# Metapath2vec: Scalable Representation Learning for Heterogeneous Networks

Yuxiao Dong, Nitesh V. Chawla, Ananthram Swami

지인선

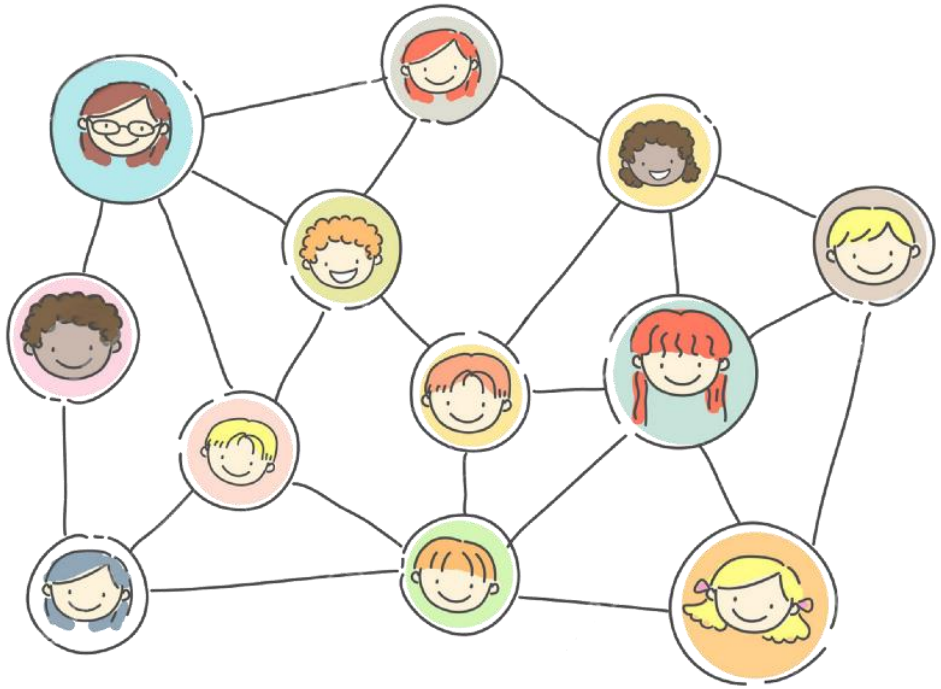
2023.01.31

# Contents

# Introduction

Homogeneous Graph: A graph with a single type of node and a single type of edge.

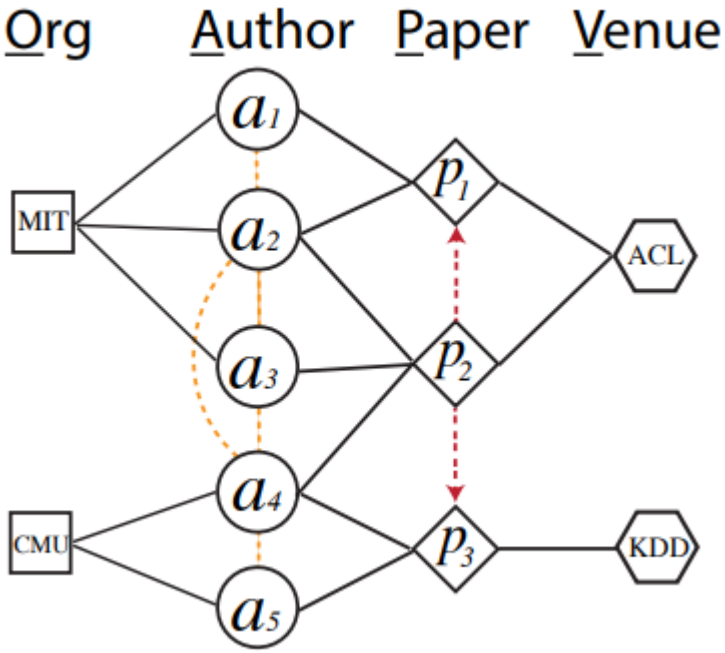
Example: Social network with nodes representing people and edges representing friendship.



