Heterogeneous and Dynamic Graph Representation Learning

Prof. O-Joun Lee

Dept. of Artificial Intelligence,
The Catholic University of Korea
ojlee@catholic.ac.kr

네트워크 과학 연구실 NETWORK SCIENCE LAB

가톨릭대학교 THE CATHOLIC UNIVERSITY OF KOREA

Contents

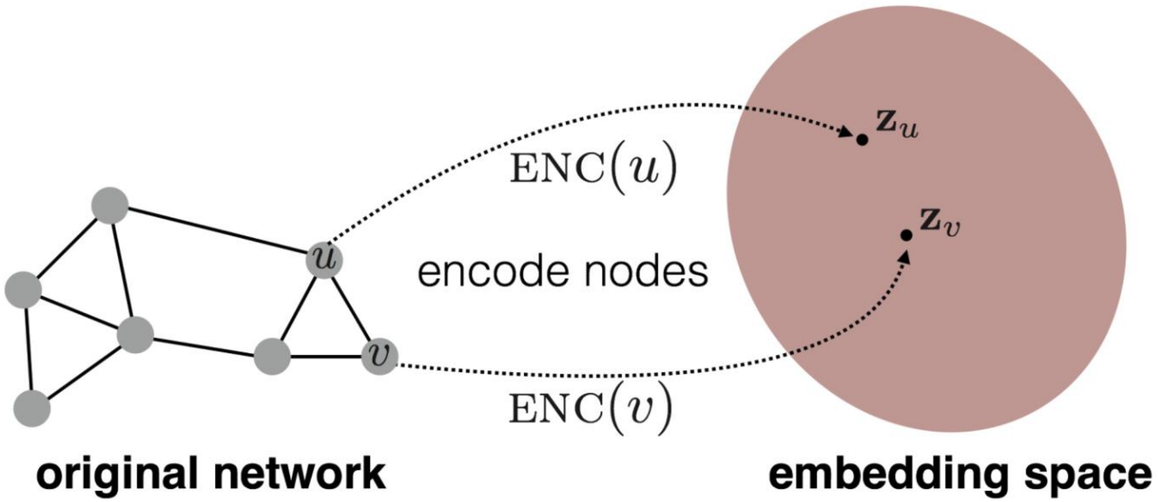
- From Homogeneous to Heterogeneous network.
- Heterogeneous Graph Representation Learning
  - metapath2vec: Scalable Representation Learning for Heterogeneous Networks
  - Are Meta-Paths Necessary?: Revisiting Heterogeneous Graph Embeddings.
  - BHIN2vec: Balancing the Type of Relation in Heterogeneous Information Network.
- Dynamic methods:
  - dynnode2vec: Scalable Dynamic Network Embedding.
  - Continuous-Time Dynamic Network Embeddings.

Reminding the Node Embeddings

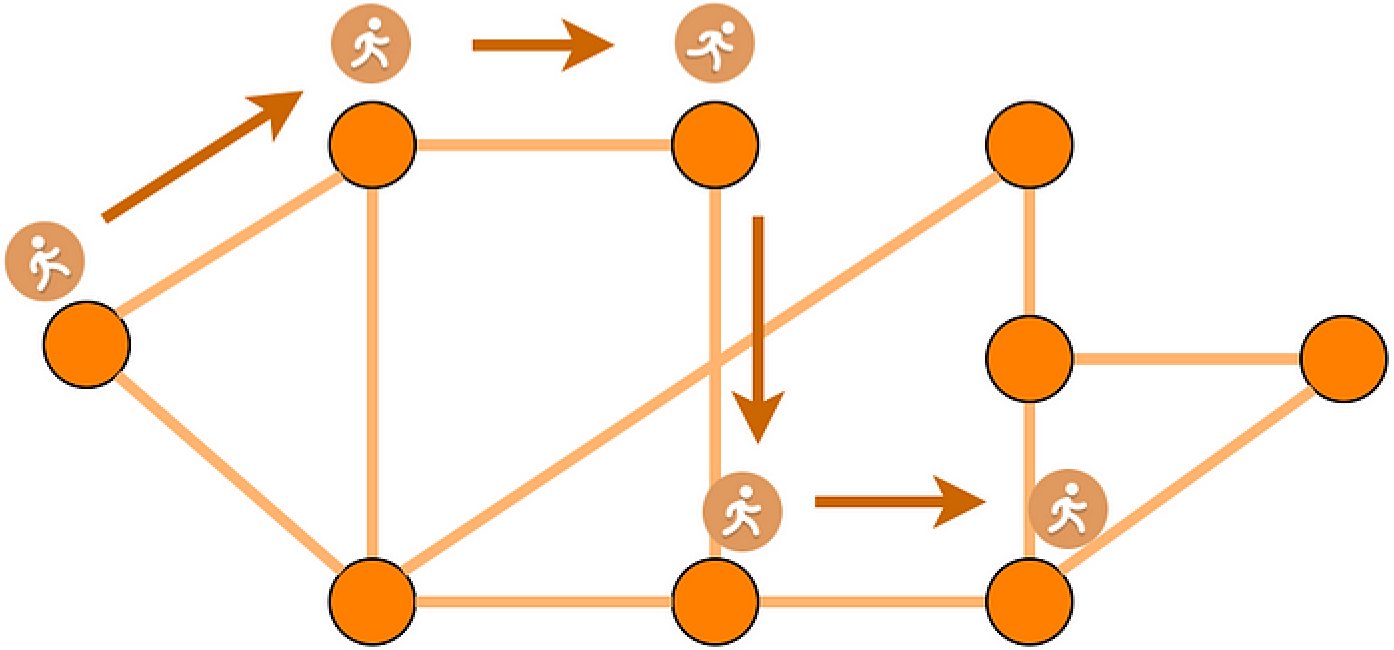
- Goal: Encode nodes → similarity in embedding space (dot product) ≈ similarity in the original graph

$$\underbrace{\text{similarity}(u, v)}_{\text{in the original network}} \approx \underbrace{\mathbf{z}_v^{\mathrm{T}} \mathbf{z}_u}_{\text{Similarity of the embedding}}$$

We need to define:
ENC(u)
Similarity(u, v)



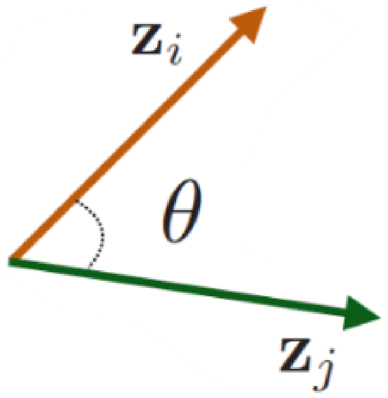
- Given a graph and a starting point, we select a neighbour of it at random, and move to this neighbour.
- Then, we select a neighbor of this point at random, and move to it,…
- The random sequence of nodes visited this way is **a random walk on the graph**



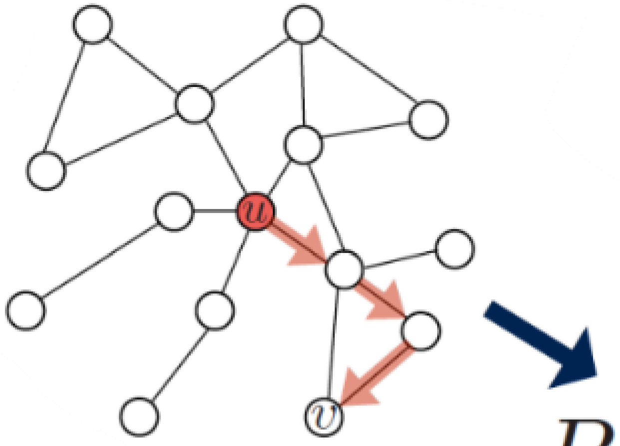
https://medium.com/analytics-vidhya/an-intuitive-explanation-of-deepwalk-84177f7f2b72

Random Walk Embeddings

- Estimate probability of visiting node v on a random walk starting from node u using some random walk strategy R.
- Optimize embeddings to encode these random walk statistics.



$$\mathbf{z}_u^{\mathrm{T}}\mathbf{z}_v \approx$$



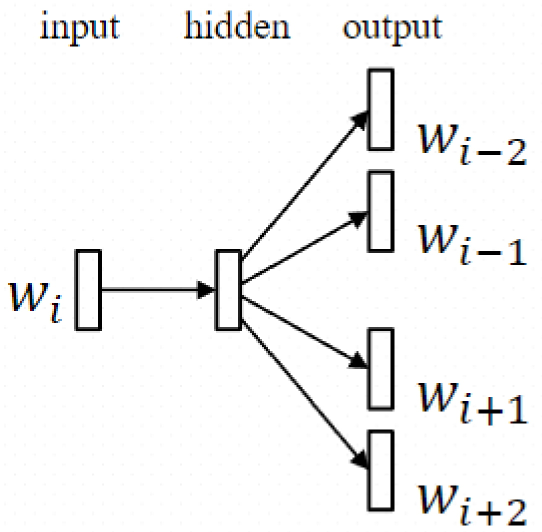
$$P_R(v|u)$$

Word Embedding in NLP

- Input: a text corpus $D = \{W\}$
- Output: $\boldsymbol{X} \in R^{|W| \times d}, d \ll |W|, d$-dim vector $\boldsymbol{X}_w$ for each word w.

