Learning Query-dependent Propagation for Knowledge Graph Reasoning

Presenter: Zhanke Zhou

Advisors: Yongqi Zhang and Quanming Yao

2022. 03. 25

Outline

- Background
- A unified framework of propagation path
- Learning to propagate for KG reasoning
- Summary

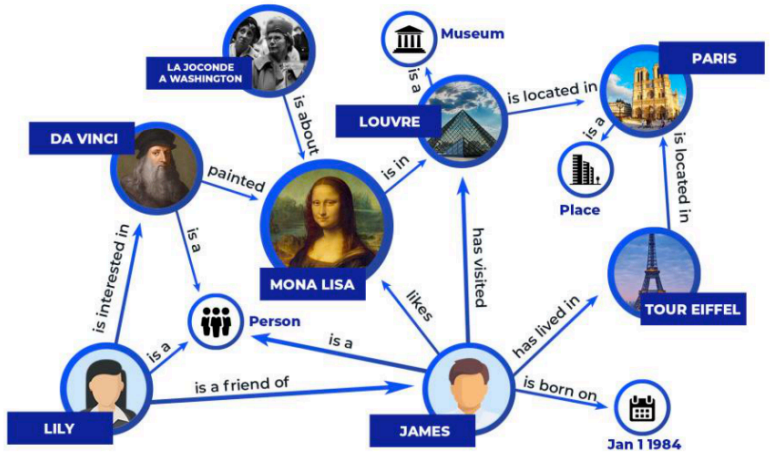
Background

A knowledge graph

- Mainly describe real world entities and relations, organized in a graph
- Allows potentially interacting arbitrary entities with each other

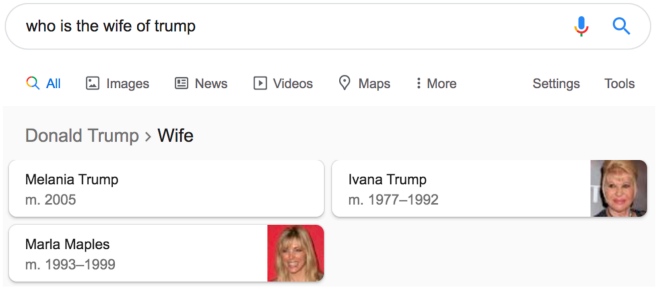
Preliminaries

- Graph representation: $\mathcal{G} = (\mathcal{E}, \mathcal{R}, \mathcal{F})$
- Entities $\mathcal{E}$
  - real world objects or concepts
- Relations $\mathcal{R}$
  - interactions between entities
- Facts $\mathcal{F}$
  - the basic unit in form of (s, r, o)
  - (subject entity, relation, object entity)

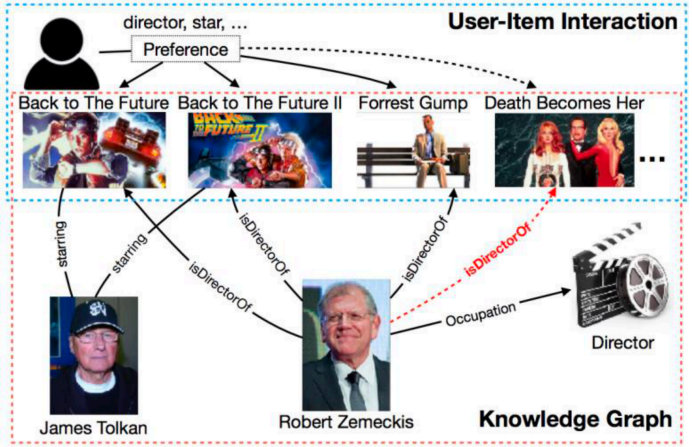


Applications

KGQA:

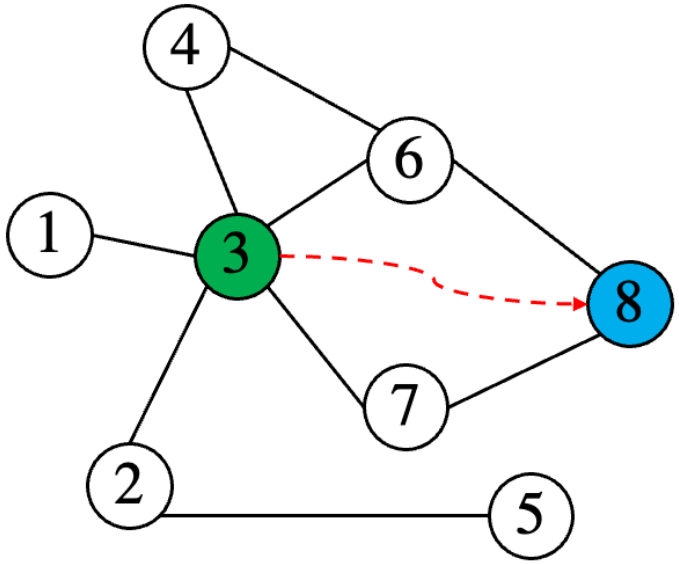


Recommendation:



Background

- Link prediction task in knowledge graph
  - give a query $(e_q, r_q, ?)$, to find the target entity e_a
  - based on observed (known) edges, to predict the latent (unknown) edges



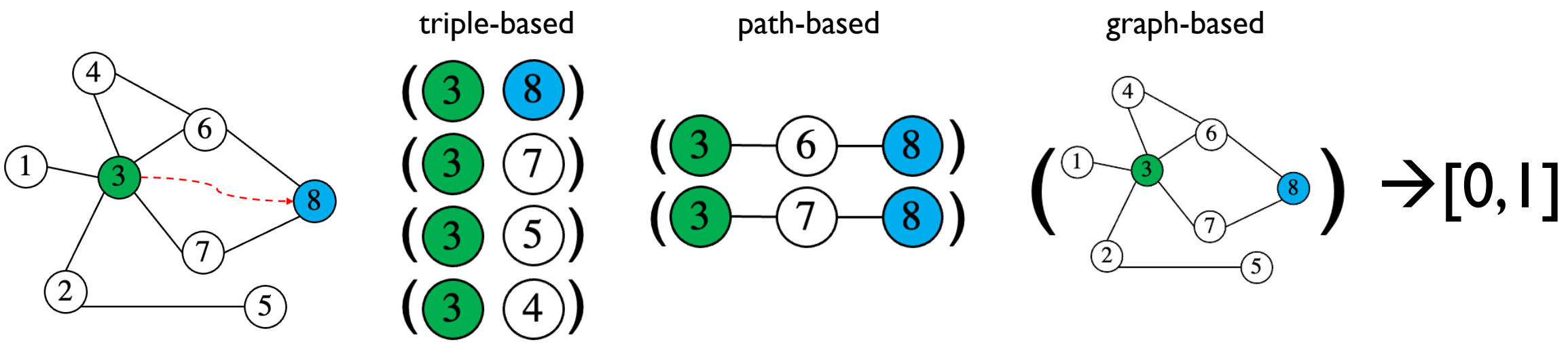
KG

$$\text{query} = (e_q, r_q, ?)$$

$$e_q \leftarrow ③, r_q \leftarrow r_1, e_a \leftarrow ⑧$$

Background

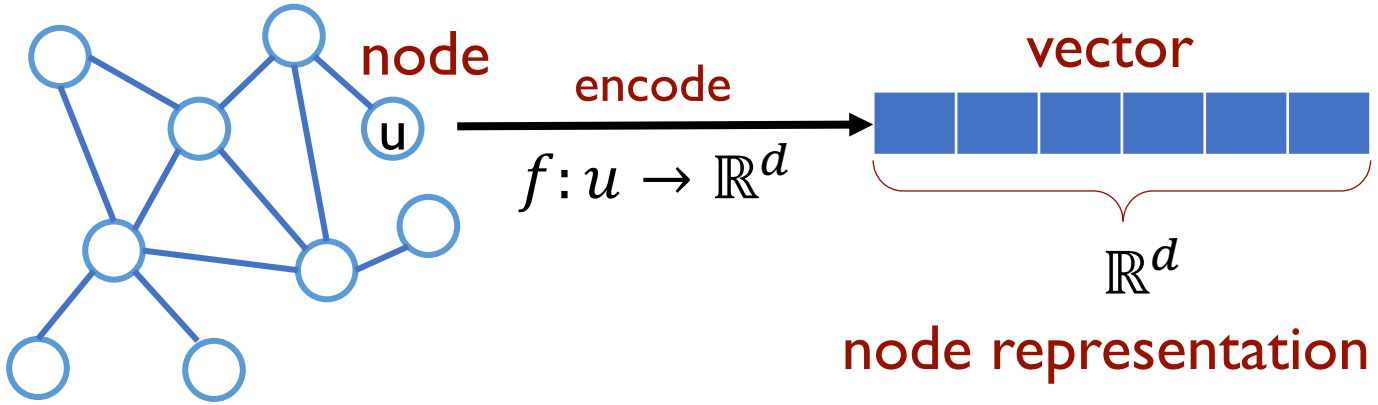
- Three classes of existing works
  - triple-based
    - $p(\mathrm{e_q}, \mathrm{r_q}, \mathrm{e_a})$ is measured by a scoring function, utilizing the representations $\mathbf{e}_\mathrm{q}, \mathbf{r}_\mathrm{q}, \mathbf{e}_\mathrm{a}$
  - path-based
    - learn the sequential order of structures by leveraging the relational paths between $\mathrm{e_q}$ and $\mathrm{e_a}$
  - graph-based
    - directly use the (sub-)graph structure for reasoning, capturing more complex semantics



Background

- Graph-based method for link prediction

Goal: efficient task-independent representations for reasoning on KG.

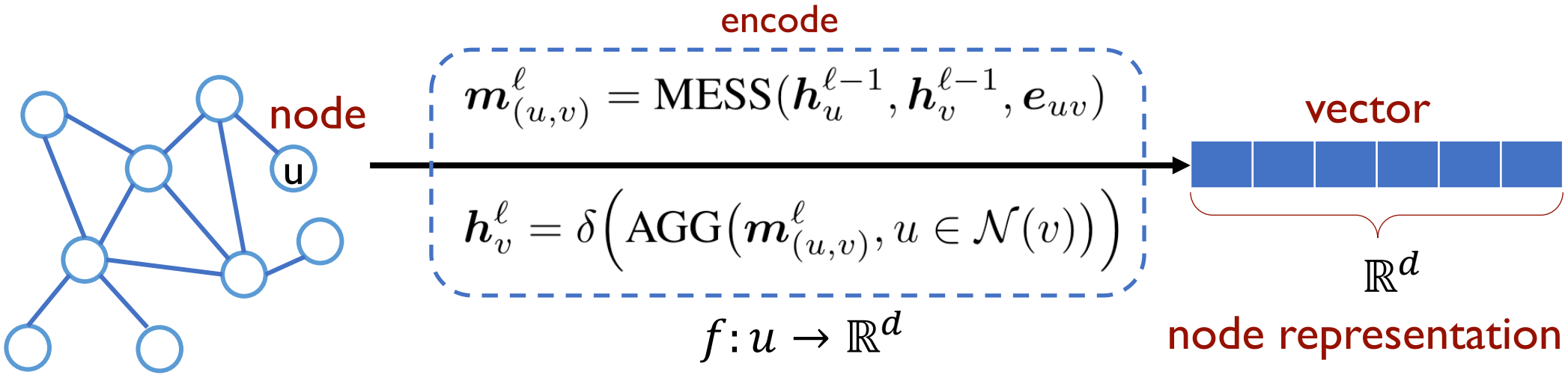


Reference: Representation Learning on Networks, snap.stanford.edu/proj/embeddings-www, WWW 2018

Background

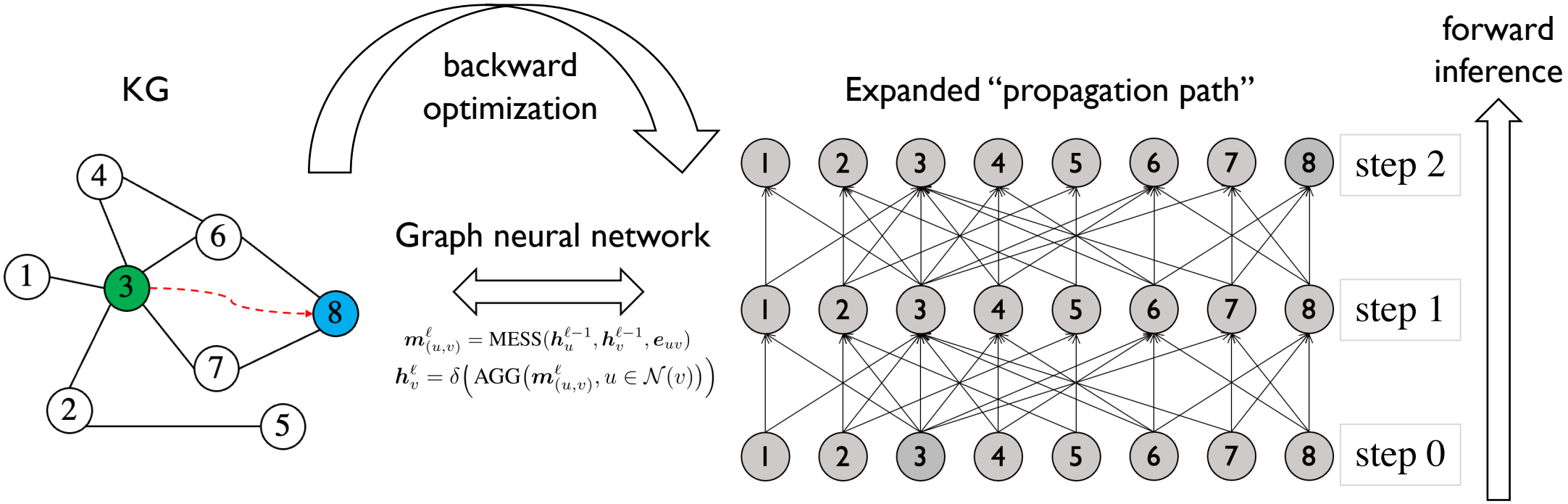
- Graph-based method for link prediction

Goal: efficient task-independent representations for reasoning on KG.