# Introduction

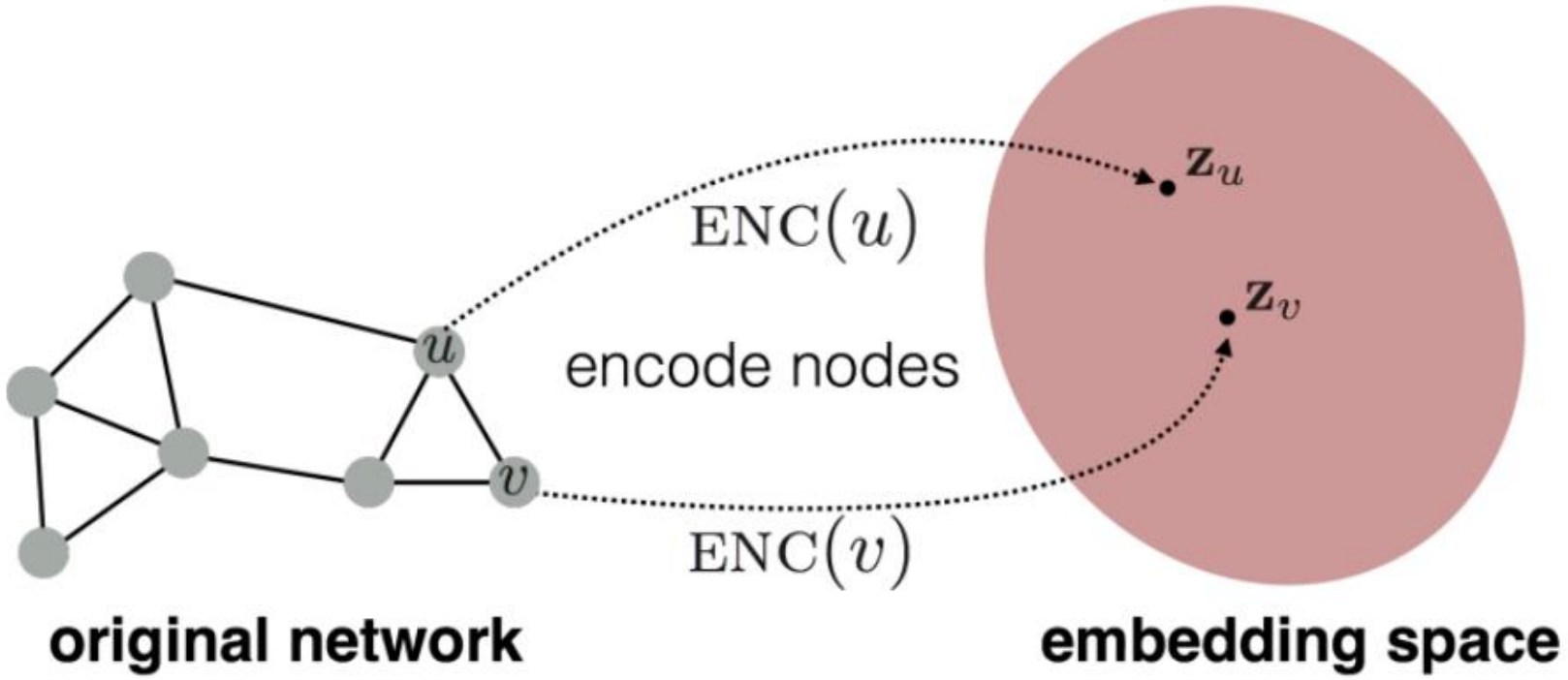
Heterogeneous Graph: A graph with two or more types of node and/or two or more types of edge.

Example: Citation graph where each node can represent author, paper, etc.



# Introduction

Graph Embedding: embed graph into a low dimensional space while preserving the graph structure and property so that the learned embeddings can be applied to the downstream network tasks.



# Introduction

DeepWalk uses Random Walk to generate node sequences and feeds them on skip−gram model to learn node embedding vector.

Node2vec uses Biased 2$^{nd}$ order Random Walk that can flexibly trade off between local and global structure of the graph.

But both of them are for homogeneous graph.

Metapath2Vec utilizes Metapath−based Random walk to construct the heterogeneous neighborhood of a node and then leverages a heterogeneous skip−gram model to perform node embeddings.

