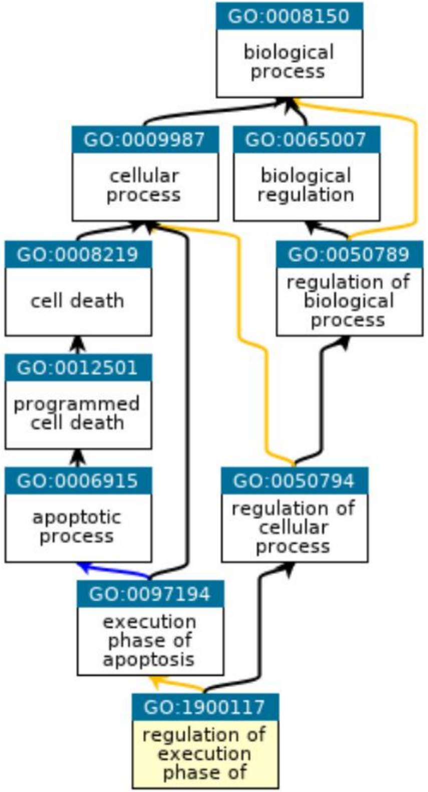
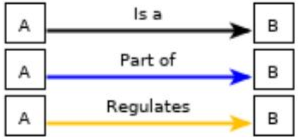
Graph embeddings

The ‘Graph’ part

Treat the ontology as a graph

Vertices and edges



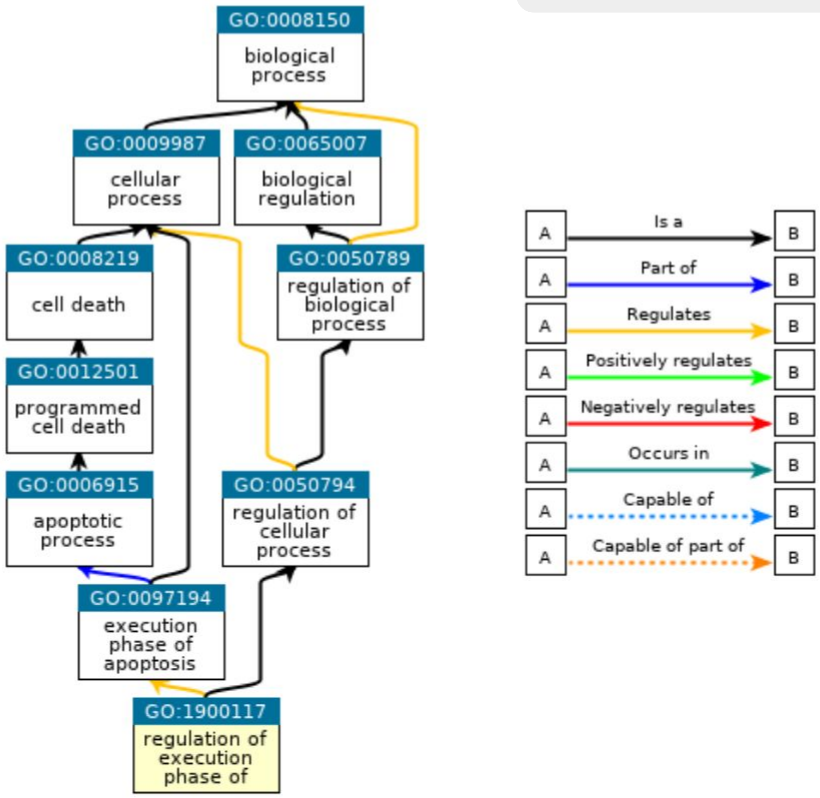


The ‘Graph’ part

Different graphs

For example:

- Using different relations

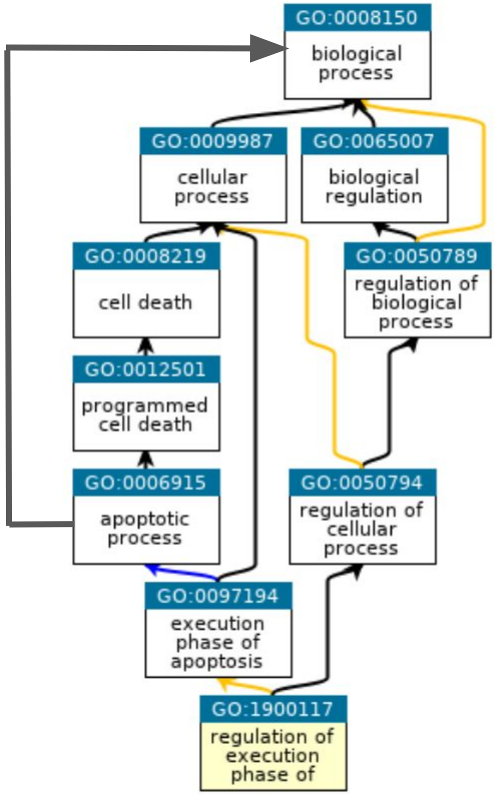


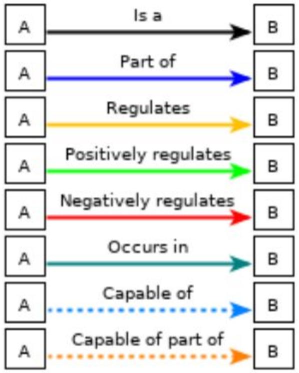
The ‘Graph’ part

Different graphs

For example:

- Using different relations
- Reasoning (Deductive closure)

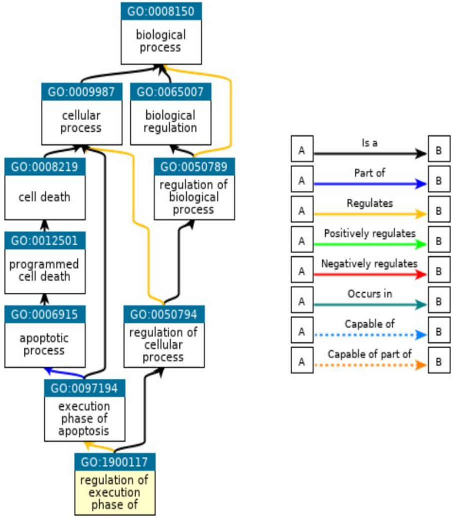




The ‘embeddings’ part

Definition

Let $KG = (V, E, L; \vdash)$ be an ontology graph with a set of vertices V, a set of edges $E \subseteq V \times V$, a label function $L : V \cup E \mapsto Lab$ that assigns labels from a set of labels Lab to vertices and edges, and an inference relation $\vdash$. An ontology graph embedding is a function $f_\eta : L(V) \cup L(E) \mapsto \mathbf{R}^n$.

