Random Walk Approaches to Learn Node Embedding

ACMS 80770: Deep Learning with Graphs

Instructor: Navid Shervani-Tabar

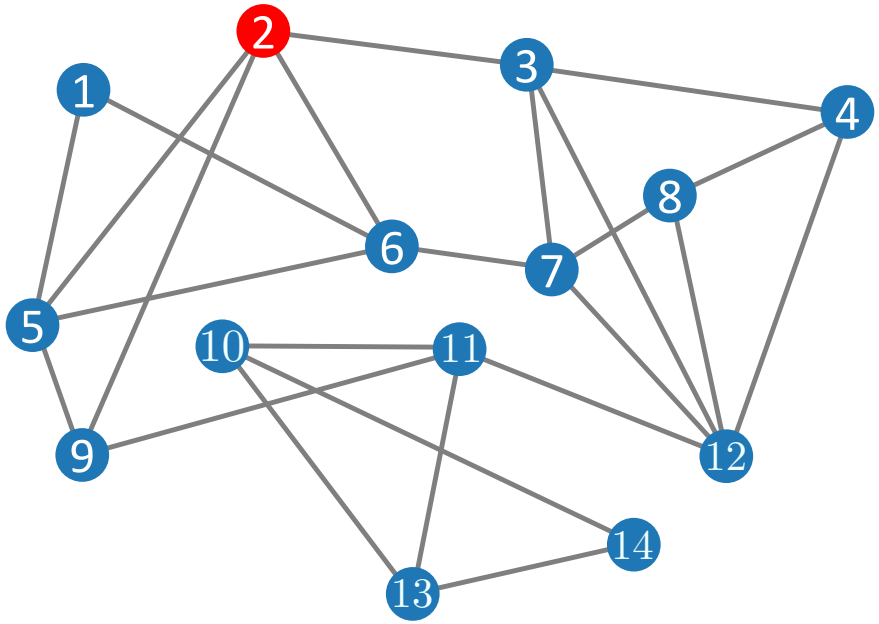
Department of Applied and Comp Math and Stats

Random Walk Embedding

- In the previous lecture, we discussed **learning node embedding** based on reconstruction goals that are a function of the adjacency matrix.
- All these measures of node-node similarity were **deterministic**.
- In this lecture, we discuss **stochastic** similarity measures.
- These measures are defined through the notion of **random walk** over the graph.

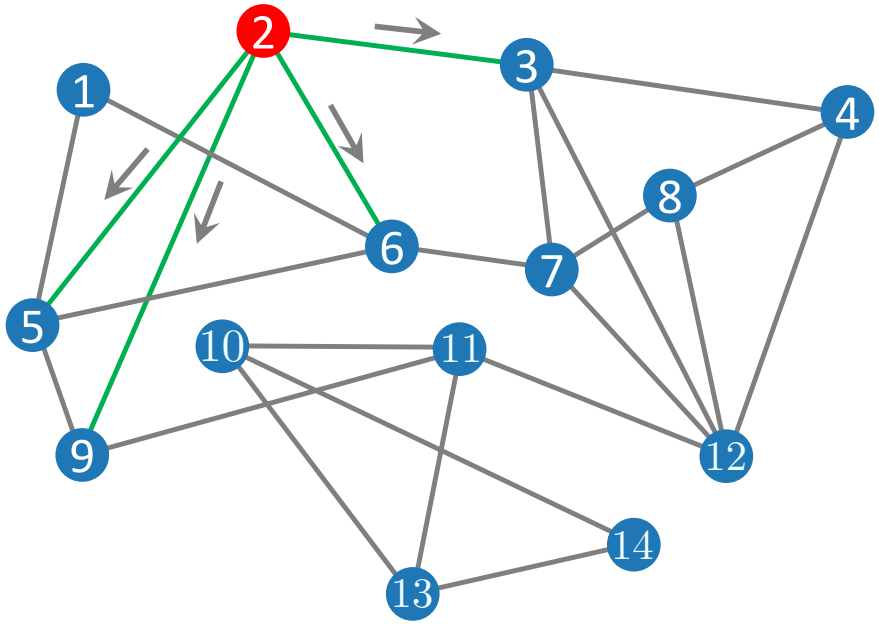
Random Walk

- Consider a walker standing on a node of the graph.



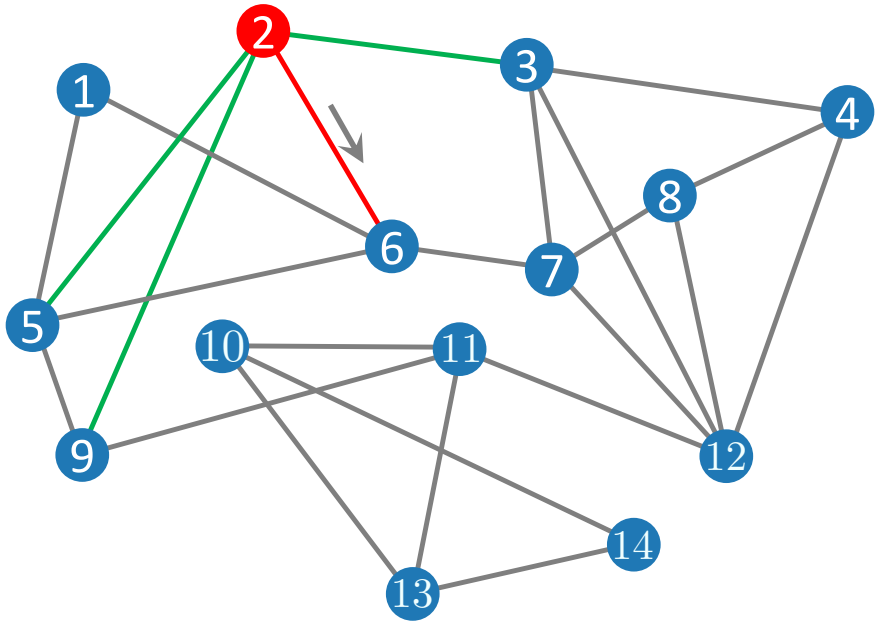
Random Walk

- Consider a **walker** standing on a node of the graph.



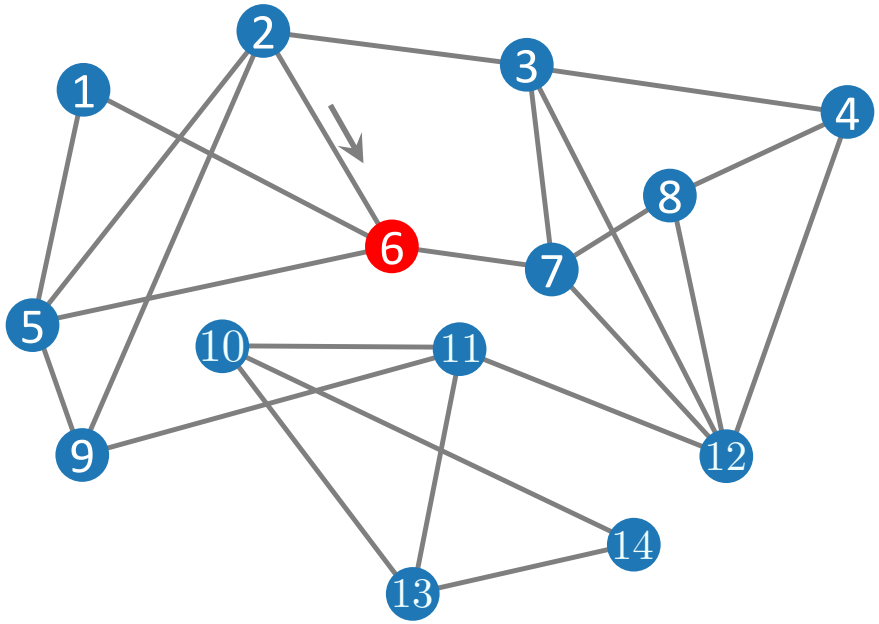
Random Walk

- Consider a **walker** standing on a node of the graph.



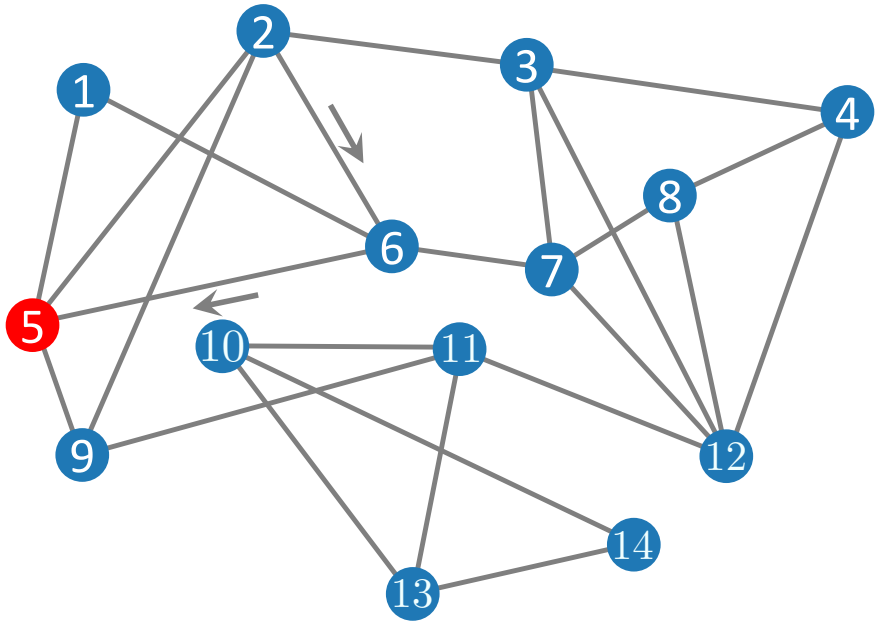
Random Walk

- Consider a **walker** standing on a node of the graph.
- At this node, walker selects one of the edges connected to that node at random and traverses it.



