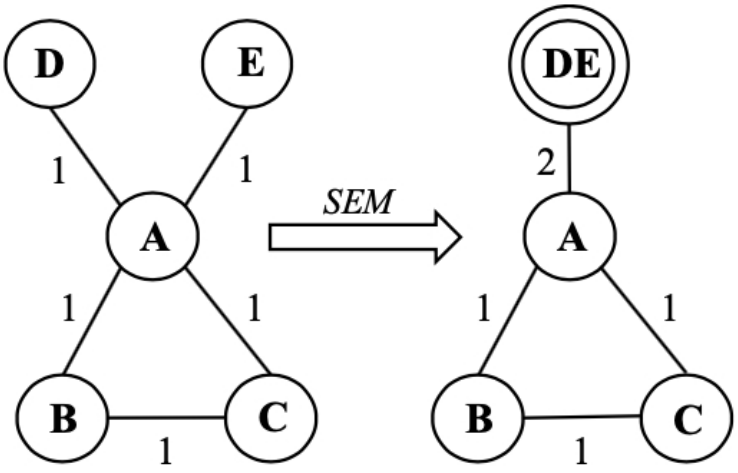

FIGURE8: D AND E ARE STRUCTURALLY EQUIVALENT

Structurally equivalent property is commutative, associative and transitive. These three properties will greatly improve the performance of graph coarsening process.

- 1st: If a vertex v is structurally equivalent to vertex u in the given unweighted graph G, then vertex u is also structurally equivalent to vertex v.
- 2nd: If a vertex v is structurally equivalent to vertex u in the given unweighted graph G, and vertex u is structurally equivalent to vertex w. Then vertex v and vertex w are also structurally equivalent to each other.
- 3rd: If a vertex v is structurally equivalent to vertex u and vertex w in the given unweighted graph G. Then vertex v is structurally equivalent to vertex w and vertex v is structurally equivalent to vertex u.

As Figure 8 illustrates, vertices D and E are viewed as structurally equivalent to each other. Therefore they merged into each other in graph coarsening procedure[24].

3.1.1.2 Normalized Heavy Edge Matching (NHEM)

Heavy edge matching is one common matching approach for graph coarsening[24]. For an unmatched node u in Gi, its heavy edge matching is a pair of vertices (u, v) such that the weight

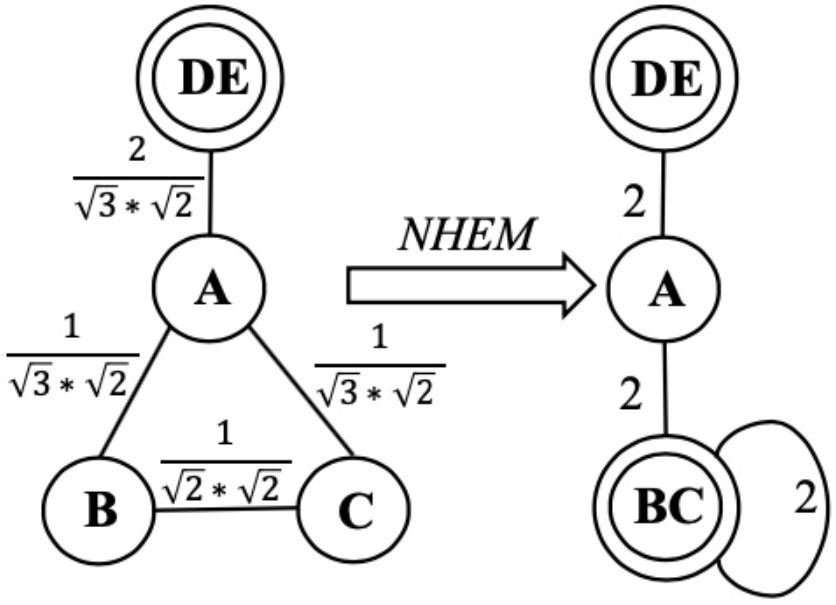

FIGURE9: B AND C ARE HEAVY EDGE MATCHED

of the edge between u and v is the largest[24]. Vertices u and v matches when the following formulas is applied: $Wi(u,v) = Ai(u,v)/\sqrt{(Di(u,u)Di(v,v))}$. In the equation, the weight of an edge is normalized by the degree of the two vertices on which the edge is incident. Intuitively, it punishes the weights of edges connected with high-degree vertices[24]. As Figure 9 illustrate, vertices B and C are heavy edge matched and then merged into each other in graph coarsening procedure.