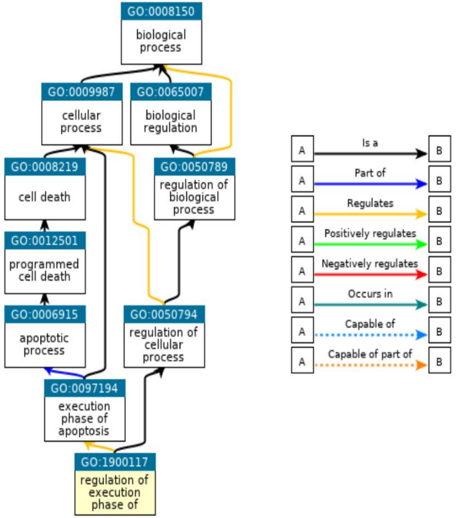


What is an embedding?

Definition

Let $KG = (V, E, L; \vdash)$ be an ontology graph with a set of vertices V, a set of edges $E \subseteq V \times V$, a label function $L : V \cup E \mapsto Lab$ that assigns labels from a set of labels Lab to vertices and edges, and an inference relation $\vdash$. An ontology graph embedding is a function $f_\eta : L(V) \cup L(E) \mapsto \mathbf{R}^n$.

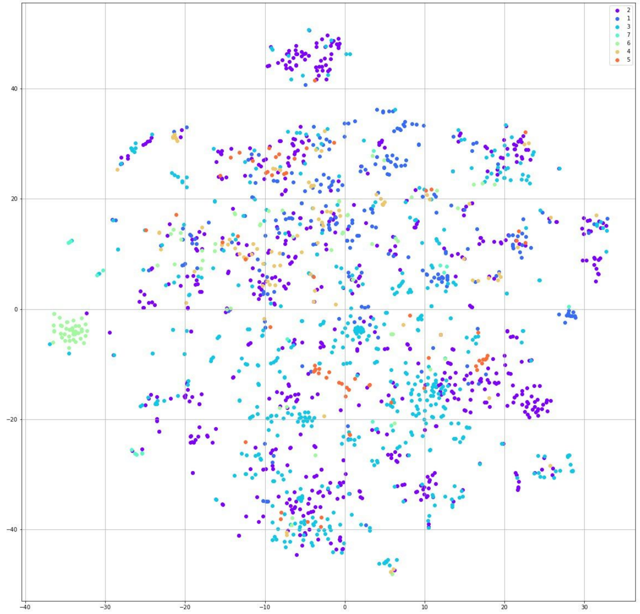
- ★ Preserve some structure of the graph in R^n
- ★ This structure is preserved under operations in R^n
- ★ These operations represent operations between the entities (connectedness, similarity, and others)

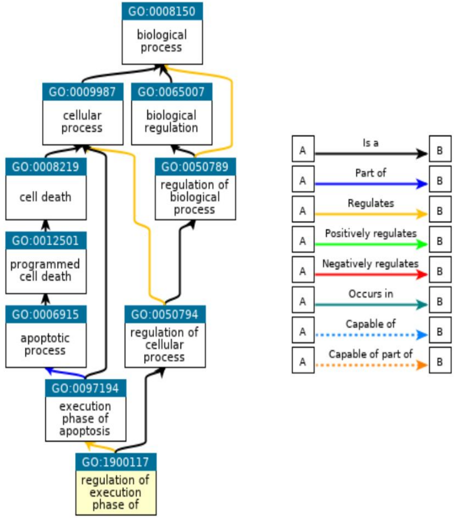


What is an embedding?

You can think of this as:

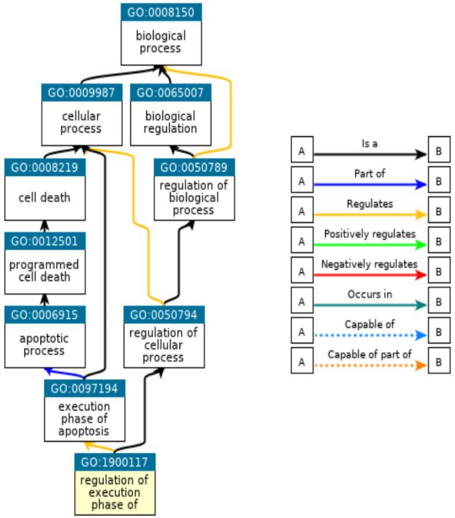
A mapping of entities to some numerical representation in some R^n





Why embeddings?

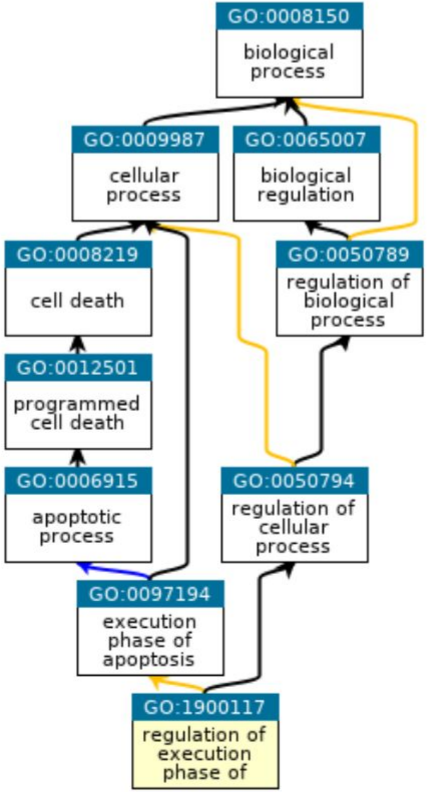
- Represent entities in a compact dimension
- Visualize entities and their relations
- Cluster entities
- Compute semantic similarity

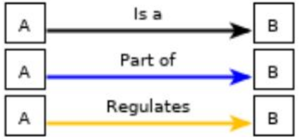


Graph Embeddings

We'd like to capture:

- Graph structure
- Adjacency
- Hub-nodes and communities

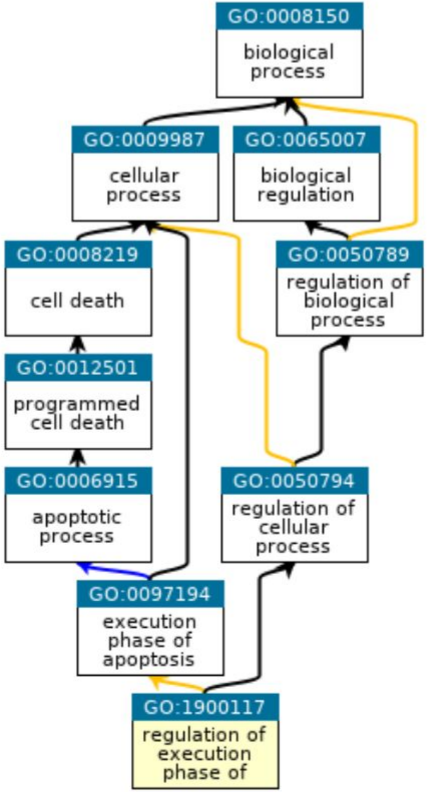


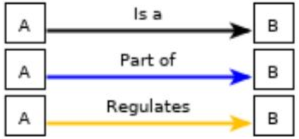


How do we capture the graph?