# Problem Definition

Definition of Heterogeneous Network:

G=(V, E, T) in which each node $v$ and each link $e$ are associated
with their mapping functions $\pi(v): V \rightarrow T_V$, $\varphi(e): E \rightarrow T_E$ respectively.

$T_V$ and $T_E$ denote the sets of object and relation types, where $|T_V| + |T_E| > 2$
(when $|T_V| + |T_E| = 2$, the graph is homogeneous)

# Problem Definition

Problem 1. Heterogeneous Network Representation Learning:

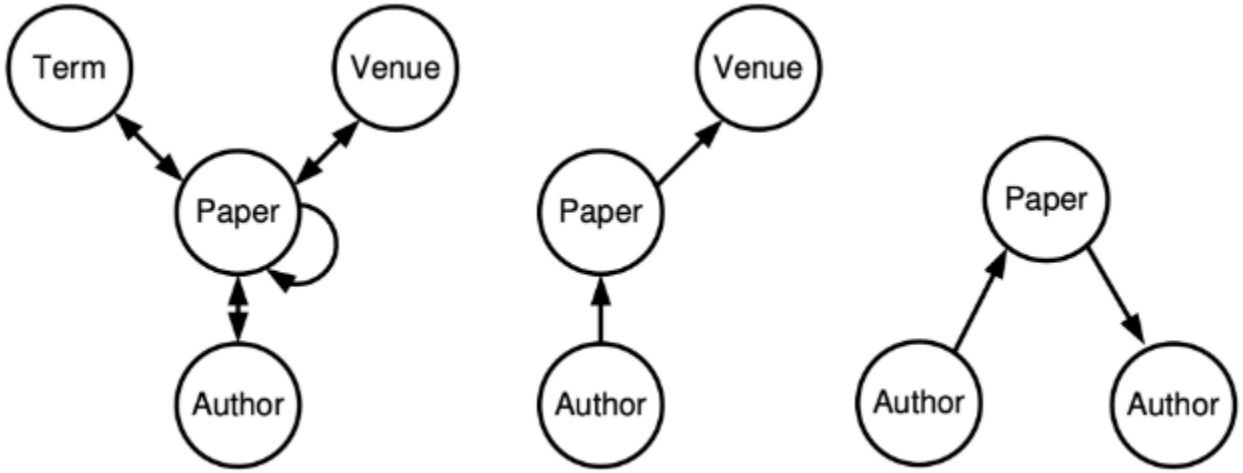
Given a heterogeneous network G, the task is to learn the $d-$dimensional latent representations $X \in R^{|V| \times d}, d \ll |V|$ that are able to capture the structural and semantic relations among them.

The output of the problem is the low−dimensional matrix $X$, with the $v$ th row—a $d-$dimensional vector $X_v$—corresponding to the representation of node v.

# Metapath2vec Framework

Definition of Metapath:

A meta−path $\Phi$ is defined as a path in the form of $A_1 \xrightarrow{R_1} A_2 \xrightarrow{R_2} \cdots \xrightarrow{R_l} A_{l+1}$
(abbreviated as $A_1 A_2 \ldots A_{l+1}$)
which describes a composite relation $R = R_1 \circ R_2 \circ \cdots R_l$ between objects $A_1$ and $A_{l+1}$
, where $\circ$ denotes the composition operator on relations.




(a) Network Schema (b) Meta-Path: APV (c) Meta-Path: APA

# Metapath2vec Framework

Metapath2vec uses Meta−Path−Based Random Walks.

If we ignore types, heterogeneous random walks are biased to highly visible types of nodes.