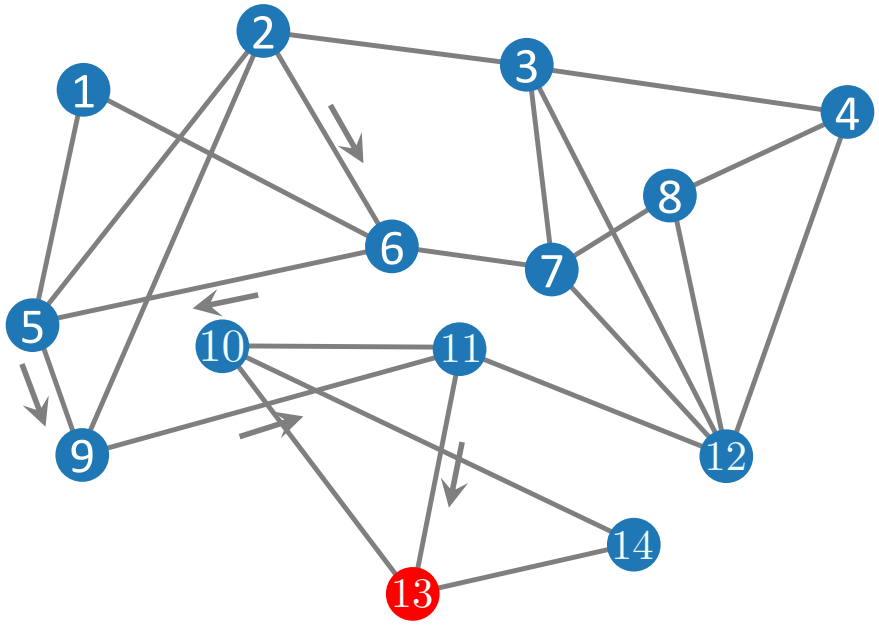
Random Walk

- Consider a **walker** standing on a node of the graph.
- At this node, walker selects one of the edges connected to that node at random and traverses it.

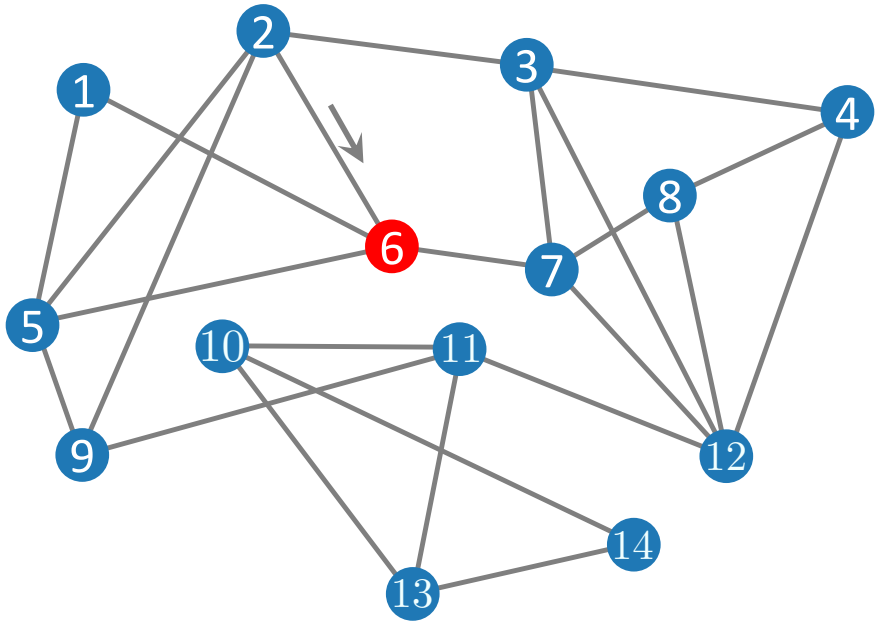


Random Walk

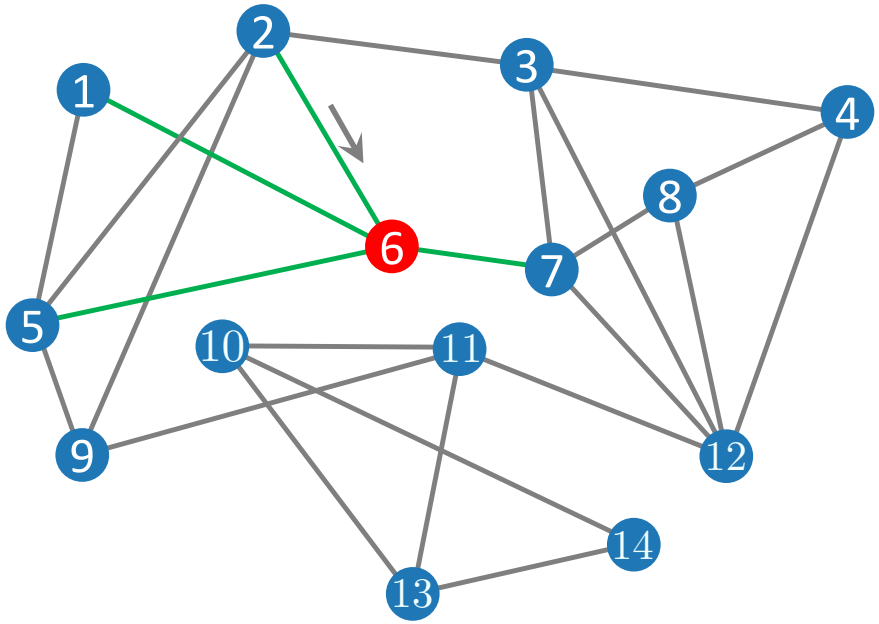
- Consider a **walker** standing on a node of the graph.
- At this node, walker selects one of the edges connected to that node at random and traverses it.
- The walker repeats this for T steps.
- This is called a length-T **random walk**.



Random Walk

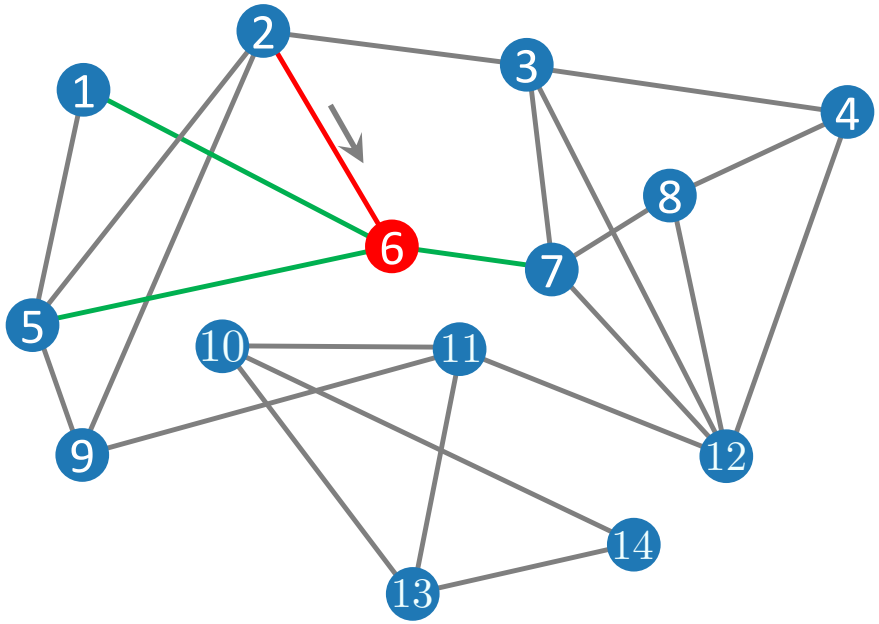


Random Walk



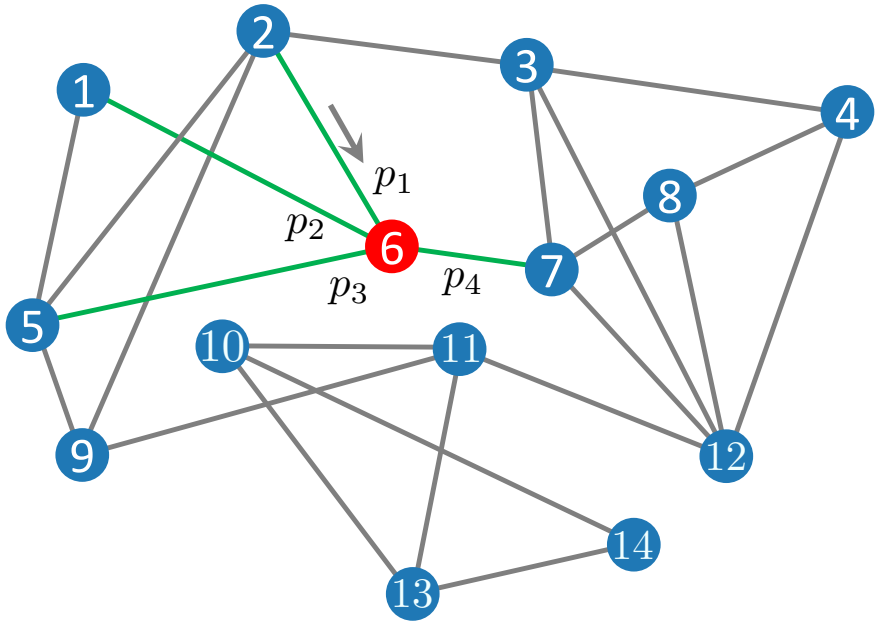
Random Walk

- A random walk can select any edges in the next step, including the one it just traversed.