To capture the graph:

Traverse it using ***walks***

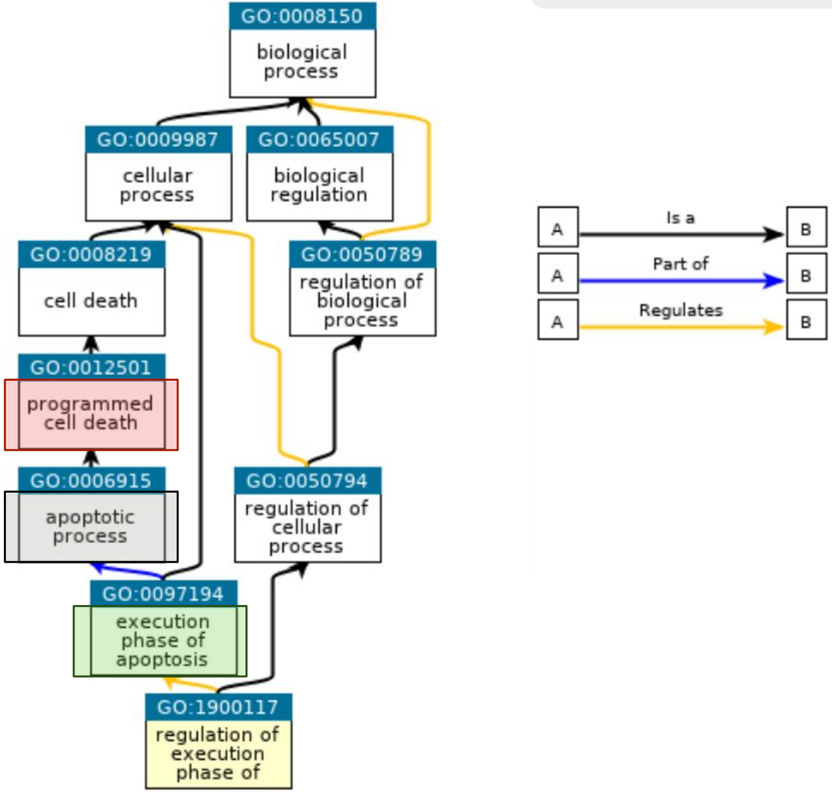




How do we capture the graph?

To capture the graph:

Traverse it using ***walks***

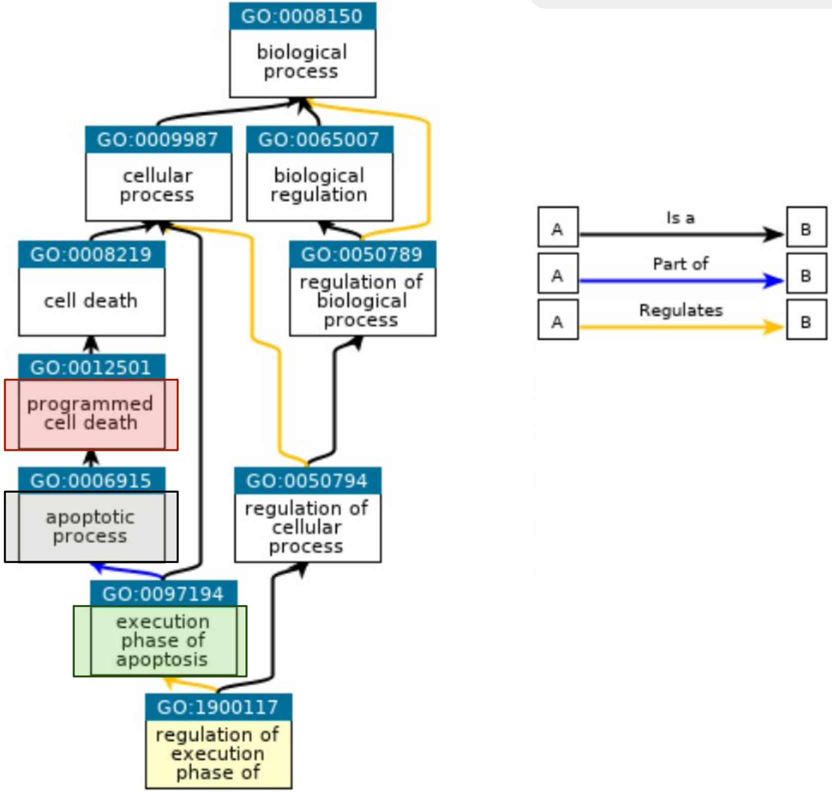


GO:1900117 **regulates** GO:0097194 **Part of** GO:0006915 **is a** GO:0012501

Why walks?

Exploring!

For a given node, walks will:



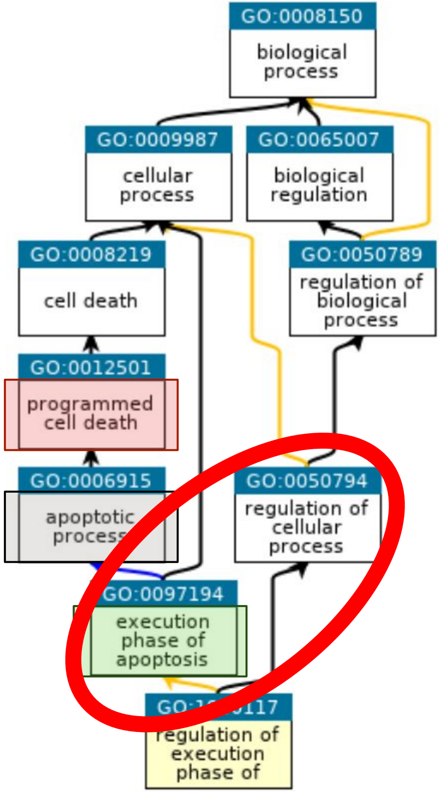
GO:1900117 **regulates** GO:0097194 **Part of** GO:0006915 **is a** GO:0012501

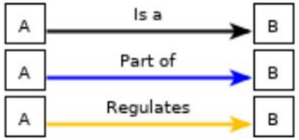
Why walks?