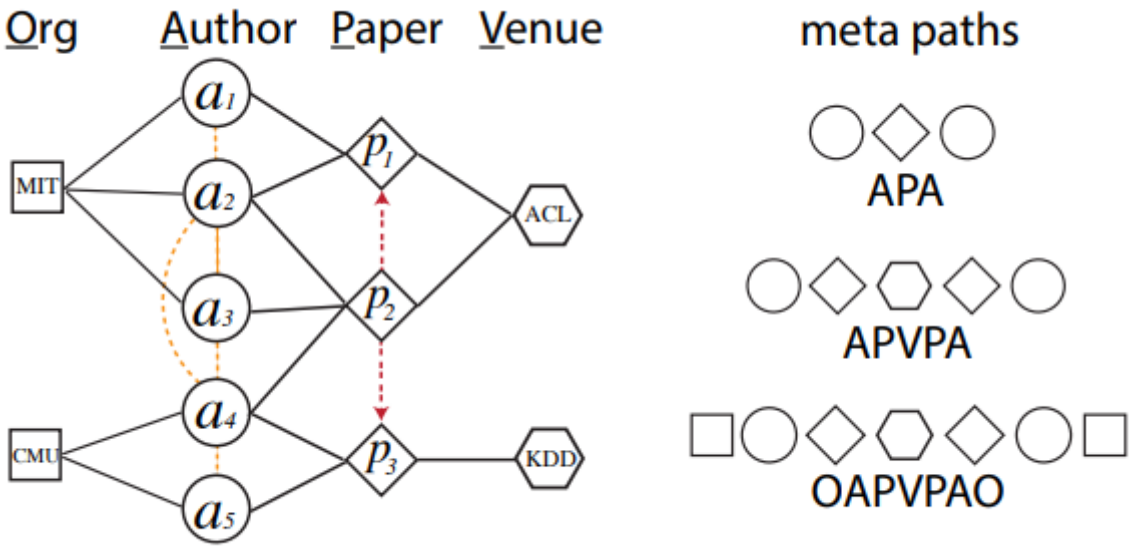
Meta−Path−Based Random Walks can generate paths that are able to capture both the semantic and structural correlations between different types of nodes.

$$p(v^{i+1}|v_t^i, \mathcal{P}) = \begin{cases} \frac{1}{|N_{t+1}(v_t^i)|} & (v^{i+1}, v_t^i) \in E, \phi(v^{i+1}) = t+1 \\ 0 & (v^{i+1}, v_t^i) \in E, \phi(v^{i+1}) \neq t+1 \\ 0 & (v^{i+1}, v_t^i) \notin E \end{cases}$$

# Metapath2vec Framework

In a traditional random walk procedure, the next step of a walker on node a4 transitioned from node CMU can be all types of nodes surrounding it—a2, a3, a5, p2, p3, and CMU.

Under the meta−path scheme 'OAPVPAO', for example, the walker is biased towards paper nodes (P) given its previous step on an organization node CMU (O), following the semantics of this path.



# Metapath2vec Framework

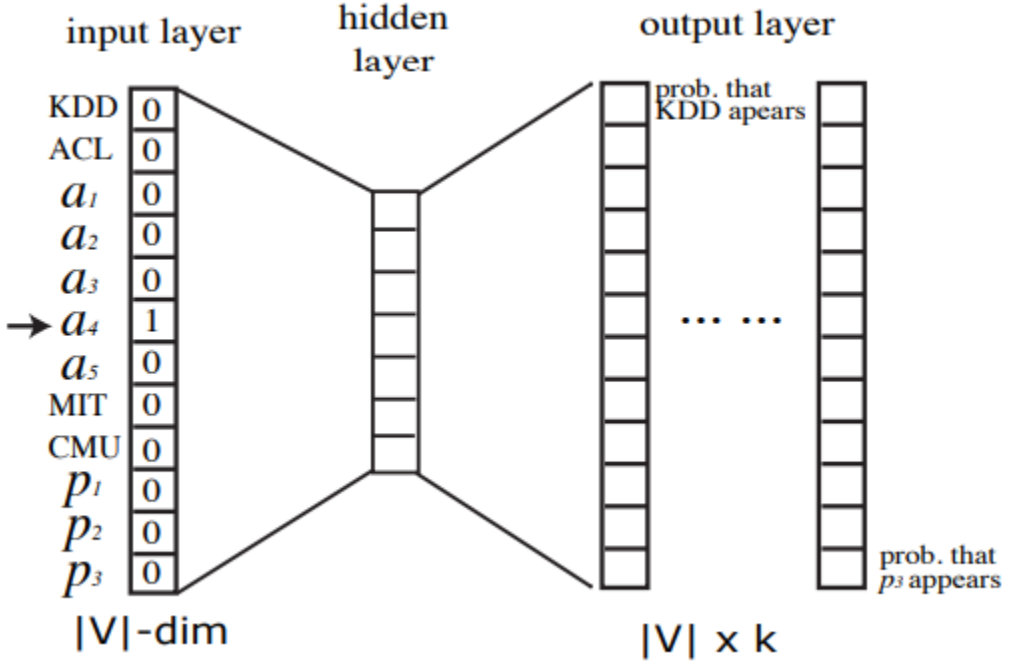
Heterogeneous Skip−gram model wants to maximize the probability of having the heterogeneous context given node v.

