MILE: A Multi-Level Framework for Scalable Graph Embedding

Credit: Jiongqian Liang, Saket Gurukar, Srinivasan Parthasarathy

Ohio State University

Presenter: Ryan McCampbell
https://qdata.github.io/deep2Read

Outline

Outline

Introduction

- We have discussed graph embeddings without end
- But how well do these methods scale?

Introduction

- Random walk-based methods
  - DeepWalk
  - Node2Vec
  - Require lots of CPU time to generate enough walks
- Matrix Factorization methods
  - GraRep
  - NetMF
  - Require large objective matrix to factor
  - Can easily require hundreds of GB

Introduction

- Real-world graphs can have millions of nodes
  - Google knowledge graph - 570M entities
  - Facebook friendship graph - 1.39B users with 1 trillion connections

Challenge

- Can we scale up existing embedding techniques in an agnostic manner so that they can be directly applied to larger datasets?
- Can the quality of embeddings be strengthened by incorporating a holistic view of the graph?

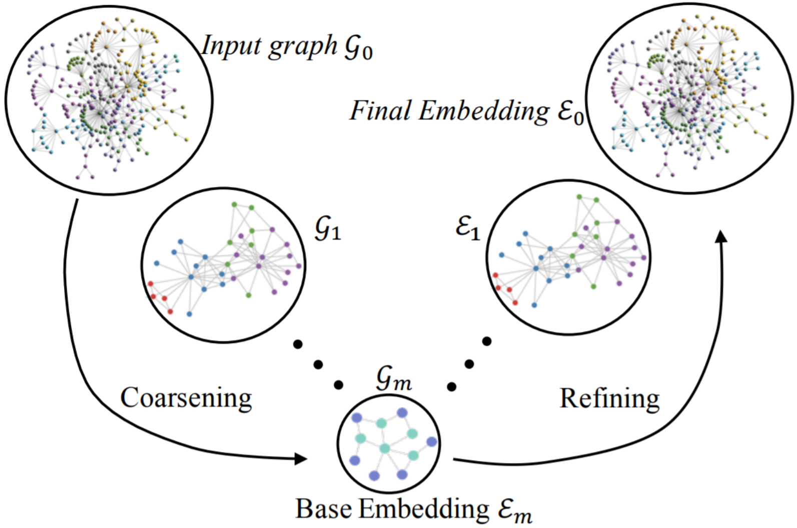
Outline

MILE - MultI Level Embedding Framework

3-step process:

1. Repeatedly coarsen graph into smaller ones using hybrid matching strategy
2. Compute embeddings on coarsest graph using existing embedding method
    - Inexpensive and less memory than full graph
    - Captures global structure
3. Novel refinement model - learn graph convolution network to refine the embeddings from the coarsest graph to the original graph

MILE Overview



Input graph $\mathcal{G}_0$
Final Embedding $\mathcal{E}_0$
$\mathcal{G}_1$
$\mathcal{E}_1$
$\mathcal{G}_m$
Coarsening
Refining
Base Embedding $\mathcal{E}_m$

Outline

Graph Coarsening

- Graph G_0 repeatedly coarsened into smaller graphs $G_1, \ldots, G_m$
- To coarsen G_i to G_{i+1}, multiple nodes are collapsed to form super-nodes
- Edges on super-nodes are union of original nodes' edges
- Set of nodes forming super-node called *matching*

Structural Equivalence Matching (SEM)

- Two nodes are *structurally equivalent* if they are incident on the same set of neighborhoods
- If two vertices u and v in an unweighted graph G are structurally equivalent, then their node embeddings derived from G will be identical.
- *Structural equivalence matching* is set of nodes that are structurally equivalent to each other

Normalized Heavy Edge Matching (NHEM)

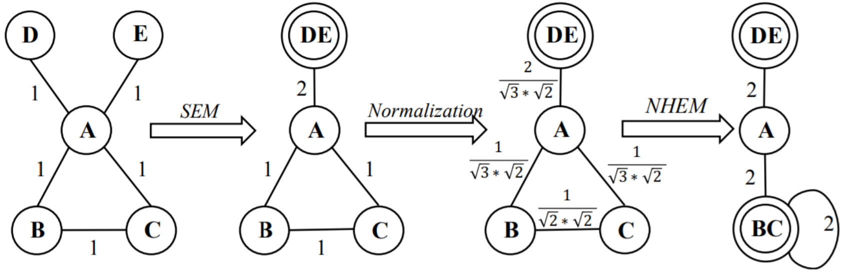
- Heavy edge matching is pair of vertices with largest weight edge between them
- Normalize weights:

$$W_i(u, v) = \frac{A_i(u, v)}{\sqrt{D_i(u, u) D_i(v, v)}}$$

  - Penalize weights of edges connected to high-degree nodes

Hybrid Matching Method

- Combine SEM and NHEM



D
E
A
B
C
1
1
1
1
1
SEM
DE
2
A
B
C
1
1
1
Normalization
DE
$\frac{2}{\sqrt{3}*\sqrt{2}}$
A
$\frac{1}{\sqrt{3}*\sqrt{2}}$
$\frac{1}{\sqrt{3}*\sqrt{2}}$
B
C
$\frac{1}{\sqrt{2}*\sqrt{2}}$
NHEM
DE
2
A
2
BC
2

(a) Using SEM and NHEM for graph coarsening

Matching Matrix

- Matching matrix $M_{i,i+1}$ stores matching info from G_i to G_{i+1}
- Adjacency matrix is constructed from match matrix:

$$A_{i+1} = M_{i,i+1}^T A_i M_{i,i+1}$$

Matching Matrix

$$A_0 = \begin{array}{ccccc} \textbf{A} & \textbf{B} & \textbf{C} & \textbf{D} & \textbf{E} \\ \hline \end{array} \begin{pmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

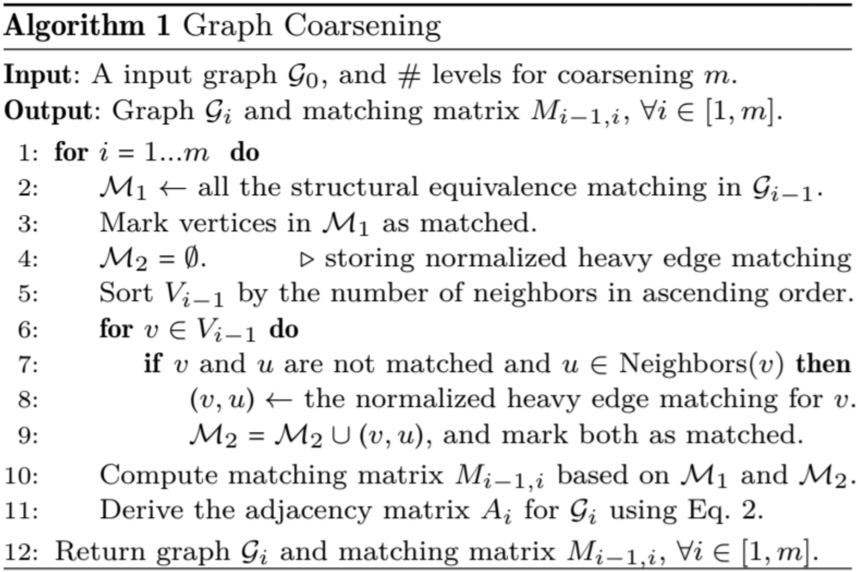
$$M_{0,1} = \begin{array}{cccc} \textbf{A} & \textbf{BC} & \textbf{DE} & \\ \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} & & & \end{array}$$

$$M_{0,1} = \left(\begin{array}{ccc|c} \textbf{A} & \textbf{BC} & \textbf{DE} & \\ 1 & 0 & 0 & \textbf{A} \\ 0 & 1 & 0 & \textbf{B} \\ 0 & 1 & 0 & \textbf{C} \\ 0 & 0 & 1 & \textbf{D} \\ 0 & 0 & 1 & \textbf{E} \end{array}\right)$$

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

(b) Adjacency matrix and matching matrix

Graph Coarsening Algorithm



Algorithm 1 Graph Coarsening

Input: A input graph $\mathcal{G}_0$, and # levels for coarsening m.
Output: Graph $\mathcal{G}_i$ and matching matrix $M_{i-1,i}$, $\forall i \in [1, m]$.

```
 1: for $i = 1...m$ do
 2:     $\mathcal{M}_1 \leftarrow$ all the structural equivalence matching in $\mathcal{G}_{i-1}$.
 3:     Mark vertices in $\mathcal{M}_1$ as matched.
 4:     $\mathcal{M}_2 = \emptyset$.          $\triangleright$ storing normalized heavy edge matching
 5:     Sort $V_{i-1}$ by the number of neighbors in ascending order.
 6:     for $v \in V_{i-1}$ do
 7:         if $v$ and $u$ are not matched and $u \in$ Neighbors($v$) then
 8:             $(v, u) \leftarrow$ the normalized heavy edge matching for $v$.
 9:             $\mathcal{M}_2 = \mathcal{M}_2 \cup (v, u)$, and mark both as matched.
10:     Compute matching matrix $M_{i-1,i}$ based on $\mathcal{M}_1$ and $\mathcal{M}_2$.
11:     Derive the adjacency matrix $A_i$ for $\mathcal{G}_i$ using Eq. 2.
12: Return graph $\mathcal{G}_i$ and matching matrix $M_{i-1,i}$, $\forall i \in [1, m]$.
```