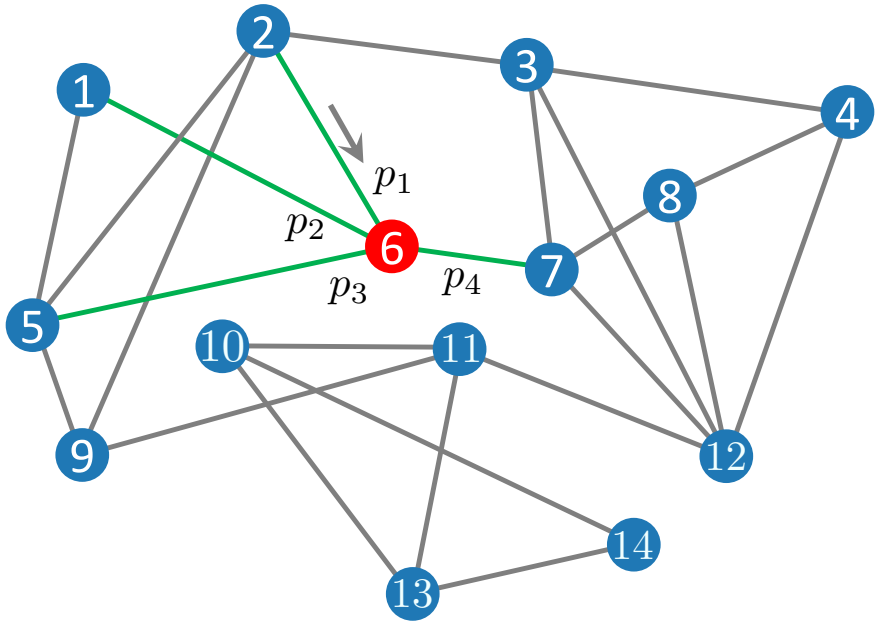
Random Walk Strategies

- A random walk can select any edges in the next step, including the one it **just traversed**.



Random Walk Strategies

- A random walk can select any edges in the next step, including the one it **just traversed**.
- A random walker can use different strategies to walk on graph.
  - It can select the next edge **uniformly** at random.
  - It can select different paths with **different probability**.



Random Walk Embedding

- We can use the random walk as an indicator of **node similarity** to learn node embeddings.
- We indicate the similarity of two nodes based on the notion of **co-occurrence**.

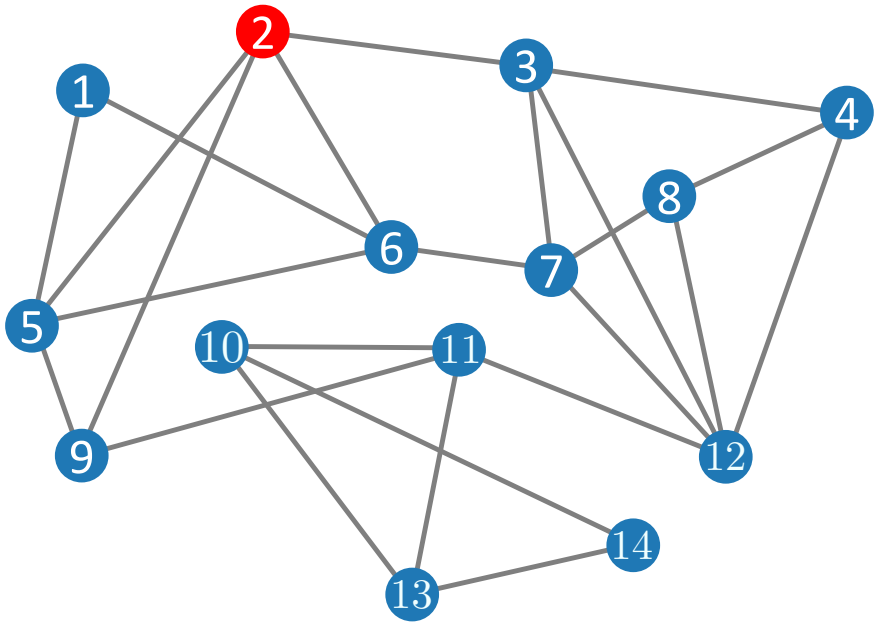
Random Walk Embedding

- We can use the random walk as an indicator of **node similarity** to learn node embeddings.
- We indicate the similarity of two nodes based on the notion of **co-occurrence**.
- The neighborhood for each node v_i is defined by a random walk rooted at that node

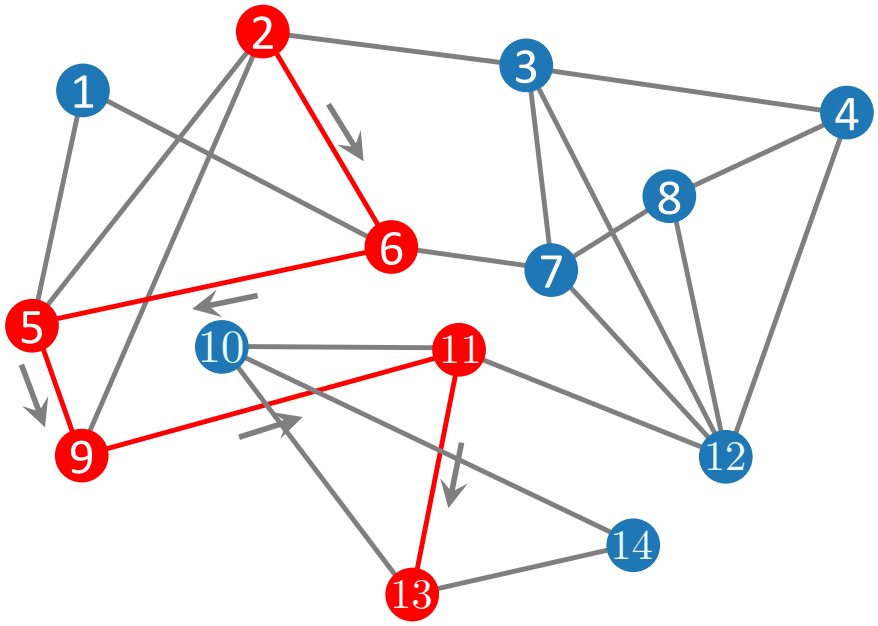
$$N_R(v_i)$$

  where R is the random walk **strategy** and v_i is the **root** node.
- If two nodes **co-occur** on a random walk, that means they are **similar to** each other.