Exploring!

For a given node, walks will:

Capture directly adjacent nodes very well





GO:1900117 **regulates** GO:0097194 **Part of** GO:0006915 **is a** GO:0012501

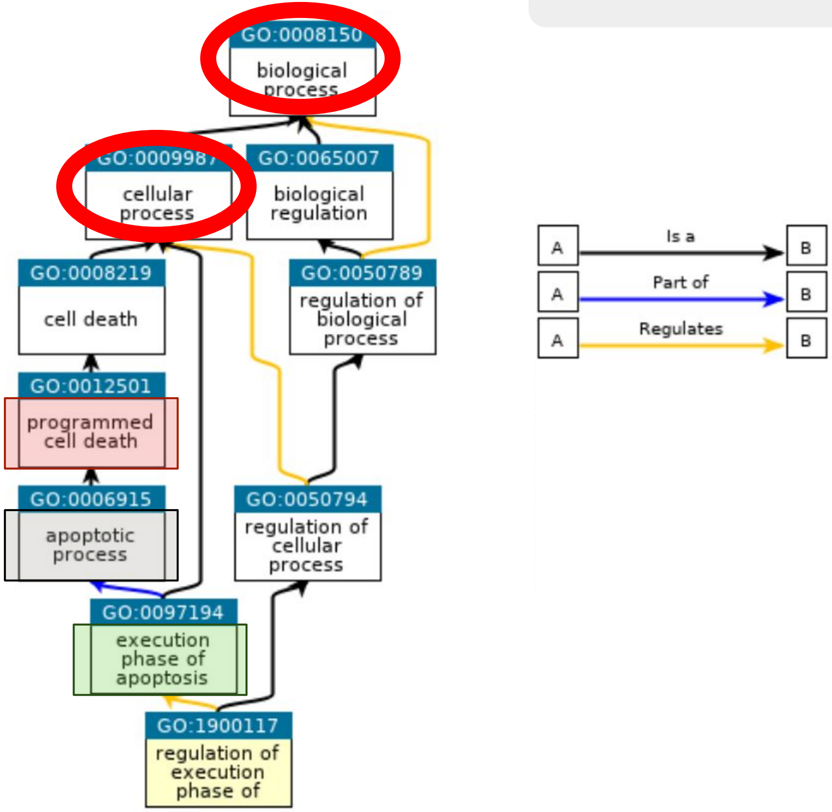
Why walks?

Exploring!

For a given node, walks will:

Capture directly adjacent nodes very well

Capture nodes with multiple paths better



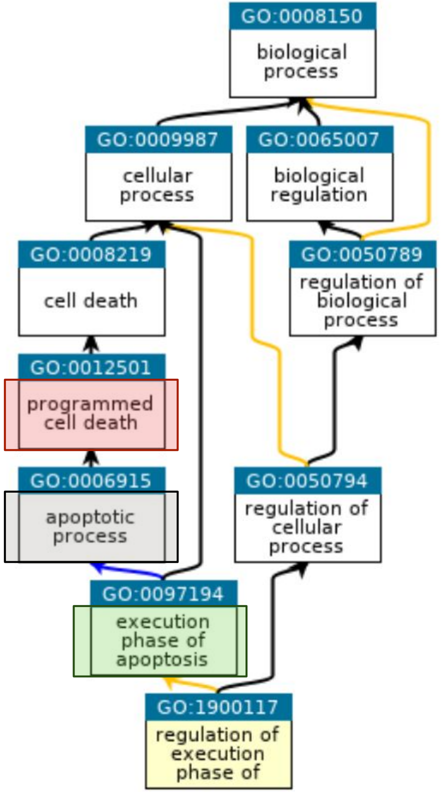
GO:1900117 **regulates** GO:0097194 **Part of** GO:0006915 **is a** GO:0012501

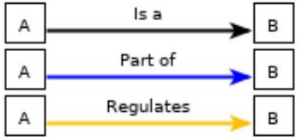
Walks

Variables to consider:

Walk length

- How long should you go?
- How many nodes and edges?





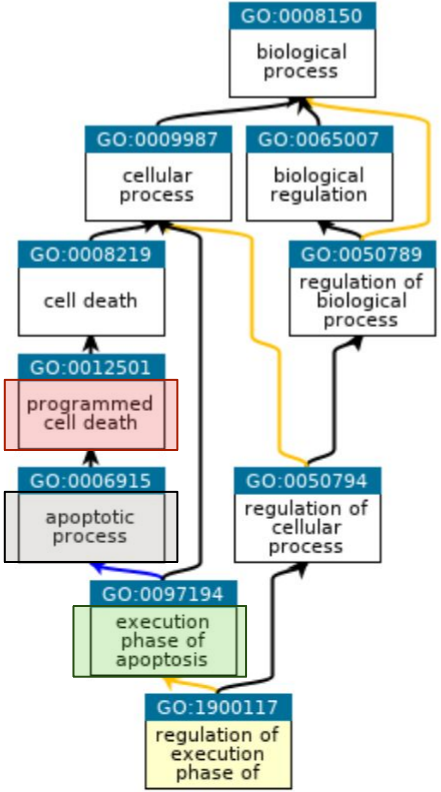
GO:1900117 **regulates** GO:0097194 **Part of** GO:0006915 **is a** GO:0012501

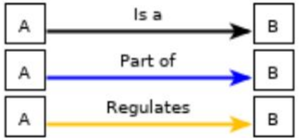
Walks

Variables to consider:

Walk direction

- Randomly (Random walk)
- With a bias (Node2Vec)