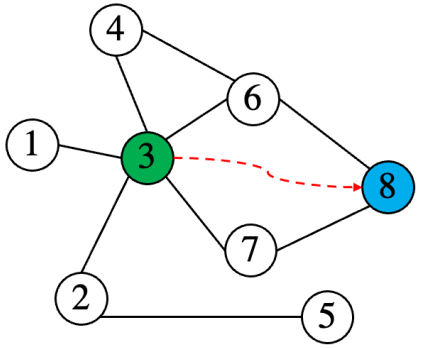
Background

- Graph-based method for link prediction
  - **propagate** the message with the KG structure
  - **update** entity representation at each propagation step



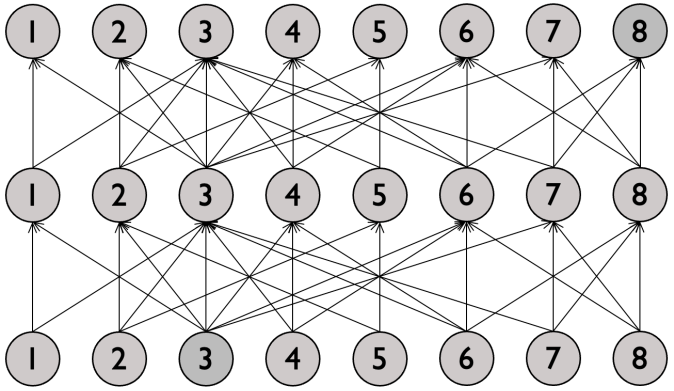
Background

- Different GNNs and their "propagation paths"

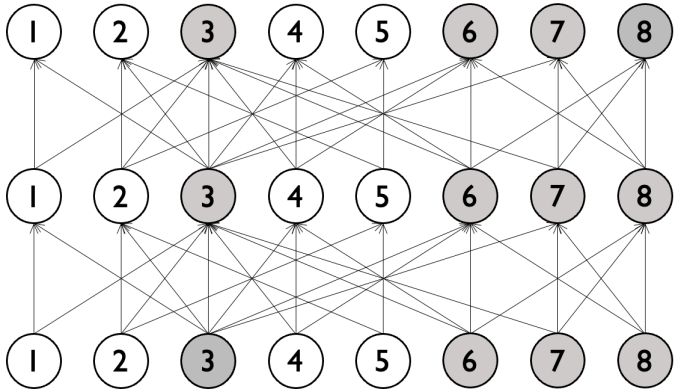


Standard

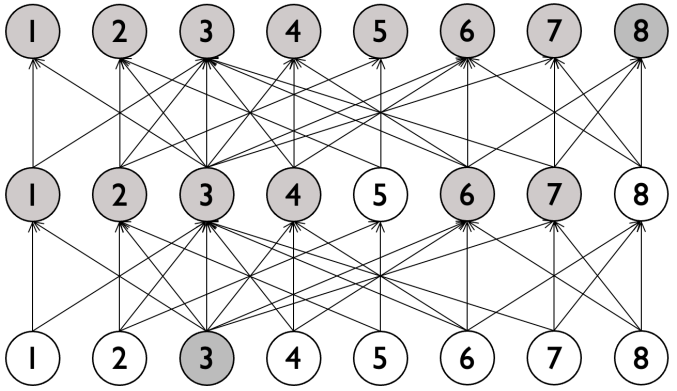
Variants



CompGCN / R-GCN

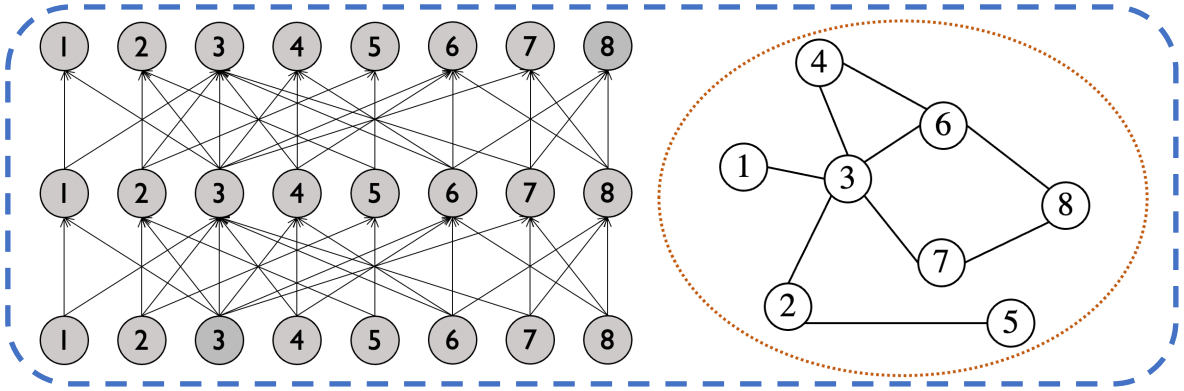


GraIL



RED-GNN

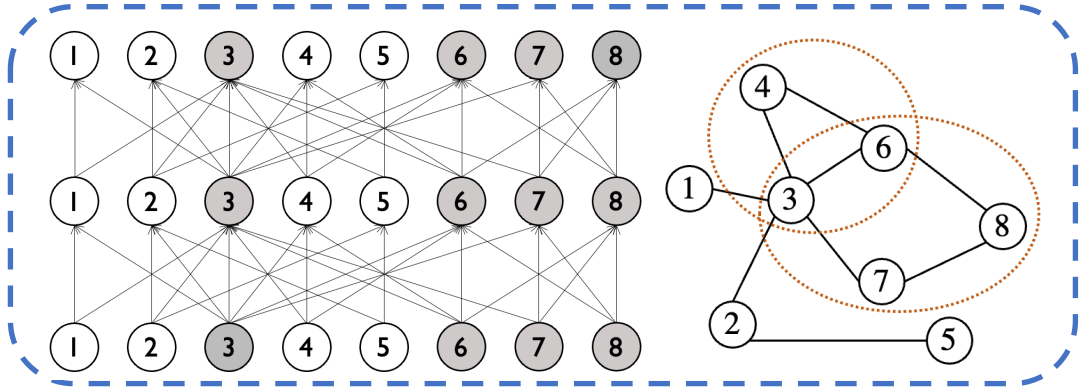
Background

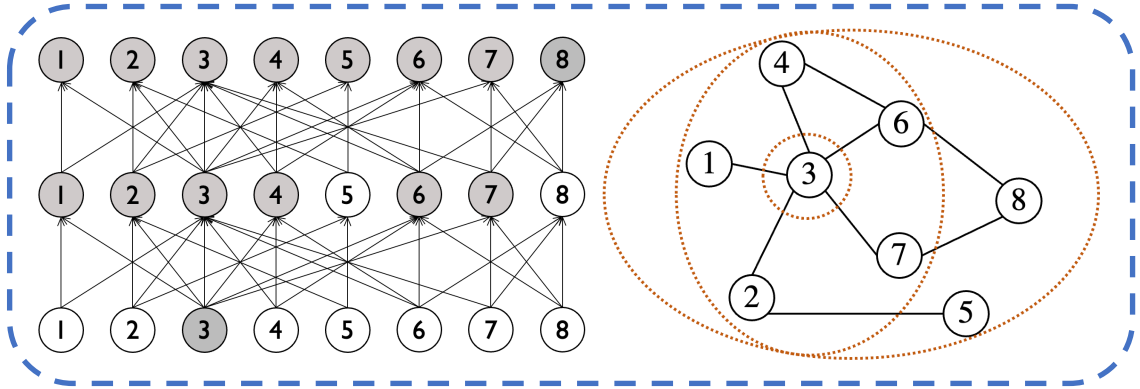


R-GCN 2017 CompGCN 2020 DPMPN 2020 NBFNet 2021

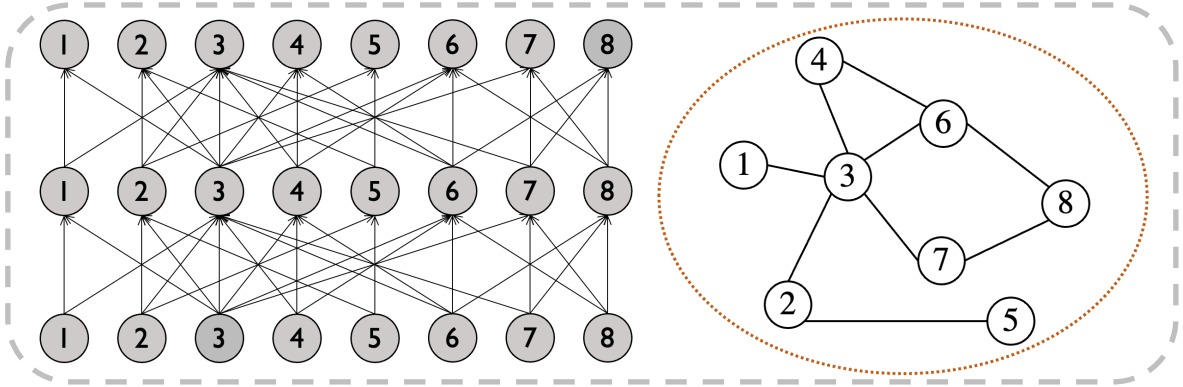
2020 GraIL

2021 RED-GNN





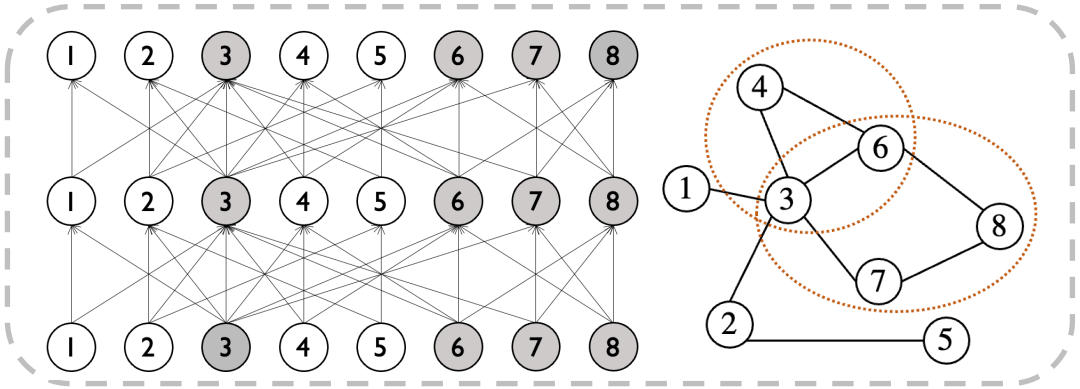
Background

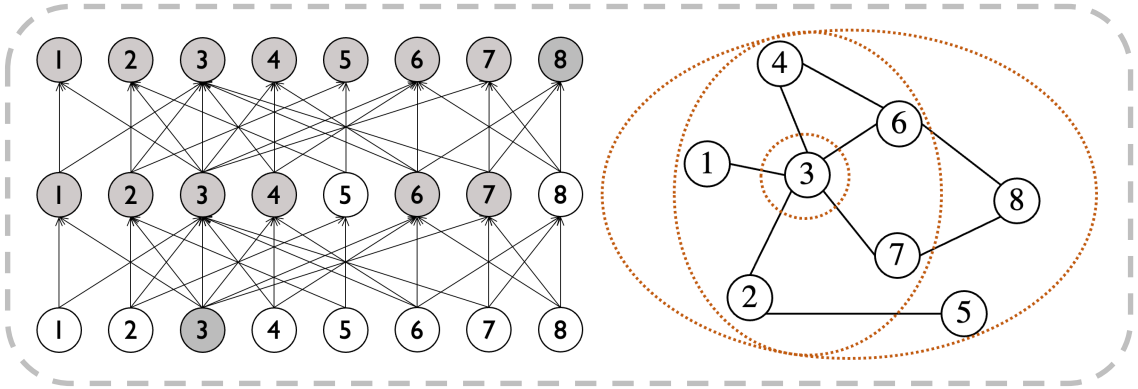


Existing work designed various propagation schemes.

We make formal definition and detailed analysis

to better disentangle the factors of the **"propagation path".**



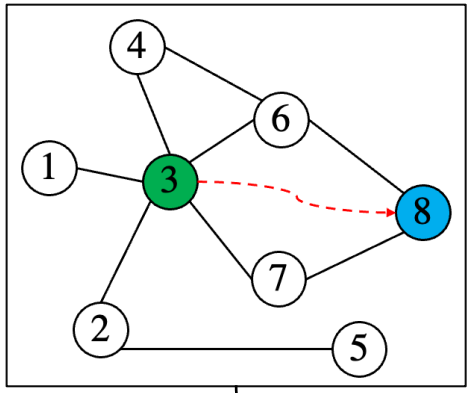


Outline

- Background
- A unified framework of propagation path
  - ➔ Q_1 What is the propagation path?
- Learning to propagate for KG reasoning
- Summary

Definition of propagation path

We define the ***propagation path*** for a query $(e_q, r_q, ?)$
as the layered and directed graph
containing the activated entities and edges at each step,
that is induced from the graph neural network
by propagating the message of ***activated entities*** to their neighbors up to ***L steps***.