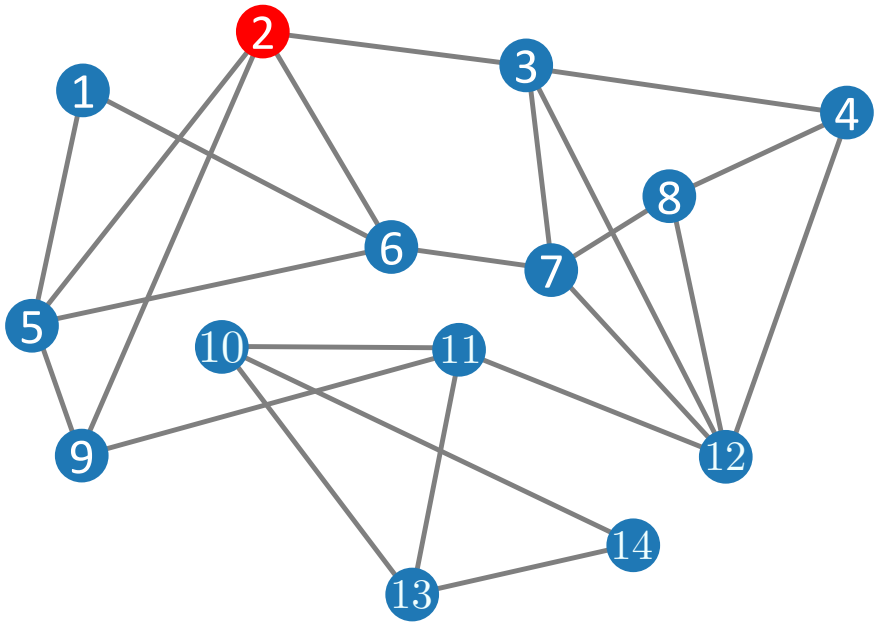
Random Walk



Random Walk

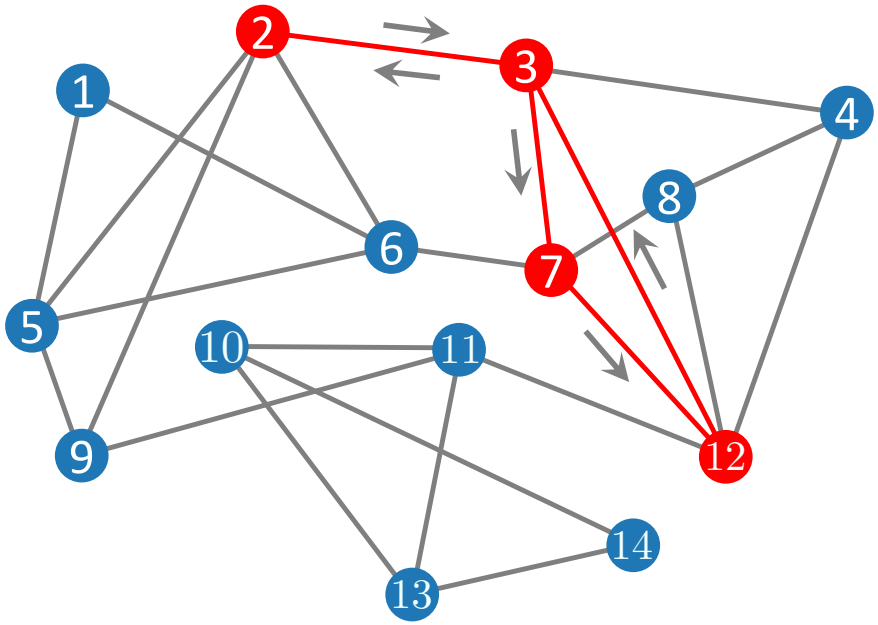


Random Walk



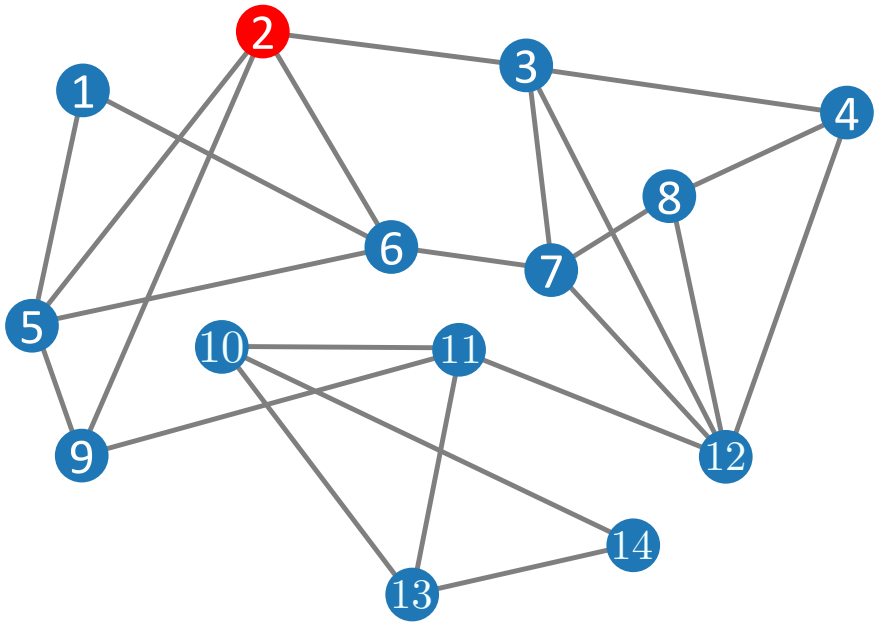
$$N_R^{(1)}(v_2) = \{\{v_2, v_6, v_5, v_9, v_{11}, v_{13}\}\}$$

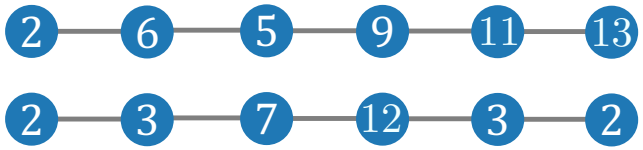
Random Walk



$$N_R^{(1)}(v_2) = \{\{v_2, v_6, v_5, v_9, v_{11}, v_{13}\}\}$$

Random Walk

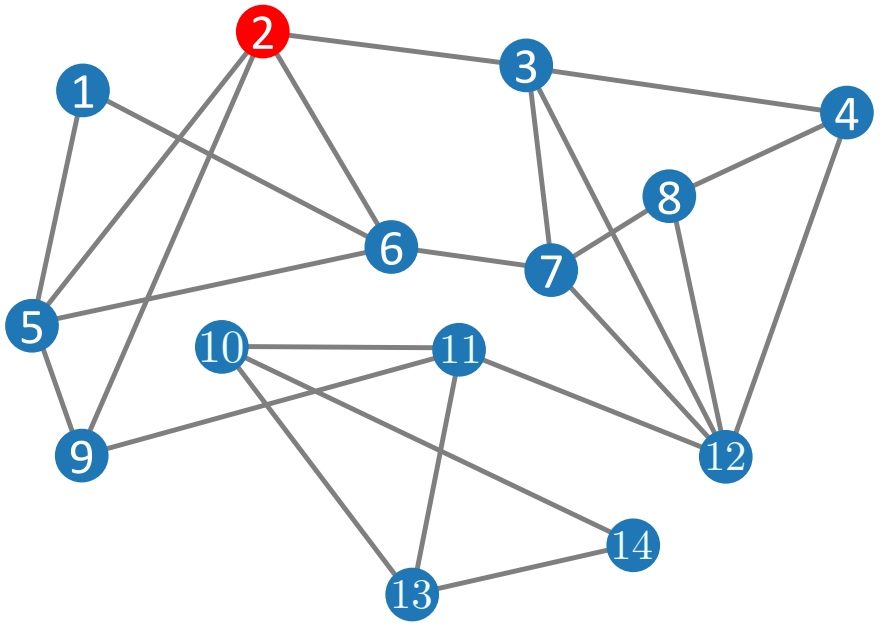


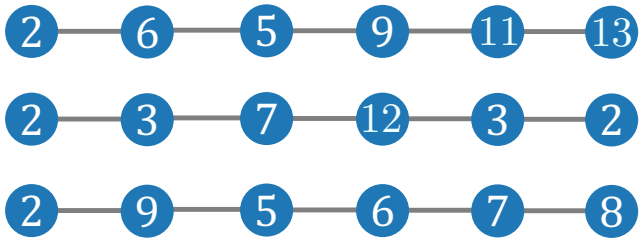


$$N_R^{(1)}(v_2) = \{\{v_2, v_6, v_5, v_9, v_{11}, v_{13}\}\}$$
$$N_R^{(2)}(v_2) = \{\{v_2, v_3, v_7, v_{12}, v_3, v_2\}\}$$

Random Walk

- To that end, we perform **multiple** random walks of different length T starting at each node on the graph.

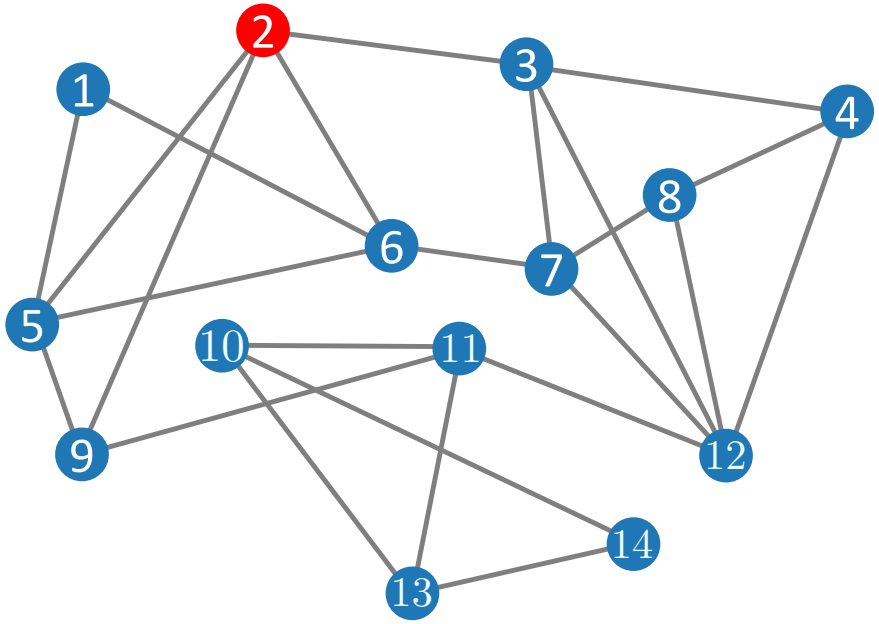


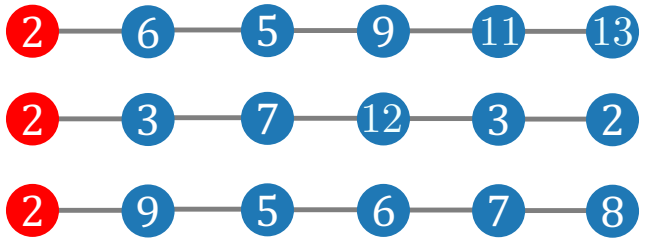


$$N_R^{(1)}(v_2) = \{\{v_2, v_6, v_5, v_9, v_{11}, v_{13}\}\}$$
$$N_R^{(2)}(v_2) = \{\{v_2, v_3, v_7, v_{12}, v_3, v_2\}\}$$
$$N_R^{(3)}(v_2) = \{\{v_2, v_9, v_5, v_6, v_7, v_8\}\}$$

Random Walk Embedding

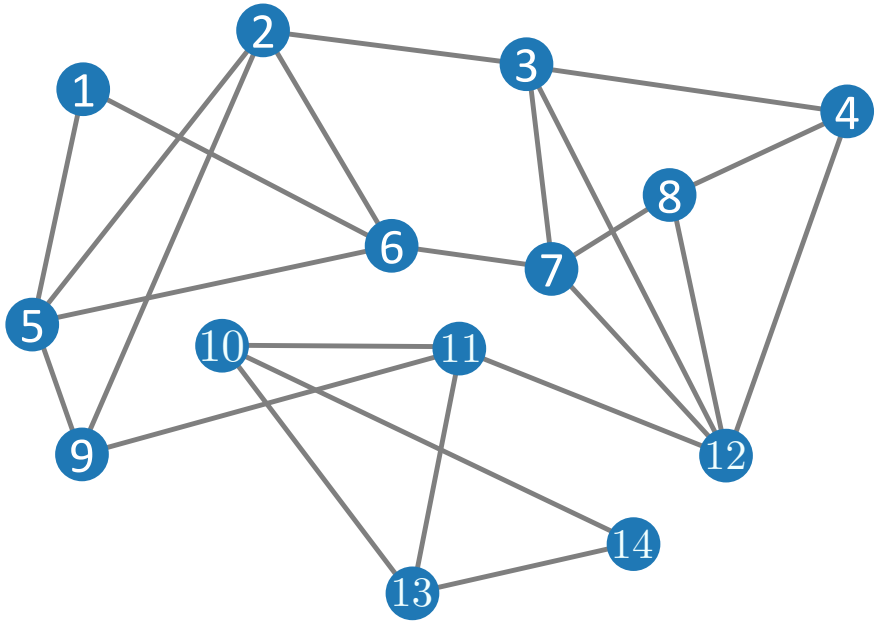
- To that end, we perform **multiple** random walks of different length T starting at each node on the graph.