3.1.1.3 Hybrid Matching(HM)

In this thesis project, a hybrid coarsening strategy is proposed consisting of two aforementioned graph coarsening techniques. In order to construct graph Gi+1 on coarsening stage i+1 from graph Gi on coarsening state i. First, the algorithm treats Gi as a undirected unweight graph and then figures out any possible structural equivalence matching in graph Gi. Second, the algorithm finds out all the normalized heavy edge matching in Gi, only this time treats Gi as an undirected weighted graph. The targeted vertices in stage i graph Gi in the aforementioned two matchings are then merged into super nodes in the next stage graph Gi+1. Third, the unmatched vertices are directly copied from stage i's Gi into stage i+1's Gi+1. Figure 10 demonstrate the entire hybrid match scheme in each round.

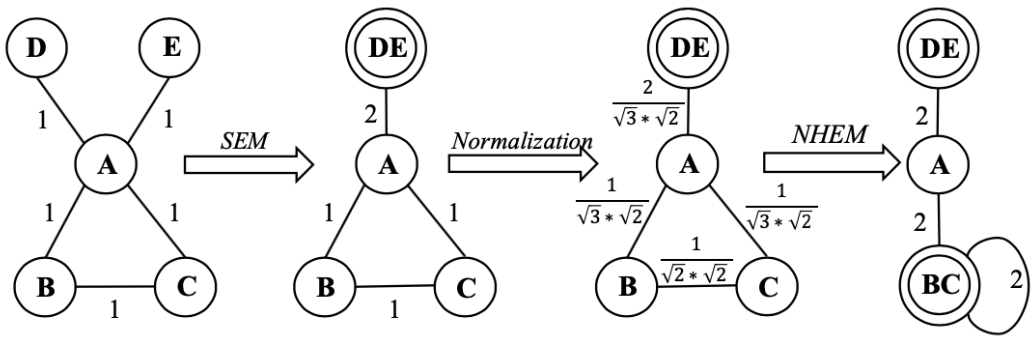

FIGURE10: HYBRID MATCHING STRATEGY

3.1.1.4 Matching Matrix

The matching matrix stores the vertices merging information from ith stage graph Gi to the next stage graph Gi+1 as a matching matrix Mi, i+1. The r-th row and c-th column of Mi, i+1 is set to

1 if node r in Gi will be collapsed to super-node c in Gi+1, and is set to 0 if otherwise[24]. Every column in matching matrix Mi, i+1 stands for a merge, and the 1st row representing the vertices

$$A_0 = \begin{array}{c} \begin{array}{ccccc} \mathbf{A} & \mathbf{B} & \mathbf{C} & \mathbf{D} & \mathbf{E} \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array} \qquad M_{0,1} = \begin{array}{c} \begin{array}{ccc} \mathbf{A} & \mathbf{BC} & \mathbf{DE} \end{array} \\ \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix} \begin{array}{c} \mathbf{A} \\ \mathbf{B} \\ \mathbf{C} \\ \mathbf{D} \\ \mathbf{E} \end{array} \end{array}$$

$$A_1 = M_{0,1}^T A_0 M_{0,1} = \begin{pmatrix} 0 & 2 & 2 \\ 2 & 2 & 0 \\ 2 & 0 & 0 \end{pmatrix}$$

FIGURE11: ADJ MATRIX AND MATCHING MATRIX

in it. All the unmatched vertices show up as a separate column in the matching matrix Mi, i+1 with the only 1 set on the diagonal. Figure 11 demonstrate the adjacency matrix and matching matrix transformation for the aforementioned graph. As we can observe, we can easily construct the stage i + 1's graph Gi+1 by applying A1 = Tran(M0,1) A0 M0,1.

3.1.2 Apply Single-Level Graph Partitioning Algorithms on Coarsened Graph

After m level of graph coarsening, the graph Gm is roughly 2^-m the size of original graph. Now apply the relative expensive single-level graph partitioning algorithms on the coarsened graph Gm. Generate the partitioned graphs Gmi…Gmn.

3.1.3 Graph Refinement

Graph refinement procedure is about to reconstruct the original graph from the already partitioned graph. In this thesis project, the algorithm needs to apply a total number of M level of refinement procedures. The reconstruction is achieved by applying the reverse matching matrix transformation where A0 = M0, 1 A1 Tran(M0, 1). Only at this point of time, the graph has already been partitioned into multiple partitions(∈m1…..∈mN) at round m with graph Gm. Reconstruction procedure happens only on each partitions separately. Figure 12 illustrate the entire m-level graph coarsening and graph refinement procedure.