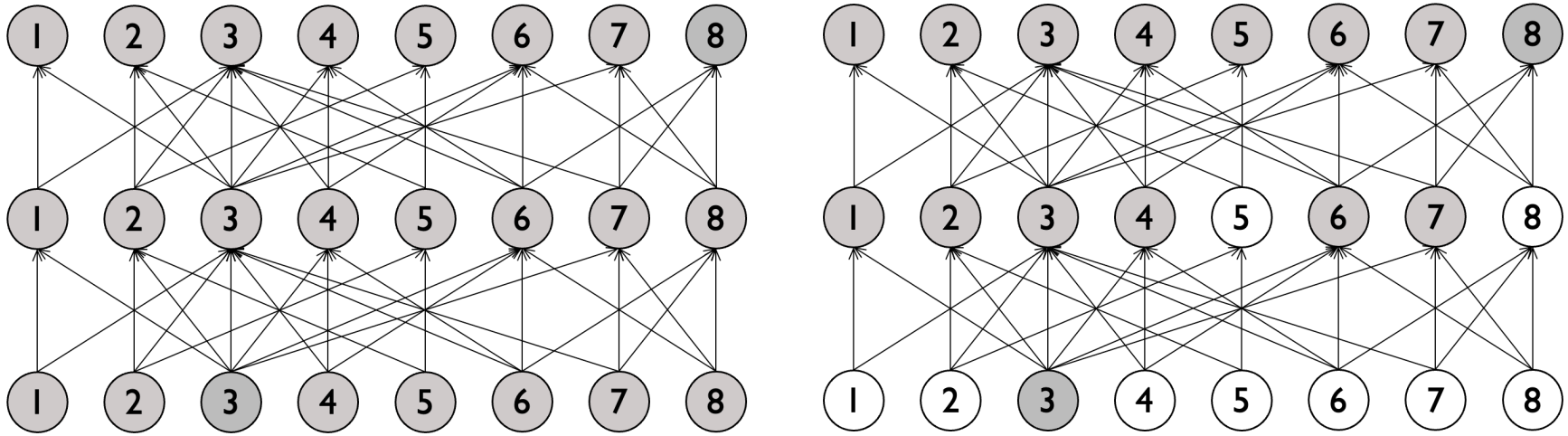


e_q= ③
r_q = r_1
(e_a= ⑧)

example1

example2

step 2
step 1
step 0

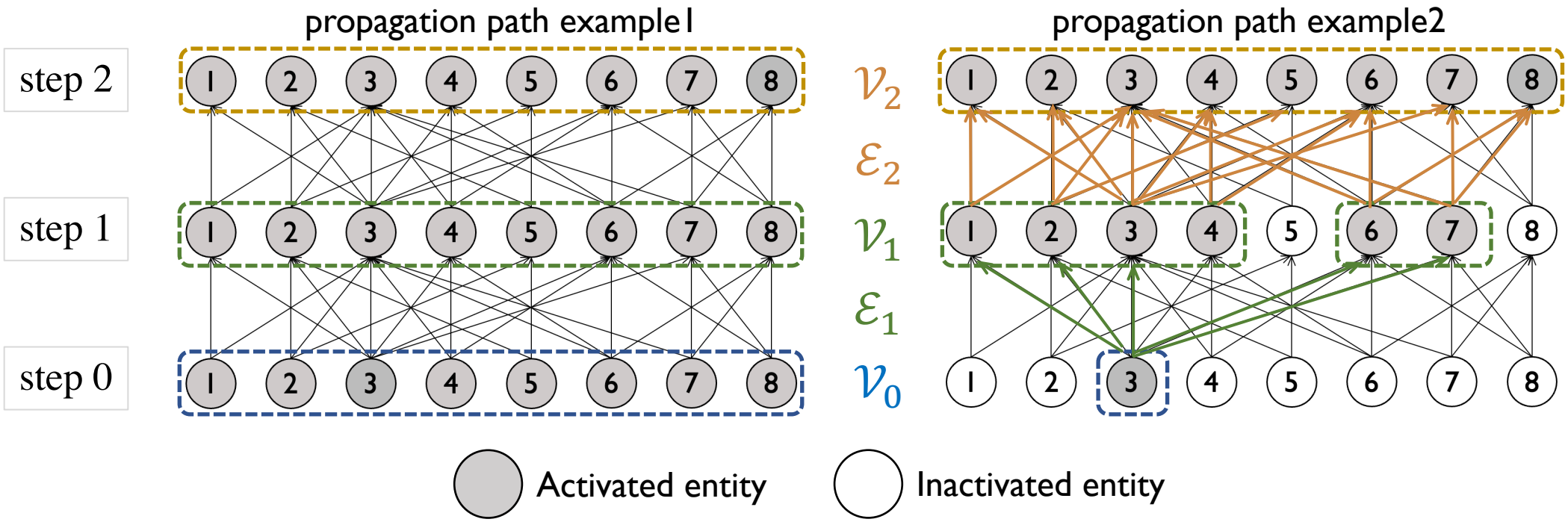


Activated entity Inactivated entity

Definition of propagation path

A propagation path with L layers $\widehat{\mathcal{G}^L} = \{\mathcal{G}^0, \mathcal{G}^1, \mathcal{G}^2, \ldots, \mathcal{G}^l\}$

- with propagation path at l-th layer $\mathcal{G}^l = (\mathcal{V}^l, \mathcal{E}^l)$



A unified learning framework
based on the propagation path

Algorithm 1 General propagation algorithm for KG reasoning.

Require: propagation path $\hat{\mathcal{G}}^L = \{\mathcal{G}^0, \mathcal{G}^1, ..., \mathcal{G}^L\}$ with $\mathcal{G}^\ell = \{\mathcal{V}^\ell, \mathcal{E}^\ell\}$.

for $\ell = 1 \ldots L$ **do**
  for $(e_s, r, e_o) \in \mathcal{E}^\ell$ **do**
    message: $\boldsymbol{m}^\ell_{(e_s,r,e_o)} := \text{MESS}(\boldsymbol{h}^{\ell-1}_{e_s}, \boldsymbol{h}^\ell_r)$;
  end for
  for $e_o \in \mathcal{V}^\ell$ **do**
    aggregate: $\boldsymbol{h}^\ell_{e_o} := \delta\left(\text{AGG}\left(\boldsymbol{m}^\ell_{(e_s,r,e_o)}, (e_s, r, e_o) \in \hat{\mathcal{E}}^\ell\right)\right)$;
  end for
end for
return $\boldsymbol{w}^\top \boldsymbol{h}^L_{e_a}$ for all $e_a \in \mathcal{V}^L$.

What we focus on

message passing

message aggregation

readout

Detailed comparison can be found in Appendix

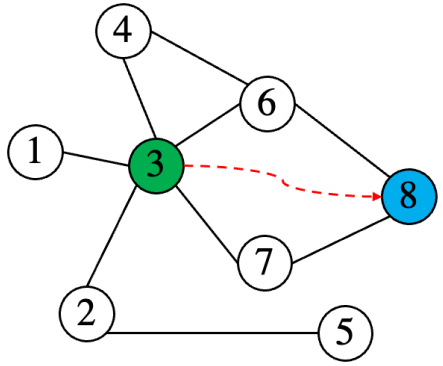
Properties of propagation path

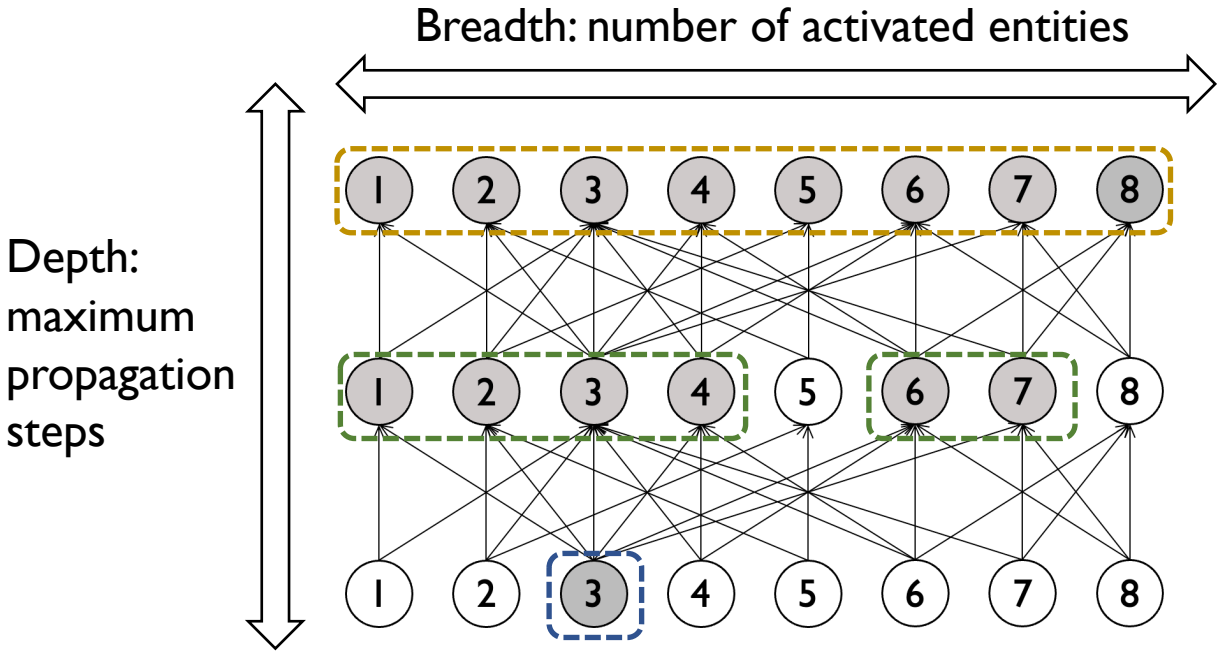
Range (structural property)

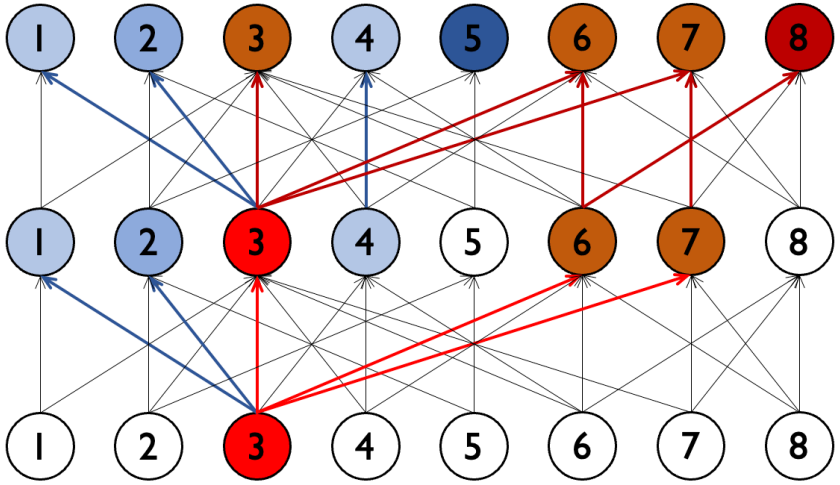
- breadth / depth / size

Relevance w.r.t. the query (semantic property)

