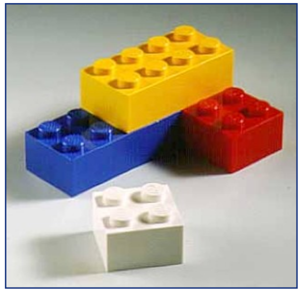
GNN Applications: Graph Mining

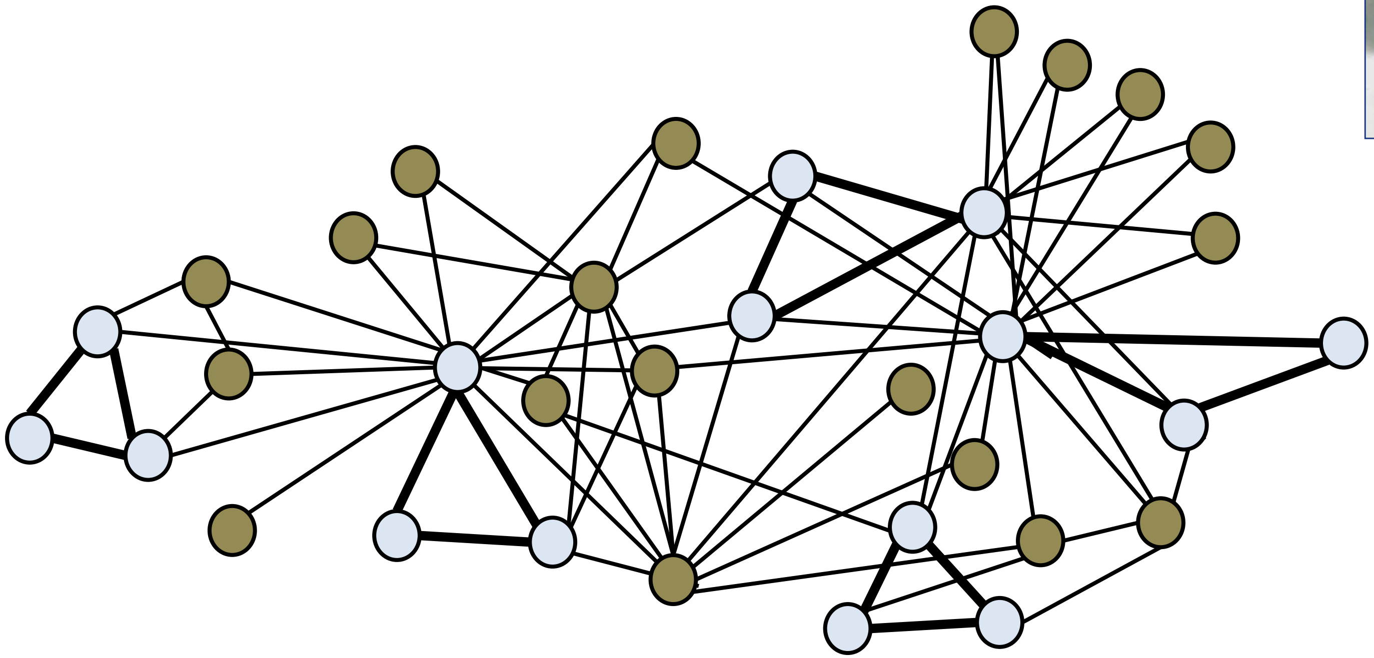
Jiaxuan You
Assistant Professor at UIUC CDS