$$\arg\max_{\theta} \sum_{v \in V} \sum_{t \in T_V} \sum_{c_t \in N_t(v)} \log p(c_t | v; \theta)$$

where $p(c_t | v; \theta) = \frac{e^{X_{c_t} \cdot X_v}}{\sum_{u \in V} e^{X_u \cdot X_v}}$

To achieve efficient optimization, we use negative sampling method. Given negative sample sizes M, the objective is

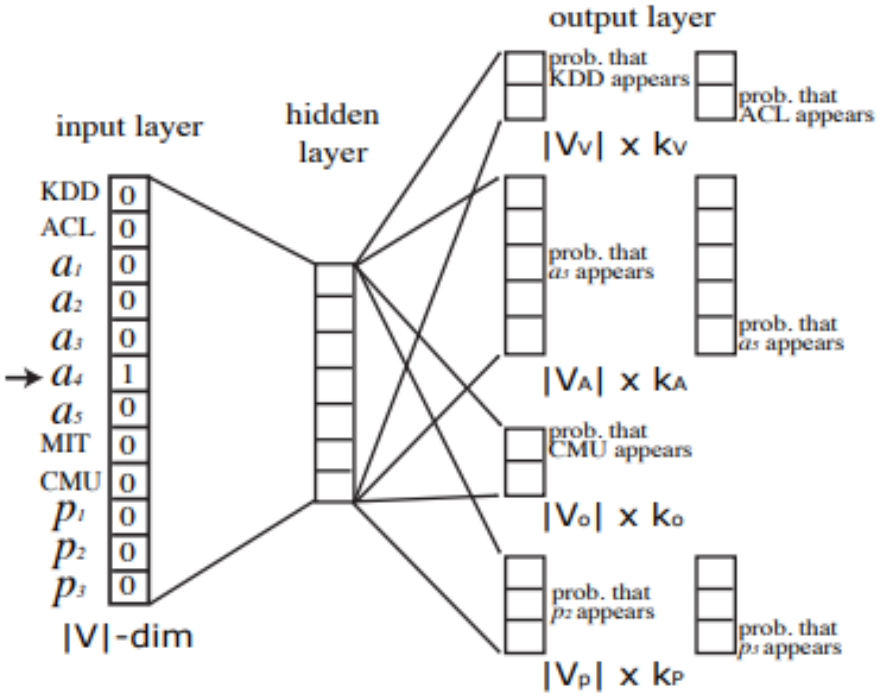
$$\log \sigma(X_{c_t} \cdot X_v) + \sum_{m=1}^{M} \mathbb{E}_{u^m \sim P(u)} [\log \sigma(-X_{u^m} \cdot X_v)]$$



# Metapath2vec++ Framework

Actually, Metapath2vec ignores the node type information in softmax function.

So, in order to infer the specific type of context given a node v,
metapath2vec actually encourages all types of negative samples,
including nodes of the same type t as well as
the other types in the heterogeneous network.