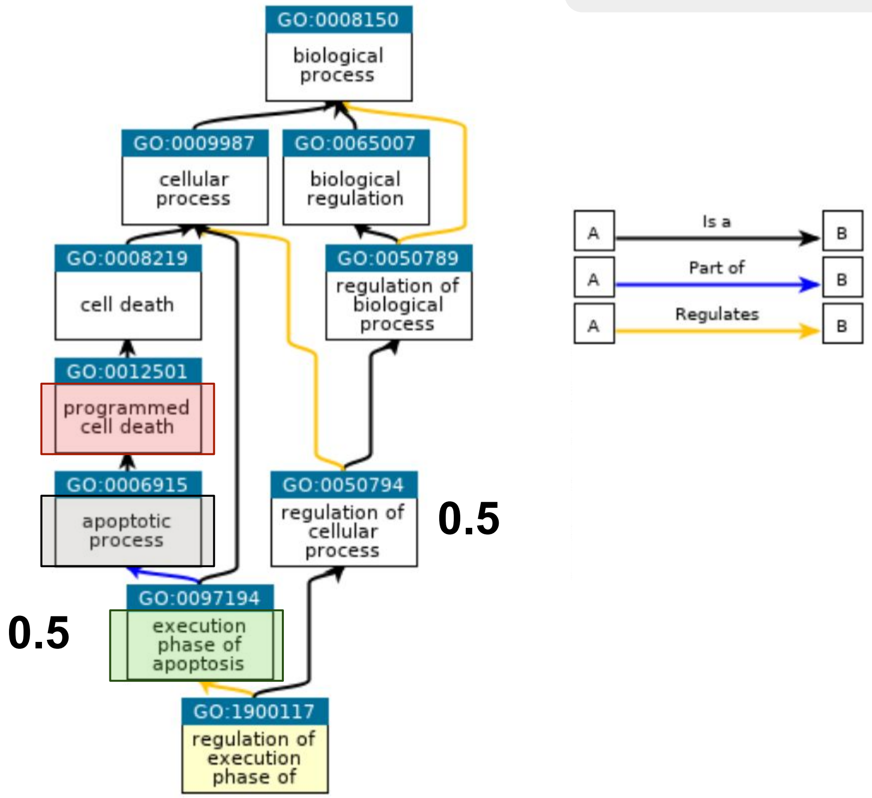
GO:1900117 **regulates** GO:0097194 **Part of** GO:0006915 **is a** GO:0012501

Walks

Variables to consider:

Walk direction

- Randomly (Random walk)
  - Candidate nodes have equal probabilities of getting chosen



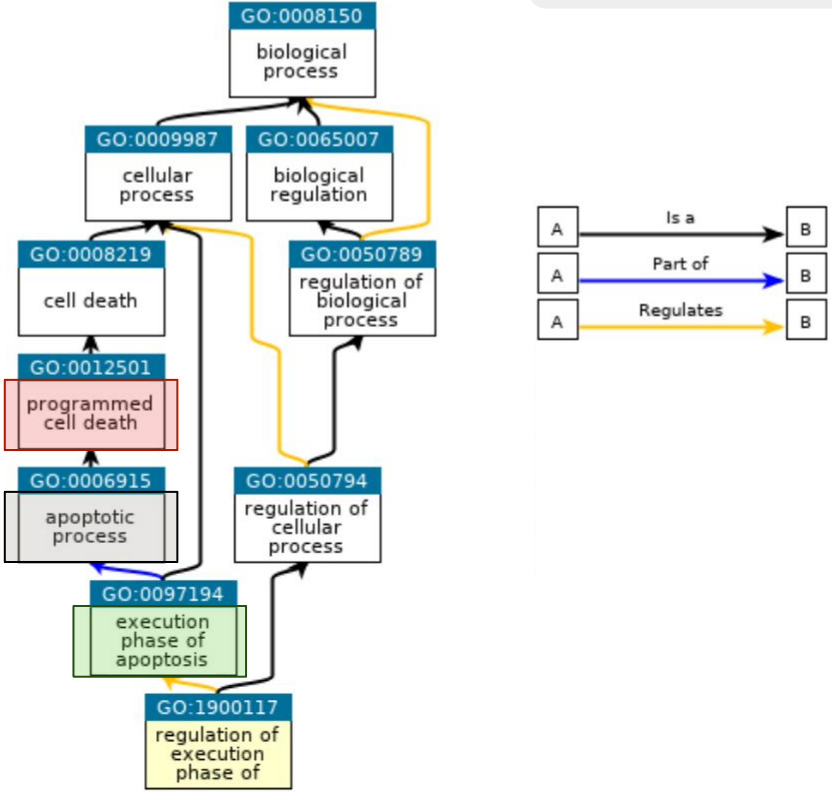
GO:1900117 **regulates** GO:0097194 **Part of** GO:0006915 **is a** GO:0012501

Walks

Variables to consider:

Walk direction

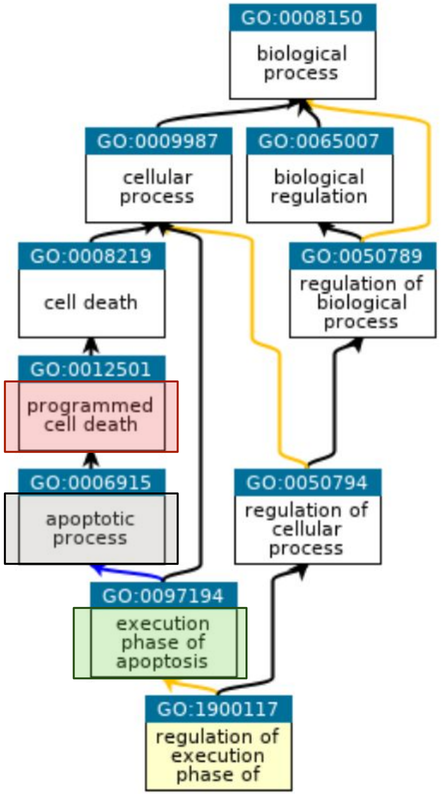
- With a bias (Node2Vec)
  - P probability to go back to previous node
  - Q probability to explore further nodes

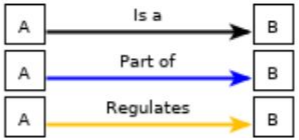


GO:1900117 **regulates** GO:0097194 **Part of** GO:0006915 **is a** GO:0012501

Walks → embeddings

How do we go from walks to embeddings?





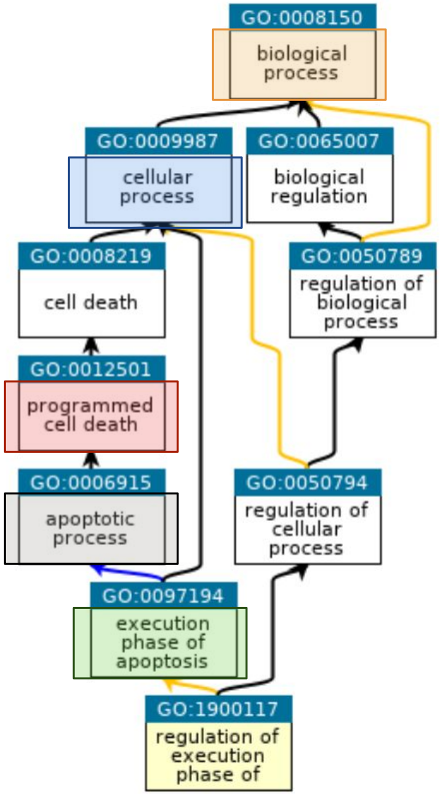
GO:1900117 **regulates** GO:0097194 **Part of** GO:0006915 **is a** GO:0012501

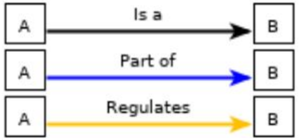
Walks → embeddings

How do we go from walks to embeddings?