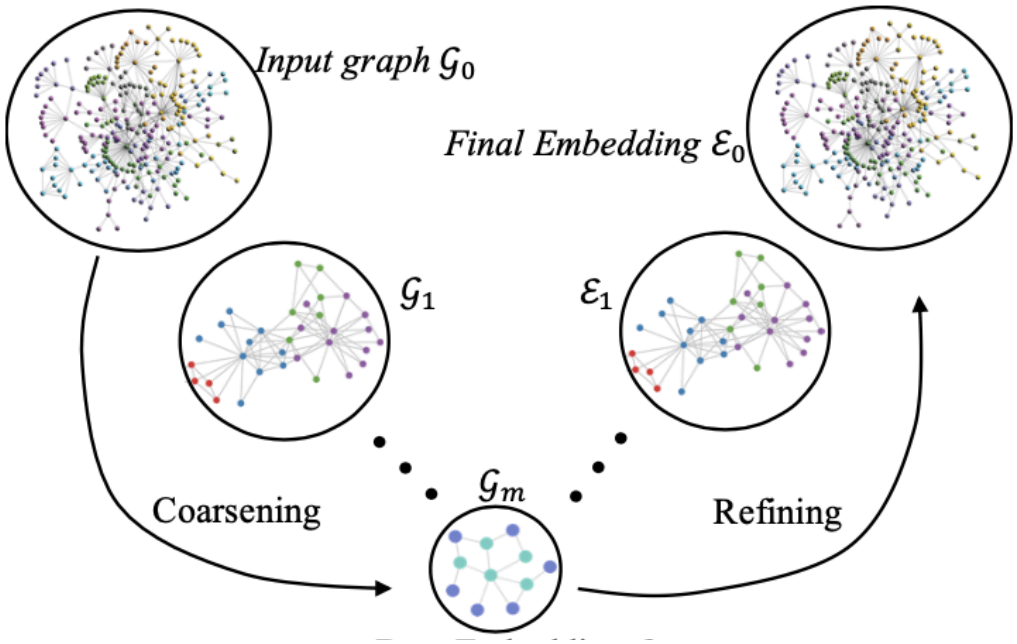

FIGURE12: M LEVEL GRAPH COARSENING AND REFINEMENT

3.1.4 Multi-Level Graph Partitioning Algorithm Experiment

For testing and experimenting purposes, I compare the performance of Multi-Level graph partitioning algorithm to the performance of Single-Level graph partitioning algorithm. I compare the performance of them in three different categories. 1st metrics is accuracy in terms of actual edge cutting. 2nd metrics is how even the algorithms actually partitioned the graph. 3rd metrics is the performance in terms of time complexity. For the sake of fairness, I choose the Spectral Partitioning Algorithm as the plugin graph partitioning algorithm for the multi-level graph partitioning algorithm and the single-level graph partitioning algorithm. The dataset I use is email-eu with more than 1,200 vertices and more than 20,000 edges. The following table shows the experiment result.

Algorithm	#Partitions	% Edge Cuts	%Unevenly Distributed	Performance
Multi-level	10	9%	6%	67ms
Multi-level	20	11%	7%	57ms
Multi-level	40	16%	9%	89ms
Multi-level	80	17%	9%	46ms
Spectral Partitioning[21]	10	12%	9%	108ms
Spectral Partitioning[21]	20	12%	8%	121ms
Spectral Partitioning[21]	40	14%	11%	176ms
Spectral Partitioning[21]	80	15%	12%	287ms
Kernighan-Lin[19]	10	16%	11%	97ms
Kernighan-Lin[19]	20	16%	13%	182ms

Algorithm	#Partitions	% Edge Cuts	%Unevenly Distributed	Performance
Kernighan-Lin[19]	40	19%	9%	199ms
Kernighan-Lin[19]	80	25%	6%	330ms
Fiduccia-Mattheyses[20]	10	11%	12%	103ms
Fiduccia-Mattheyses[20]	20	16%	11%	218ms
Fiduccia-Mattheyses[20]	40	19%	8%	549ms
Fiduccia-Mattheyses[20]	80	19%	8%	501ms

3.2 Graph Embedding Space Mapping

The main challenge for multi-level network embedding is focused on how to accommodate different embedding spaces. The algorithm could either build a pairwise embedding space mapping between two every combination of embedding spaces or map all embedding spaces into an identical network embedding space. The original scheme may need creating a pairwise mapping for each pair which may be computationally costly while the latter way needs assembling about m mappings where m is the amount of subgraphs. Disregard of the underlying algorithms applied on the network embedding spaces mapping, the fundamental method is wherewith to map between two network embedding spaces. To map different network embedding spaces, the algorithm requires remarkable connections between spaces. The connection is achieved by extracting the shared vertices, known as the anchor vertices, between different subgraphs.