- Computational lens on big social and information networks.
- The connections between individuals form the structural ...
- In a network sense, individuals matters in the ways in which ...
- Accordingly, this thesis develops computational models to investigating the ways that ...
- We study two fundamental and interconnected directions: user demographics and network diversity
- ... ...

Sentences



Word2vec

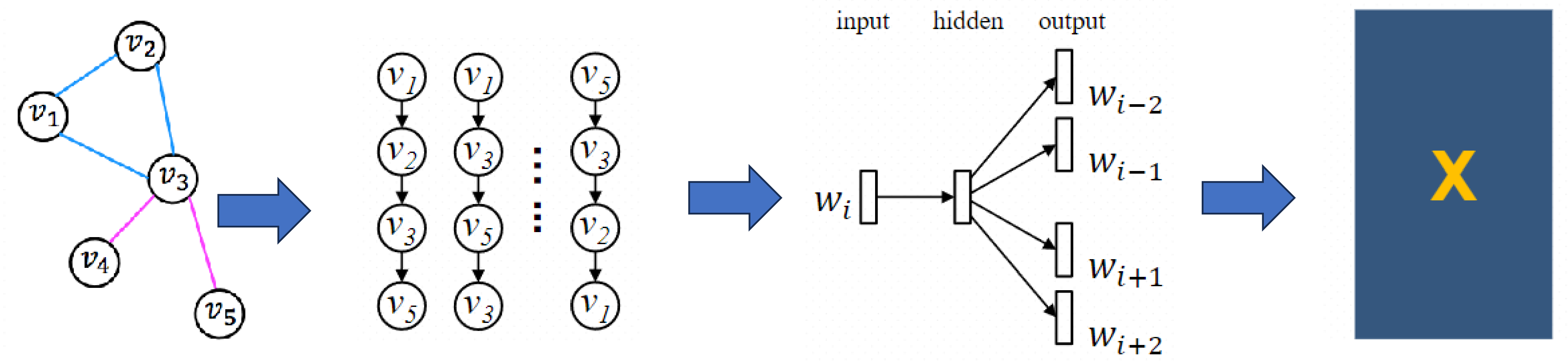
Latent representation vector

- geographically close words: a word and its context words -- in a sentence or document exhibit interrelations in human natural language.

T. Mikolov, I Sutskever, K Chen, GS Corrado, J Dean. Distributed representations of words and phrases and their compositionality. In NIPS '13, pp. 3111-31119

DeepWalk

- Input: a network $\mathrm{G} = (V, E)$
- Output: $\boldsymbol{X} \in R^{|V| \times d}$, $d \ll |V|$, d-dim vector $\boldsymbol{X}_v$ for each node V.



Random walk paths (sentences)

Word2vec (Skip-gram)

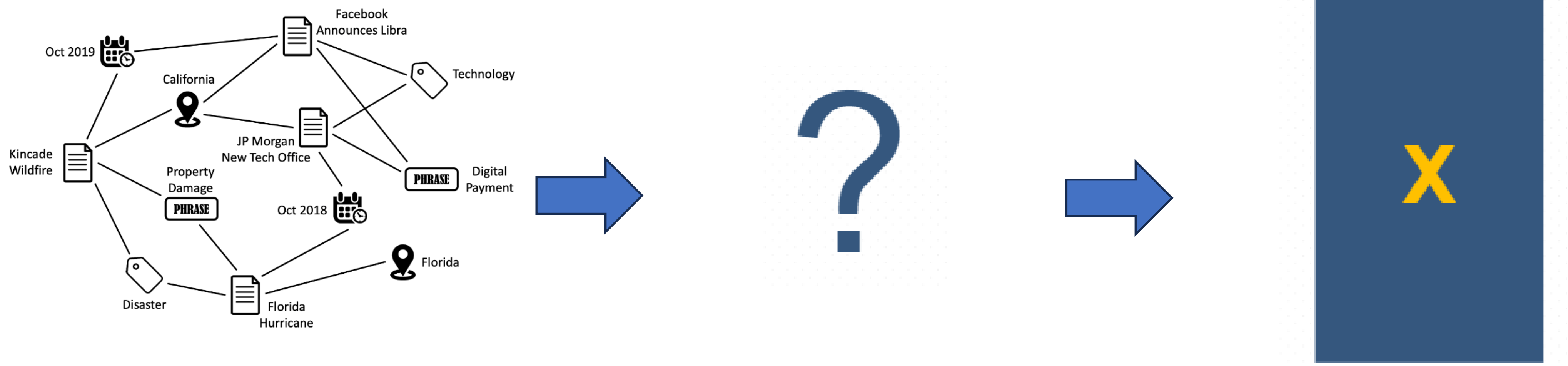
Latent representation vector

- Employ random walks on the graph to discover the structure

Bryan Perozzi et al: DeepWalk: Online Learning of Social Representations.

From Homogeneous to Heterogeneous Network Embedding: Problem

- Previous works are considering homogeneous network.
  - only one type of nodes and edges.
  - real world is ubiquitous. For example: social media websites like Facebook contain a set of node types, such as users, posts, groups and, tags.
  - special case of heterogeneous network.
- Input: a heterogeneous information network $G = (V, E, T)$. T is object relation type.
- Output: $\boldsymbol{X} \in R^{|V| \times d}, d \ll |V|$, d-dim vector $\boldsymbol{X}_v$ for each node V.

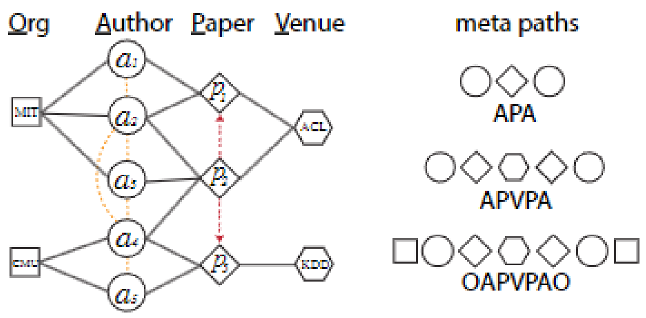


Latent representation vector

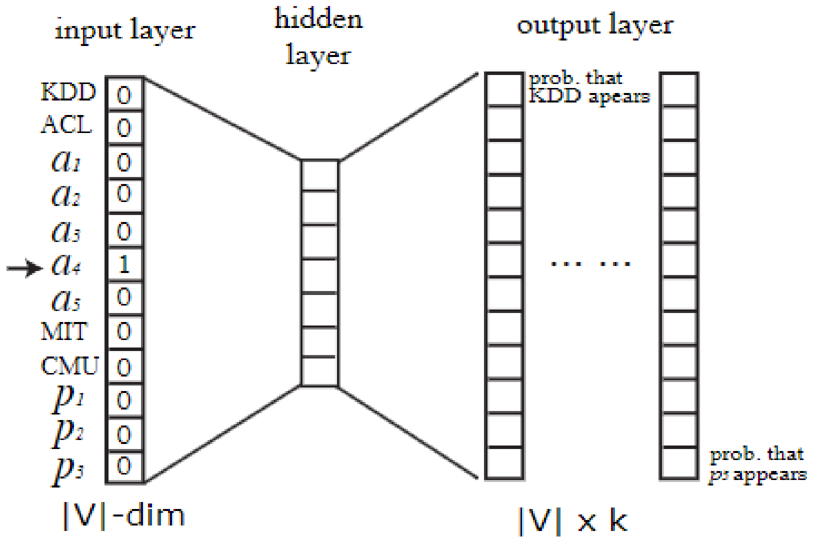
Heterogeneous Network Embedding: Challenges

- How do we effectively preserve the concept of “node-context” among multiple types of nodes?
  - e.g., users, posts, groups and, tags in social heterogeneous networks.
  - or authors, papers, & venues in academic heterogeneous networks.
- Can we directly apply homogeneous network embedding architectures to heterogeneous networks?
- It is also difficult for conventional meta-path-based methods to model similarities between nodes without connected meta-paths.