- Generate many such walks

(Many walks will give us an idea about the graph adjacency and structure)





GO:1900117 **regulates** GO:0097194 **Part of** GO:0006915 **is a** GO:0012501

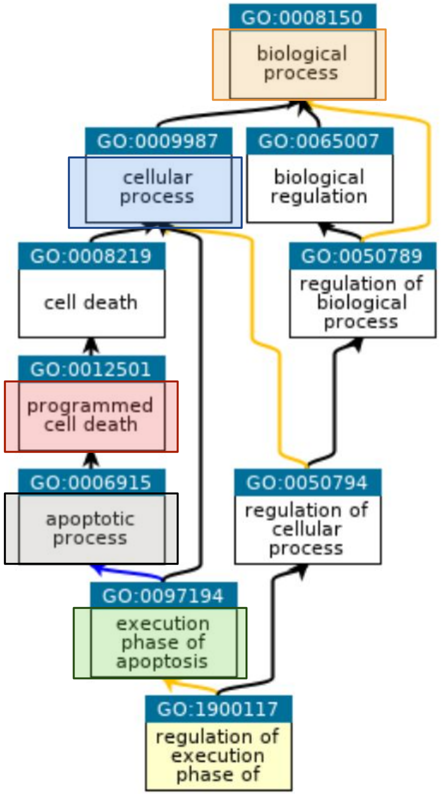
....

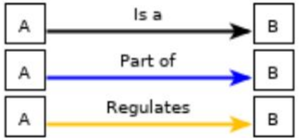
GO:1900117 **regulates** GO:0097194 **is a** GO:0009987 **is a** GO:0008150

Walks → embeddings

How do we go from walks to embeddings?

- Generate many such walks
- Each walk is a sentence
- Sentences form a corpus





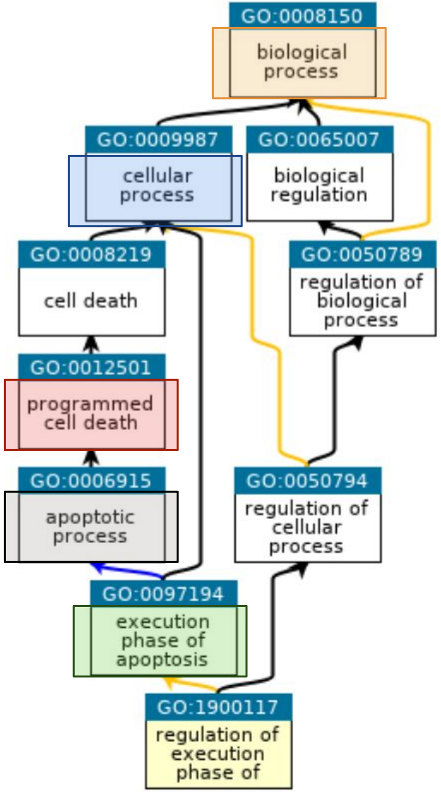
GO:1900117 **regulates** GO:0097194 **Part of** GO:0006915 **is a** GO:0012501

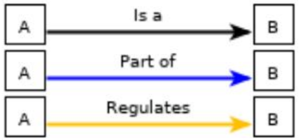
....

GO:1900117 **regulates** GO:0097194 **is a** GO:0009987 **is a** GO:0008150

Word2Vec

- Well-known method
- Generates embeddings that capture co-occurrences based on a corpus
- Embeddings are in the form of *n*-dimensional vectors





GO:1900117 **regulates** GO:0097194 **Part of** GO:0006915 **is a** GO:0012501

....

GO:1900117 **regulates** GO:0097194 **is a** GO:0009987 **is a** GO:0008150

Word2Vec

Word2Vec captures co-occurrences

Given a node:

- Capture the nodes it frequently co-occurred with in the given walks

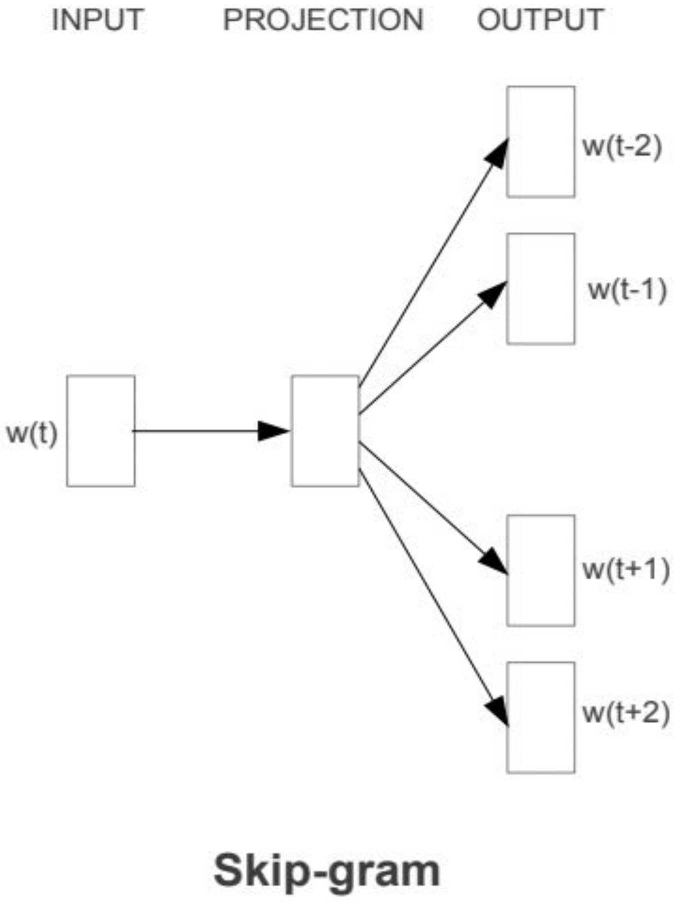


Figure from the original paper: Efficient Estimation of Word Representations in Vector Space, Mikolov et al.