- weights of entities and edges



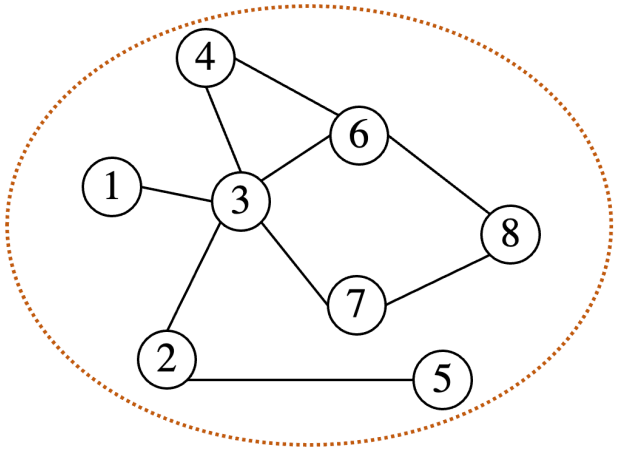


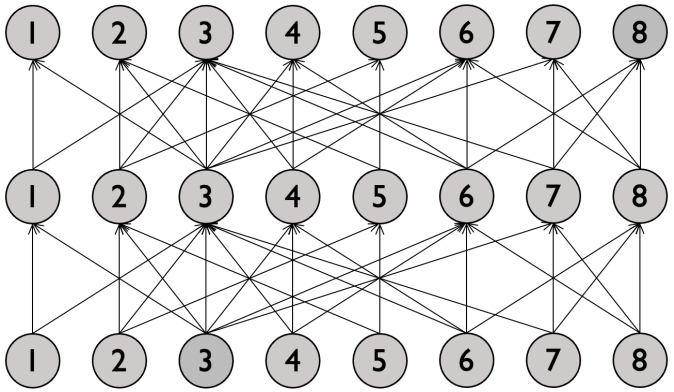


Taxonomy | w.r.t. Range

Full propagation

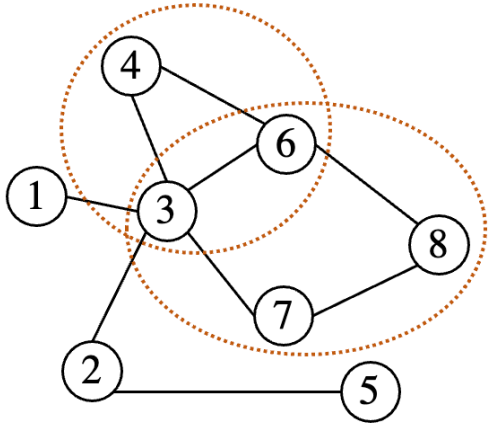
- CompGCN / R-GCN
- NBFNet / DPMPN

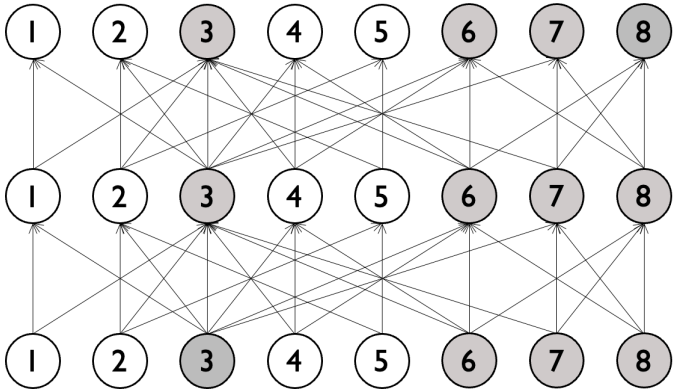




Limited propagation

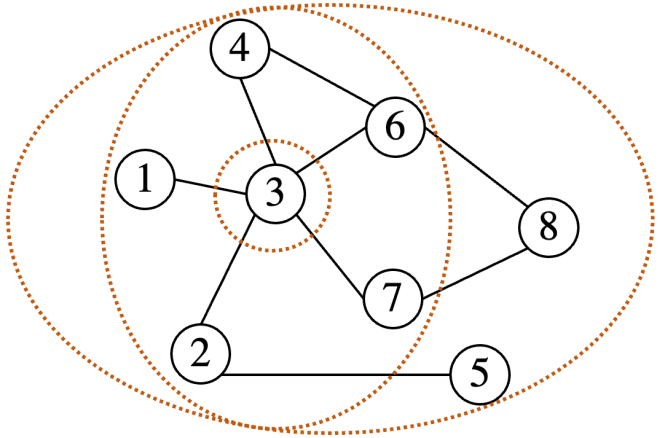
- GraIL

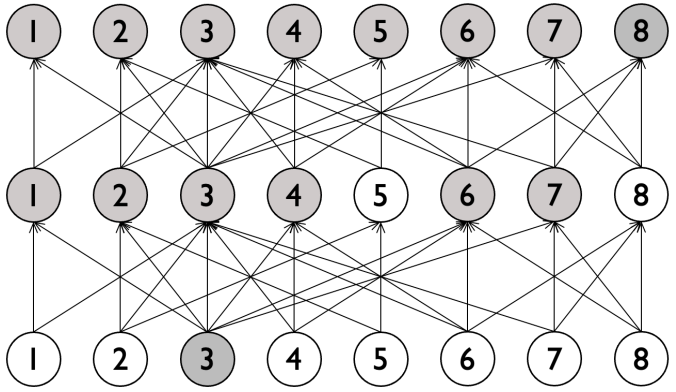




Progressive propagation

- RED-GNN

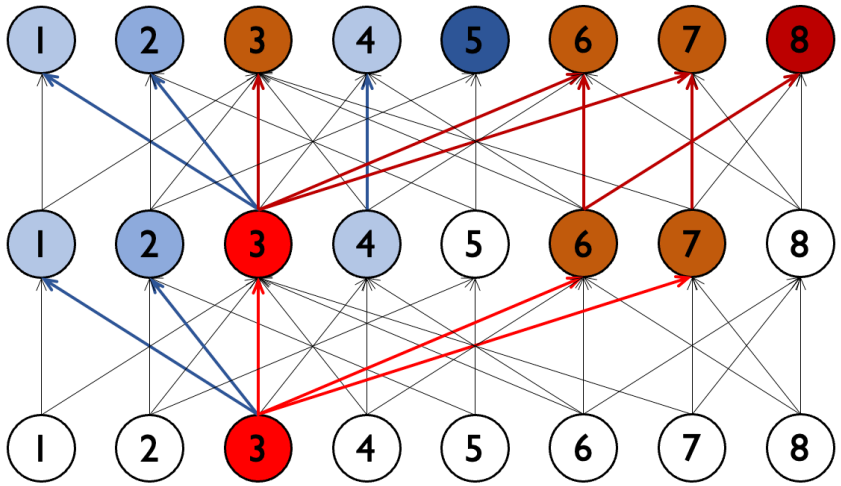




Taxonomy | w.r.t. Relevance

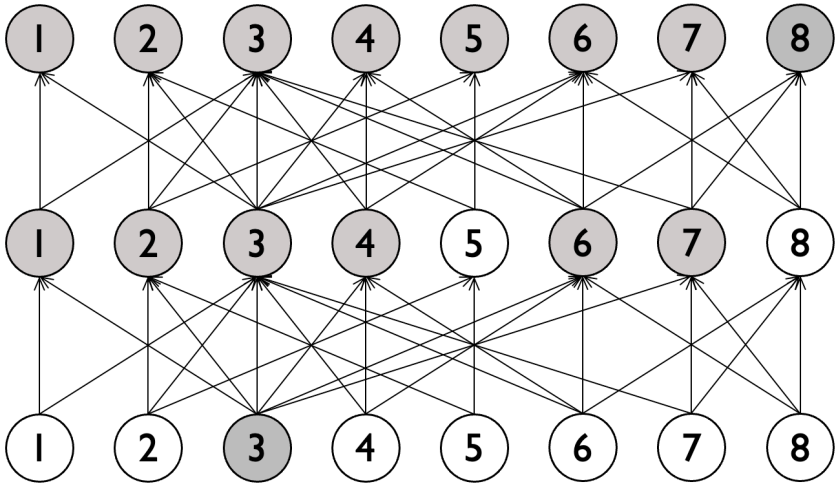
Query-dependent

- RED-GNN / NBFNet
- DPMPN /GraIL



Not query-dependent

- R-GCN / CompGCN



Comparison of existing works

		Semantic	
		Query-dependent	**Not query-dependent**
Structural	**Full propagation**	NBFNet DPMPN	R-GCN CompGCN
	Limited propagation	GraIL	NA
	Progressive propagation	RED-GNN	NA

Comparison of existing works

Semantic

Query-dependent is better than not query-dependent

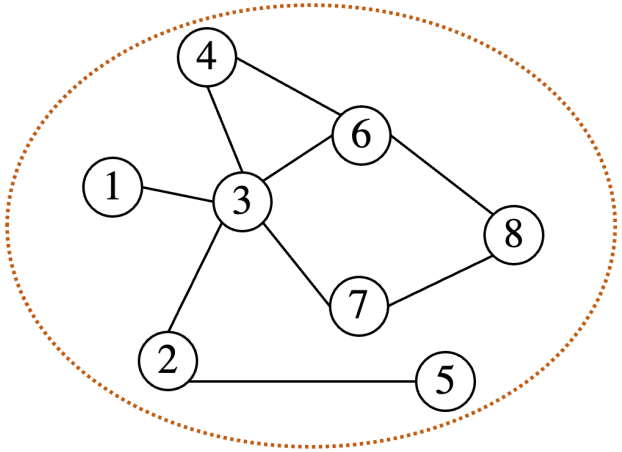
Structural			
	Full propagation	NBFNet DPMPN	R-GCN CompGCN
	Limited propagation	GraIL	NA
	Progressive propagation	RED-GNN	NA

Comparison of existing works

Structural

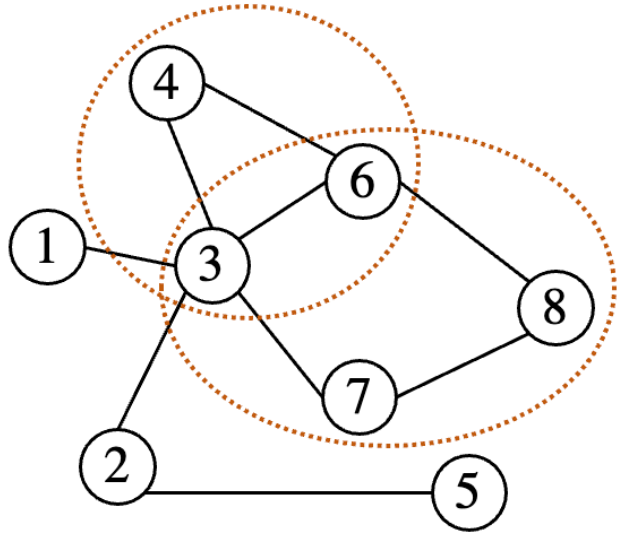
Full propagation	NBFNet DPMPN
Limited propagation	GraIL
Progressive propagation	RED-GNN

- Too expensive.
- Hard to propagate deeper.



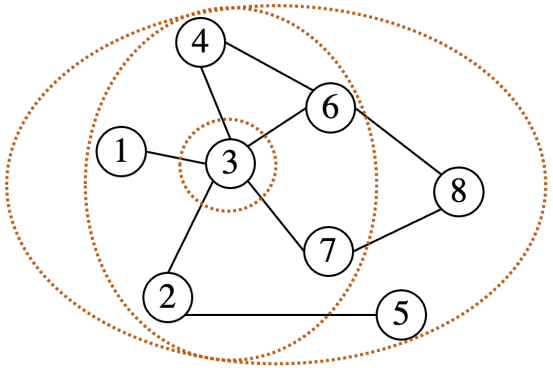
Comparison of existing works

Structural	
Full propagation	NBFNet DPMPN
Limited propagation	GraIL
Progressive propagation	RED-GNN



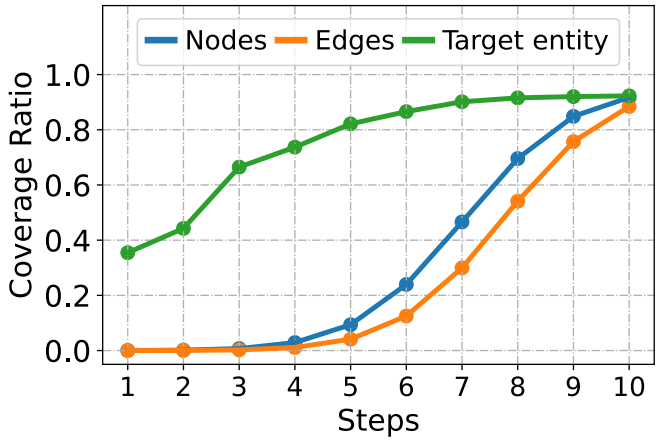
- Requires query entity and answer entity.
- Extracting enclosing subgraphs is inefficient.

Comparison of existing works

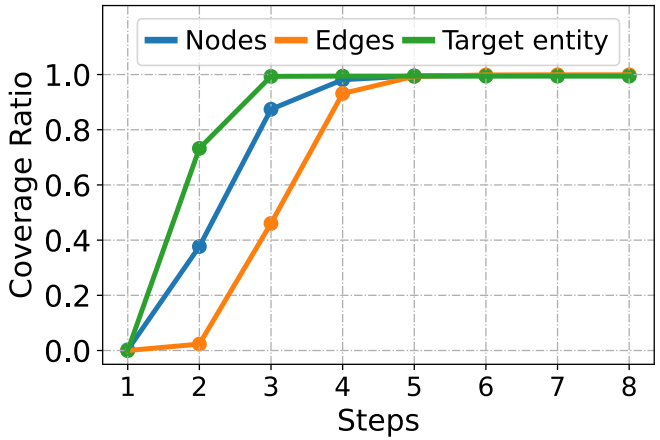


Structural	
Full propagation	NBFNet DPMPN
Limited propagation	GraIL
Progressive propagation	RED-GNN

- Will degenerate to the full propagation.
- Suffer in a dense graph.



WN18RR



FB15k-237

Comparison of existing works

	Query-dependent	Not query-dependent
Full propagation	Too expensive. Hard to propagate deeper.	
Limited propagation	Requires query entity and answer entity. Extracting enclosing subgraphs is inefficient. Inapplicable for large graph.	
Progressive propagation	Will degenerate to the full propagation. Suffer in a dense graph.	

Large size but shallow step.

Expensive and inefficient.

Outline

- Background
- A unified framework of propagation path
  - ➔ Q_1 What is the propagation path?
  - ➔ Q_2 What kinds of propagation paths do we need?
- Learning to propagate for KG reasoning
- Summary

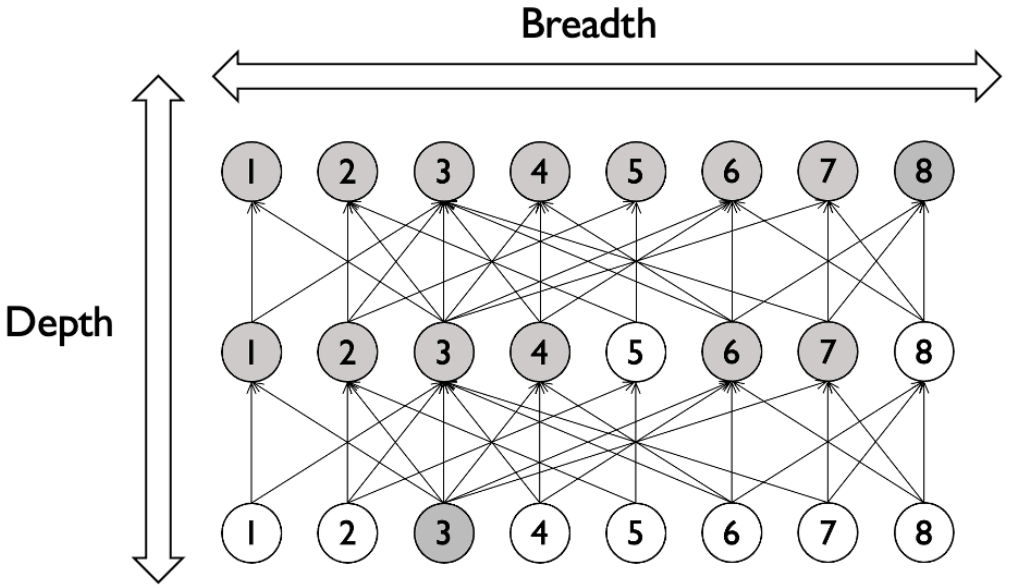
Motivations

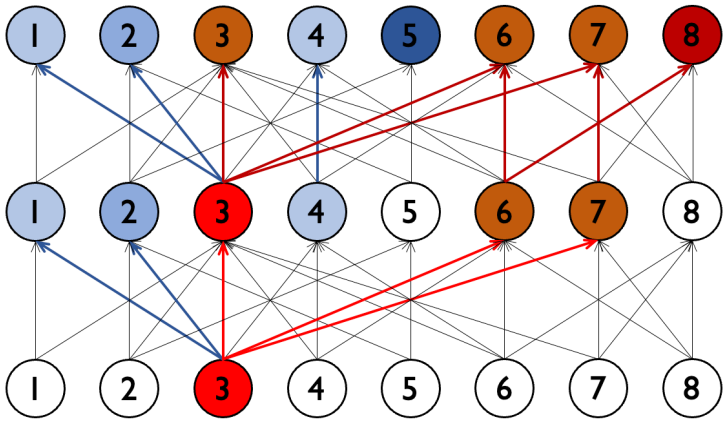
Q_2: what kinds of propagation paths do we need? 🤔