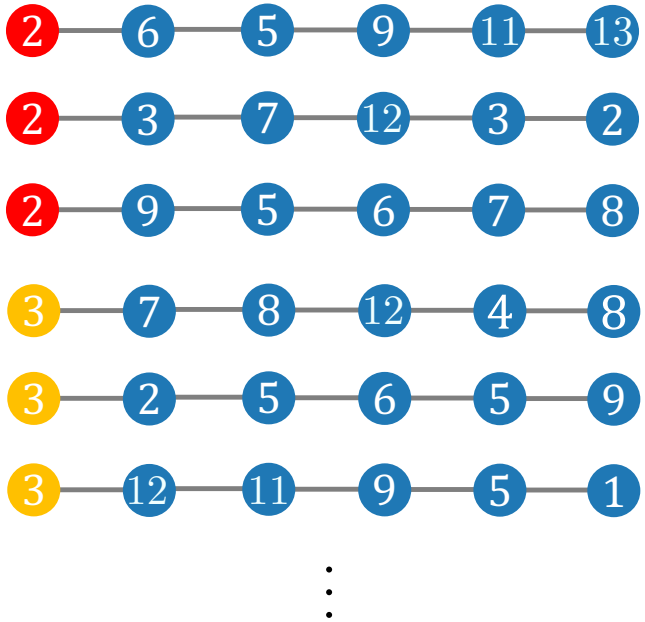




Random Walk Embedding

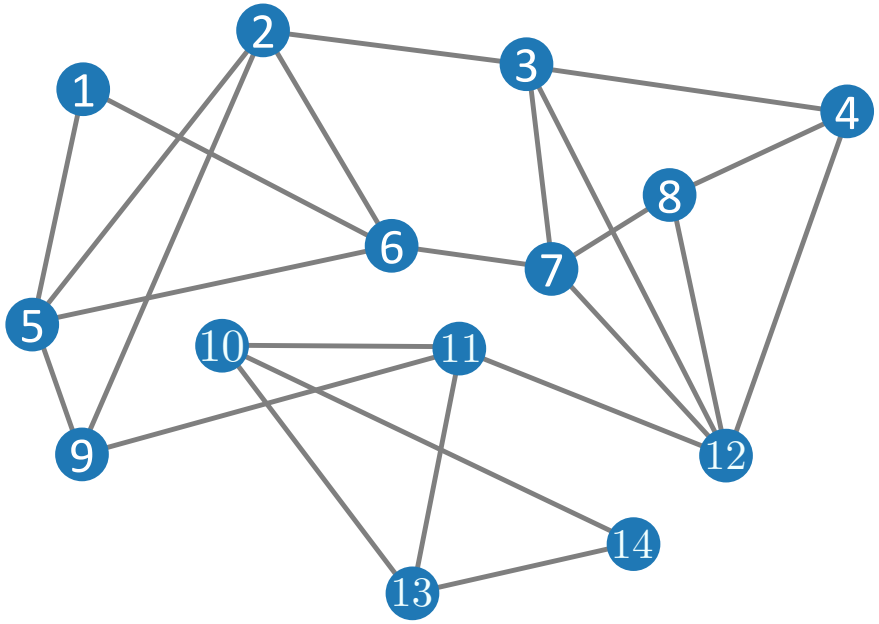
- To that end, we perform **multiple** random walks of different length T starting at each node on the graph.
- These random walks then capture **local neighborhood** of each node and the **topology** of the **graph**.

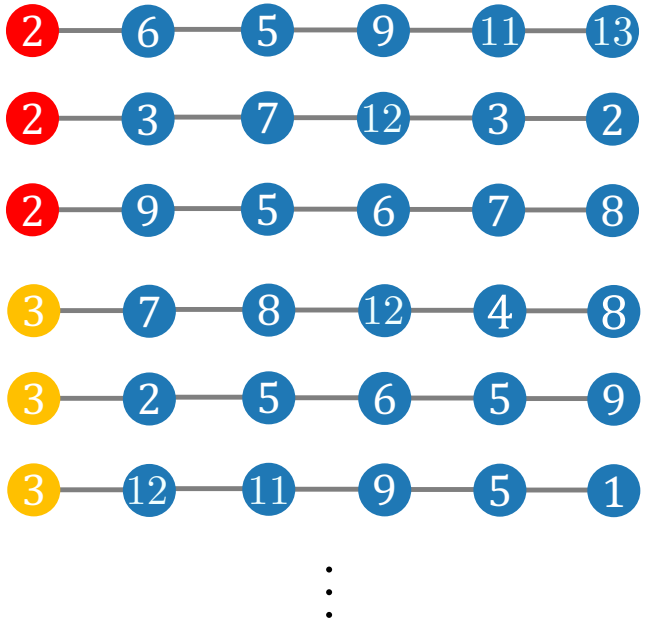




Random Walk Embedding

- To that end, we perform **multiple** random walks of different length T starting at each node on the graph.
- These random walks then capture **local neighborhood** of each node and the **topology** of the **graph**.
- By performing random walks rooted at different nodes, we **decompose** the graph into a set of random walks.





Random Walk Embedding

- In the previous lecture, we discussed **deterministic** node-node similarity measures.
- These functions were defined as a deterministic function of the **adjacency** matrix $\boldsymbol{A}$.

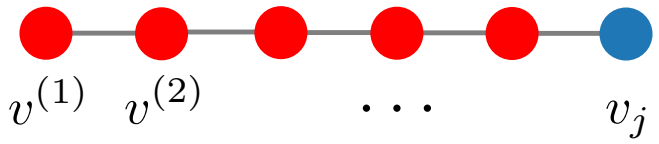
Random Walk Embedding

- In the previous lecture, we discussed **deterministic** node-node similarity measures.
- These functions were defined as a deterministic function of the **adjacency** matrix $\boldsymbol{A}$.
- In this lecture, our measure of similarity is defined by the **statistics of Random Walk**.

Random Walk Embedding

- In the previous lecture, we discussed **deterministic** node-node similarity measures.
- These functions were defined as a deterministic function of the **adjacency** matrix $\boldsymbol{A}$.
- In this lecture, our measure of similarity is defined by the **statistics of Random Walk**.
- The likelihood of observing node v_j on a random walk conditioned on the nodes visited before v_j is

$$p(v_j|\{v^{(1)},\ldots,v^{(t-1)}\})$$



Encoder-Decoder Framework

- In the previous lecture, we discussed **deterministic** node-node similarity measures.
- These functions were defined as a deterministic function of the **adjacency** matrix $\boldsymbol{A}$.
- In this lecture, our measure of similarity is defined by the **statistics of Random Walk**.
- The likelihood of observing node v_j on a random walk conditioned on the nodes visited before v_j is

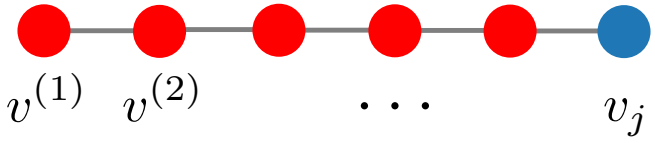
$$p(v_j|\{v^{(1)}, \ldots, v^{(t-1)}\})$$

- We want to learn an **embedding** $\boldsymbol{z}_i$ for each node v_i
- We can **reformulate** the problem as estimating the probability

$$p(v_j|\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(t-1)}\})$$

Encoder-Decoder Framework

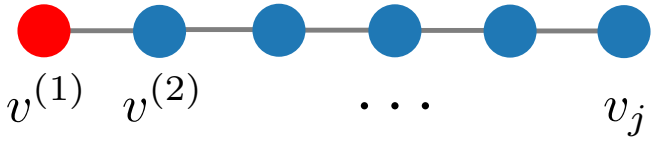
- As **walk grows**, computing this conditional probability becomes unfeasible.



Encoder-Decoder Framework

- As **walk grows**, computing this conditional probability becomes unfeasible.
- To make this objective tractable, we only condition the neighbor node on the **root** node.

$$p(v_j|\boldsymbol{z}_i^{(1)})$$



Encoder-Decoder Framework

- As **walk grows**, computing this conditional probability becomes unfeasible.
- To make this objective tractable, we only condition the neighbor node on the **root** node.

$$p(v_j|\boldsymbol{z}_i^{(1)})$$

- The **similarity** measure is the neighborhood $N_R(v_i)$ defined by the random walks.
- The resulting measure of similarity is **stochastic** and asymmetric.
- The objective is to **learn** the probability of **co-occurrence** of two nodes on a random-walk starting from one of them.
- For node v_i, we want to **maximize** the probability of observing a neighborhood $N_R(v_i)$ given embedding $\boldsymbol{z}_i$.

$$p(N_R(v_i)|\boldsymbol{z}_i)$$

Encoder-Decoder Framework

- With the conditional independence assumption, we factorize the likelihood as

$$p(N_R(v_i)|\boldsymbol{z}_i) = \prod_{v_j \in N_R(v_i)} p(v_j|\boldsymbol{z}_i)$$

- The **decoder reconstructs** the statistics of the set of random walk neighbors $N_R(v_i)$ for all nodes $v_i \in V$ on the graph.

Encoder-Decoder Framework

- With the conditional independence assumption, we factorize the likelihood as

$$p(N_R(v_i)|\boldsymbol{z}_i) = \prod_{v_j \in N_R(v_i)} p(v_j|\boldsymbol{z}_i)$$

- The **decoder reconstructs** the statistics of the set of random walk neighbors $N_R(v_i)$ for all nodes $v_i \in V$ on the graph.

- To that end, the decoder takes a pair of node embeddings z_i and $\boldsymbol{z}_j$ as input and returns the probability of co-occurrence of nodes v_i and v_j on a random walk rooted at v_i.

$$f_d(\boldsymbol{z}_i, \boldsymbol{z}_j) = p(v_j|\boldsymbol{z}_i)$$

Encoder-Decoder Framework

- Consider using the dot product between two embedding vectors $\boldsymbol{z}_i$ and $\boldsymbol{z}_j$ as the decoder.
- In this case, probability of observing v_j on a random walk rooted at v_i is

$$p(v_j|\boldsymbol{z}_i) = [\mathbf{Z}^T\mathbf{Z}]_{ij}$$

Encoder-Decoder Framework

- Consider using the dot product between two embedding vectors $\boldsymbol{z}_i$ and $\boldsymbol{z}_j$ as the decoder.
- In this case, probability of observing v_j on a random walk rooted at v_i is

$$p(v_j|\boldsymbol{z}_i) = [\mathbf{Z}^T\mathbf{Z}]_{ij}$$

- The dot product $\boldsymbol{z}_i\boldsymbol{z}_j^T$ is not necessarily a probability value and may **not lie** between 0 and 1.
- Also, the vector $[\boldsymbol{Z}^T\boldsymbol{Z}]_i$ may not represent a probability vector and does **not** necessarily **sum to** 1.
- To address this, we take advantage of the softmax function.