Outline

Base Embedding

- After coarsening the graph for m iterations, apply graph embedding f on coarsest graph G_m
- Agnostic to graph embedding method: can use any embedding algorithm
- They tried DeepWalk, Node2Vec, GraRep, and NetMF

Outline

Embeddings Refinement

- Need to derive embeddings E for G_0 from G_m
- Infer embeddings E_i from E_{i+1}
- Projected embeddings:

$$E_i^p = M_{i,i+1} E_{i+1}$$

    - Embedding of super-node copied to original nodes
    - Nodes will share same embeddings

Graph Convolution Network

- Graph convolution:

$$X *_G g = U\theta_g U^T X$$

  - θ: Spectral multipliers, U: eigenvectors of normalized Laplacian
- Fast approximation:

$$X *_G g \approx \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}X\Theta$$

  - $\tilde{A} = A + \lambda D$, $\tilde{D}(i,i) = \sum_j \tilde{A}(i,j)$
  - λ hyper-parameter
  - Θ trainable weight

Graph Convolution Network

- Refine embeddings using graph convolution network

$$E_i = R(E_i^p, A_i) = H^{(l)}(E_i^p, A_i)$$

$$H^{(k)}(X, A) = \sigma(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}H^{(k-1)}(X, A)\Theta^{(k)})$$

Model Learning

- We can run base embedding on G_i to generate "ground-truth" embeddings for training loss
- Learn $\Theta^{(k)}$ on coarsest graph and reuse them across all levels
- Use base embedding on coarsest graph to get ground truth embeddings $E_m = f(G_m)$
- Further coarsen G_m to G_{m+1} and get another embedding $E_{m+1} = f(G_{m+1})$
- Predict $R(E_m^p, A_m) = H^{(l)}(M_{m,m+1}E_{m+1}, A_m)$
- Loss:

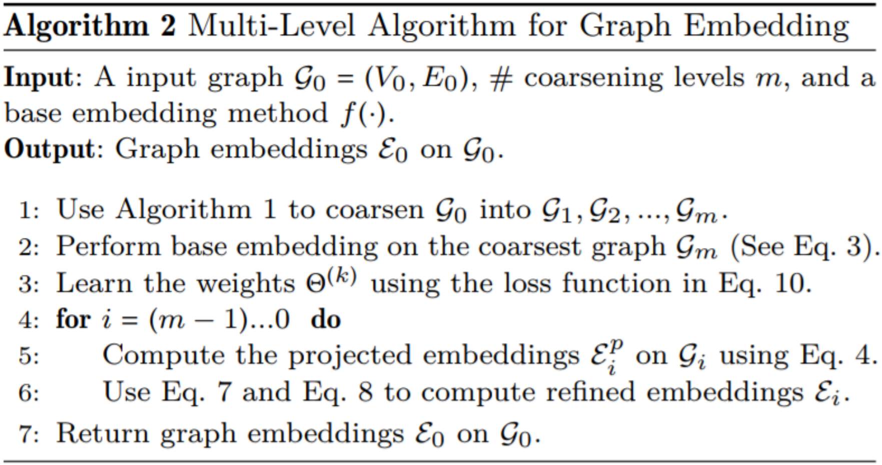
$$L = \frac{1}{|V_m|}\left\|E_m - H^{(l)}(M_{m,m+1}E_{m+1}, A_m)\right\|^2$$