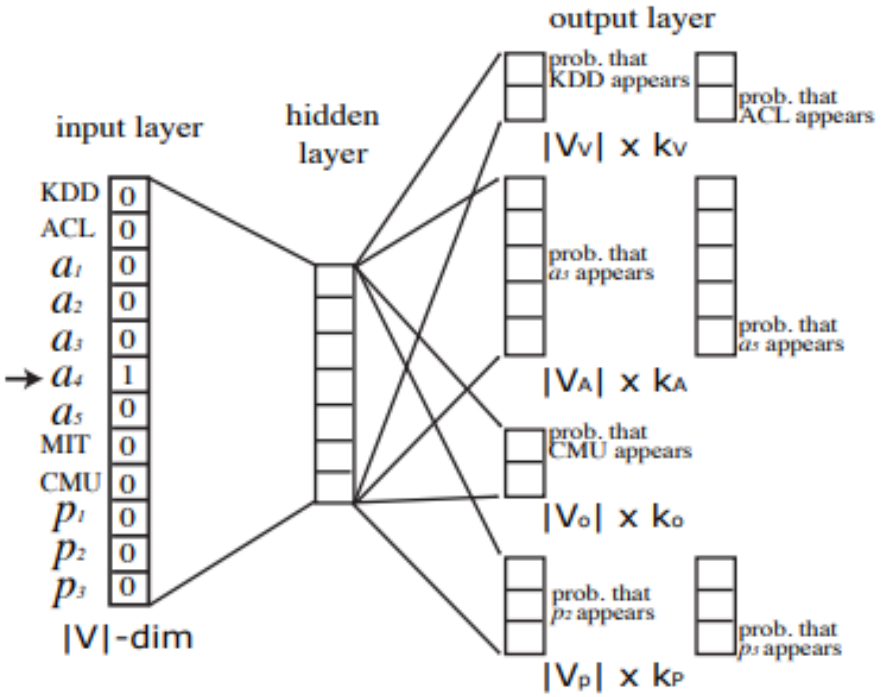
# Metapath2vec++ Framework

In Metapath2vec++ framework, the softmax function is normalized with respect to the node type of the context.

$$p(c_t|v;\theta) = \frac{e^{X_{c_t}\cdot X_v}}{\sum_{u_t\in V_t} e^{X_{u_t}\cdot X_v}}$$

Also, the negative sampling distribution is also specified by the node type of the neighbor that is targeted to predict

$$\log\sigma(X_{c_t}\cdot X_v) + \sum_{m=1}^{M} \mathbb{E}_{u_t^m\sim P_t(u_t)}[\log\sigma(-X_{u_t^m}\cdot X_v)]$$



**Input:** The heterogeneous information network $G = (V, E, T)$, a meta-path scheme $\mathcal{P}$, #walks per node $w$, walk length $l$, embedding dimension $d$, neighborhood size $k$

**Output:** The latent node embeddings $\mathbf{X} \in \mathbb{R}^{|V| \times d}$

initialize $\mathbf{X}$ ;
**for** $i = 1 \rightarrow w$ **do**
  **for** $v \in V$ **do**
    $MP$ = MetaPathRandomWalk($G$, $\mathcal{P}$, $v$, $l$) ;
    $\mathbf{X}$ = HeterogeneousSkipGram($\mathbf{X}$, $k$, $MP$) ;
  **end**
**end**
return $\mathbf{X}$ ;

**MetaPathRandomWalk**($G$, $\mathcal{P}$, $v$, $l$)
$MP[1] = v$ ;
**for** $i = 1 \rightarrow l-1$ **do**
  draw u according to Eq. 3 ;
  $MP$[i+1] = $u$ ;
**end**
return $MP$ ;

**HeterogeneousSkipGram**($\mathbf{X}$, $k$, $MP$)
**for** $i = 1 \rightarrow l$ **do**
  $v = MP$[i] ;
  **for** $j = max(0, i-k) \rightarrow min(i+k, l)$ & $j \neq i$ **do**
    $c_t = MP$[j] ;
    $X^{new} = X^{old} - \eta \cdot \frac{\partial O(\mathbf{X})}{\partial X}$ (Eq. 7) ;
  **end**
**end**

# Experiments

AMiner dataset:

AMiner CS dataset consists of 9,323,739 computer scientists and 3,194,405 papers from 3,883 computer science venues—both conferences and journals—held until 2016.

There are three types of nodes: authors, papers, and venues.

Links represent different types of relationships among three sets of nodes,

such as collaboration relationships on a paper.