Structural **Deeper step** ➔ to capture longer-range semantics

Smaller size ➔ to decrease complexity and cost

Semantic **Query-dependent** ➔ to distinguish different entities and edges





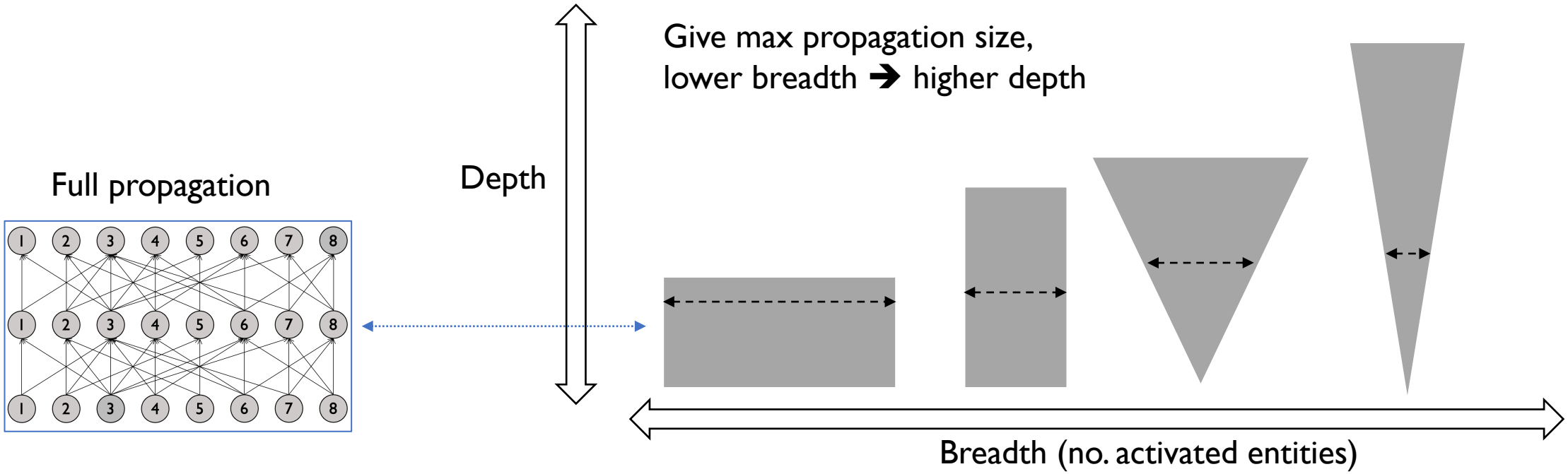
Motivations

Q_2: what kinds of propagation paths do we need? 🤔

Structural **Deeper step** ➔ to capture longer-range semantics

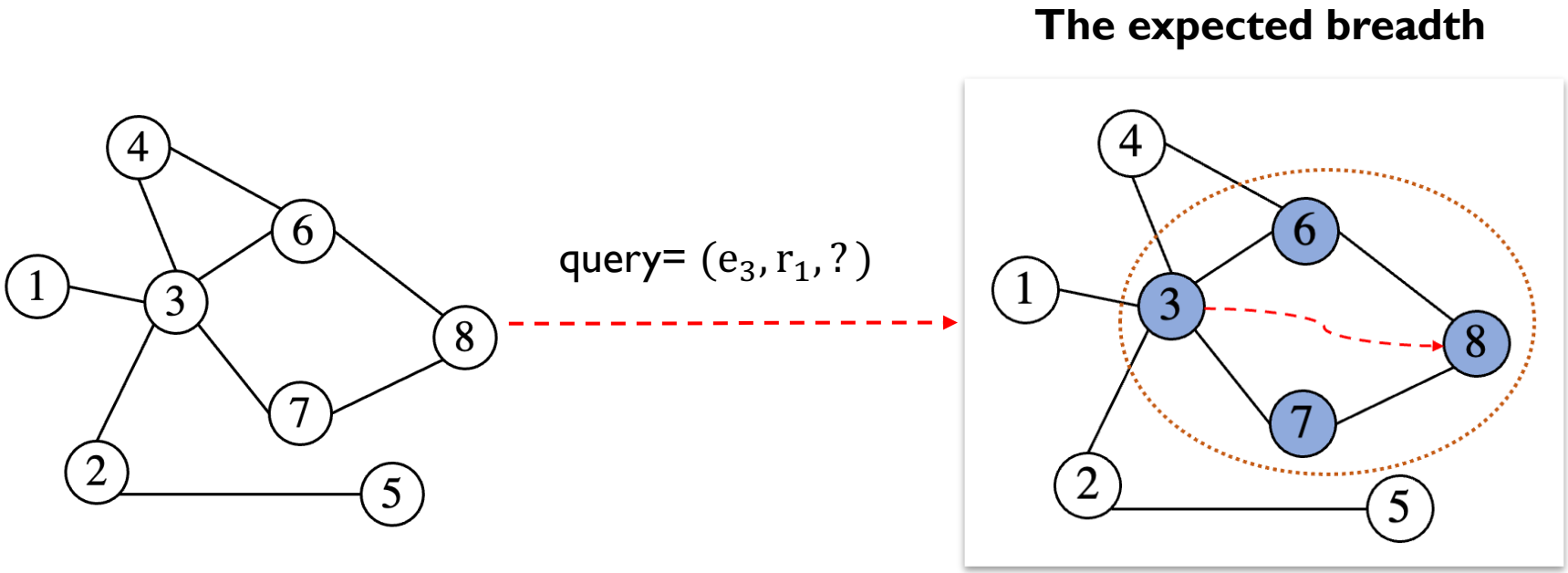
Smaller size ➔ to decrease complexity and cost

Semantic **Query-dependent** ➔ to distinguish different entities and edges



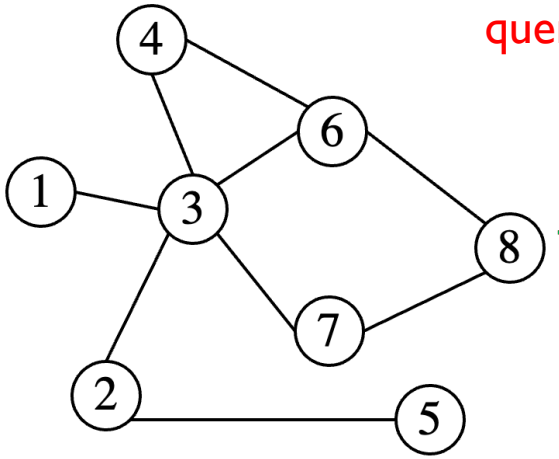
Motivations

Q_2: what kinds of propagation paths do we need?



Motivations

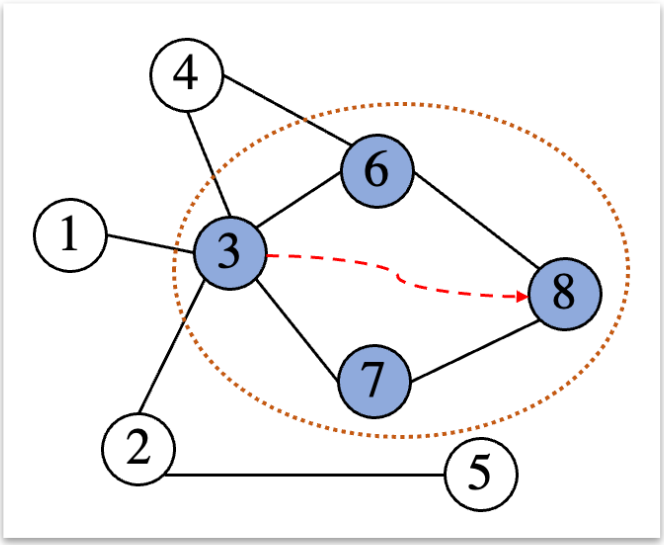
Q_2: what kinds of propagation paths do we need?

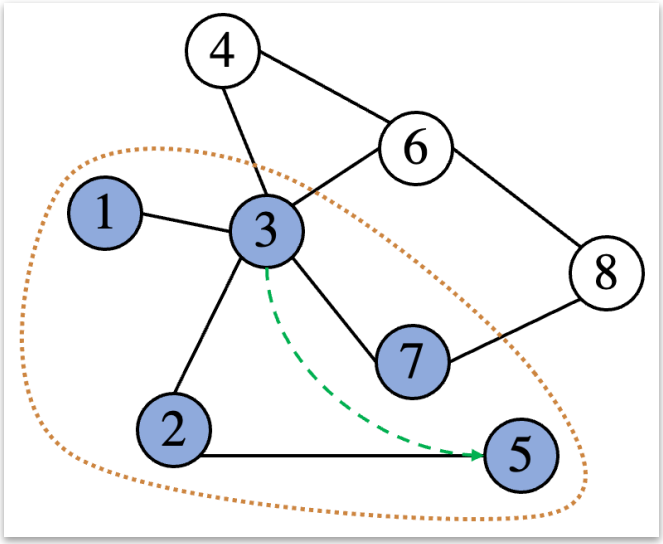


query1 = $(e_3, r_1, ?)$

query2=$(e_3, r_2, ?)$

The expected breadth





Outline

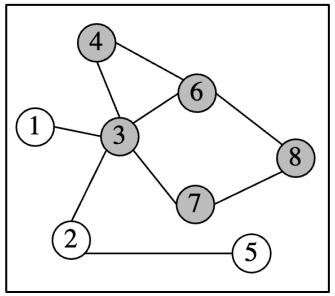
- Background
- A unified framework of propagation path
  - ➔ Q_1 What is the propagation path?
  - ➔ Q_2 What kinds of propagation paths do we need?
- Learning to propagate for KG reasoning
  - ➔ Q_3 How to learn the expected propagation paths?
- Summary

Overview

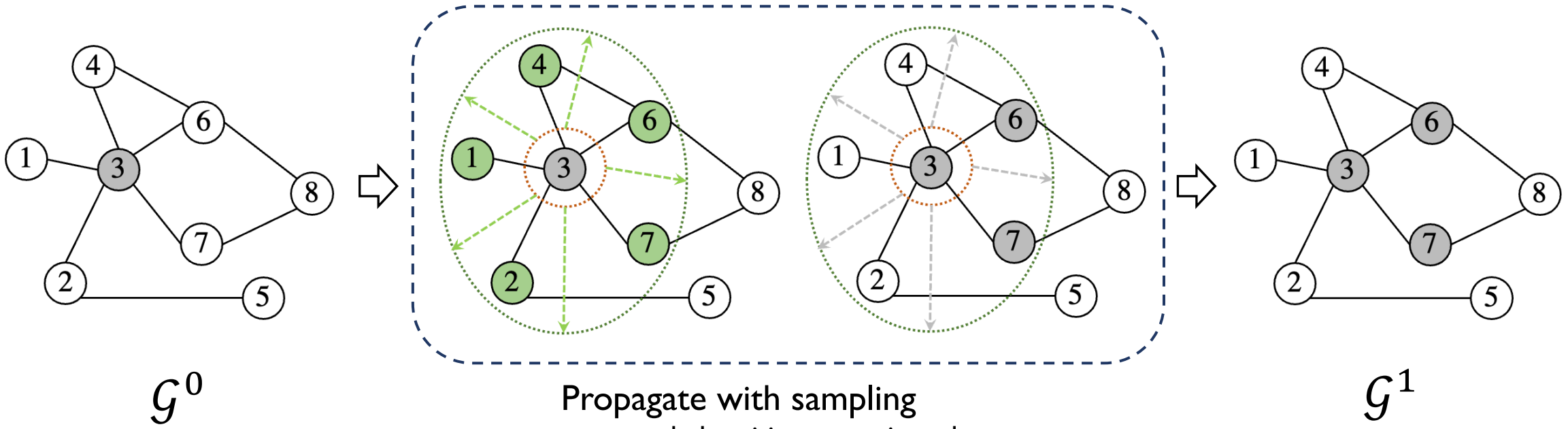
Q_3. How to learn the expected propagation paths?

$\mathcal{G}^0$ $\mathcal{G}^L$

Overview



To sample the "good" entities and discard the "bad" ones in each propagation step.