3.2.1 Anchor Nodes

As I explained previously, the the anchor vertices are the connections between different subgraphs. After the algorithm constructed the network embeddings for each subgraph independently, the anchor nodes ought to produce different network embeddings values. The anchor nodes will be assigned to different network embedding values because they exist in different network embedding spaces in spite of the fact that they are the same set of vertices. Figure 13 demonstrates the overall procedure for this algorithm, where all the stars(anchor nodes) are aligned relatively well. Consequently, the anchor nodes bridge the gap between different graph embedding spaces. In other words, the algorithm rotates different network embedding spaces into each other such that all the anchor nodes shared by subgraphs are well aligned.

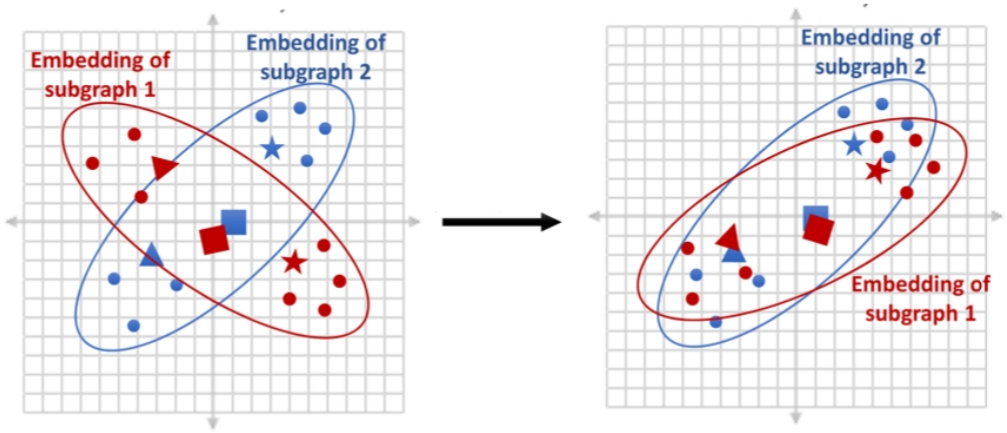

FIGURE13: EMBEDDING SPACE MAPPING

3.2.1.1 Extract High-Degree Vertices

High-Degree vertices are the vertices are very well connected to other vertices. Therefore, there is a very high chance that all the high-degree vertices will be repeated in different subgraphs.

This discovery coincides well with the nature of anchor nodes. As a result, the first step of embedding space mapping is to extract the high-degree vertices as the anchor nodes from the original network.

3.2.2 Embedding Space Mapping

The main purpose of partition space mapping is to learn a mapping function Mi' = H(Mi), where Mi' is the mapped embedding result and all the anchor nodes have roughly the same embedding result across different subgraphs. I use multiple layered perceptron as the tool and utilize Stochastic gradient descent(SGD) to minimize the difference of anchor nodes' embedding results for different partitions[24]. A MLP has of two or more layers of nonlinearly-activating nodes.

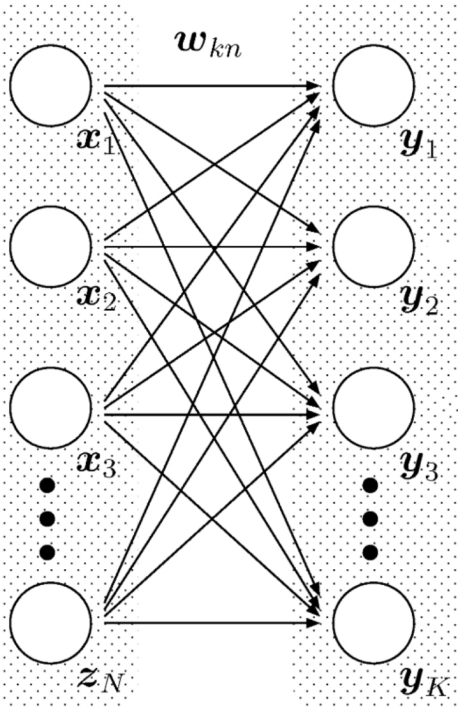

FIGURE14: 2-LAYERED MLP MAP EMBEDDING SPACES

Since MLPs are fully connected, each node in one layer connects with a certain weight to every node in the following layer[25]. As Figure 14 demonstrates, the two layered MLP maps anchor

nodes from embedding space X to embedding space Y. Because each perceptron in one layer connects with a certain weight to every perceptron in the following layer[25]. If we have two layers of MLP with N perceptrons on each layer and with no hidden layers, the weights in the MLP forms a N by N matrix to transform a network embedding Mi to another embedding Mj. If I feed the the 2-layered MLP with all the anchor nodes[26], and minimize mean squared error between the absolute difference Mi and Mj, then I learnt a MLP that best maps the different embedding spaces.