CS598: Deep Learning with Graphs, 2024 Fall
https://ulab-uiuc.github.io/CS598/

Subgraphs

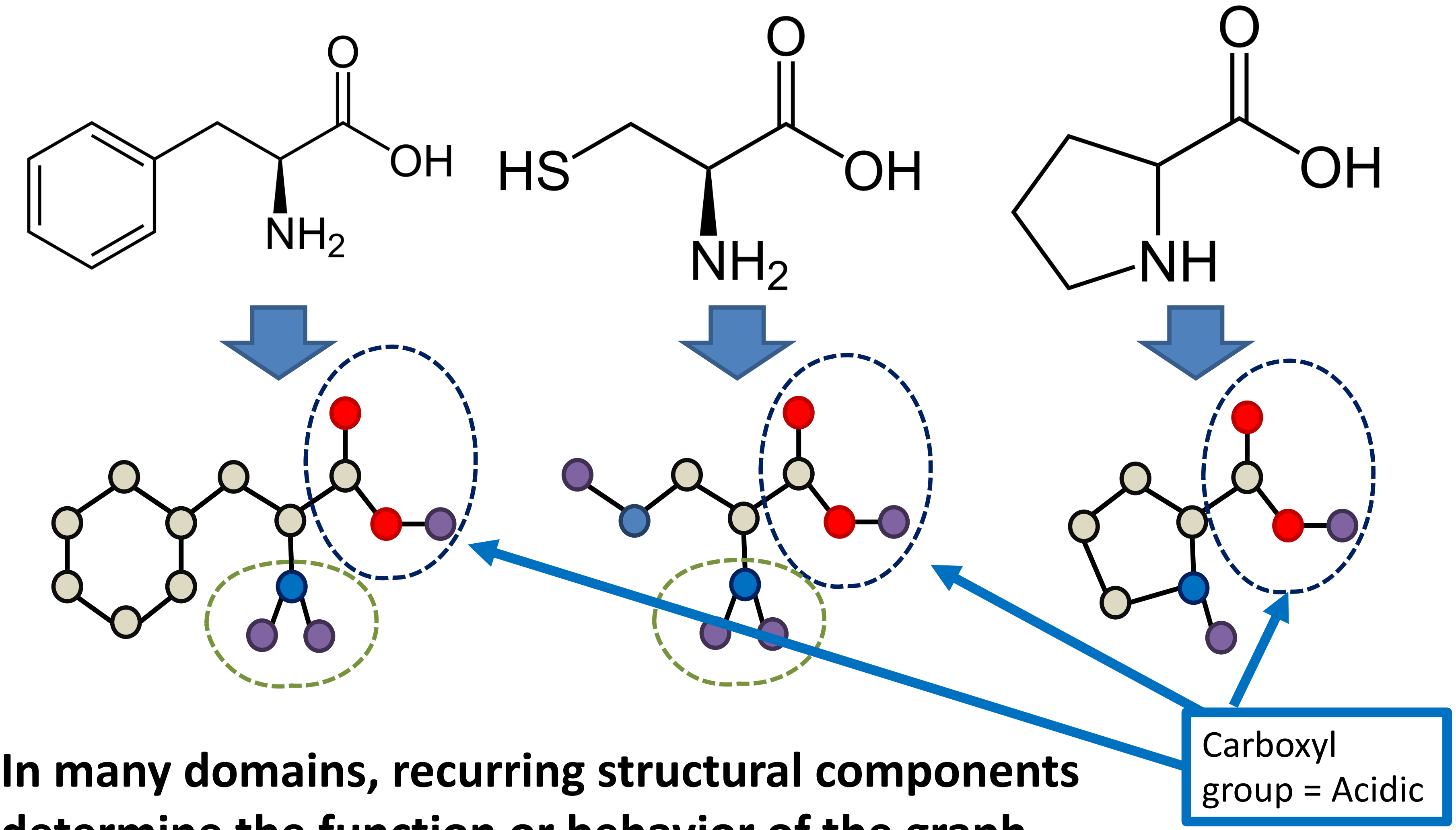
- **Subgraphs** are the building blocks of networks:





- They have the power to **characterize** and **discriminate** networks

Building Blocks of Networks



In many domains, recurring structural components determine the function or behavior of the graph

Plan for Today

1) Subgraphs and motifs

- Defining Subgraphs and Motifs
- Determining Motif Significance

2) Neural Subgraph Representations

3) Mining Frequent Motifs

GNN Applications: Graph Mining

Subgraphs and Motifs

Definition: Subgraph (1)

Two ways to formalize "network building blocks"

- Given graph $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$:

Def 1. Node-induced subgraph: Take subset of the **nodes** and all edges induced by the nodes:

- $G' = (V', E')$ is a node induced subgraph iff
  - $V' \subseteq V$
  - $E' = \{(u, v) \in E \mid u, v \in V'\}$
  - G' is the subgraph of G **induced** by V'

- **Alternate terminology:** "induced subgraph"

Definition: Subgraph (2)

Two ways to formalize "network building blocks"

- Given graph $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$:

Def 2. Edge-induced subgraph: Take subset of the edges and all corresponding nodes

- $G' = (V', E')$ is an edge induced subgraph iff
  - $E' \subseteq E$
  - $V' = \{v \in V \mid (v, u) \in E' \text{ for some } u\}$

- **Alternate terminology:** "non-induced subgraph" or just "subgraph"