Model Learning

- Problems with "double-base" embedding
  - Requires extra coarsening and base embedding
  - Embeddings may not be comparable since learned independently
- Better method:
  - Copy G_m, eliminate extra coarsening and embedding
  - $E_{m+1} = E_m$

$$L = \frac{1}{|V_m|} \left\| E_m - H^{(l)}(E_m, A_m) \right\|^2$$

Full Algorithm



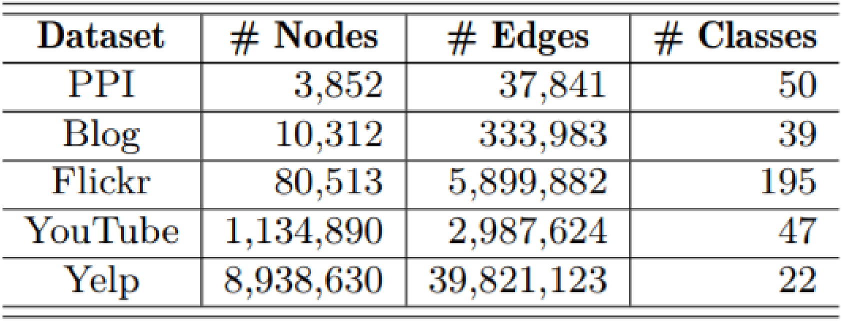
Algorithm 2 Multi-Level Algorithm for Graph Embedding

Input: A input graph $\mathcal{G}_0 = (V_0, E_0)$, # coarsening levels m, and a base embedding method $f(\cdot)$.
Output: Graph embeddings $\mathcal{E}_0$ on $\mathcal{G}_0$.

```
1: Use Algorithm 1 to coarsen $\mathcal{G}_0$ into $\mathcal{G}_1, \mathcal{G}_2, ..., \mathcal{G}_m$.
2: Perform base embedding on the coarsest graph $\mathcal{G}_m$ (See Eq. 3).
3: Learn the weights $\Theta^{(k)}$ using the loss function in Eq. 10.
4: for $i = (m-1)...0$ do
5:     Compute the projected embeddings $\mathcal{E}_i^p$ on $\mathcal{G}_i$ using Eq. 4.
6:     Use Eq. 7 and Eq. 8 to compute refined embeddings $\mathcal{E}_i$.
7: Return graph embeddings $\mathcal{E}_0$ on $\mathcal{G}_0$.
```

Outline

Results



Dataset	# Nodes	# Edges	# Classes
PPI	3,852	37,841	50
Blog	10,312	333,983	39
Flickr	80,513	5,899,882	195
YouTube	1,134,890	2,987,624	47
Yelp	8,938,630	39,821,123	22

Table 2: Dataset Information

Results

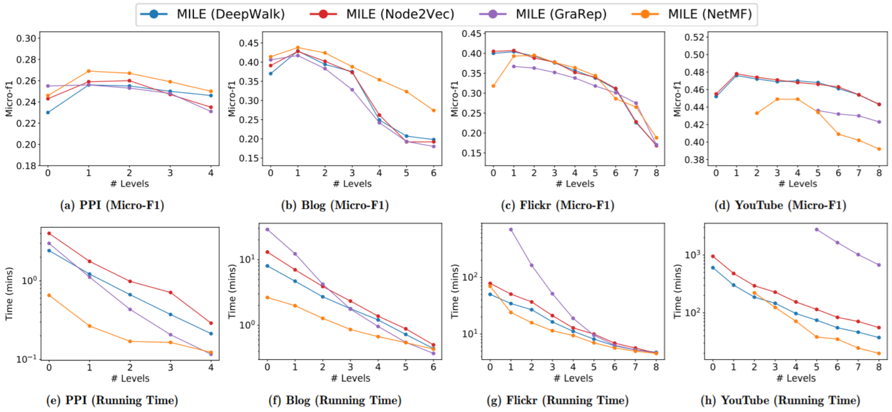
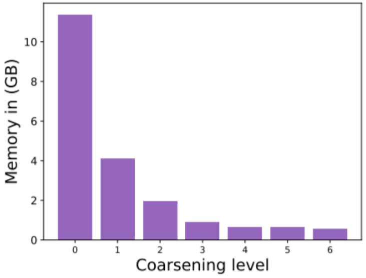
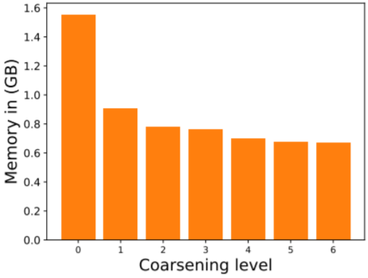


Figure: Performance of different methods and number of levels

Results



(a) MILE (GraRep)



(b) MILE (NetMF)

Figure: Memory consumption

Outline

Conclusions

- MILE is:
  - scalable
  - improves embedding quality
  - supports multiple embedding strategies
- This makes graph learning applicable to much broader and more interesting applications