3.3 Multi-Level Network Embedding Framework

As I explained before, the entire framework has three major steps. The framework start with partitioning the graph with the multi-level graph partitioning algorithm. Then, the framework uses Deepwalk or Node2Vec to calculate the embedding values for each subgraphs. Finally, the framework map the different embedding spaces into one by using anchor nodes.

3.3.1 Graph Partitioning Step

The framework needs to first extract the anchor nodes. Then perform the aforementioned multi-level graph partitioning algorithm. Finally, connect anchor nodes back to all the subgraphs. Figure 15 demonstrates the pseudocode for graph partitioning step. Where line 5, graph partition algorithm is the Multi-Level graph partition in our framework. Step 1 through 4 is the anchor nodes extraction, and step 6 through 8 is the subgraphs reconstruction step by add anchor nodes back into subgraphs.

input : Graph $G = (V, E)$;
number of subgraphs k;
degree threshold δ
output: k subgraphs $H_1, H_2, \ldots, H_k$

// Select anchor nodes
1 $C = \{v \in V | deg(v) > \delta\}$;
// Generate induced graph from G and C
2 $V' = V \setminus C$;
3 $E' = \{(u, v) | (u, v) \in E \wedge u \in V' \wedge v \in V'\}$;
4 $H = (V', E')$;
// Non-anchor graph partition
5 $(V_1, E_1), (V_2, E_2) \ldots, (V_k, E_k) = graph\_partition(H, k)$;
// Merge with anchor nodes
6 **for** $i \in [1, k]$ **do**
7 $H_i = (V_i \cup C, \{(u, v) | (u, v) \in E \wedge u \in V_i \cup C \wedge v \in V_i \cup C\})$;
8 return $H_1, H_2, \ldots, H_k$

FIGURE15: GRAPH PARTITIONING STEP

3.3.2 SubGraph Embedding Step

At this step, the framework uses the deep learning based graph embedding algorithm on everyone of the subgraphs to get the network embeddings in each embedding space. Figure16 is the pseudocode for DeepWalk. Where G(V, E) is the input subgraph with vertices V and edges E in the subgraph. The notion d is the designated embedding size. The notion γ represents the number of walks per vertex. The notion t represent the walk depth per sentences. Pay attention to step 3 through 9, if user wants to have an average of 128 number of walks per vertex with walk depth of 128. If the given graph G has a vertices size 53,000. Then the algorithm will spend at least 2 ^ 1 * 2 ^ 7 * 2 ^ 7 * 2 ^ 19 = 2 ^ 34, 16 giga bytes of memory to calculate the embeddings for graph G. Memory consumption is one of the main reasons why the thesis project need to partition the big network into smaller ones, then apply DeepWalk or Node2Vec to all the subgraphs.

Input: network $G(V, E)$
window size w
embedding size d
walks per vertex γ
walk length t
Output: matrix of vertex representations $\Phi \in \mathbb{R}^{|V| \times d}$

Initialization: Sample Φ from $\mathcal{U}^{|V| \times d}$
Build a binary Tree T from V
for $i = 0$ to γ **do**
  $\mathcal{O} = \text{Shuffle}(V)$
  for each $v_i \in \mathcal{O}$ **do**
    $\mathcal{W}_{v_i} = RandomWalk(G, v_i, \text{t})$
    SkipGram(Φ, $\mathcal{W}_{v_i}$, w)
  end for
end for

FIGURE16: DEEPWALK

3.3.3 Embedding Space Mapping Step

As I mentioned previously, embedding space mapping is achieved by training a MLP with the anchor nodes as inputs. There is a trade off to consider, if the embedding dimensionality is N, then the framework needs a two layered, N perceptrons per layer, no hidden layer MLP to perform network embedding space mapping. There are N ^ 2 parameters to tune in this MLP, therefore, the framework need at least 2 * N ^ 2 anchor nodes to achieve any meaningful training results. After the MLP is learnt, we can apply MLP to all the vertices to attain the global network embeddings for the entire network.

3.4 Benchmark Plan

3.4.1 Dataset Choices

Co-authorship network[27] is the major test dataset I use in this project. First, Co-authorship network is large enough with 402392 vertices and 1234019 edges. It makes only makes sense to use multi-layered network embedding technology when the graph is larger than 5000 vertices and 15000 edges[27]. Second, Co-authorship network is sufficiently different than randomly generated graph. In terms degree, although the average degree is around 3, however the first 20 percent of the vertices consists of 80 percent the edges. Therefore, Co-authorship network makes link prediction and clustering benchmarks possible[27].

