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

1 Introduction

2 MILE
   • Graph Coarsening
   • Base Embedding
   • Embeddings Refinement

3 Results

4 Conclusions
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
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

1. Introduction

2. MILE
   - Graph Coarsening
   - Base Embedding
   - Embeddings Refinement

3. Results

4. Conclusions
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 $G_0$

Final Embedding $\mathcal{E}_0$

$G_1$

$\mathcal{E}_1$

Coarsening

$G_m$

Refining

Base Embedding $\mathcal{E}_m$

Credit: Jiongqian Liang, Saket Gurukar, Srinivasan Parthasarathy (Ohio State University)
Outline

1 Introduction

2 MILE
   - Graph Coarsening
   - Base Embedding
   - Embeddings Refinement

3 Results

4 Conclusions
Graph Coarsening

- Graph $G_0$ repeatedly coarsened into smaller graphs $G_1, ..., G_m$
- 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*
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

(a) Using SEM and NHEM for graph coarsening
Matching matrix $M_{i,i+1}$ stores matching info from $G_i$ to $G_{i+1}$

- Adjacency matrix is constructed from match matrix:

$$A_{i+1} = M_{i,i+1}^T A_i M_{i,i+1}$$
(b) Adjacency matrix and matching matrix

\[ A_0 = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}, \quad \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \]

\[ A_1 = M_{0,1}^T A_0 M_{0,1} = \begin{pmatrix} 0 & 2 & 2 \\ 2 & 2 & 0 \\ 2 & 0 & 0 \end{pmatrix} \]
Algorithm 1 Graph Coarsening

**Input:** A input graph $G_0$, and $\#$ levels for coarsening $m$.

**Output:** Graph $G_i$ and matching matrix $M_{i-1,i}$, $\forall i \in [1, m]$.

1: for $i = 1...m$ do
2: \hspace{1em} $M_1 \leftarrow$ all the structural equivalence matching in $G_{i-1}$.
3: \hspace{1em} Mark vertices in $M_1$ as matched.
4: \hspace{1em} $M_2 = \emptyset$. ▷ storing normalized heavy edge matching
5: \hspace{1em} Sort $V_{i-1}$ by the number of neighbors in ascending order.
6: for $v \in V_{i-1}$ do
7: \hspace{2em} if $v$ and $u$ are not matched and $u \in \text{Neighbors}(v)$ then
8: \hspace{3em} $(v, u) \leftarrow$ the normalized heavy edge matching for $v$.
9: \hspace{3em} $M_2 = M_2 \cup (v, u)$, and mark both as matched.
10: Compute matching matrix $M_{i-1,i}$ based on $M_1$ and $M_2$.
11: Derive the adjacency matrix $A_i$ for $G_i$ using Eq. 2.
12: Return graph $G_i$ and matching matrix $M_{i-1,i}$, $\forall i \in [1, m]$. 

Credit: Jiongqian Liang, Saket Gurukar, SriniMILE: A Multi-Level Framework for Scalable Presenter: Ryan McCampbell https://qdata.github.io/deep2Read
Outline

1 Introduction

2 MILE
   - Graph Coarsening
   - Base Embedding
   - Embeddings Refinement

3 Results

4 Conclusions
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

1 Introduction

2 MILE
   - Graph Coarsening
   - Base Embedding
   - Embeddings Refinement

3 Results

4 Conclusions
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

Credit: Jiongqian Liang, Saket Gurukar, Srini\textregistered MILE: A Multi-Level Framework for Scalable Presenter: Ryan McCampbell  \url{https://qdata.github.io/deep2Read}
Graph Convolution Network

- Graph convolution:
  \[ X \ast_G g = U \theta_g U^T X \]
  - \( \theta \): Spectral multipliers, \( U \): eigenvectors of normalized Laplacian
- Fast approximation:
  \[ X \ast_G g \approx \tilde{D}^{-1/2} \tilde{A} \tilde{D}^{-1/2} X \Theta \]
  - \( \tilde{A} = A + \lambda D \), \( \tilde{D}(i, i) = \sum_j \tilde{A}(i, j) \)
  - \( \lambda \) hyper-parameter
  - \( \Theta \) trainable weight
Refine embeddings using graph convolution network

\[ E_i = R(E_i^p, A_i) = H^{(l)}(E_i^p, A_i) \]

\[ H^{(k)}(X, A) = \sigma(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}H^{(k-1)}(X, A)\Theta^{(k)}) \]
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 $R(E^p_m, A_m) = H^{(l)}(M_{m,m+1}E_{m+1}, A_m)$.

Loss:

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

$$L = \frac{1}{|V_m|} \left\| E_m - H^{(l)}(E_m, A_m) \right\|^2$$
Algorithm 2 Multi-Level Algorithm for Graph Embedding

Input: A input graph $G_0 = (V_0, E_0)$, # coarsening levels $m$, and a base embedding method $f(\cdot)$.
Output: Graph embeddings $E_0$ on $G_0$.

1: Use Algorithm 1 to coarsen $G_0$ into $G_1, G_2, ..., G_m$.
2: Perform base embedding on the coarsest graph $G_m$ (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 $E_i^p$ on $G_i$ using Eq. 4.
6: Use Eq. 7 and Eq. 8 to compute refined embeddings $E_i$.
7: Return graph embeddings $E_0$ on $G_0$. 
Outline

1. Introduction

2. MILE
   - Graph Coarsening
   - Base Embedding
   - Embeddings Refinement

3. Results

4. Conclusions
### Table 2: Dataset Information

<table>
<thead>
<tr>
<th>Dataset</th>
<th># Nodes</th>
<th># Edges</th>
<th># Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>PPI</td>
<td>3,852</td>
<td>37,841</td>
<td>50</td>
</tr>
<tr>
<td>Blog</td>
<td>10,312</td>
<td>333,983</td>
<td>39</td>
</tr>
<tr>
<td>Flickr</td>
<td>80,513</td>
<td>5,899,882</td>
<td>195</td>
</tr>
<tr>
<td>YouTube</td>
<td>1,134,890</td>
<td>2,987,624</td>
<td>47</td>
</tr>
<tr>
<td>Yelp</td>
<td>8,938,630</td>
<td>39,821,123</td>
<td>22</td>
</tr>
</tbody>
</table>
Figure: Performance of different methods and number of levels
Results

**Figure:** Memory consumption

(a) MILE (GraRep)

(b) MILE (NetMF)
Outline

1 Introduction

2 MILE
   - Graph Coarsening
   - Base Embedding
   - Embeddings Refinement

3 Results

4 Conclusions
MILE is:

- scalable
- improves embedding quality
- supports multiple embedding strategies

This makes graph learning applicable to much broader and more interesting applications