Softmax

- A softmax function takes a vector of real values as input and returns a vector of probability values, with higher probability assigned to the higher value.

$$\text{softmax}(\boldsymbol{x}) := \left[\frac{\exp(x_1)}{\sum_{c=1}^{C} \exp(x_c)}, \ldots, \frac{\exp(x_C)}{\sum_{c=1}^{C} \exp(x_c)} \right]$$

  where $\text{softmax}: \mathbb{R}^C \to [0,1]^C$ with C total number of possible outcomes.

- Note that

$$0 \leq \text{softmax}(x)_c \leq 1$$

  and

$$\sum_{c=1}^{C} \text{softmax}(\boldsymbol{x})_c = 1$$

- We refer to the inputs $\boldsymbol{x}$ of the softmax function as logits.

Decoder

- To avoid requiring the **decoder** directly predict a probability vector, we pass d_f into the **softmax** function.
- We use **softmax** function to return the **probability** of a length-T random walk rooted at node v_i on a graph G visiting a node v_j.
- Therefore, the decoder is formulated as

$$p_{G,T}\left(v_j \mid \boldsymbol{z}_i\right) = f_d\left(\boldsymbol{z}_i, \boldsymbol{z}_j\right) = \frac{\exp\left(\boldsymbol{z}_i^T \boldsymbol{z}_j\right)}{\sum_{v_k \in V} \exp\left(\boldsymbol{z}_i^T \boldsymbol{z}_k\right)}$$

- The length of the walk T is usually in the interval $2 \leq T \leq 10$.

Objective

- Given the stochastic definition of neighborhood $N_R(v_i)$ for node v_i, we want the decoder to assign high probability to true output $v_j \in N_R(v_i)$ for the corresponding root node v_i.
- For each root node $v_i \in V$, our data is defined as the set of M random walks.

$$\mathcal{D} = \left\{ N_R^{(s)}(v_i) \mid 1 \leq s \leq M \right\}$$

- The objective is then to **maximize** conditional probability

$$p\left(N_R^{(s)}(v_i) \mid \boldsymbol{z}_i\right) = \prod_{v_j \in N_R^{(s)}(v_i)} p(v_j \mid \boldsymbol{z}_i)$$

- Alternatively, for all random walks rooted at v_i

$$\boldsymbol{z}_i = \arg\max_{\mathbf{Z}} \sum_{s=1}^{M} \sum_{v_j \in N_R^{(s)}(v_i)} \log p(v_j \mid \boldsymbol{z_i})$$

Objective

- We can rewrite the objective for all nodes $v_i \in V$ as minimizing the negative log likelihood

$$\mathcal{L} = -\sum_{v_i \in V} \sum_{s} \sum_{v_j \in N_R^{(s)}(v_i)} \log p\left(v_j \mid \boldsymbol{z}_i\right)$$

where conditional probability $p(v_j|\boldsymbol{z}_i)$ is estimated by the decoder

$$f_d\left(\boldsymbol{z}_i, \boldsymbol{z}_j\right) = \frac{\exp\left(\boldsymbol{z}_i^T \boldsymbol{z}_j\right)}{\sum_{v_k \in V} \exp\left(\boldsymbol{z}_i^T \boldsymbol{z}_k\right)}$$

Objective

- We plug in the probability computed by the decoder into the loss function and try to minimize it.

$$\mathcal{L} = -\sum_{v_i \in V} \sum_{s} \sum_{v_j \in N_R^{(s)}(v_i)} \log \left(\frac{\exp\left(\boldsymbol{z}_i^T \boldsymbol{z}_j\right)}{\sum_{v_k \in V} \exp\left(\boldsymbol{z}_i^T \boldsymbol{z}_k\right)} \right)$$

Objective

- We plug in the probability computed by the decoder into the loss function and try to minimize it.

$$\mathcal{L} = -\sum_{v_i \in V} \sum_{s} \sum_{v_j \in N_R^{(s)}(v_i)} \log \left(\frac{\exp\left(\boldsymbol{z}_i^T \boldsymbol{z}_j\right)}{\sum_{v_k \in V} \exp\left(\boldsymbol{z}_i^T \boldsymbol{z}_k\right)} \right)$$

- Computing conditional probability for all nodes is expensive as estimating the denominator for all nodes requires $O(|V|^2)$ computations.
- Based on the loss approximation method and random walk strategy, we can categorize random walk node embeddings as
  - DeepWalk model
  - node2vec model

Random Walk Embeddings

- DeepWalk model
  - performs unbiased random walks.
  - Uses hierarchical softmax to approximate loss.
- Node2vec model
  - Uses 2nd order biased random walks.
  - The objective function is estimated using negative sampling.
- Negative sampling is a fast approximation method that instead of normalizing the probability using all nodes $v_k \in V$, it uses only few negative samples $v_k \in V'$ to approximate the objective.

Negative Sampling

- The key idea is to reformulate conditional probability as a **binary classification**. Let
  - $D = 1$ be the event that the pair of nodes co-occur in the data $\mathcal{D}$, with the binary probability

$$p(D = 1|(v_i, v_j))$$

  - $D = 0$ be the event that the pair of nodes do not co-occur, with the binary probability

$$p(D = 0|(v_i, v_j))$$

- Instead of maximizing conditional probability $p(v_j|v_i)$, we maximize

$$p(D = 1|(v_i, v_j))$$

for $v_j \in N_R(v_i)$.

Negative Sampling

- To avoid a trivial solution, we rewrite this as

$$p(D = 1|(v_i, v_j)) \prod_{v_k \in V'} p(D = 0|(v_i, v_k))$$

where $V' \subset V$ is a subset of nodes that do not co-occur with v_i.

- We refer to $v_k \in V'$ as noise nodes.

Negative Sampling

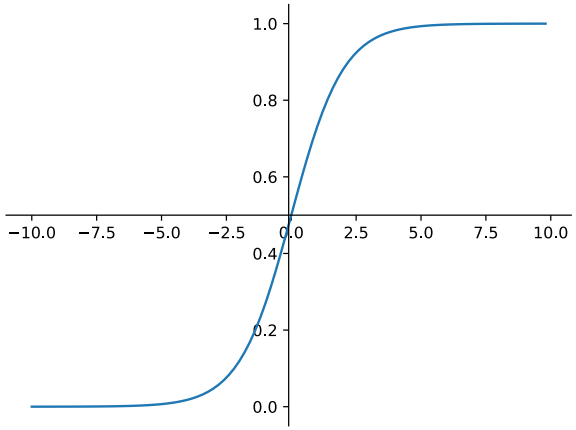
- To avoid a trivial solution, we rewrite this as

$$p(D=1|(v_i,v_j)) \prod_{v_k \in V'} p(D=0|(v_i,v_k))$$

where $V' \subset V$ is a subset of nodes that do not co-occur with v_i.

- We refer to $v_k \in V'$ as noise nodes.
- We can represent the binary probabilities using a sigmoid function

$$\sigma(x) = \frac{1}{1+\exp(-x)}$$



Negative Sampling

- The objective is then to maximize

$$p\left(D=1 \mid (v_i, v_j)\right) \prod_{v_k \in V'} p\left(D=0 \mid (v_i, v_k)\right)$$

$$p\left(D=1 \mid (v_i, v_j)\right) \prod_{v_k \in V'} \left[1 - p\left(D=1 \mid (v_i, v_k)\right)\right]$$

Negative Sampling

- The objective is then to maximize

$$p\left(D=1 \mid (v_i, v_j)\right) \prod_{v_k \in V'} p\left(D=0 \mid (v_i, v_k)\right)$$

$$p\left(D=1 \mid (v_i, v_j)\right) \prod_{v_k \in V'} \left[1 - p\left(D=1 \mid (v_i, v_k)\right)\right]$$

$$\log\left[p\left(D=1 \mid (v_i, v_j)\right)\right] + \sum_{v_k \in V'} \log\left[1 - p\left(D=1 \mid (v_i, v_k)\right)\right]$$

- We use sigmoid to compute binary classification probabilities

$$p\left(D=1 \mid (v_i, v_j)\right) = \sigma(\boldsymbol{z}_i^T \boldsymbol{z}_j)$$

Negative Sampling

- The objective is then to maximize

$$p\left(D=1 \mid (v_i, v_j)\right) \prod_{v_k \in V'} p\left(D=0 \mid (v_i, v_k)\right)$$

$$p\left(D=1 \mid (v_i, v_j)\right) \prod_{v_k \in V'} \left[1 - p\left(D=1 \mid (v_i, v_k)\right)\right]$$

$$\log\left[p\left(D=1 \mid (v_i, v_j)\right)\right] + \sum_{v_k \in V'} \log\left[1 - p\left(D=1 \mid (v_i, v_k)\right)\right]$$

- We use sigmoid to compute binary classification probabilities

$$p\left(D=1 \mid (v_i, v_j)\right) = \sigma(\boldsymbol{z}_i^T \boldsymbol{z}_j)$$

- Plugging this to the objective, we can rewrite is as the minimization of the loss function

$$\mathcal{L}_i = -\log(\sigma(\boldsymbol{z}_i^T \boldsymbol{z}_j)) - \sum_{v_k \in V'} \log(1 - \sigma(\boldsymbol{z}_i^T \boldsymbol{z}_k))$$