3.4.2 Application And Method

Apply CNN[28] on network embedding result to perform link prediction is one of the most commonly used benchmarks to evaluate the accuracy of network embeddings. I use Co-authorship network to perform link prediction as the benchmark. First, randomly remove 50,000 existing edges between high degree nodes, randomly generate 50,000 edges between 0 degree nodes[27]. Second, partition the network into subgraphs with 256 anchor nodes, then embed the sub-graphs and super-graph. Third, label all the edges between high degree nodes and the edges between 0 degree nodes, then use the embeddings to train a CNN[28].

3.4.3 Metrics And Evaluation

For the purpose of evaluation, I focus on four major types.

- 1st: Precision of the prediction
- 2nd: Recall of the prediction
- 3rd: F-Measure of the prediction
- 4th: The time spent on calculating the embedding result

Chapter 4 Results And Discussion

4.1 Strategy

As I mentioned before, the strategy has three major steps. First, randomly remove 50,000 existing edges between high degree nodes, randomly generate 50,000 edges between 0 degree nodes[27]. Second, partition the network into subgraphs with 256 anchor nodes, then embed the sub-graphs and super-graph. Third, label all the edges between high degree nodes and the edges between 0 degree nodes, then use the embeddings to train a prediction CNN[28]. The CNN has a SGD solver as relu activator, with 4 layers networks.

4.1 Dataset

As I argued in 3.4.1, Co-authorship network[27] is an appropriate dataset to use in order to test the performance of the framework. But, due to the fact that Co-authorship network[27] is large enough that non of the single-level network embedding algorithm are able to run to finish. Therefore, I also have email-Eu as the dataset to test the accuracy metrics of multi-layered graph partitioning algorithm to the single-layered graph partitioning algorithm.

4.3 Environment

I use 16 cores intel i7 machine with 64 GIGA-BYTE DDR4 ram. I configure 16 threads for the embedding and model training algorithm.

4.4 Result

4.4.1 Co-authorship Network Metrics

The following table listed the result of the multi-level embedding framework's performance in terms of accuracy and running time. Especially the scaling when the number of partitions increases. But, due to the fact that single partition never runs to finished because of the memory overflow system error. The system can not compare the accuracy against the single partitioned network embeddings.

Embedding Algorithm	Number Of Partitions	Precision	Recall	F1	Running Time
Node2Vec	**1**	Nan	Nan	Nan	Memory Overflow
Node2Vec	**10**	71.2%	82.3%	76.3%	13.5 Hours
Node2Vec	**20**	68.9%	80.2%	74.1%	11.8 Hours
Node2Vec	**40**	69.6%	81.6%	75.2%	9.5 Hours
Node2Vec	**80**	71.8%	83.5%	77.2%	8.0 Hours
Node2Vec	**160**	72.0%	83.7%	77.4%	7.2 Hours

Embedding Algorithm	Number Of Partitions	Precision	Recall	F1	Running Time
DeepWalk	**1**	Nan	Nan	Nan	Memory Overflow
DeepWalk	**10**	72.2%	83.1%	76.9%	14.1 Hours
DeepWalk	**20**	68.0%	80.0%	73.9%	11.5 Hours
DeepWalk	**40**	67.6%	78.9%	72.8%	10.0 Hours
DeepWalk	**80**	72.9%	82.6%	77.5%	9.1 Hours
DeepWalk	**160**	73.1%	82.8%	77.6%	8.1 Hours

4.4.2 Email-Eu Network Metrics

For the dataset email-eu, which is considerable smaller than the co-authorship dataset, the purpose to testing the framework on email-eu dataset is to gain the accuracy metrics difference between the multi-level network embedding framework and the single network embedding algorithms. The following table shows the accuracy results change when the number of partitions kept doubling.

Algorithm	Number Of Partitions	Precision	Recall
Node2Vec	1	76.4%	87.1%
Node2Vec	10	76.1%	86.6%
Node2Vec	20	74.3%	80.3%
Node2Vec	40	65.1%	73.8%