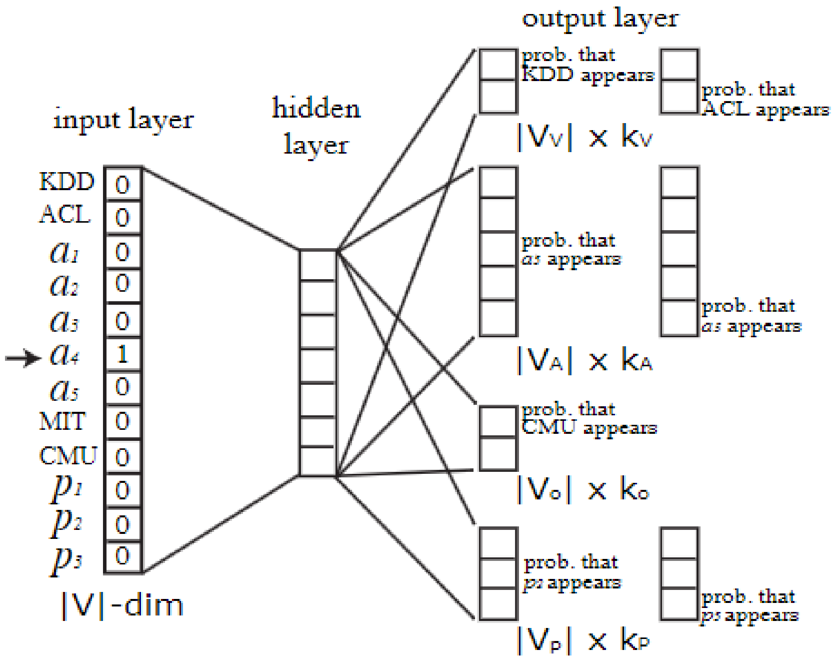
- **Solution**: meta-path based random walk (inspired by DeepWalk and Node2Vec)
  - metapath2vec and metapath2vec++.
- **Goal**: to generate paths that can capture both the semantic and structural correlations between different types of nodes, facilitating the transformation of heterogeneous network structures into skip-gram.



Heterogenous network



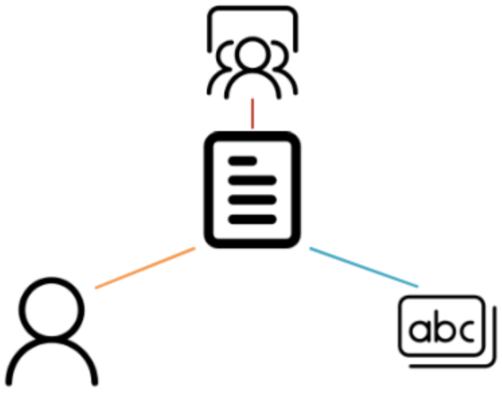
Skip-gram in metapath2vec



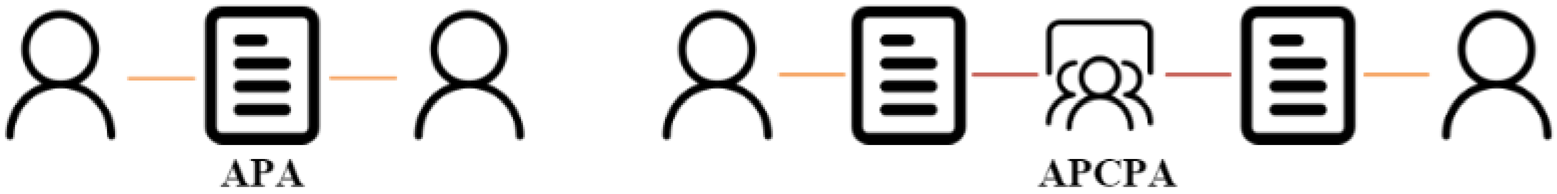
Skip-gram in metapath2vec++

Yuxiao Dong et al:"metapath2vec: Scalable representation learning for heterogeneous networks." *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, 2017.

- Network schema $S = (V, R)$ of graph G
  - directed graph defined over node types V and with edges as relations from R



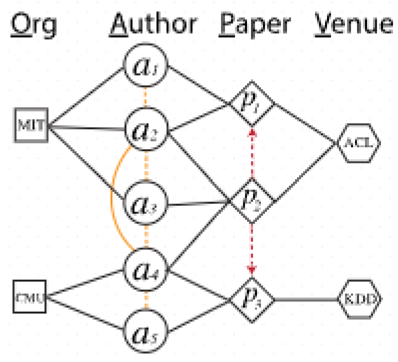
- meta-path: based on network schema S.
  - Denoted as $\boxed{V_1 \xrightarrow{R_1} V_2 \xrightarrow{R_2} \cdots V_t \xrightarrow{R_t} V_{t+1} \cdots \xrightarrow{R_{l-1}} V_l}$
  - Node types $V_1, V_2, \ldots, V_l \in V$ and edge type $R_1, R_2, \ldots, R_{l-1} \in R$



- Each meta-path captures the proximity between the nodes on its two ends from a particular semantic perspective.

- **Metapath2Vec**:
- Given a meta-path scheme:

$$V_1 \xrightarrow{R_1} V_2 \xrightarrow{R_2} \cdots V_t \xrightarrow{R_t} V_{t+1} \cdots \xrightarrow{R_{l-1}} V_l$$



- The transition probability at step i is defined as:

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

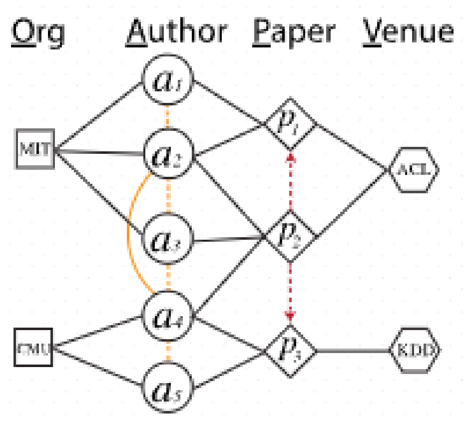
- Recursive guidance for random walkers, i.e.,

$$p(v^{i+1}|v_t^i) = p(v^{i+1}|v_1^i), \text{ if } t = l$$

- **Metapath2Vec**:
- Given a meta-path scheme (Example):

OAPVPAO

- 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 meta-path.



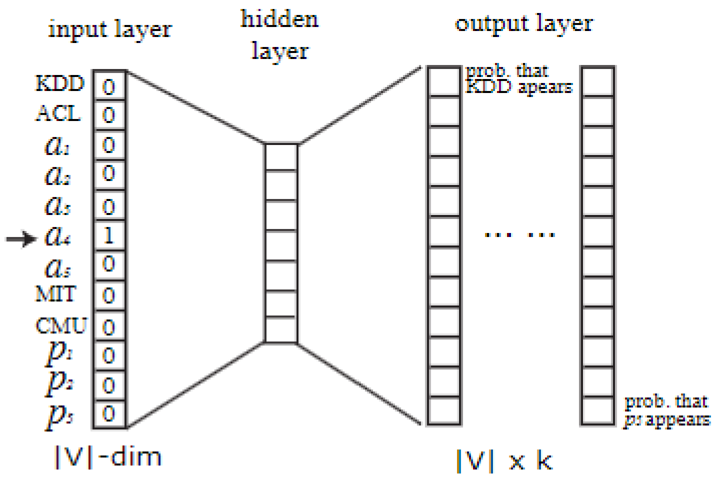
Yuxiao Dong et al:"metapath2vec: Scalable representation learning for heterogeneous networks." *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, 2017.

- Softmax in Metapath2Vec

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

Not consider node type

- The potential issue of skip-gram for heterogeneous network embedding:
  - To predict the context node c_t (type t) given a node v, metapath2vec encourages all types of nodes to appear in this context position.



- **Metapath2Vec++**: Heterogeneous Skip-Gram
- Objective function (heterogeneous negative sampling):

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

- Softmax in Metapath2Vec++

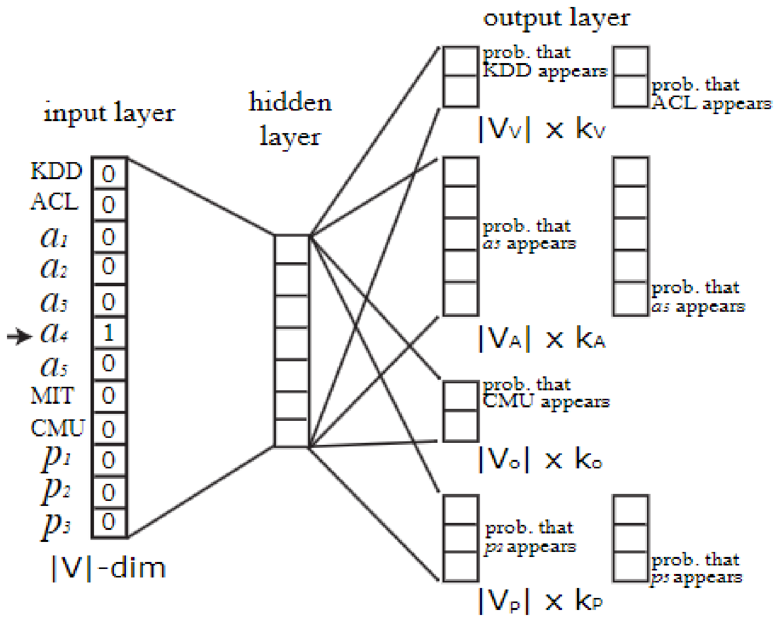
$$p(c_t|v;\theta) = \frac{e^{X_{c_t} \cdot X_v}}{\underline{\sum_{u_t \in V_t}} e^{X_{u_t} \cdot X_v}}$$