Negative Sampling

- Since

$$1 - \frac{1}{1 + \exp(-x)} = \frac{1}{1 + \exp(x)}$$

- We rewrite the objective as

$$\mathcal{L}_i = -\log(\sigma(\boldsymbol{z}_i^T \boldsymbol{z}_j)) - \sum_{v_k \in V'} \log \sigma(-\boldsymbol{z}_i^T \boldsymbol{z}_k)$$

Negative Sampling

- Since

$$1 - \frac{1}{1 + \exp(-x)} = \frac{1}{1 + \exp(x)}$$

- We rewrite the objective as

$$\mathcal{L}_i = -\log(\sigma(\boldsymbol{z}_i^T \boldsymbol{z}_j)) - \sum_{v_k \in V'} \log \sigma(-\boldsymbol{z}_i^T \boldsymbol{z}_k)$$

- We can write the objective for all data as

$$\mathcal{L} = -\sum_{v_i \in V} \sum_{s} \sum_{v_j \in N_R^{(s)}(v_i)} \left[\log(\sigma(\boldsymbol{z}_i^T \boldsymbol{z}_j)) + \sum_{v_k \in V'} \log(\sigma(-\boldsymbol{z}_i^T \boldsymbol{z}_k)) \right]$$

where V' is the set of noise nodes and sampled from

$$v_k \sim p(V)$$

Negative Sampling

- Probability distribution $p(V)$ is a distribution over the set of nodes V.
- The node distribution $p(V)$ can either be
  - Uniform.
  - Proportional to node degrees.

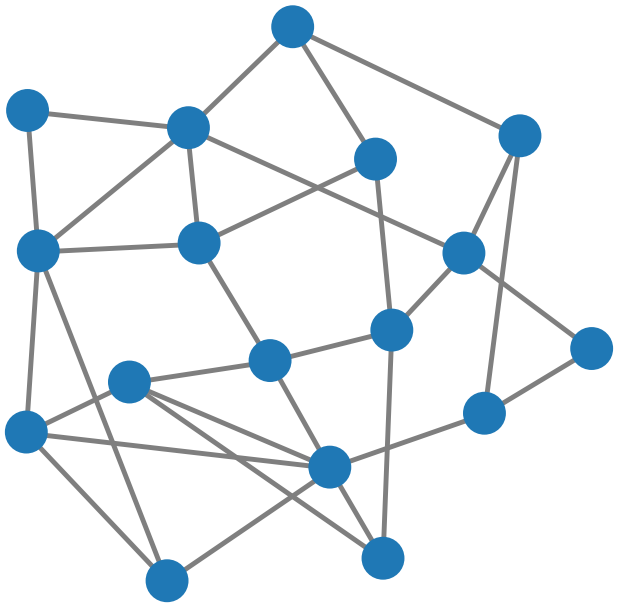
Negative Sampling

- Probability distribution $p(V)$ is a distribution over the set of nodes V.
- The node distribution $p(V)$ can either be
  - Uniform.
  - Proportional to node degrees.
- We use k negative sample nodes $v_n \sim p(V)$ to normalize the conditional probability instead of normalizing against all nodes $v_k \in V$ in the softmax function.

$$\mathcal{L} = -\sum_{v_i \in V} \sum_{s} \sum_{v_j \in N_R^{(s)}(v_i)} \log \left(\frac{\exp\left(\boldsymbol{z}_i^T \boldsymbol{z}_j\right)}{\sum_{v_k \in V} \exp\left(\boldsymbol{z}_i^T \boldsymbol{z}_k\right)} \right)$$

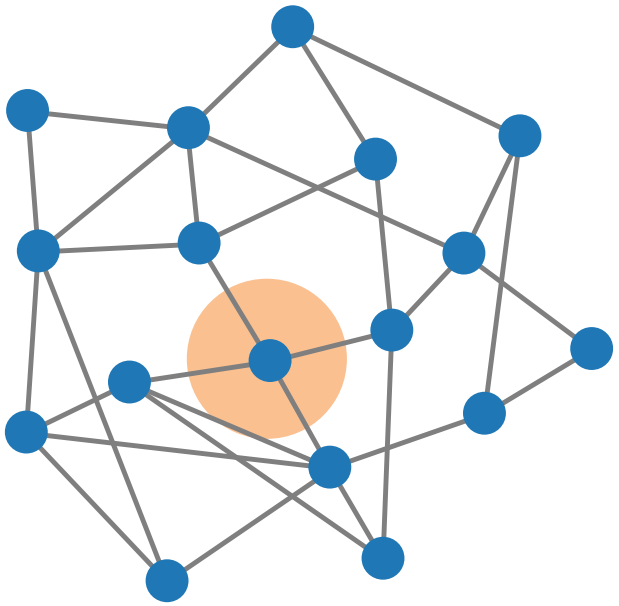
Walk Strategy

- BFS is a classical local search strategy used for computing shortest distance between two nodes.



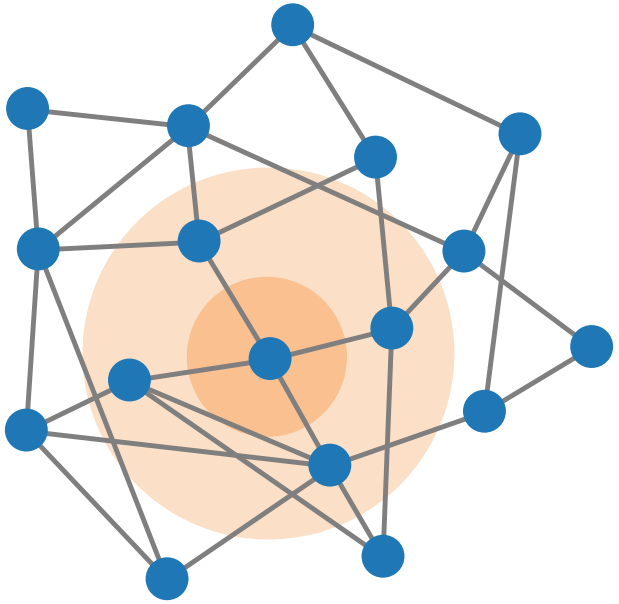
Walk Strategy

- BFS is a classical local search strategy used for computing shortest distance between two nodes.
- In this approach, we start from a node and label it 0.



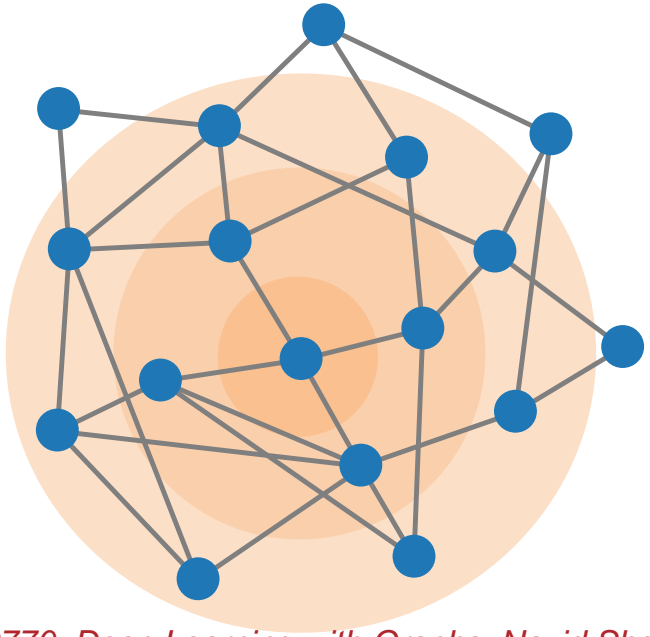
Walk Strategy

- BFS is a classical local search strategy used for computing shortest distance between two nodes.
- In this approach, we start from a node and label it 0.
- Then we go through all neighbors of node 0 and label them 1.



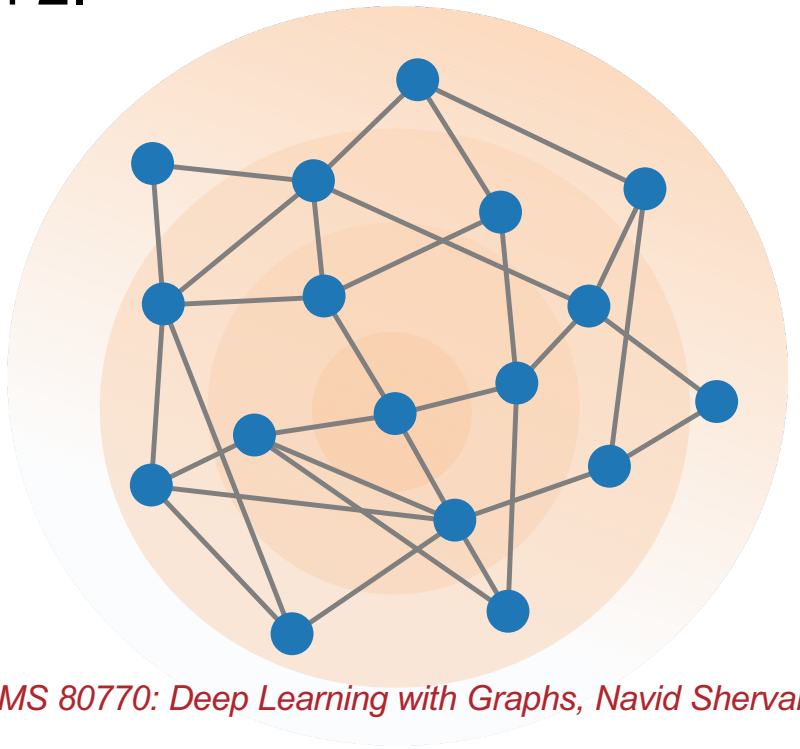
Walk Strategy

- BFS is a classical local search strategy used for computing shortest distance between two nodes.
- In this approach, we start from a node and label it 0.
- Then we go through all neighbors of node 0 and label them 1.
- Then, we go through all unlabeled neighbors of nodes labeled 1 and label them 2.