$\mathcal{G}^0$

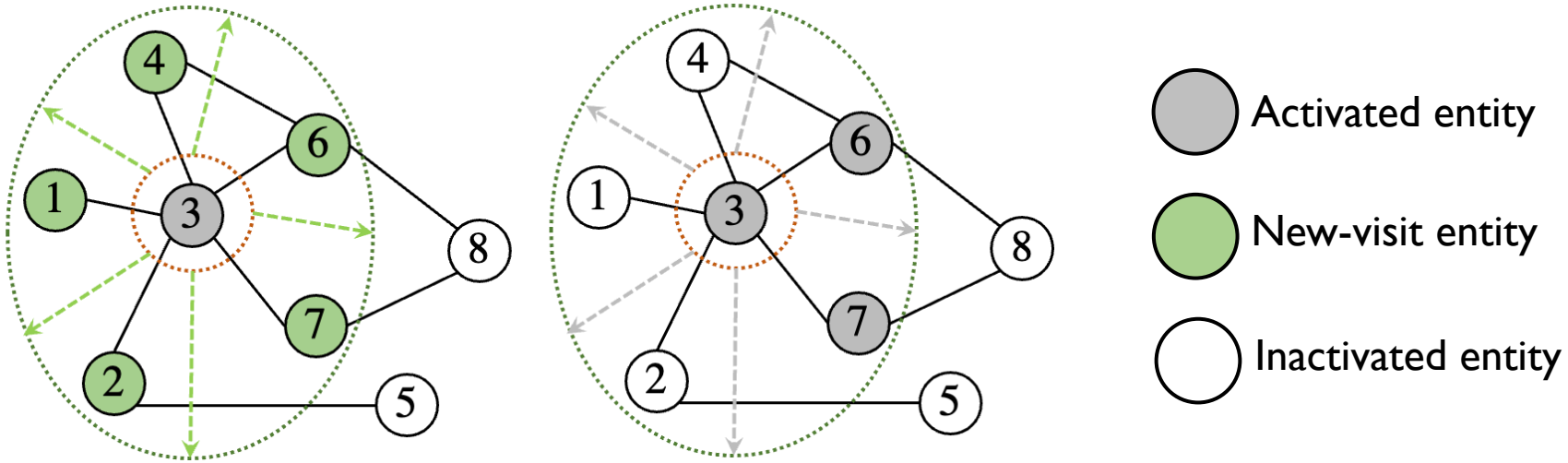
Propagate with sampling

- sampled entities are activated
- discarded entities stay inactivated

$\mathcal{G}^1$

Activated entity New-visit entity Inactivated entity

Sampling scheme



The sampling problem: $\mathrm{SAMP}(\mathrm{CAND}(\mathcal{V}^{l-1})) \rightarrow \mathcal{V}^{l}$

- (CAND) determine the candidate entities for sampling.
- (SAMP) give the sampling probabilistic distribution.

Sampling scheme | CAND

$\mathrm{SAMP}(\mathrm{CAND}(\mathcal{V}^{l-1})) \to \mathcal{V}^{l}$

- (CAND) determine the candidate entities for sampling.
- (SAMP) give the sampling probabilistic distribution.

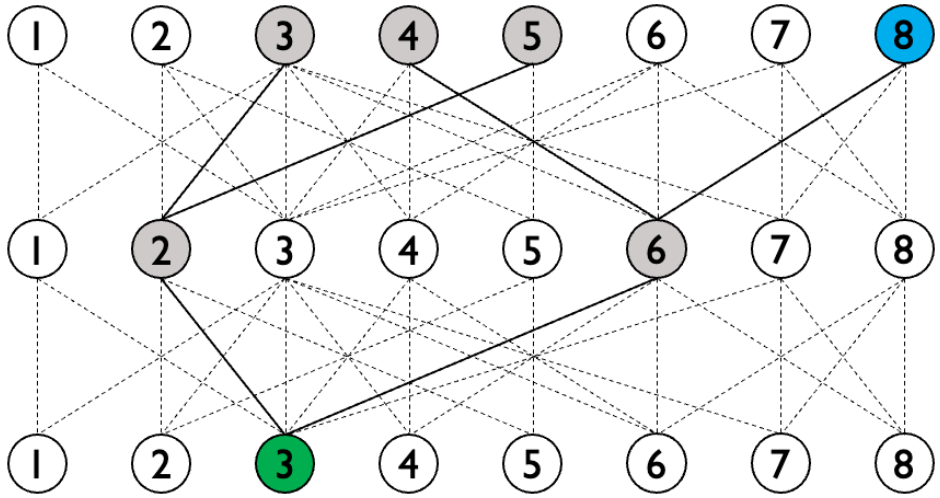
Node-wise sampling

- sample K neighbors for each entity in $\mathcal{V}^{l-1}$
- $\mathrm{CAND}(\mathcal{V}^{l-1}) = \{\mathcal{N}(e) \mid e \in \mathcal{V}^{l-1}\}$
- $|\mathcal{V}^{l}| = K|\mathcal{V}^{l-1}| = K^2|\mathcal{V}^{l-2}|$ (max number)
- exponential explosion in no. entity

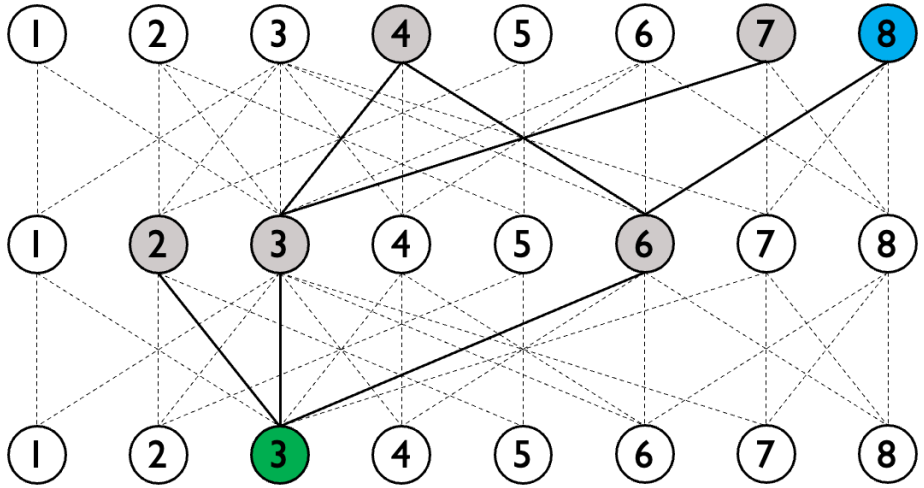
Layer-wise sampling

- sample K entities in each layer
- $\mathrm{CAND}(\mathcal{V}^{l-1}) = \mathcal{V}$
- sparse connection
- low node usage

Node-wise, K=2



Layer-wise, K=3



Sampling scheme | CAND

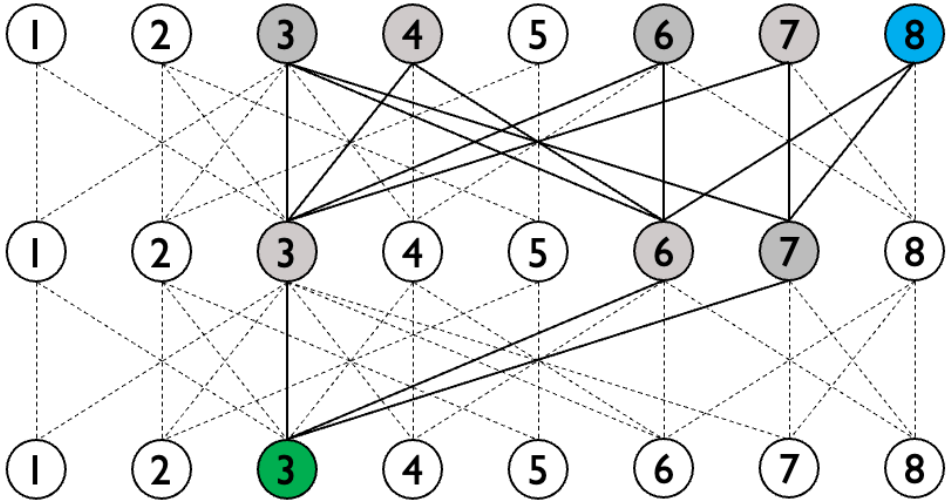
$\text{SAMP}(\text{CAND}(\mathcal{V}^{l-1})) \rightarrow \mathcal{V}^{l}$

- (CAND) determine the candidate entities for sampling.
- (SAMP) give the sampling probabilistic distribution.

Incremental sampling

- keep all the activated entities in the last layer
- sample K new-visit entities in the current layer
- $\ddot{\mathcal{V}}_l = \text{CAND}(\mathcal{V}^{l-1}) = \{e_n \mid e_n \in \mathcal{N}(e), e_n \notin \mathcal{V}^{l-1}, e \in \mathcal{V}^{l-1}\}$
- $\text{SAMP}(\ddot{\mathcal{V}}^{l}) \cup \mathcal{V}^{l-1} \rightarrow \mathcal{V}^{l}$

Incremental, K=2



Sampling scheme | CAND

$\mathrm{SAMP}(\mathrm{CAND}(\mathcal{V}^{l-1})) \rightarrow \mathcal{V}^{l}$

- (CAND) determine the candidate entities for sampling.
- (SAMP) give the sampling probabilistic distribution.

Node-wise sampling

- sample K neighbors for each entity in $\mathcal{V}^{l-1}$
  - exponential explosion in no. entity

✅ Increase speed can be directly controlled by K

Incremental sampling

- keep all the activated entities in the last layer
- sample K new-visit entities in the current layer

Layer-wise sampling

- sample K entities in each layer
  - sparse connection, low entity usage

✅ Every entity in $\mathcal{V}^{l-1}$ is connected to $\mathcal{V}^{l}$

Sampling scheme | SAMP

$\mathrm{SAMP}(\mathrm{CAND}(\mathcal{V}^{l-1})) \rightarrow \mathcal{V}^{l}$

- (CAND) determine the candidate entities for sampling.
- (SAMP) give the sampling probabilistic distribution.

Definition 2 (The sampling problem). [10] *Given a set of entities* $\bar{\mathcal{V}}^{\ell}$ *and their learned representations* $\boldsymbol{h}_{e_o}^{\ell}$, *we define the sampling distribution*

$$p^{\ell}(e_o) = \frac{\exp\left(g^{\ell}(e_o)/\tau\right)}{\sum_{e' \in \bar{\mathcal{V}}^{\ell}} \exp\left(g^{\ell}(e')/\tau\right)}, \tag{4}$$

where τ *is the temperature value and* $g^{\ell}(e_o)$ *is the logit of entity* e_o *that can be obtained by its local structure or representation. Based on this distribution, the sampling problem is to sample* $K \leq |\bar{\mathcal{V}}^{\ell}|$ *entities without replacement from* $\bar{\mathcal{V}}^{\ell}$.

➔ How to sample K entities from the candidate set $\bar{\mathcal{V}}_l$

Sampling scheme | SAMP

$\mathrm{SAMP}(\mathrm{CAND}(\mathcal{V}^{l-1})) \rightarrow \mathcal{V}^{l}$

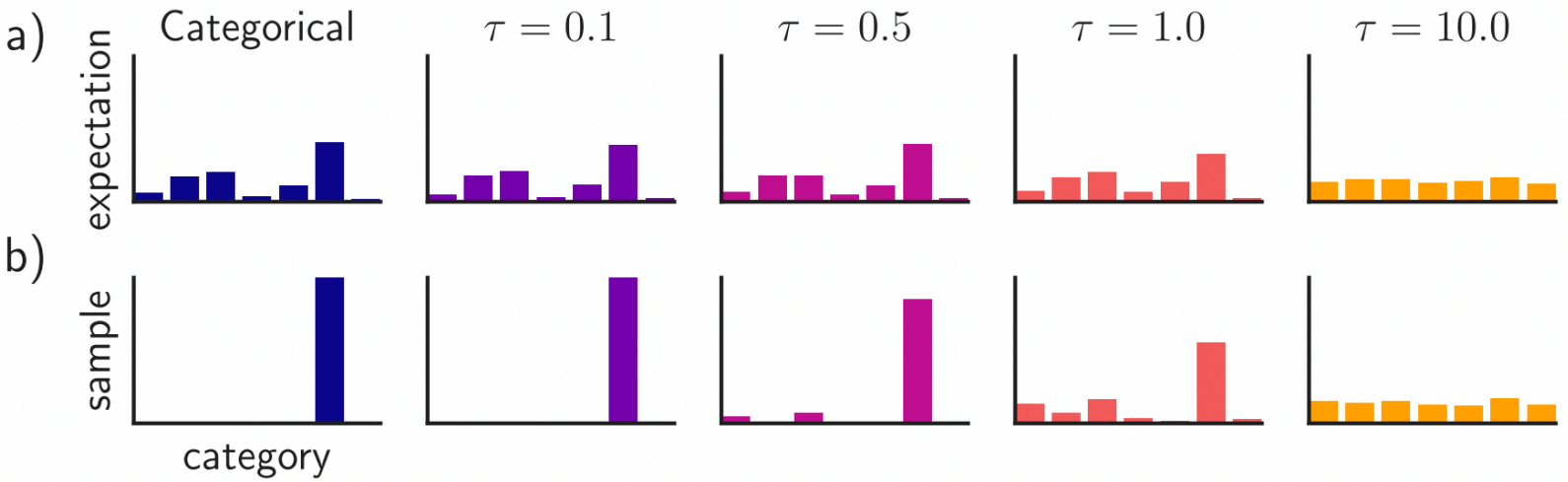
- (CAND) determine the candidate entities for sampling.
- (SAMP) give the sampling probabilistic distribution.