Consider node type t

- Stochastic gradient descent

$$\frac{\partial O(\mathrm{X})}{\partial X_{u_t^m}} = (\sigma(X_{u_t^m} \cdot X_v - \mathbb{I}_{c_t}[u_t^m]))X_v$$

$$\frac{\partial O(\mathrm{X})}{\partial X_v} = \sum_{m=0}^{M} (\sigma(X_{u_t^m} \cdot X_v - \mathbb{I}_{c_t}[u_t^m]))X_{u_t^m}$$



Metapath2Vec: Metapath2Vec++ Algorithm

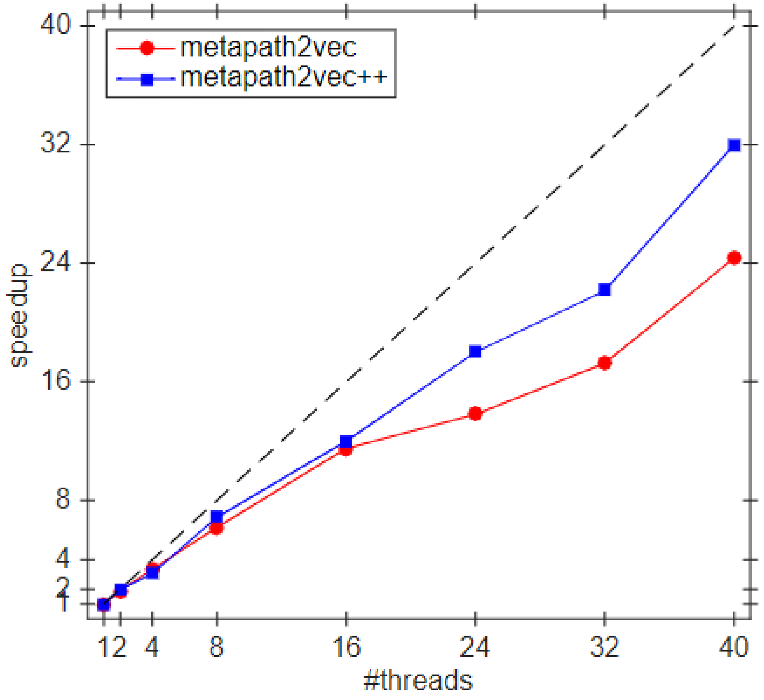
Input: The heterogeneous information network $G = (V, E, T)$, a meta-path scheme $\mathcal{P}$, #walks per node w, walk length l, embedding dimension d, neighborhood size k
Output: The latent node embeddings $\mathbf{X} \in \mathbb{R}^{|V| \times d}$

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

- Every sub-procedure is easy to parallelize.
- 24-32X speedup by using 40 cores.



Yuxiao Dong et al:"metapath2vec: Scalable representation learning for heterogeneous networks." *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, 2017.

Sample code of Random Walk of DeepWalk and Metapath2Vec in Karate Graph

```python
def random_walk(adj_list, node, walk_length):
  walk = [node]        # Walk starts from this node

  for i in range(walk_length-1):
    node = adj_list[node][random.randint(0,len(adj_list[node])-1)]
    walk.append(node)

  return walk

# Perform random walks on the graph
num_walks = 6

for node in karate_graph.nodes():
  print("Node " + str(node) + " :" + str(random_walk(adjacency_matrix_array, node, num_walks)))
```

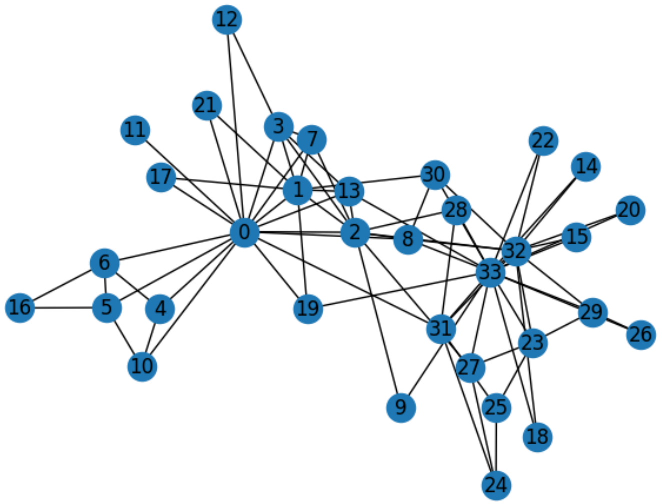
```
Node 0 :[0, 0, 2, 0, 4, 0]
Node 1 :[1, 0, 0, 0, 3, 0]
Node 2 :[2, 0, 0, 3, 0, 0]
Node 3 :[3, 3, 0, 0, 3, 0]
Node 4 :[4, 0, 3, 0, 0, 0]
Node 5 :[5, 0, 0, 0, 0, 0]
Node 6 :[6, 0, 0, 3, 0, 0]
Node 7 :[7, 0, 0, 0, 0, 2]
Node 8 :[8, 0, 3, 0, 2, 0]
Node 9 :[9, 0, 0, 5, 0, 0]
Node 10 :[10, 0, 2, 0, 0, 0]
Node 11 :[11, 0, 2, 0, 2, 0]
Node 12 :[12, 0, 3, 0, 3, 0]
Node 13 :[13, 3, 0, 5, 0, 0]
```

Deep Walk

```python
# Define the metapaths
metapaths = [["Member", "Club", "Member"], ["Member", "Club", "Member", "Club", "Member"]]

def random_walk_metapath2vec(adj_list, node, metapath, walk_length):
  walk = [node]        # Walk starts from this node

  for i in range(walk_length-1):
    next_hop_candidates = []
    current_node = walk[-1]
    for j in range(len(metapath)):
      if current_node in adj_list and metapath[j] in adj_list[current_node]:
        next_hop_candidates.extend(adj_list[current_node][metapath[j]])
    if len(next_hop_candidates) > 0:
      next_node = random.choice(next_hop_candidates)
    else:
      next_node = random.choice(adj_list[current_node])
    walk.append(next_node)

  return walk

# Perform random walks on the graph using metapath2vec
num_walks = 6

for node in karate_graph.nodes():
  for metapath in metapaths:
    print("Node " + str(node) + " using metapath " + str(metapath) + " :" + str(random_walk_metapath2vec(adjacency_matrix_array, node, metapath, num_walks)))
```

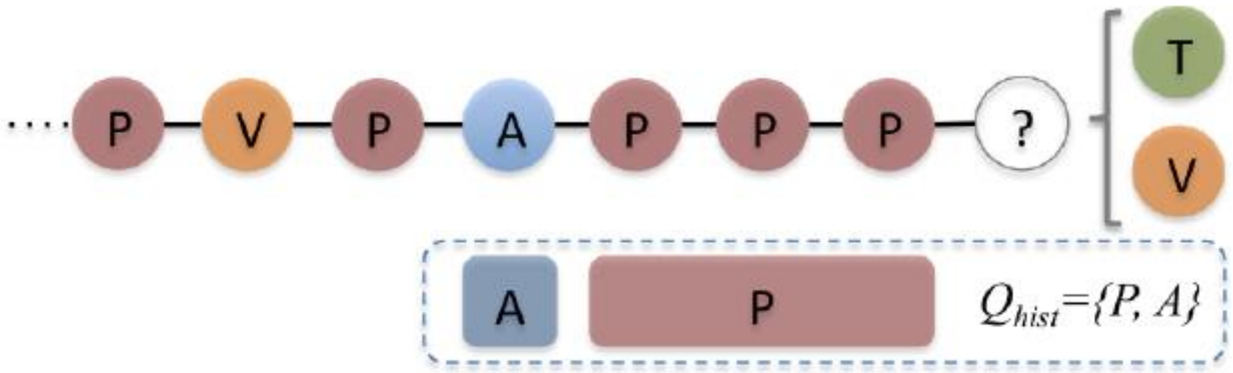
```
Node 0 using metapath ['Member', 'Club', 'Member'] :[0, 0, 0, 0, 3, 0]
Node 0 using metapath ['Member', 'Club', 'Member', 'Club', 'Member'] :[0, 3, 3, 0, 2, 0]
Node 1 using metapath ['Member', 'Club', 'Member'] :[1, 0, 0, 0, 0, 2]
Node 1 using metapath ['Member', 'Club', 'Member', 'Club', 'Member'] :[1, 0, 5, 0, 5, 0]
Node 2 using metapath ['Member', 'Club', 'Member'] :[2, 6, 0, 2, 0, 3]
Node 2 using metapath ['Member', 'Club', 'Member', 'Club', 'Member'] :[2, 0, 2, 0, 2, 0]
Node 3 using metapath ['Member', 'Club', 'Member'] :[3, 0, 3, 0, 0, 2]
Node 3 using metapath ['Member', 'Club', 'Member', 'Club', 'Member'] :[3, 0, 3, 0, 2, 0]
Node 4 using metapath ['Member', 'Club', 'Member'] :[4, 0, 3, 0, 0, 0]
Node 4 using metapath ['Member', 'Club', 'Member', 'Club', 'Member'] :[4, 3, 0, 2, 0, 0]
```

Metapath2Vec



- Meta-paths must be manually customized based on task and dataset, hence requiring domain knowledge.
- They fail to capture more complex relationships such as motifs, i.e. patterns of interconnections occurring in complex networks at numbers that are significantly higher than those in randomized networks.
- The usage of meta-path is limited to the discrete space. So, if two vertices are not structurally connected in the graph, metapath-based methods cannot capture their relations.