Walk Strategy

- BFS is a classical local search strategy used for computing shortest distance between two nodes.
- In this approach, we start from a node and label it 0.
- Then we go through all neighbors of node 0 and label them 1.
- Then, we go through all unlabeled neighbors of nodes labeled 1 and label them 2.

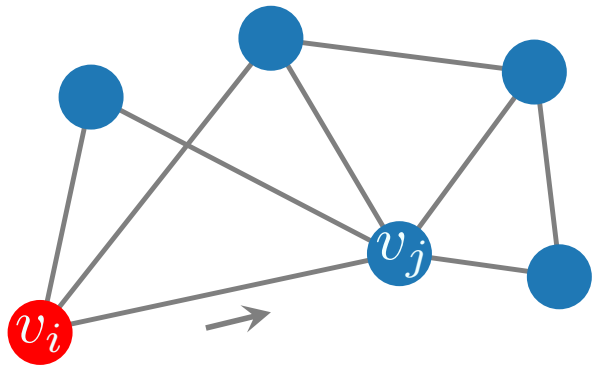


Walk Strategy

- BFS and DFS are two ends of local search spectrum.
- In node2vec approach, a biased random walk exploration strategy is used to interpolate between BFS-like and DFS-like search strategies.

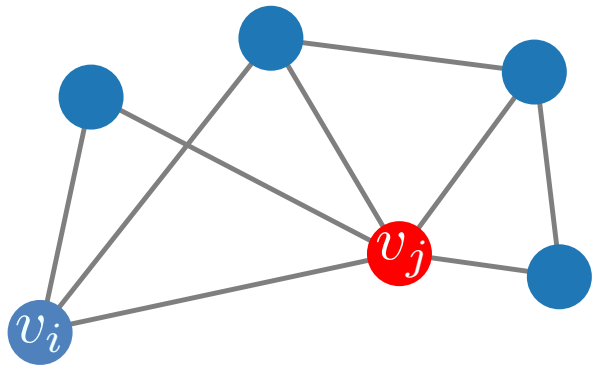
Walk Strategy

- BFS and DFS are two ends of local search spectrum.
- In node2vec approach, a biased random walk exploration strategy is used to interpolate between BFS-like and DFS-like search strategies.
- Consider a walker moving from a node v_i to a neighbor v_j.



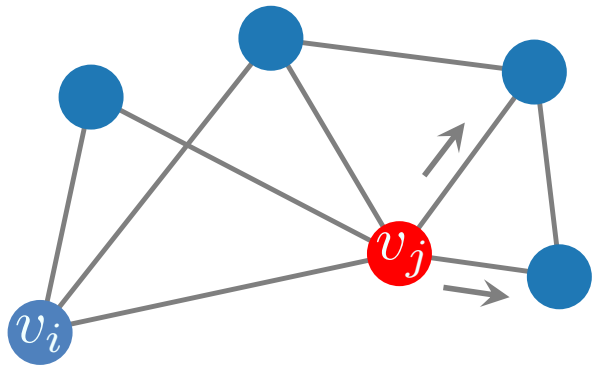
Walk Strategy

- Consider a walker moving from a node v_i to a neighbor v_j.
- In the next step, walker has the following choices:



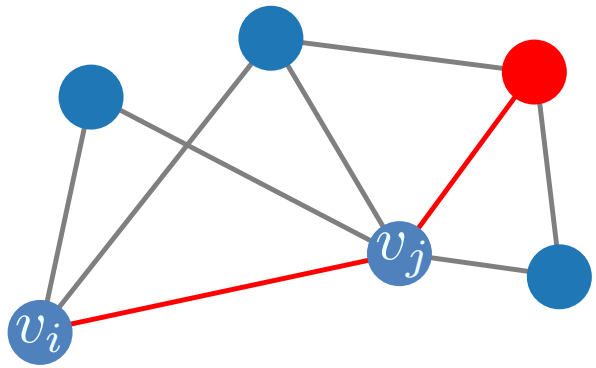
Walk Strategy

- Consider a walker moving from a node v_i to a neighbor v_j.
- In the next step, walker has the following choices:



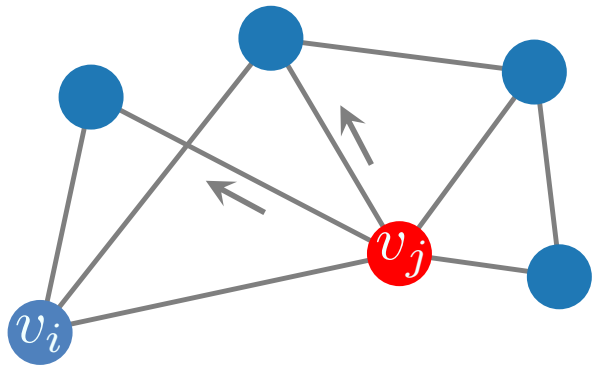
Walk Strategy

- Consider a walker moving from a node v_i to a neighbor v_j.
- In the next step, walker has the following choices:
  - Move away to node v_k, where the distance between walker and node v_i is 2.



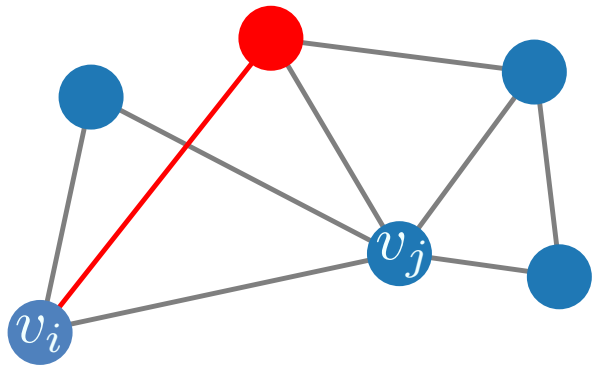
Walk Strategy

- Consider a walker moving from a node v_i to a neighbor v_j.
- In the next step, walker has the following choices:
  - Move away to node v_k, where the distance between walker and node v_i is 2.



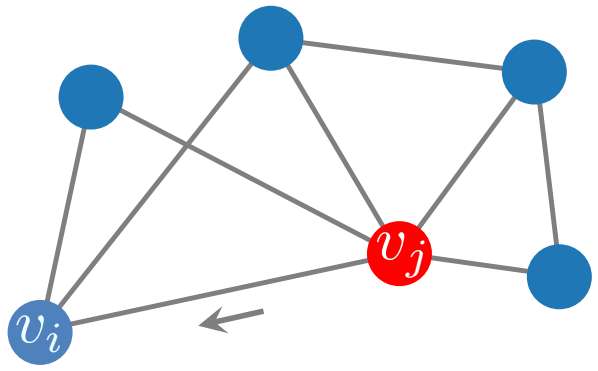
Walk Strategy

- Consider a walker moving from a node v_i to a neighbor v_j.
- In the next step, walker has the following choices:
  - Move away to node v_k, where the distance between walker and node v_i is 2.
  - Move to node v_n where the walker stays at distance 1 from node v_i.



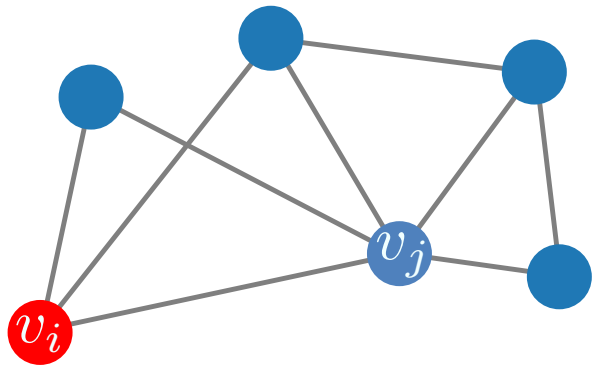
Walk Strategy

- Consider a walker moving from a node v_i to a neighbor v_j.
- In the next step, walker has the following choices:
  - Move away to node v_k, where the distance between walker and node v_i is 2.
  - Move to node v_n where the walker stays at distance 1 from node v_i.



Walk Strategy

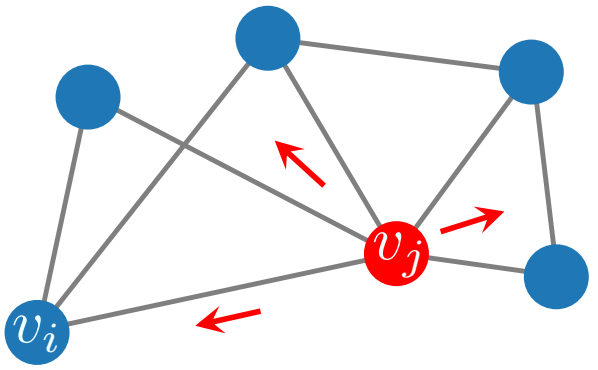
- Consider a walker moving from a node v_i to a neighbor v_j.
- In the next step, walker has the following choices:
  - Move away to node v_k, where the distance between walker and node v_i is 2.
  - Move to node v_n where the walker stays at distance 1 from node v_i.
  - Move back to node v_i. then the distance between v_i and walker is 0.



Walk Strategy

- In the node2vec approach, a parameterized random walk strategy introduces search bias and guides the random walk.
- Use parameter p to control probability of returning to the initial point.
- Use parameter q to control probability of moving away from initial point.
- Using these, the walker has the following probability for choosing from the three directions

$$\begin{cases} \frac{1}{p} & l(v_x, v_i) = 0 \\ 1 & l(v_x, v_i) = 1 \\ \frac{1}{q} & l(v_x, v_i) = 2 \end{cases}$$



Summary

- Random walk
- Random walk embedding
- Random walk embedding with encoder-decoder approach
  - Negative log likelihood objective
  - Decoder
- Softmax
- DeepWalk
- Node2Vec
- Negative Sampling
- Breadth-first Search
- Biased Walk Strategy