Word2Vec

Word2Vec captures co-occurrences

Given a node:

- Capture the nodes it frequently co-occurred with in the given walks
- Minimize the cross-entropy loss

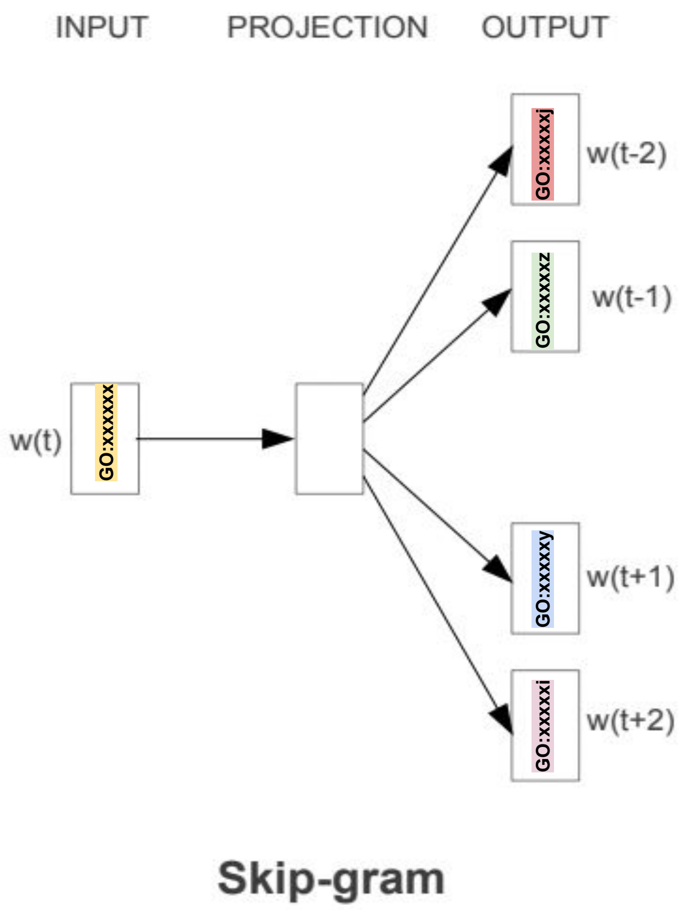


Figure from the original paper: Efficient Estimation of Word Representations in Vector Space, Mikolov et al.

Word2Vec

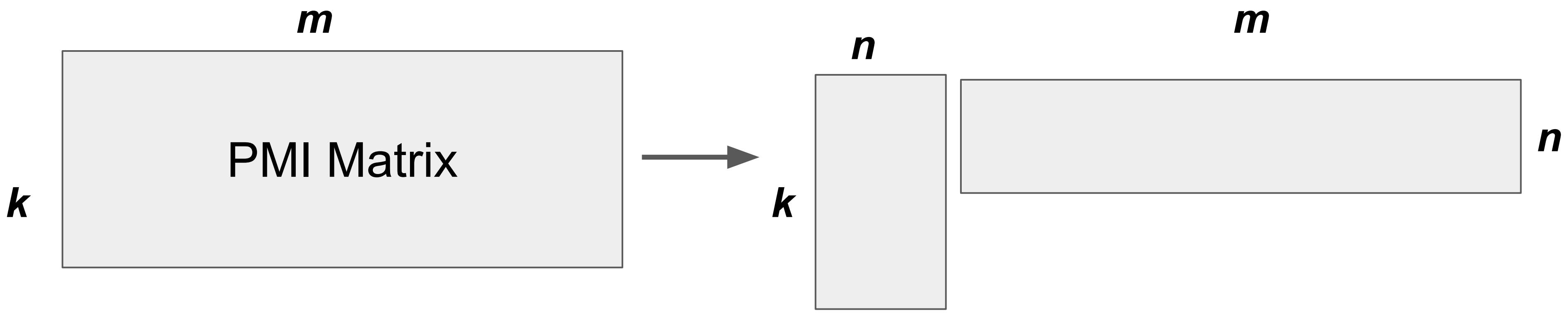
- You can this of it as a factorization of a Pointwise Mutual Information (PMI) matrix

	GO:xxxxxx	GO:xxxxxj	GO:xxxxxz	GO:xxxxxy	GO:xxxxxi
GO:xxxxxx	0	2	1	1	4
GO:xxxxxj	2	0	1	1	2
GO:xxxxxz	1	1	0	2	2
GO:xxxxxy	1	1	2	0	3
GO:xxxxxi	4	2	2	3	0

$$\operatorname{pmi}(x;y) \equiv \log \frac{p(x,y)}{p(x)p(y)}$$

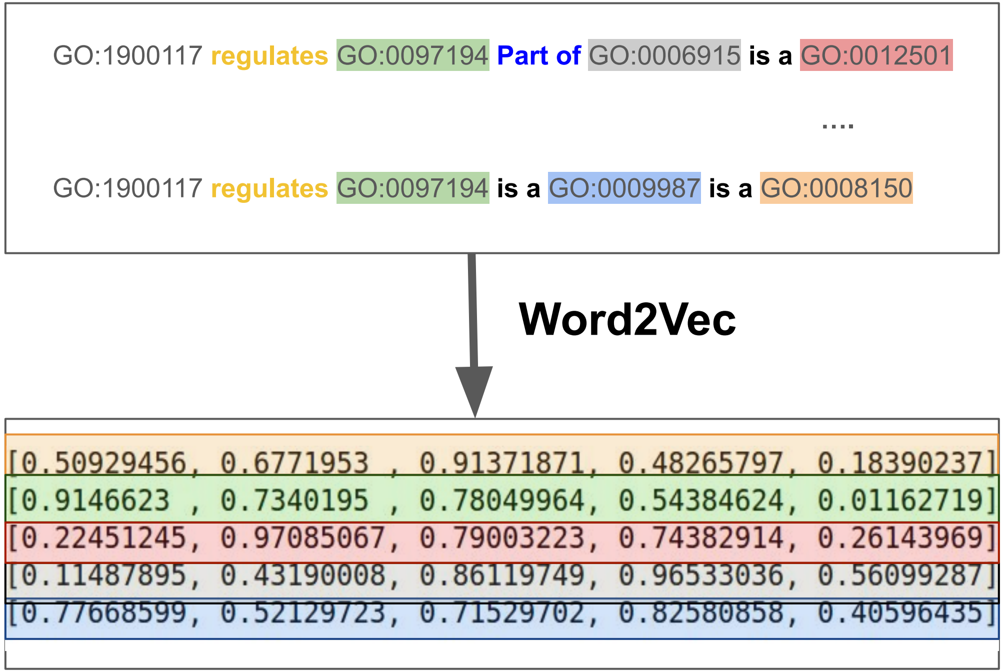
Word2Vec

You can this of it as an efficient factorization of a Pointwise Mutual Information (PMI) matrix