- **Solution**: propose JUST, which performs random walks by probabilistically deciding whether to "jump" to a different node type or "stay" in the same node type
  - no rely on predefined meta-paths.
  - **balancing** between **homogeneous** edges (same node type) and **heterogeneous** edges (across node types).
  - balances the node distribution over different node types by controlling the jumping behavior.



- Perform random walks on the heterogeneous graph, but probabilistically decide at each step.
  - **Jump** to a target domain q: uniformly sampling one node from those in a target domain q connected to v_i via heterogeneous edge

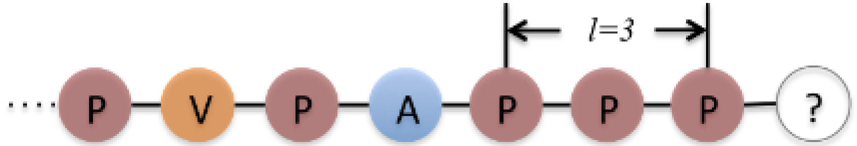
$$V_{jump}^{q}(v_i) = \{v|(v_i, v) \in E_{he} \vee (v_i, v) \in E_{he}, \phi(v) = q$$

  - **Stay** in the current domain: uniformly sampling one node from those connected to vi via homogeneous edges

$$V_{stay}(v_i) = \{v|(v_i, v) \in E_{ho} \vee (v_i, v) \in E_{ho}$$

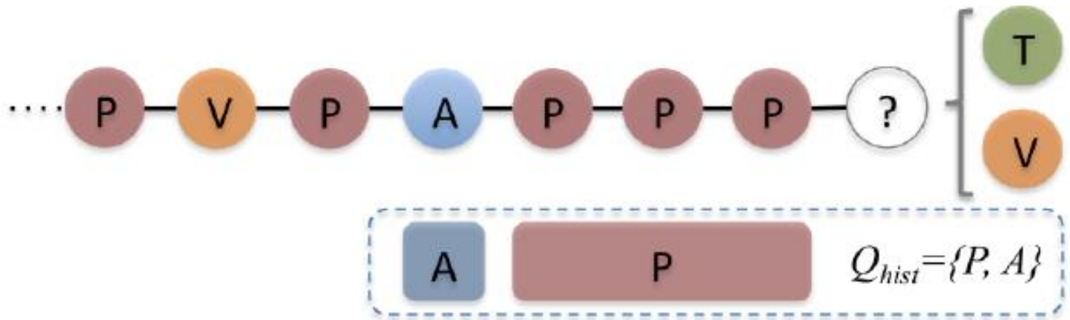
- **Probability** for stay or jump is controlled by an exponential decay function:
  - Stay:

$$Pr_{stay}(v_i) = \begin{cases} 0, \text{ if } V_{stay}(v_i) = \emptyset \\ 1, \text{ if } \{V^q_{jump}(v_i) | q \in Q, q \neq \phi(v_i)\} = \emptyset \\ \alpha^l, \ otherwise \end{cases}$$



  - Jump:

$$Q_{Jump}(v_i) = \begin{cases} \{q | q \in Q \wedge q \notin Q_{hist}, V^q_{jump}(v_i) \neq \emptyset\}, \text{ if not empty} \\ \{q | q \in Q, q \neq \phi(v_i), V^q_{jump}(v_i) \neq \emptyset\}, \ otherwise \end{cases}$$



Algorithm 1 Truncated Random Walk with Jump & Stay

Require: A heterogeneous graph $G = (V, E)$, initial stay probability α, number of memorized domains m, number of random walks per node r, maximum walk length L_{max}

Initialize an empty set of walks $\mathcal{W} = \emptyset$
for $i = 1$ to r **do**
    for each $v \in V$ **do**
        Initialize a random walk by adding v;
        Initialize Q_{hist} by adding $\phi(v)$;
        while $|W| < L_{max}$ **do**
            Pick a Jump or Stay decision according to Eq. 1;
            if Stay **then**
                Continue W by staying;
            else if Jump **then**
                Sample a target domain q from Eq. 2;
                Continue W by jumping to domain q;
                Update Q_{hist} by keeping only last m domains;
            end if
        end while
        Add W to $\mathcal{W}$
    end for
end for
return The set of random walks $\mathcal{W}$

- Apply Skip-gram model same as metapth2vec.
- Different with Metapath2Vec:
  - Decide probabilistically whether to "jump" across node types via a heterogeneous edge or "stay" in the same node type via a homogeneous edge.
  - When jumping, select the target node type probabilistically while trying to avoid recently visited node types.

- **Motivation**: random walk produces an imbalanced training of heterogeneous networks
  - the major relation types take a large portion of training samples.
  - dominate the training and minor relation types will hardly be learned.
- **Idea**: Formulating heterogeneous network embedding as a multi-task learning problem, where each task corresponds to a relation type in the network.
  - allowing handling the imbalance by focusing on under-trained relation types
- **Solution**: Introducing an inverse training ratio tensor that quantifies how well each relation type is represented in the embedding space based on the task losses.
  - Proposing a biased random walk strategy that uses the inverse training ratios to generate walks containing more of the under-trained relation types.

- **Biased random walk generator**:
  - determine the type for the next node by sampling and do another sampling for the next node that has the sampled type.
- A stochastic matrix to store transition probabilities between node types

$$P_{ij} = p\left(t_j | t_i\right) \text{ such that } \sum_j P_{ij} = 1$$

- For k steps:

$$(P^k)_{ij} = p\left(t_j | t_i, k\right)$$

- **From inverse training ratio tensor to stochastic matrix**:
  - Sample more of less-trained relations so that the less-trained relations would be reflected more in the embedding space.
- Perturbation approach with uniform probability:

$$P_{uni_{xy}} = \begin{cases} \frac{1}{degree(t_x)} & \text{if } (t_x, t_y) \in E_{meta} \\ 0 & \text{otherwise} \end{cases}$$

- Probability to move from t_i to t_j in k steps

Perturbation parameter

$$L_{stochastic} = \sum_{i=0}^{k-1} \left| P^{i+1} - \left(P_{uni}^{i+1} + \alpha \left(I_i - 1 \right) \right) \right|_F^2$$

- Update stochastic matrix: update nonzero values and clip the values between zero and one
  - preserve the property in thestochastic matrix,

- **Skip-gram model**: learn embedding table Q where $f(v_i) = Q[i]$.
- The skip-gram loss for one random walk w

$$L = -\sum_{i=1}^{l}\sum_{j=1}^{k}\log p\left(w_{i+j} \mid w_i\right) = -\sum_{i=1}^{l}\sum_{j=1}^{k}\log \frac{e^{f\left(w_{i+j}\right)^{\top} f\left(w_i\right)}}{\sum_{v_n}^{V} e^{f\left(v_n\right)^{\top} f\left(w_i\right)}}.$$

Negative log-likelihood (Softmax function)

Take m samples

$$L = -\sum_{i=1}^{l}\sum_{j=1}^{k}\left(L_p\left(w_{i+j}, w_i\right) + \sum_{v_o}^{N_V} L_n\left(v_o, w_i\right)\right)$$

$$L_p\left(v_c, v_s\right) = \log\sigma\left(f\left(v_c\right)^{\top} f\left(v_s\right)\right)$$

$$L_n\left(v_c, v_s\right) = \log\sigma\left(-f\left(v_c\right)^{\top} f\left(v_s\right)\right),$$

Sigmoid function

- **Multi-task setting**: possible task set is defined

$$\mathfrak{J}_{possible} = \left\{ J_{xyz} \mid \left(A^{z}\right)_{xy} > 0 \right\}$$

$$A = \text{the adjacency matrix of } G_{meta}$$

- Balance in multitasks:

$$L\left[J_{xyz}\right] = -\frac{\sum_{i=1}^{l}\left(L\left[J_{xyz}\right]_{p}\left(w_{i+z+1}, w_{i}\right) + \sum_{v_o}^{N_V} L\left[J_{xyz}\right]_{n}\left(v_o, w_i\right)\right)}{\sum_{i=1}^{l}\left(\mathbb{I}\left[J_{xyz}\right]\left(w_{i+z+1}, w_{i}\right) + \sum_{v_o}^{N_V} \mathbb{I}\left[J_{xyz}\right]\left(v_o, w_i\right)\right)}$$

$$L\left[J_{xyz}\right]_{p}\left(v_c, v_s\right) = \begin{cases} L_p\left(v_c, v_s\right) & \text{if } \phi\left(v_c\right) = t_y \wedge \phi\left(v_s\right) = t_x \\ 0 & \text{otherwise} \end{cases}$$

$$L\left[J_{xyz}\right]_{n}\left(v_c, v_s\right) = \begin{cases} L_n\left(v_c, v_s\right) & \text{if } \phi\left(v_c\right) = t_y \wedge \phi\left(v_s\right) = t_x \\ 0 & \text{otherwise} \end{cases}$$

$$\mathbb{I}\left[J_{xyz}\right]\left(v_c, v_s\right) = \begin{cases} 1 & \text{if } \phi\left(v_c\right) = t_y \wedge \phi\left(v_s\right) = t_x \\ 0 & \text{otherwise} \end{cases}$$