Standard categorical distribution:

- Given n logits $[\phi_1, \phi_2, \ldots, \phi_n]$, and a sampling distribution $p_i = \frac{\exp(\phi_i)}{\sum_{j=1}^{n} \exp(\phi_j)}$

The Gumbel softmax trick:

- add the noise $u_i \sim U(0,1)$ to the logits with $\phi_i' = \phi_i - \log(-\log u_i)$
- get top-k ϕ_i' is an approximation of sampling from $p_i' = \frac{\exp(\phi_i'/\tau)}{\sum_{j=1}^{n} \exp(\phi_j'/\tau)}$



Sampling scheme | Optimization

Straight through gradient estimator

➔ multiply the sampling logits to the corresponding entity representations

$$\boldsymbol{\phi}_{e_i} \leftarrow \sigma(\boldsymbol{W}^l_{samp}\boldsymbol{h}^l_{e_i})$$

$$\boldsymbol{p}^l(e_i) \leftarrow \text{Gumbel\_softmax}\left(\boldsymbol{\phi}_{e_i} \mid \{\boldsymbol{\phi}_{e_j}, e_j \in \ddot{\mathcal{V}}_l\}\right)$$

$$\boldsymbol{h}^l_{e_i} \leftarrow \boldsymbol{h}^l_{e_i} * (1 - \text{no\_grad}(\boldsymbol{p}^l(e_i)) + \boldsymbol{p}^l(e_i))$$

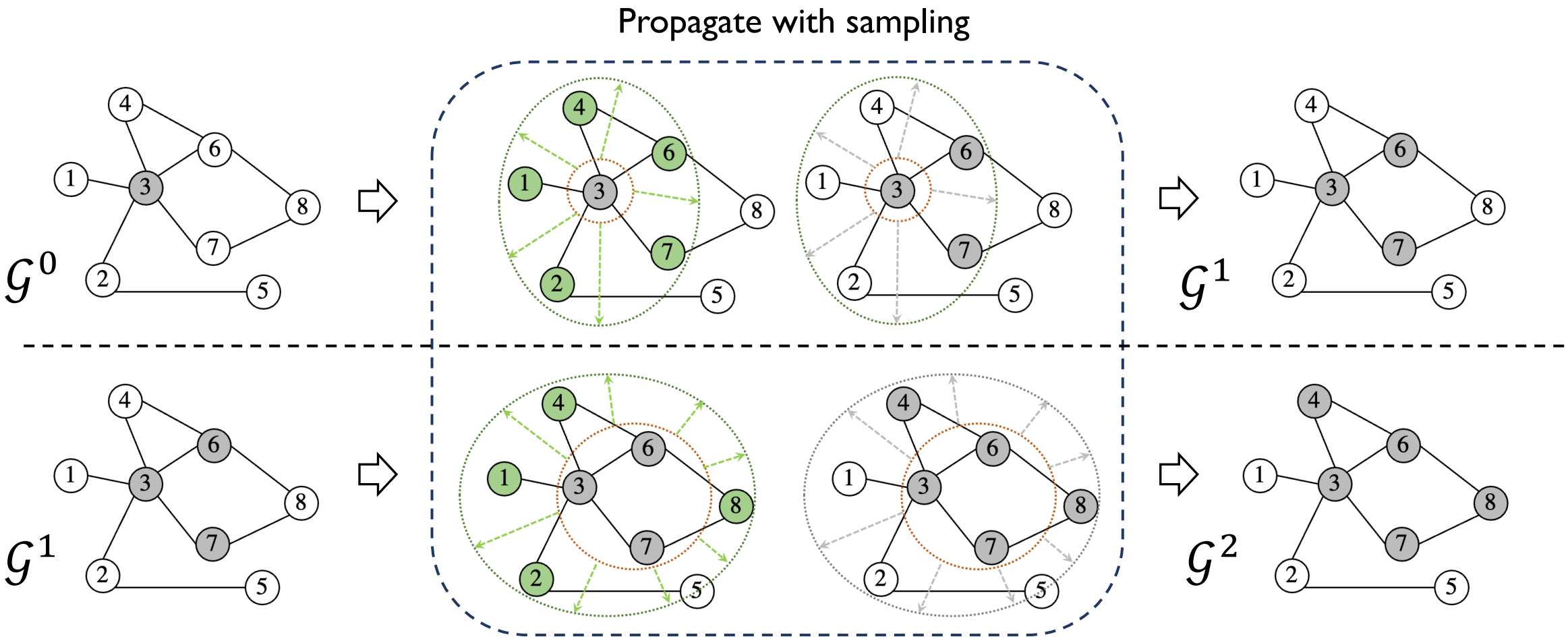
Full algorithm

Algorithm 3 PRINCE: propagation with incremental sampling.

Require: a knowledge graph $\mathcal{K} = \{\mathcal{V}, \mathcal{R}, \mathcal{E}, \mathcal{Q}\}$, query $(e_q, r_q, ?)$, number of steps L and K_I entities each incremental step.

initialize $\boldsymbol{h}^0_{e_q}(e_q, r_q) = \text{IND}(e_q, r_q, e_a)$ and the entity set $\tilde{\mathcal{V}}^0_{e_q} = \{e_q\}$;
for $\ell = 1 \ldots L$ **do**
  collect the ℓ-step edges $\bar{\mathcal{E}}^\ell_{e_q} = \{(e_s, r, e_o) \in \mathcal{F} | e_s \in \tilde{\mathcal{V}}^{\ell-1}_{e_q}\}$ and entities $\bar{\mathcal{V}}^\ell_{e_q} = \{e_o | (e_s, r, e_o) \in \bar{\mathcal{E}}^\ell_{e_q}\}$; — CAND
  for $(e_s, r, e_o) \in \bar{\mathcal{E}}^\ell_{e_q}$ **do**
    message: $\boldsymbol{m}^\ell_{(e_s,r,e_o)} := \text{MESS}(\boldsymbol{h}^{\ell-1}_{e_s}, \boldsymbol{h}^{\ell-1}_{e_o}, \boldsymbol{w}_q(e_s, r, e_o, e_q, r_q))$;
  end for
  for $e_o \in \bar{\mathcal{V}}^\ell_{e_q}$ **do** — Message propagation
    aggregate: $\boldsymbol{h}^\ell_{e_o}(e_q, r_q) := \text{AGG}\big(\boldsymbol{m}^\ell_{(e_s,r,e_o)}, (e_s, r, e_o) \in \hat{\mathcal{E}}^\ell_{e_q}\big)$;
    logits: $g^\ell(e_o) := g\big(\boldsymbol{h}^\ell_{e_o}(e_q, r_q); \boldsymbol{\theta}^\ell\big)$;
  end for
  obtain the new entities $\ddot{\mathcal{V}}^\ell_{e_q} = \bar{\mathcal{V}}^\ell_{e_q} - \tilde{\mathcal{V}}^{\ell-1}_{e_q}$;
  for $e_o \in \ddot{\mathcal{V}}^\ell_{e_q}$ **do** — SAMP
    Gumbel logits: $G_{e_o} = g^\ell(e_o) - \log(-\log U_{e_o})$ with $U_{e_o} \sim \text{Uniform}(0, 1)$;
  end for
  sample $\tilde{\ddot{\mathcal{V}}}^\ell_{e_q} = \{\arg\text{top}_{K_I} G_{e_o}, \text{for } e_o \in \ddot{\mathcal{V}}^\ell_{e_q}\}$;
  for $e_o \in \ddot{\mathcal{V}}^\ell_{e_q}$ **do** — Optimization
    Approximated probability: $p^\ell(e_o \in \tilde{\ddot{\mathcal{V}}}^\ell_{e_q}) = \frac{K_I \cdot \exp(g^\ell(e_o)/\tau)}{\sum_{e' \in \ddot{\mathcal{V}}^\ell_{e_q}} \exp(g^\ell(e')/\tau)}$;
    ST: $\boldsymbol{h}^\ell_{e_o}(e_q, r_q) = \big(no\_grad(1 - p^\ell(e_o \in \tilde{\ddot{\mathcal{V}}}^\ell_{e_q})) + p^\ell(e_o \in \tilde{\ddot{\mathcal{V}}}^\ell_{e_q})\big) \cdot \boldsymbol{h}^\ell_{e_o}(e_q, r_q)$;
  end for
  update $\tilde{\mathcal{V}}^\ell_{e_q} = \tilde{\mathcal{V}}^{\ell-1}_{e_q} \cup \ddot{\mathcal{V}}^\ell_{e_q}$;
end for
return $\boldsymbol{w}^\top \boldsymbol{h}^L_{e_a}(e_q, r_q)$ for all $e_a \in \tilde{\mathcal{V}}^L_{e_q}$.

Full algorithm



Outline

- Background
  - ➔ Q_1 What is the propagation path?
- A unified framework of propagation path
  - ➔ Q_2 What kinds of propagation paths do we need?
- Learning to propagate for KG reasoning
  - ➔ Q_3 How to learn the expected propagation paths?
- Summary

Summary

Three problems to study

- Q_1 What is the propagation path.
- Q_2 What kinds of propagation paths do we need.
- Q_3 How to learn the expected propagation paths.

Three major contributions

- A **unified framework** of propagation path of GNN models for KG reasoning.
- A novel GNN with **incremental sampling strategy** that can learn query-dependent propagation path.
- The **state-of-the-arts** performance in both transductive and inductive reasoning settings.

Q&A

Thanks for your attention!

Appendix

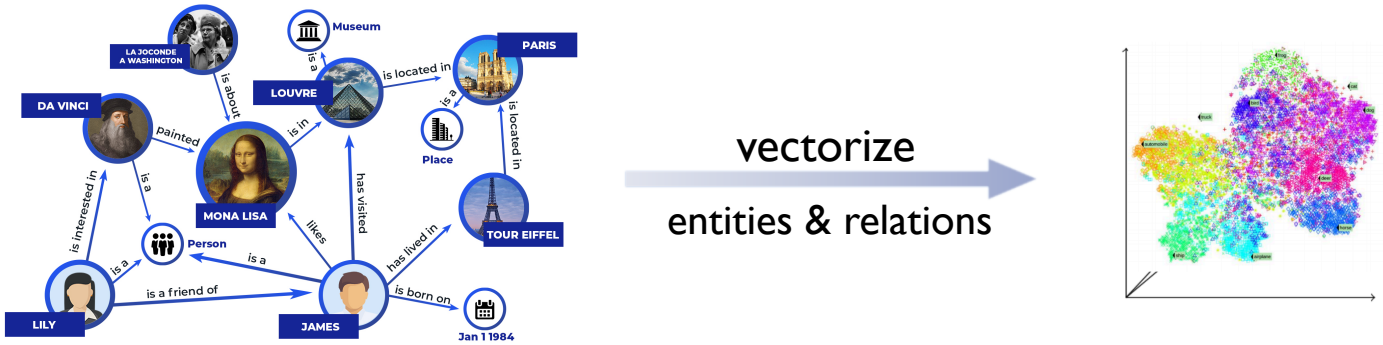
A unified learning framework
based on the propagation path

Model	Year	$\mathcal{V}_0$	$\mathcal{V}_l$	$\mathcal{E}_1$	$\mathcal{E}_l$
R-GCN	2017	$\mathcal{V}$	$\mathcal{V}$	$\mathcal{E}$	$\mathcal{E}$
CompGCN	2020	$\mathcal{V}$	$\mathcal{V}$	$\mathcal{E}$	$\mathcal{E}$
GraIL	2020	$\mathcal{V}_s \subseteq \mathcal{V}$	$\mathcal{V}_s \subseteq \mathcal{V}$	$\mathcal{E}_s \subseteq \mathcal{E}$	$\mathcal{E}_s \subseteq \mathcal{E}$
NBFNet	2021	$\mathcal{V}$	$\mathcal{V}$	$\mathcal{E}$	$\mathcal{E}$
RED-GNN	2021	$\{e_q\}$	$\{e_o \mid (e_s, r, e_o) \in \mathcal{E}_l\}$	$\{(e_q, r, e_o) \in \mathcal{E}\}$	$\{(e_s, r, e_o) \in \mathcal{E} \mid e_s \in \mathcal{V}_{l-1}\}$

Model	MESS( ) $\rightarrow m^l$	AGG( ) $\rightarrow h^l_{e_o}$	Complexity
R-GCN	$\boldsymbol{W^l h^{l-1}_{e_s}}$	$\boldsymbol{\sigma}(\sum m^{\boldsymbol{l}}_{\boldsymbol{e_o}} + \boldsymbol{W^l_0 h^{l-1}_{e_o}})$	$L\lvert\mathcal{V}\rvert$
CompGCN	$\boldsymbol{W^l}\phi(\boldsymbol{h^{l-1}_{e_s}}, \boldsymbol{h^l_r})$	$\boldsymbol{\sigma}(\sum m^{\boldsymbol{l}}_{\boldsymbol{e_o}} + \boldsymbol{W^l_0 h^{l-1}_{e_o}})$	$L\lvert\mathcal{V}\rvert$
GraIL	$\boldsymbol{W^l}\phi(\boldsymbol{h^{l-1}_{e_s}}, \boldsymbol{h^l_r}, \boldsymbol{h^l_{r_q}})$	$\boldsymbol{\sigma}(\sum m^{\boldsymbol{l}}_{\boldsymbol{e_o}} + \boldsymbol{W^l_0 h^{l-1}_{e_o}})$	$L\lvert\mathcal{V}_s\rvert$
NBFNet	$\boldsymbol{W_{r_q}}\phi(\boldsymbol{h^{l-1}_{e_s}}, \boldsymbol{h^l_r})$	$\boldsymbol{\sigma}(\boldsymbol{W^l}(\sum m^{\boldsymbol{l}}_{\boldsymbol{e_o}}, \boldsymbol{h^0_{e_o}}))$	$L\lvert\mathcal{V}\rvert$
RED-GNN	$\boldsymbol{W^l}\phi(\boldsymbol{h^{l-1}_{e_s}}, \boldsymbol{h^l_r}, \boldsymbol{h^l_{r_q}})$	$\boldsymbol{\sigma}(\boldsymbol{W^l}\sum m^{\boldsymbol{l}}_{\boldsymbol{e_o}})$	$\sum_{i=1}^{L}\lvert\mathcal{V}_i\rvert$

Background – Knowledge Graph Embedding (KGE)