Hyperparameter:

(1) number of walks per node w: 1000

(2) walk length : 100

(3) vector dimension d: 128

(4) neighborhood size k: 7

(5) size of negative samples: 5

# Experiments

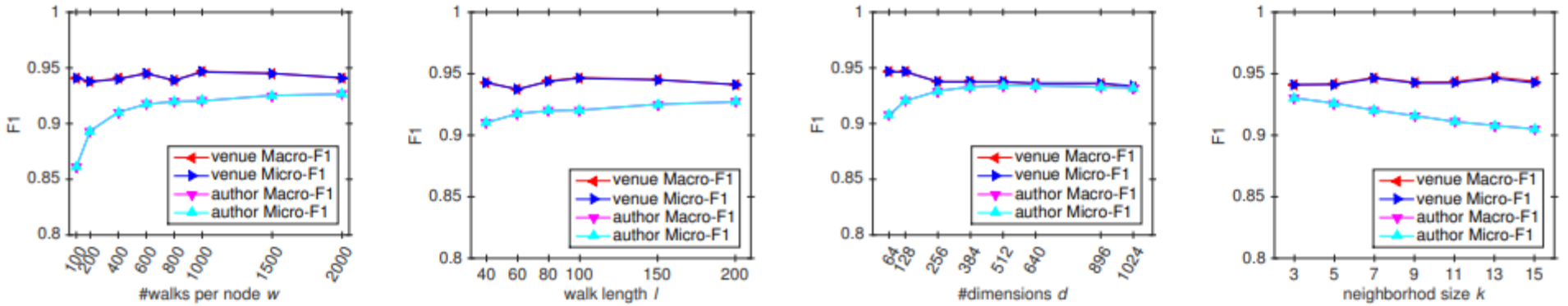
**Table 2: Multi-class venue node classification results in AMiner data.**

| Metric | Method | 5% | 10% | 20% | 30% | 40% | 50% | 60% | 70% | 80% | 90% |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Macro-F1 | DeepWalk/node2vec | 0.0723 | 0.1396 | 0.1905 | 0.2795 | 0.3427 | 0.3911 | 0.4424 | 0.4774 | 0.4955 | 0.4457 |
| | LINE (1st+2nd) | 0.2245 | 0.4629 | 0.7011 | 0.8473 | 0.8953 | 0.9203 | 0.9308 | 0.9466 | 0.9410 | 0.9466 |
| | PTE | 0.1702 | 0.3388 | 0.6535 | 0.8304 | 0.8936 | 0.9210 | 0.9352 | 0.9505 | 0.9525 | 0.9489 |
| | *metapath2vec* | 0.3033 | 0.5247 | 0.8033 | 0.8971 | 0.9406 | 0.9532 | 0.9529 | 0.9701 | 0.9683 | 0.9670 |
| | *metapath2vec++* | 0.3090 | 0.5444 | 0.8049 | 0.8995 | 0.9468 | 0.9580 | 0.9561 | 0.9675 | 0.9533 | 0.9503 |
| Micro-F1 | DeepWalk/node2vec | 0.1701 | 0.2142 | 0.2486 | 0.3266 | 0.3788 | 0.4090 | 0.4630 | 0.4975 | 0.5259 | 0.5286 |
| | LINE (1st+2nd) | 0.3000 | 0.5167 | 0.7159 | 0.8457 | 0.8950 | 0.9209 | 0.9333 | 0.9500 | 0.9556 | 0.9571 |
| | PTE | 0.2512 | 0.4267 | 0.6879 | 0.8372 | 0.8950 | 0.9239 | 0.9352 | 0.9550 | 0.9667 | 0.9571 |
| | *metapath2vec* | 0.4173 | 0.5975 | 0.8327 | 0.9011 | 0.9400 | 0.9522 | 0.9537 | 0.9725 | 0.9815 | 0.9857 |
| | *metapath2vec++* | 0.4331 | 0.6192 | 0.8336 | 0.9032 | 0.9463 | 0.9582 | 0.9574 | 0.9700 | 0.9741 | 0.9786 |

Macro−F1 score: $\frac{sum\ of\ F1\ scores}{num\ of\ Classes}$ Micro−F1 score: $\frac{TP}{TP+\frac{FP+FN}{2}}$

Micro F1 gives equal importance to each observation, while Macro F1 gives each class equal importance.