- The inverse training ratios (some relation types are not contained in a random walk by chance)

$$\tilde{L}\left[J_{xyz}\right](t) = L\left[J_{xyz}\right] / L_{initial}\left[J_{xyz}\right]$$
$$r\left[J_{xyz}\right](t) = \tilde{L}\left[J_{xyz}\right](t) / \mathbb{E}_{\mathcal{J}_{possible}}\left[\tilde{L}\left[J\right](t)\right],$$

- The tasks that always not occur in a random walk

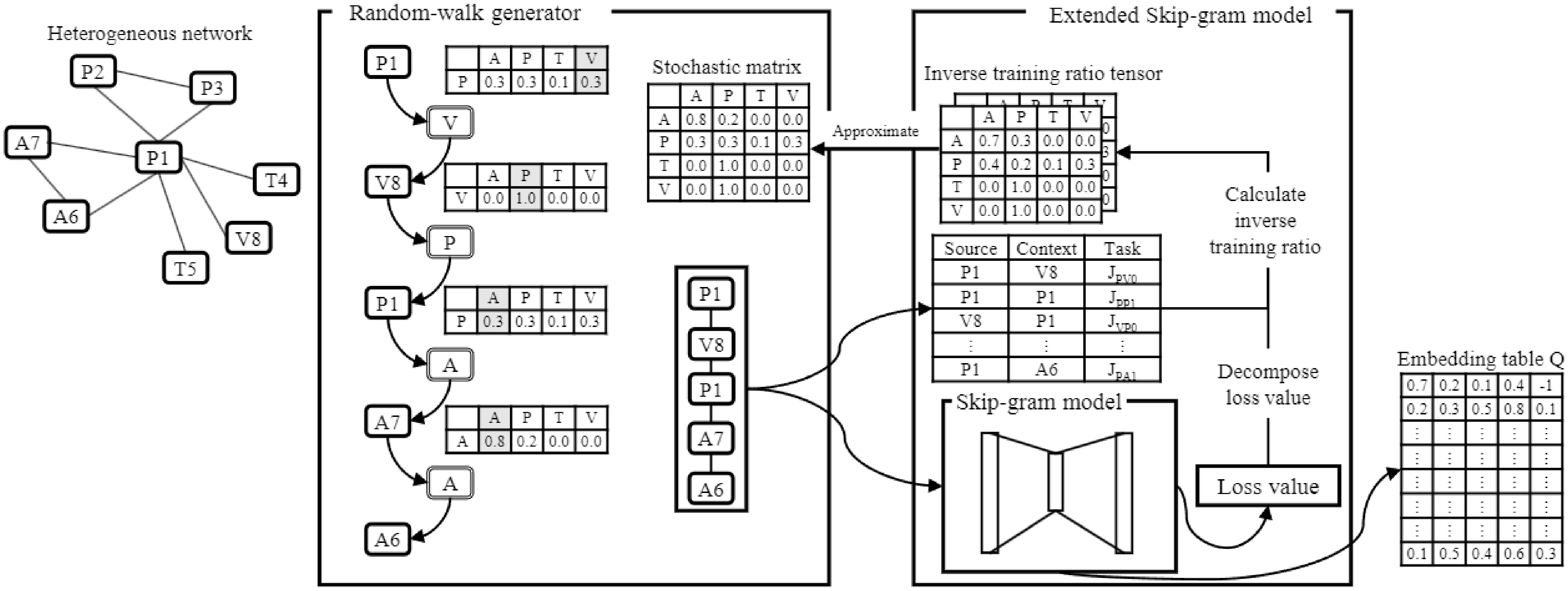
$$I_{zxy}(t) = \begin{cases} r[J_{xyz}](t) & \text{if } J_{xyz} \in \mathcal{J}_{possible} \\ 1 & \text{otherwise} \end{cases}$$

- Heterogeneous skip-gram model: sample negative nodes which have the same type with positive node

$$L_p(v_c, v_s) = \log\sigma\left(\left(\sqrt{r}\odot f(v_c)\right)^{\top}\left(\sqrt{r}\odot f(v_s)\right)\right)$$
$$L_n(v_c, v_s) = \log\sigma\left(-\left(\sqrt{r}\odot f(v_c)\right)^{\top}\left(\sqrt{r}\odot f(v_s)\right)\right)$$
$$r = f_R(k, \phi(v_c), \phi(v_s))$$

Seonghyeon Lee et al:"BHIN2vec: Balancing the Type of Relation in Heterogeneous Information Network."

BHIN2vec: Algorithm’s Diagram



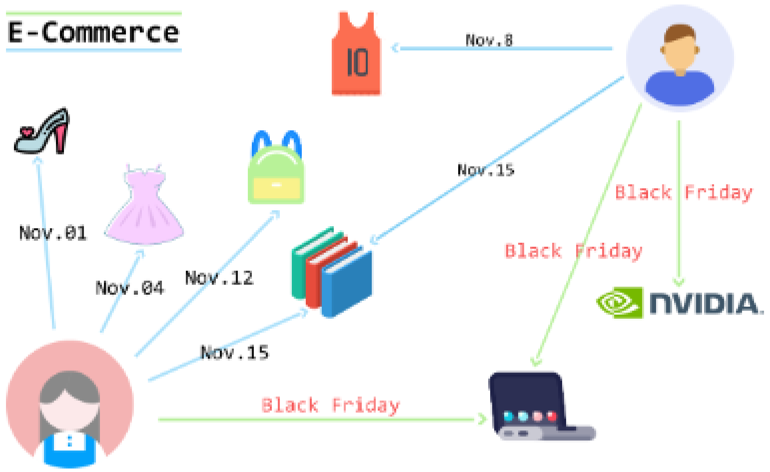
Biased Random Walk

Skip-gram model

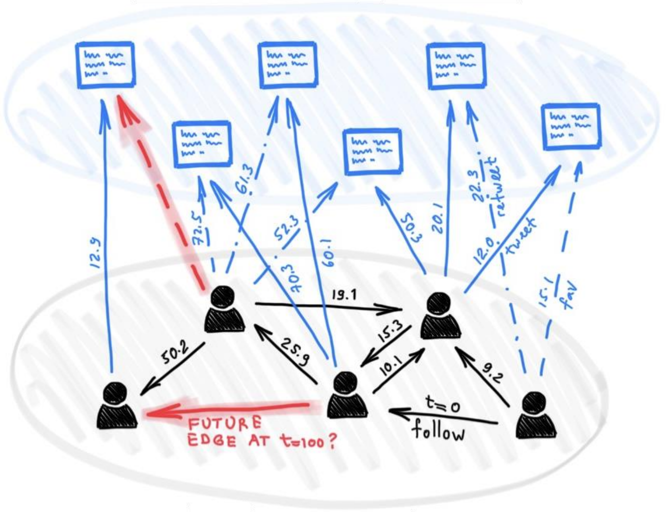
Seonghyeon Lee et al:"BHIN2vec: Balancing the Type of Relation in Heterogeneous Information Network.”

Dynamic Network Embedding

- Previous random walk methods – DeepWalk, Node2Vec, LINE, Struc2Vec, Metapath2Vec, etc – are relied on static graph.
- However, the networks in the real world are dynamic
  - evolving over time.



An illustration of user-item graph.



An illustration of social graph.

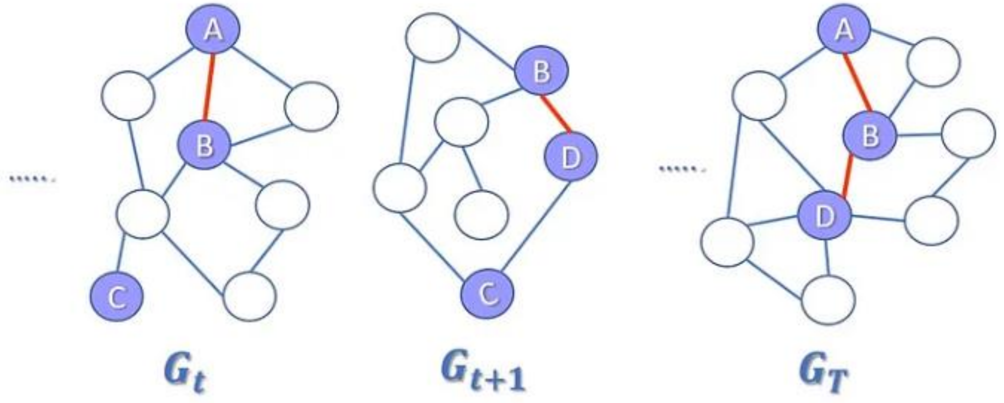
Guotong Xue et al:"Dynamic Network Embedding Survey."