4.5 Discussion

- 1st: For both DeepWalk and Node2Vec, the original network with 402392 vertices and 1234019 edges never run to finish due to memory overflow. This result concurs with our observation when introducing the second step in the framework. When user wants to have an average of 128 number of walks per vertex with walk depth of 128. If the given graph G has a vertices size 53,000. Then the algorithm will spend at least 2 ^ 1 * 2 ^ 7 * 2 ^ 7 * 2 ^ 19 = 2 ^ 34, 16 giga bytes of memory to calculate the embeddings for graph G[3.3.2].
- 2nd: For both DeepWalk and Node2Vec, when the number of partitions increases, both precision and recall went down. Mostly because, the embedding result becomes more inaccurate when number of partitions went up. But both precision and recall went up when the number of partitions becomes 80. It's most because that the number of vertices in each subgraph becomes 4000, and the number of anchor nodes are 512. When the ratio of anchor nodes to number of vertices in graph G becomes big enough. The reconstruction process after multi-level partitioning made the connections between subgraphs strong enough to produce even better embedding result.
- 3rd: The running time becomes less and less when the number of partitions increase. This is mainly because that when graph becomes smaller, the embedding algorithm will perform better in terms of time complexity.
- 4th: The accuracy for email-eu dataset drops considerably when the number of partitions keeping doubling. This is most likely due to the fact that there are simply not enough anchor nodes to be shared. Email-eu has only about 1,200 nodes, if the framework choose to have 256 anchor nodes shared by subgraphs, then network embedding dimensionality is limited by 8. Eight dimensions are not complicated enough to present email-eu. And when the partition number becomes 80, there simply aren't 256 anchor node can be shared within 80 partitions.

Chapter 5 Conclusions

5.1 Summary And Conclusion

5.1.1 Performance Conclusion

The multi-level graph partitioning algorithm definitely performs well and partition the graph sensibly. The subgraphs are evenly partitioned, with the minimized the edge cuttings and fast convergence speed. With the multi-level graph partitioning algorithm inlace, the framework is not limited by the time complexity and memory complexity of the network embedding algorithms. As we can clearly observe from the experiment results, the more partitions we have, the smaller amount of time it takes to solve the problem.

5.1.2 Accuracy Conclusion

The experiment clearly demonstrates that, in a small network, the accuracy doesn't drop much when there are enough anchor nodes to be shared with different partitions. But the accuracy drops drastically when then number of anchor nodes are not enough.

5.1.3 Summary

The multi-level network embedding framework can achieve high accuracy and high performance scalability given certain constraints. The multi-level network embedding framework can also

achieve plugin mode with different network embedding algorithms and graph partitioning algorithms.

5.2 Possible Future Work

5.2.1 Hyper-Parameter Tunings

There are two major places where hyper-parameter tunings are extremely important.

- 1st: In Node2Vec algorithm, p and q are the two probability parameters to guide the random walks either prefer DFS or BFS. For different networks, different p and q will generate very different embedding results. It's extremely valuable to spend some time on studying the tuning of p and q.
- 2nd: The embedding space mapping procedure need to tune a MLP based on the given anchor nodes. There is a trade-off between the embedding dimensionality and the ability to map different embedding spaces. It makes sense to study what is the proper number of embedding dimensionality when given a network. The framework need to use reinforcement learnings to generate knowledge on how to tune the dimensionality hyper-parameter.

5.2.2 Engineering Effort

Currently, the entire framework still needs a lot of human intervention, such as copying intermediate result files from places to places, or constantly monitoring jobs because memory expensive jobs are likely to fail, etc. More engineering efforts need to be made if I want to make this framework to be stabilized.

5.2.3 Parallelization

After graph partitioning, the framework may become highly parallelized in nature. Because all the embedding algorithms can run independently without any synchronizations. The process is only limited by the memory consumptions. Therefore, map-reduce can be a very good choice to implement a parallelized version of multi-level network embedding framework.

Appendix A

A sample of embedded graph with 16 dimension embeddings.emb.

A trained model in node2vec according to the same given graph embeddings.model

Co-authorship sample dataset is in data/coauthor

Appendix B

C++ program to build the graph, multi-level partition the graph is under /code/c.

C++ program to map embedding spaces is under /code/c.

Python code to calculate graph embedding is under /code/python

Bibliography

[1]Cui, P., Wang, X., Pei, J. and Zhu, W. (2019). A Survey on Network Embedding. [online] arXiv.org. Available at: https://arxiv.org/abs/1711.08752 [Accessed 20 Mar. 2019].

[2]W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. In NIPS'17.

[3]Cs.cmu.edu, 2019. [Online]. Available: https://www.cs.cmu.edu/~elaw/papers/pca.pdf. [Accessed: 23- Oct- 2019].

[4]J. Qiu, Y. Dong, H. Ma, J. Li, K. Wang, and J. Tang. Network embedding as matrix factorization: Unifyingdeepwalk, line, pte, and node2vec. In WSDM'18.

[5]Liang, J., Gurukar, S. and Parthasarathy, S. (2017). MILE: A Multi-Level Framework for Scalable Graph Embedding. [online] arXiv.org. Available at: https://arxiv.org/abs/ 1802.09612 [Accessed 12 Mar. 2019].