# Experiments



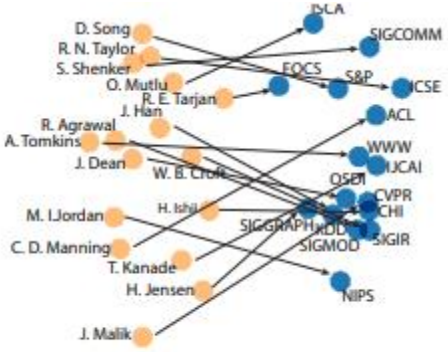

(a) #walks per node $w$ (b) walk length $l$ (c) #dimensions $d$ (d) neighborhood size $k$

**Figure 3: Parameter sensitivity in multi-class node classification.** 50% as training data and the remaining as test data.

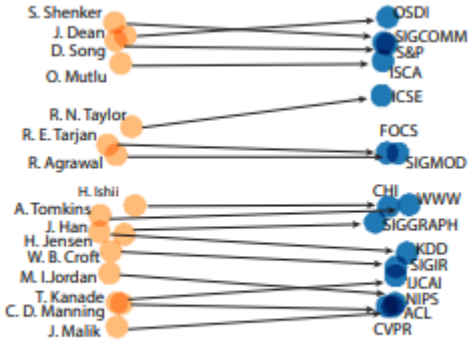
# Conclusion

1) Metapath2vec successfully adopts RandomWalk-based Embedding framework to heterogeneous graph.

2) Metapath2vec++ successfully differentiates different types of nodes and preserves their relations.




(a) DeepWalk / node2vec




(d) *metapath2vec++*

3) But, metapath2vec needs prior knowledge about metapath dependent on dataset.
(Graph Transformer(2019 NIPS) can generate metapath without prior knowledge)

# Implementation

## Aminer dataset

```
dataset = AMiner(path)
data = dataset[0]
```

```
print(data)
```

```
HeteroData(
  author={
    y=[246678],
    y_index=[246678],
    num_nodes=1693531
  },
  venue={
    y=[134],
    y_index=[134],
    num_nodes=3883
  },
  paper={ num_nodes=3194405 },
  (paper, written_by, author)={ edge_index=[2, 9323605] },
  (author, writes, paper)={ edge_index=[2, 9323605] },
  (paper, published_in, venue)={ edge_index=[2, 3194405] },
  (venue, publishes, paper)={ edge_index=[2, 3194405] }
)
```

## construct Adj matrix for each metapath

```
for keys, edge_index in edge_index_dict.items():
  shape = (num_nodes_dict[keys[0]], num_nodes_dict[keys[-1]])
  row, col = edge_index
  adj = SparseTensor(row=row, col=col, sparse_sizes = shape)
  adj = adj.to('cpu')
  adj_dict[keys] = adj
```

# Implementation

## Generate positive random walk

```
batch = batch.repeat(self.walks_per_node)
rws = [batch]
for i in range(self.walk_length):
  keys = self.metapath[i % len(self.metapath)]
  batch = sample(self.adj_dict[keys], batch, dummy_idx = self.dummy_idx).view(-1)
  rws.append(batch)
```

## Also, negative random walk

```
batch = batch.repeat(self.walks_per_node * self.negative_size)

rws = [batch]

for i in range(self.walk_length):
  keys = self.metapath[i%len(self.metapath)]
  batch = torch.randint(0, self.num_nodes_dict[keys[-1]], (batch.size(0), ), dtype=torch.long)
  rws.append(batch)
```

# Implementation

Compute positive loss/negative loss

```
positive_loss=-torch.log(torch.sigmoid(torch.einsum('bij,bkj->bk', [center_embedding,neighbor_embedding]).view(-1))+1e-15).mean()
```

$$\log \sigma(X_{c_t} \cdot X_v) +$$

```
negative_loss=-torch.log(1-torch.sigmoid(torch.einsum('bij,bkj->bk', [center_embedding,neighbor_embedding]).view(-1))+1e-15).mean()
```

$$\sum_{m=1}^{M} \mathbb{E}_{u^m \sim P(u)} [\log \sigma(-X_{u^m} \cdot X_v)]$$

# Implementation

Use embedding from metapath2vec for downstream task(multi class logistic regression)

| train ratio | 0.1 | 0.3 | 0.5 | 0.9 |
|---|---|---|---|---|
| Accuracy | 0.8973 | 0.9270 | 0.9256 | 0.9301 |