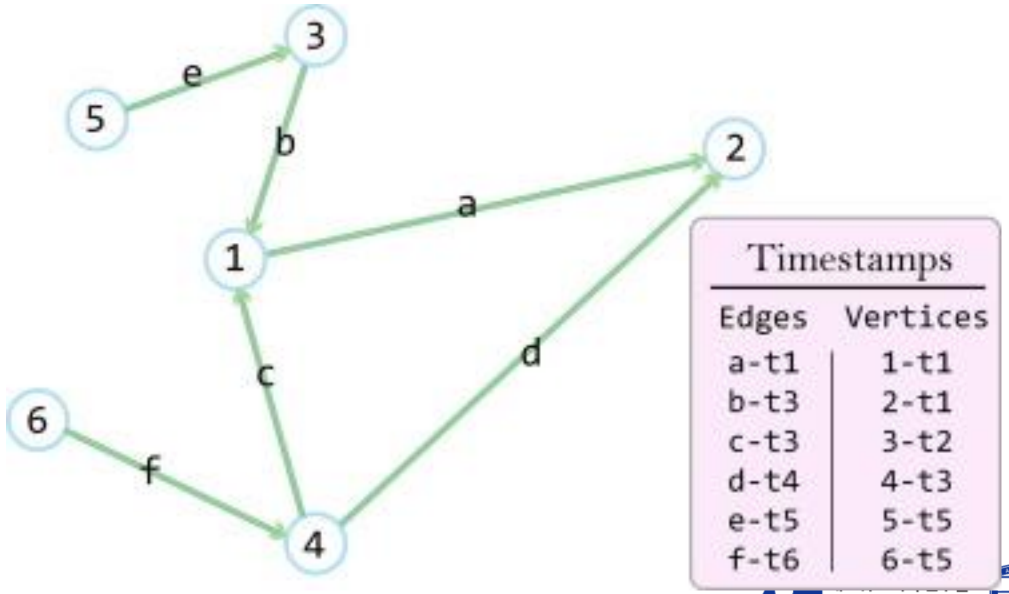
Dynamic Network Embedding: Problem and Challenges

- **Problem**: Learning dynamic node representations.
- **Challenges**:
  - Time-varying graph structures: links and node can emerge and disappear; communities are changing all the time.
    - requires the node representations capture both structural proximity (as in static cases) and their temporal evolution.
    - Time intervals of events are uneven.
  - Causes of the change: can come from different aspects, e.g.in co-authorship network, research community & career stage perspectives.
    - requires modeling multi-faceted variations.

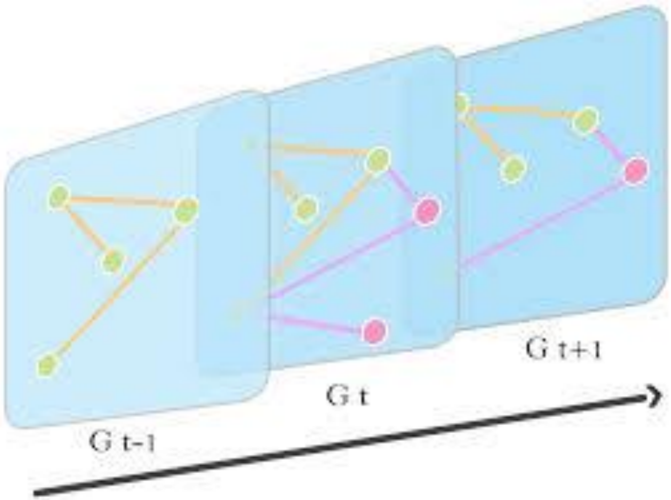
Dynamic Network Embedding: Problem and Challenges

- **2 ways to model dynamic**: discrete model and continuous model.
  - Discrete model: sequence of network snapshots within a given time interval
    - $G = \{G_1, \ldots, G_T\}$, where T is the number of snapshots. Each snapshot $G_t = (V_t, E_t)$ is a static network recorded at time t.
  - Continuous model: a network with edges and nodes annotated with timestamps
    - We have $G = (V_T, E_T, \mathfrak{I})$ where $\mathfrak{I} : V, E \rightarrow \mathbb{R}^+$ is a function that maps each edge and node to a corresponding timestamp.





- **Motivation**: Networks in the real world are always evolving
  - new users (new vertices) in social networks, new citations (new edges) in citation networks.
  - Users may delete friends (delete edges) or some users may leave the network (delete nodes).
- The static graph embedding methods are not capable of dealing with the critical challenge involved in dynamic networks.
  - **Disadvantage**: embedding vectors for each timestamp are in different spaces.
    - Leading to learn embedding vectors separately is a time-consuming process.
- **Solution**: dynnode2vec method to modify the node2vec method
  - employing the previous learned embedding vectors as initials weights for the skip-gram model.



- Given a dynamic graph as a sequence $G_1, G_2, \ldots, G_T$ from timestamp 1 to T.
- Each graph at time t is defined as $G_t = (V_t, E_t)$
- **Evolving Random Walk Generation:**
  - only generates random walks for the set of "evolving nodes" (ΔV_t) that have changed between consecutive timestamps t and t+1:

$$\Delta V_t = V_{add} \cup \{v_i \in V_t | \exists e_i = (v_i, v_j) \in (E_{add} \cup E_{del})\}$$

    - Change: new nodes added, existing nodes deleted, or edges added/removed for existing nodes).
- Dynamic Skip-gram Model:
  - initializes the skip-gram model at timestamp t with the pre-trained embedding vectors from the previous timestamp t-1.
  - vocabulary is updated based on the new evolving random walks, and Skip-gram t is retrained using only the new evolving random walks on the evolving node set ΔV_t.

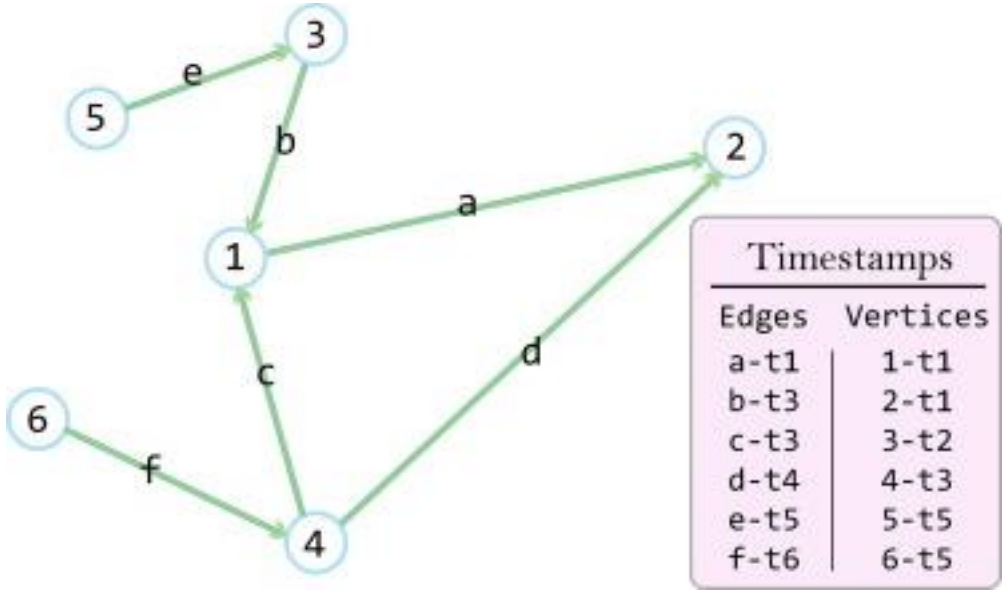
dynnode2vec: Algorithm

- Run static node2vec on the initial graph G_1 to get embedding vectors Z_1.
- For each subsequent timestamp t=2 to T:
  - a) Find evolving node set ΔV_t
  - b) Sample new random walks only for ΔV_t
  - c) Train Skip-gram using new walks, initialized with Skip-gram t-1
  - d) Obtain embedding vectors Z_t

Algorithm 1 :Algorithm: Dynnode2vec

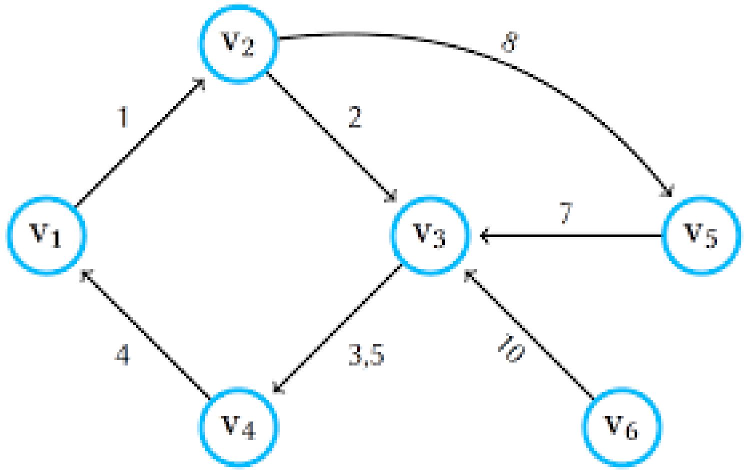
```
Input: Graphs G = G_1, G_2, ..., G_T
Output: Embedding vectors Z_1, Z_2, ..., Z_T
Run static node2vec for the Graph G_1
for t = 2 to N do
    Find a set of evolving nodes, ΔV_t,
    Sample new random walks (Walk_n) for ΔV_t
    Train Skip-Gram Skip_t with Walk_n and obtain Z_t
end for
```

- **Motivation**: traditional methods often treat dynamic network as a sequence of static snapshots, which loss important temporal information.
- **Solution**: treat network as a continuous-time dynamic network, capturing the exact times of interaction.