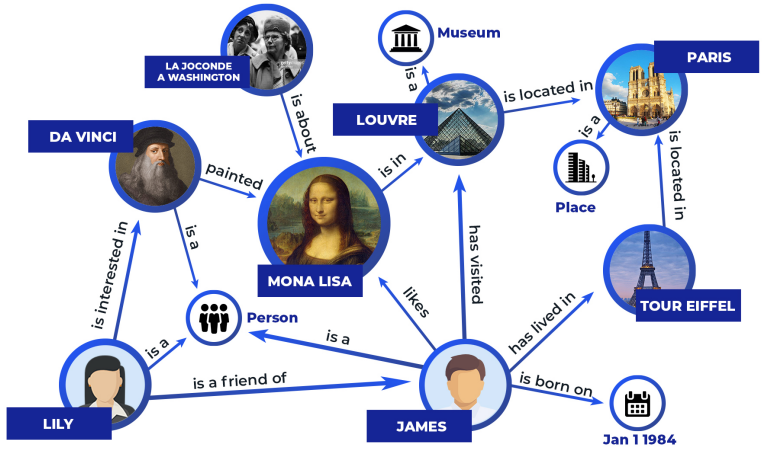
- Knowledge Graph Embedding
  - Encode entities and relations in KG into low-dimensional vectors space
  - while capturing nodes' and edges' connection properties



- Most KGE models based on embeddings define a scoring function F to estimate the plausibility of any fact (s, r, o) using their embeddings:

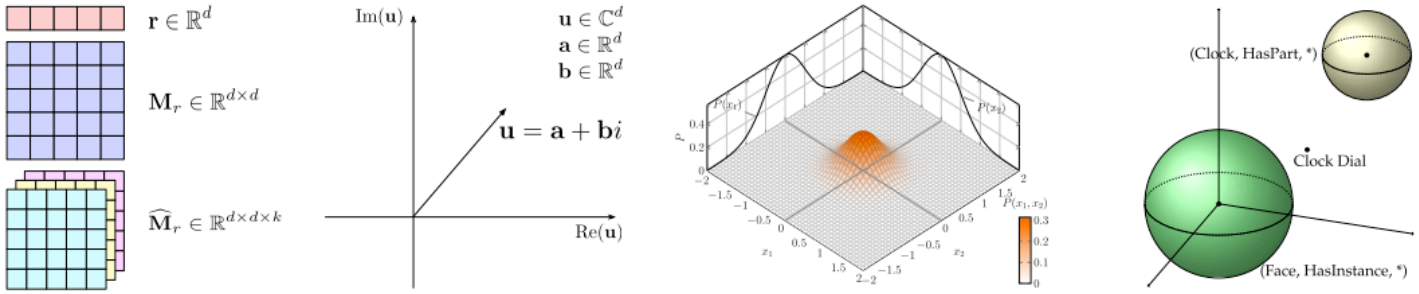
$$F(\boldsymbol{s}, \boldsymbol{r}, \boldsymbol{o}) = \mathrm{p}(\mathbf{o}|\mathbf{s}, \mathbf{r}) \text{ or } \mathrm{p}(\mathbf{s}|\mathbf{o}, \mathbf{r}) \text{ or } \mathrm{p}(\mathbf{r}|\mathbf{s}, \mathbf{o})$$

Background – review of KGE models

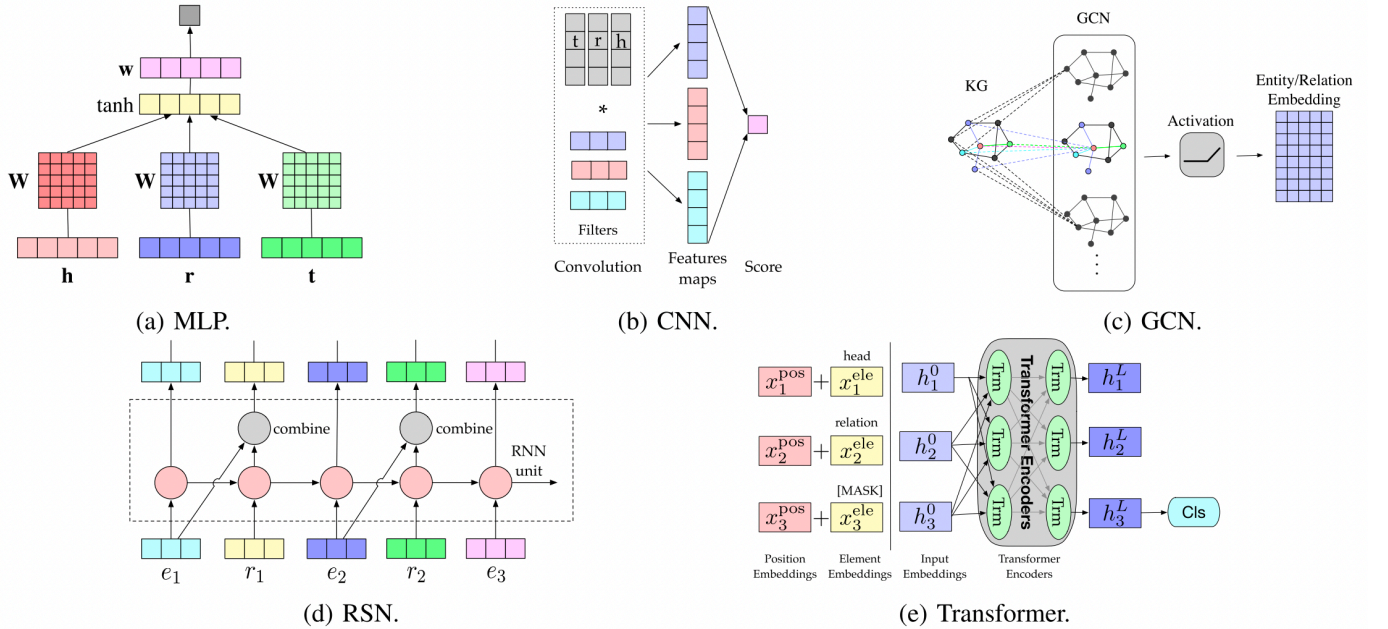


Simple Taxonomy

- Embedding models
  - tensor decomposition models
  - geometric models
  - deep learning models
- Structure-based models
  - rule-mining
  - path/graph-based



(a) Point-wise space. (b) Complex vector space. (c) Gaussian distribution. (d) Manifold space.



(a) MLP. (b) CNN. (c) GCN.

(d) RSN. (e) Transformer.

$$F(s, r, o)$$

KGE training and testing

- Training
  - S^+: $positive\ samples$ S^-: $negative\ samples$
  - Objectives: $max\ \phi^+(S^+) + \phi^-(S^-)$
- Prediction
  - For link prediction, given an incomplete triple $(h, r, ?)$
  - the missing tail is inferred as the entity that results in the highest score:

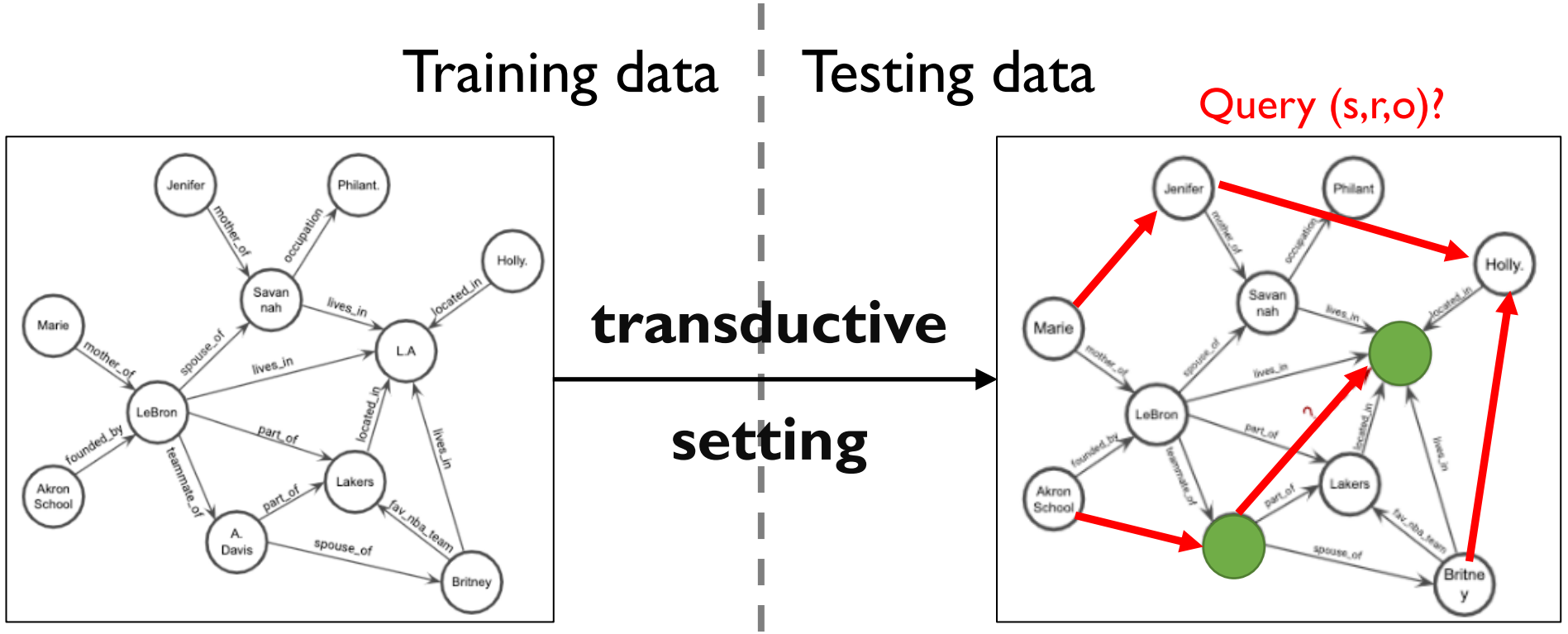
$$t = \underset{e \in \mathcal{E}}{\operatorname{argmax}}\, \phi(\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{e})$$

- Evaluation
  - Three common metrics
  - ***q*:** the rank of correct entity

$$MR = \frac{1}{|Q|}\sum_{q \in Q} q \qquad MRR = \frac{1}{|Q|}\sum_{q \in Q} \frac{1}{q} \qquad H@K = \frac{|\{q \in Q : q \leq K\}|}{|Q|}$$

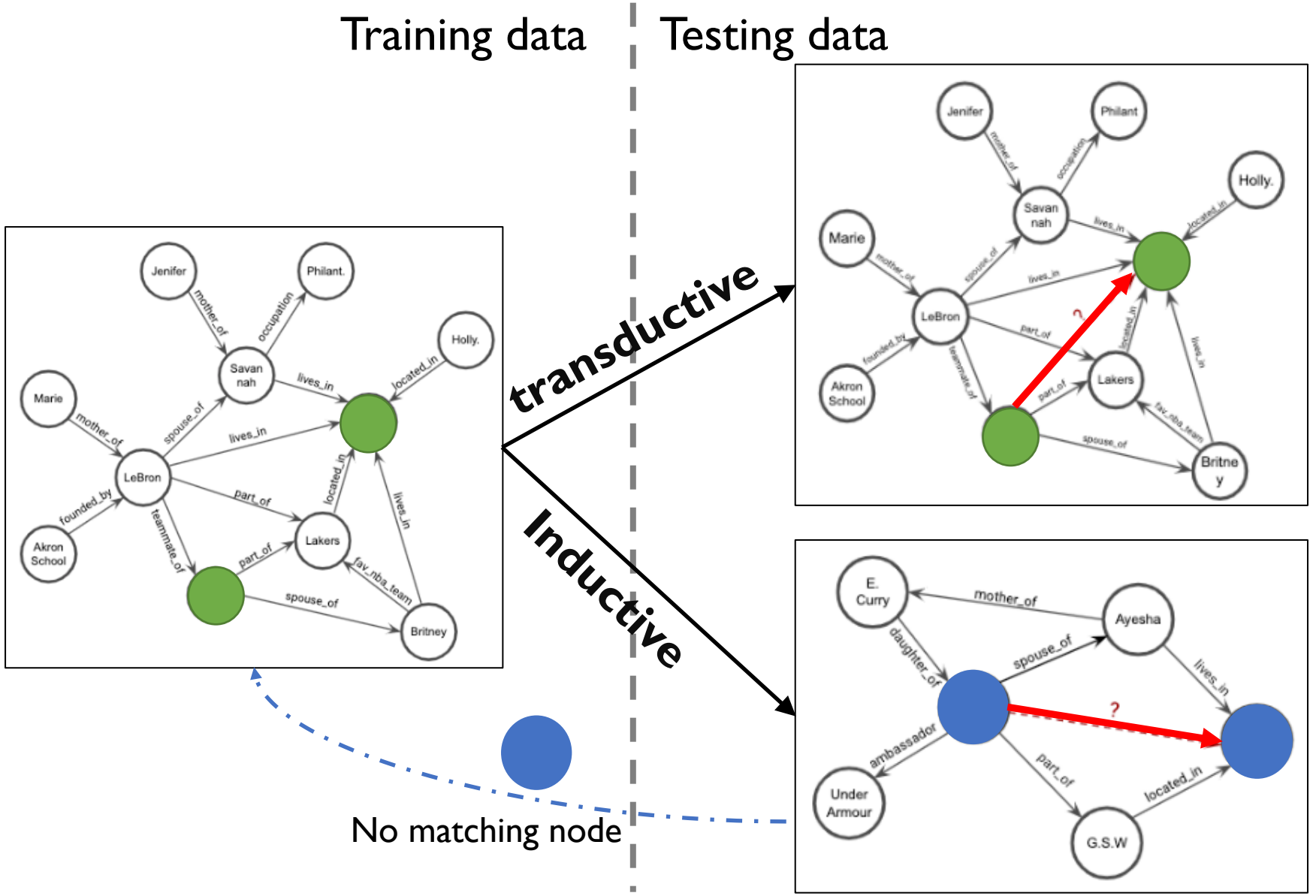
Background – Knowledge Graph Embedding (KGE)

Transductive learning | Popular learning paradigm in KGE



The same set of entities (nodes)
for the training and testing data

Two learning paradigms: Inductive and transductive



The same set of entities (nodes) as the training data

Different set of entities to the training data

e.g.,

- new users and products on e-commerce platforms
- new molecules in biomedical KG