Neural Word Embedding as Implicit Matrix Factorization by Levy Omer and Goldberg Yoav

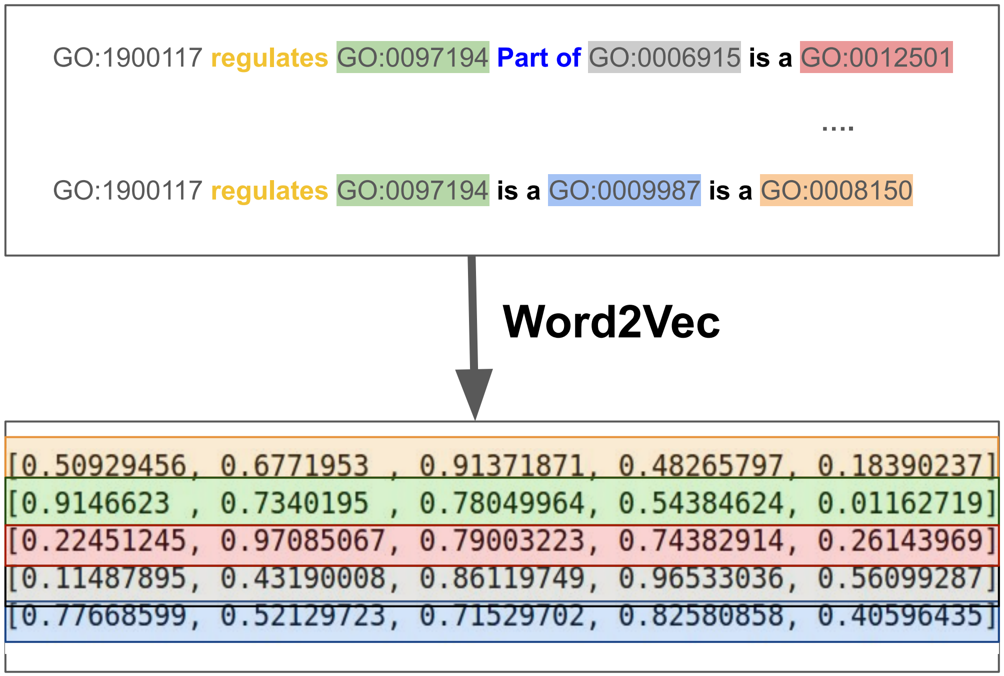
Walks → embeddings



What do these vectors capture?

Remember: Co-occurrence

- The graph structure
- Adjacency
- Hub-nodes and communities



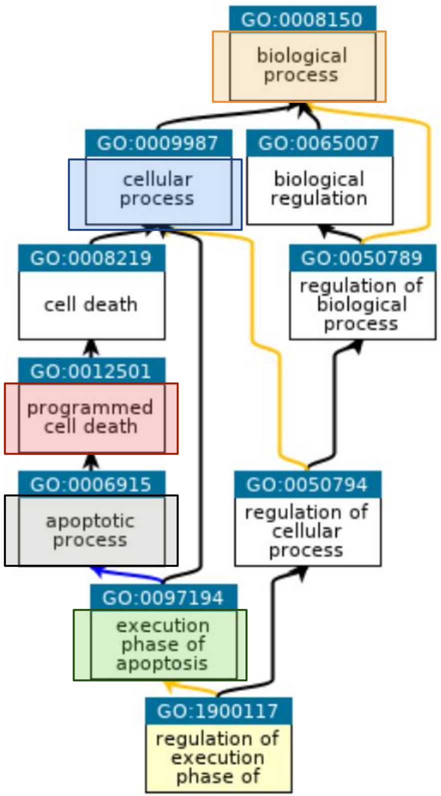
More parameters

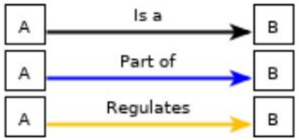
Again we are faced with more parameters:

- How many walks per node?
- Restart probability

Recall:

- Length of walk





GO:1900117 **regulates** GO:0097194 **Part of** GO:0006915 **is a** GO:0012501

....

GO:1900117 **regulates** GO:0097194 **is a** GO:0009987 **is a** GO:0008150

Ontology graph embedding methods

Many methods

- OWL2Vec
  - Recall

Axiom of condition 1	Axiom or triple(s) of condition 2	Projected triple(s)
$A \sqsubseteq \square r.D$ or $\square r.D \sqsubseteq A$	$D \equiv B \mid B_1 \sqcup ... \sqcup B_n \mid B_1 \sqcap ... \sqcap B_n$	$\langle A, r, B \rangle$ or
$\exists r.\top \sqsubseteq A$ (domain)	$\top \sqsubseteq \forall r.B$ (range)	$\langle A, r, B_i \rangle$ for $i \in 1, ..., n$
$A \sqsubseteq \exists r.\{b\}$	$B(b)$	
$r \sqsubseteq r'$	$\langle A, r', B \rangle$ has been projected	
$r' \equiv r^-$	$\langle B, r', A \rangle$ has been projected	
$s_1 \circ ... \circ s_n \sqsubseteq r$	$\langle A, s_1, C_1 \rangle ... \langle C_n, s_n, B \rangle$ have been projected	
$B \sqsubseteq A$	–	$\langle B, rdfs:subClassOf, A \rangle$ $\langle A, rdfs:subClassOf^-, B \rangle$
$A(a)$	–	$\langle a, rdf:type, A \rangle$ $\langle A, rdf:type^-, a \rangle$
$r(a, b)$	–	$\langle a, r, b \rangle$

Ontology graph embedding methods

Many methods

- DL2Vec

Condition 1	Condition 2	Triple(s)
$A \sqsubseteq QR_0 \ldots QR_m D$ $A \equiv QR_0 \ldots QR_m D$	$D := B_1 \sqcup \ldots \sqcup B_n \mid B_1 \sqcap \ldots \sqcap B_n$	$\langle A, (R_0 \ldots R_m), B_i \rangle$ for $i \in 1 \ldots n$
$A \sqsubseteq B$		$\langle A, SubClassOf, B \rangle$
$A \equiv B$		$\langle A, EquivalentTo, B \rangle$

Ontology graph embedding methods