- A continuous-time dynamic network is defined as $G = (V, E_T, \mathfrak{I})$.
  - V: set of nodes.
  - $E_T \subseteq V \times V \times R^+$: set of temporal edges between vertices in V.
  - $\mathfrak{I}: \mathrm{E} \rightarrow \mathrm{R}^+$: a function that maps each edge to a corresponding timestamp.

- Temporal walk:
  - A temporal walk from v_1 to v_k in G is a sequence of vertices $\langle v_1, v_2, \dots, v_k \rangle$ such that
    - $\langle v_i, v_{i+1} \rangle$ for $1 \leq i < k$ and
    - $\mathfrak{I}(v_i, v_{i+1}) \leq \mathfrak{I}(v_{i+1}, v_{i+2})$ for $1 \leq i < (k-1)$



- **Temporal random walk**:
- The set of temporal neighbors of a node v at time t:

$$\Gamma_t(v) = \left\{(w, t') \mid e = (v, w, t') \in E_T \wedge \mathcal{T}(e) > t\right\}$$

- Unbiased selection: each temporal neighbor w of node v at time t is selected

$$\Pr(w) = 1/|\Gamma_t(v)|$$

- Biased selection: sampling the next node in a temporal walk via temporally weighted distribution based on

$$\Pr(w) = \frac{\exp\left[\tau(w) - \tau(v)\right]}{\sum_{w' \in \Gamma_t(v)} \exp\left[\tau(w') - \tau(v)\right]}$$

Exponential decay: selecting a neighbor decreases exponentially with time

$$\Pr(w) = \frac{\delta(w)}{\sum_{w' \in \Gamma_t(v)} \delta(w')}$$

Linear decay: sorts temporal neighbors in descending order time-wise.

Continuous-Time Dynamic Network Embeddings

- Temporal context windows: To handle the temporal nature of walks - a walk to run out of temporally valid edges to traverse.
  - A walk must have a minimum length ω and can extend up to a maximum length L. The number of context windows is defined

$$\beta = \sum_{i=1}^{k} |S_{t_i}| - \omega + 1$$

- Learning time-preserving embeddings: maximize the likelihood of observing temporal context windows given the embeddings

$$\max_{f} \log \Pr \left(W_T = \{v_{i-\omega}, \cdots, v_{i+\omega}\} \setminus v_i \mid f(v_i) \right)$$

→ Utilizing stochastic gradient descent.

Continuous-Time Dynamic Network Embeddings: Algorithm

Algorithm 1 Continuous-Time Dynamic Network Embeddings

Input:
a (un)weighted and (un)directed dynamic network $G = (V, E_T, \mathcal{T})$,
temporal context window count β, context window size ω,
embedding dimensions D,

Set maximum walk length $L = 80$
Initialize set of *temporal walks* $\mathcal{S}_T$ to $\emptyset$
Initialize number of context windows $C = 0$
Precompute sampling distribution $\mathbb{F}_s$ using G
    $\mathbb{F}_s \in \{\text{Uniform, Exponential, Linear}\}$
$G' = (V, E_T, \mathcal{T}, \mathbb{F}_s)$
while $\beta - C > 0$ **do**
    Sample an edge $e_* = (v, u)$ via distribution $\mathbb{F}_s$
    $t = \mathcal{T}(e_*)$
    $S_t = \text{TemporalWalk}(G', e_* = (v, u), t, L, \omega + \beta - C - 1)$
    if $|S_t| > \omega$ **then**
        Add the *temporal walk* S_t to $\mathcal{S}_T$
        $C = C + (|S_t| - \omega + 1)$
end while
$\mathbf{Z} = \text{StochasticGradientDescent}(\omega, D, \mathcal{S}_T)$
return the *dynamic* node embedding matrix $\mathbf{Z}$

Algorithm 2 Temporal Random Walk

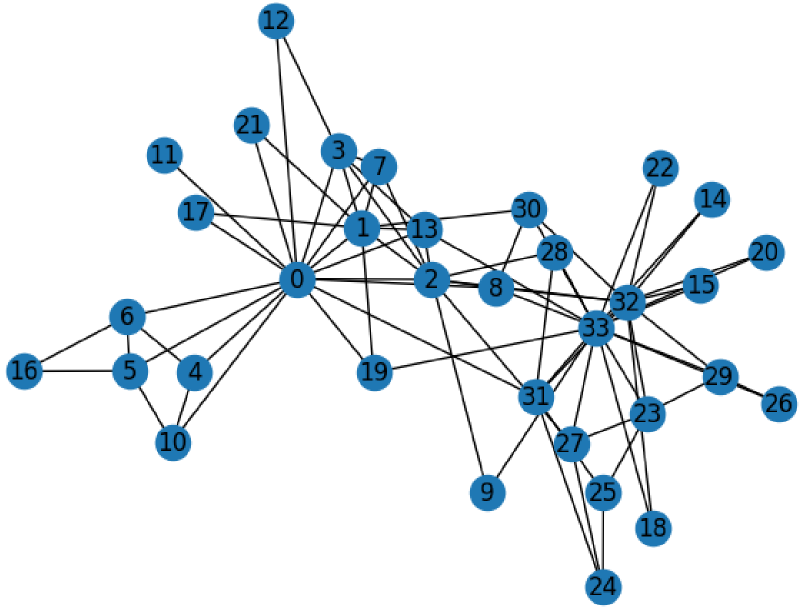
procedure $\text{TemporalWalk}(G', e = (s, r), t, L, C)$
    Initialize temporal walk $S_t = [\, s, r \,]$
    Set $i = r$ ▷ current node
    for $p = 1$ **to** $min(L, C) - 1$ **do**
        $\Gamma_t(i) = \{(w, t') \mid e = (i, w, t') \in E_T \wedge \mathcal{T}(i) > t\}$
        if $|\Gamma_t(i)| > 0$ **then**
            Select node j from distribution $\mathbb{F}_\Gamma(\Gamma_t(i))$
            Append j to S_t
            Set $t = \mathcal{T}(i, j)$
            Set $i = j$
        else terminate temporal walk
    return temporal walk S_t of length $|S_t|$ rooted at node s

- Initialize parameters and precompute sampling distributions.
- Sample temporal walks using the specified distributions.
- Use stochastic gradient descent to optimize the embeddings.

Sample code of Temporal Random Walk in Karate Graph

```python
karate_graph = nx.karate_club_graph()
# Convert the Karate Club graph to an edgelist
edgelist = nx.to_edgelist(karate_graph)
# Print the edgelist
print(edgelist)
# Create a list of edges with timestamps
edges_with_timestamps = []
for edge in edgelist:
  edges_with_timestamps.append((edge[0], edge[1], datetime.datetime.now()))
# Define the time window
time_window = datetime.timedelta(days=1)
# Perform temporal random walks
num_walks = 6
for node in karate_graph.nodes():
  for i in range(num_walks):
    walk = [node]
    current_time = datetime.datetime.now()
    while current_time < datetime.datetime.now() + time_window:
      next_hop_candidates = []
      for edge in edges_with_timestamps:
        if edge[0] == walk[-1] and edge[2] <= current_time:
          next_hop_candidates.append(edge[1])
      if len(next_hop_candidates) > 0:
        next_node = random.choice(next_hop_candidates)
      else:
        break
      walk.append(next_node)
      current_time = datetime.datetime.now()
    print("Node " + str(node) + " walk " + str(i+1) + ": " + str(walk))
```

```
[(0, 1, {'weight': 4}), (0, 2, {'weight': 5}), (0, 3, {'weight': 3}), (0, 4, {'w
Node 0 walk 1: [0, 8, 32, 33]
Node 0 walk 2: [0, 10]
Node 0 walk 3: [0, 6, 16]
Node 0 walk 4: [0, 4, 10]
Node 0 walk 5: [0, 6, 16]
Node 0 walk 6: [0, 31, 33]
Node 1 walk 1: [1, 2, 9, 33]
Node 1 walk 2: [1, 2, 9, 33]
Node 1 walk 3: [1, 2, 28, 33]
Node 1 walk 4: [1, 21]
Node 1 walk 5: [1, 17]
```



네트워크 과학 연구실
NETWORK SCIENCE LAB

가톨릭대학교
THE CATHOLIC UNIVERSITY OF KOREA