[6]Chen, H., Perozzi, B., Al-Rfou, R. and Skiena, S. (2018). A Tutorial on Network Embeddings. [online] Arxiv.org. Available at: https://arxiv.org/pdf/1808.02590.pdf [Accessed 10 Mar. 2019].

[7]Cui, P., Wang, X., Pei, J. and Zhu, W. (2019). A Survey on Network Embedding. [online] arXiv.org. Available at: https://arxiv.org/abs/1711.08752 [Accessed 20 Mar. 2019].

[8]Borg, I., & Groenen, P. (1997). Modern multidimensional scaling: theory and applications. New York: Springer.

[9]Mathpsy.uni-tuebingen.de, 2019. [Online]. Available: http://www.mathpsy.uni-tuebingen.de/wickelmaier/pubs/Wickelmaier2003SQRU.pdf. [Accessed: 26- Oct- 2019].

[10]J. B. Tenenbaum, V. de Silva, J. C. Langford, A Global Geometric Framework for Nonlinear Dimensionality Reduction, Science 290, (2000), 2319–2323.

[11]Bengio et al. "Out-of-Sample Extensions for LLE, Isomap, MDS, Eigenmaps, and Spectral Clustering" in Advances in Neural Information Processing Systems (2004)

[12]Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social repre- sentations. In Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining, pages 701–710. ACM, 2014.

[13]Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 855–864. ACM, 2016.
[14]Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In Proceedings of the 24th International Conference on World Wide Web, pages 1067–1077. ACM, 2015.
[15]Daixin Wang, Peng Cui, and Wenwu Zhu. Structural deep network embedding. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 1225–1234. ACM, 2016.
[16]Shaosheng Cao, Wei Lu, and Qiongkai Xu. Grarep: Learning graph representations with global structural information. In Proceedings of the 24th ACM International on Conference on Information and Knowledge Management, pages 891–900. ACM, 2015.
[17]Bryan Perozzi, Vivek Kulkarni, Haochen Chen, and Steven Skiena. Don't walk, skip! online learning of multi-scale network embeddings. In 2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM). IEEE/ACM, 2017.
[18]Sami Abu-El-Haija, Bryan Perozzi, Rami Al-Rfou, and Alex Alemi. Watch your step: Learning graph embeddings through attention. arXiv preprint arXiv:1710.09599, 2017.
[19]Lin, Shen (1970). "An efficient heuristic procedure for partitioning graphs". Bell System Technical Journal. 49: 291–307. doi:10.1002/j.1538-7305.1970.tb01770.x.
[20]Fiduccia; Mattheyses (1982). "A Linear-Time Heuristic for Improving Network Partitions" (PDF). 19th Design Automation Conference. Retrieved 23 October 2019.
[21]Alzate, Carlos; Suykens, Johan A. K. (2010). "Multiway Spectral Clustering with Out-of-Sample Extensions through Weighted Kernel PCA". IEEE Transactions on Pattern Analysis and Machine Intelligence. 32 (2): 335–347. doi:10.1109/TPAMI.2008.292. ISSN 0162-8828. PMID 20075462.
[22]Knyazev, Andrew V. (2006). Multiscale Spectral Graph Partitioning and Image Segmentation. Workshop on Algorithms for Modern Massive Data Sets Stanford University and Yahoo! Research.
[23]Karypis, G.; Kumar, V. (1999). "A fast and high quality multilevel scheme for partitioning irregular graphs". SIAM Journal on Scientific Computing. 20 (1): 359. CiteSeerX 10.1.1.39.3415. doi:10.1137/S1064827595287997

[24]Liang, J., Gurukar, S. and Parthasarathy, S. (2017). MILE: A Multi-Level Framework for Scalable Graph Embedding. [online] arXiv.org. Available at: https://arxiv.org/abs/ 1802.09612 [Accessed 12 Oct. 2019].
[25]R. Collobert and S. Bengio (2004). Links between Perceptrons, MLPs and SVMs. Proc. Int'l Conf. on Machine Learning (ICML).
[26]THOMAS KIPF, Graph convolutional networks. (n.d.). [ebook] Available at: https://tkipf.github.io/graph-convolutional-networks/ [Accessed 23 Oct. 2019].
[27]"SNAP: Network datasets: DBLP collaboration network", Snap.stanford.edu, 2019. [Online]. Available: https://snap.stanford.edu/data/com-DBLP.html. [Accessed: 26- Oct- 2019].
[28]Neural networks. II. What are they and why is everybody so interested in them now; Wasserman, P.D.; Schwartz, T.; Page(s): 10-15; IEEE Expert, 1988, Volume 3, Issue 1