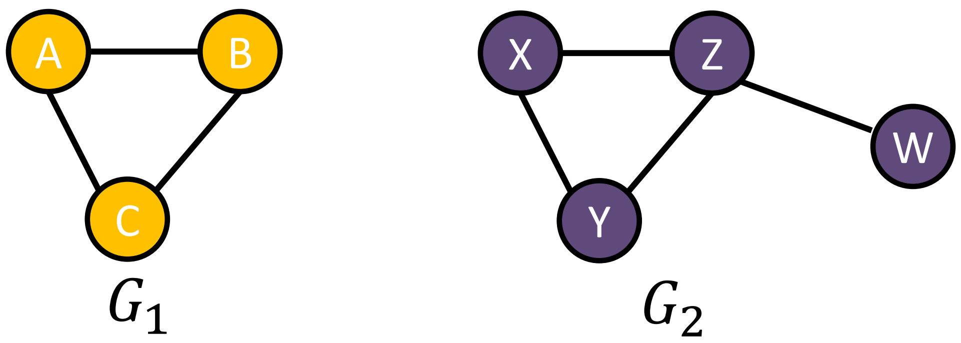
Definition: Subgraph (3)

Two ways to formalize "network building blocks"

- The best definition depends on the domain! **Examples:**
  - **Chemistry**: Node-induced (functional groups)
  - **Knowledge graphs**: Often edge-induced (focus is on edges representing logical relations)

Definition: Subgraph (4)

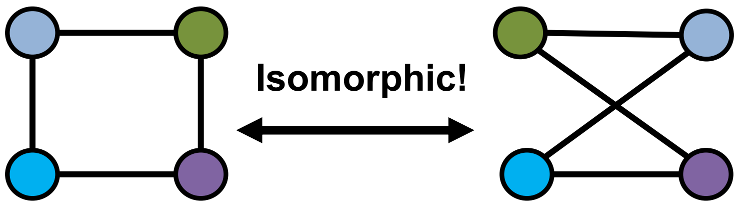
- The preceding definitions define subgraphs when $V' \subseteq V$ and $E' \subseteq E$, i.e. nodes and edges are taken from the original graph G.
- **What if V' and E' come from a totally different graph?** Example:

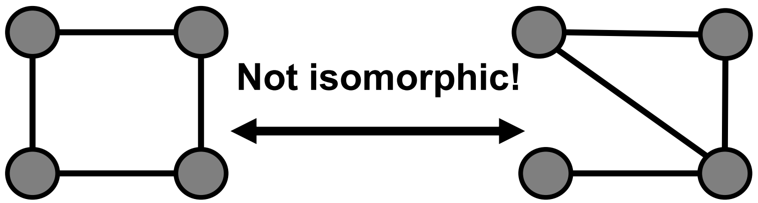


- **We would like to say that G_1 is "contained in" G_2**

Graph Isomorphism

- **Graph isomorphism problem:** Check whether two graphs are identical:
  - $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are **isomorphic** if there exists a bijection $f: V_1 \to V_2$ such that $(u, v) \in E_1$ iff $\big(f(u), f(v)\big) \in E_2$
    - f is called the **isomorphism:**

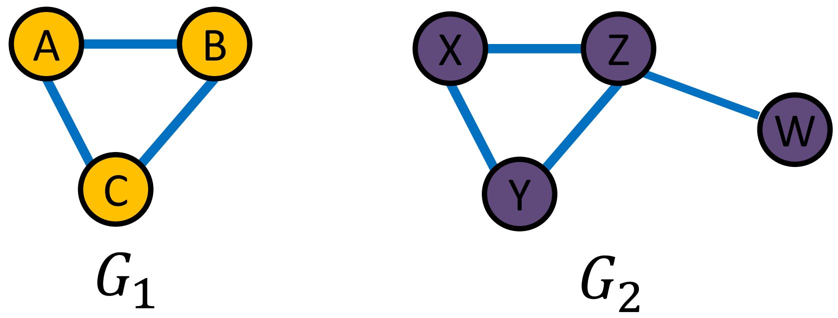




- We do not know if graph isomorphism is NP-hard, nor is any polynomial algorithm found for solving graph isomorphism.

Subgraph Isomorphism

- G_2 is **subgraph-isomorphic** to G_1 if some subgraph of G_2 is isomorphic to G_1
  - We also commonly say G_1 is a subgraph of G_2
  - We can use either the node-induced or edge-induced definition of subgraph
  - **This problem is NP-hard**



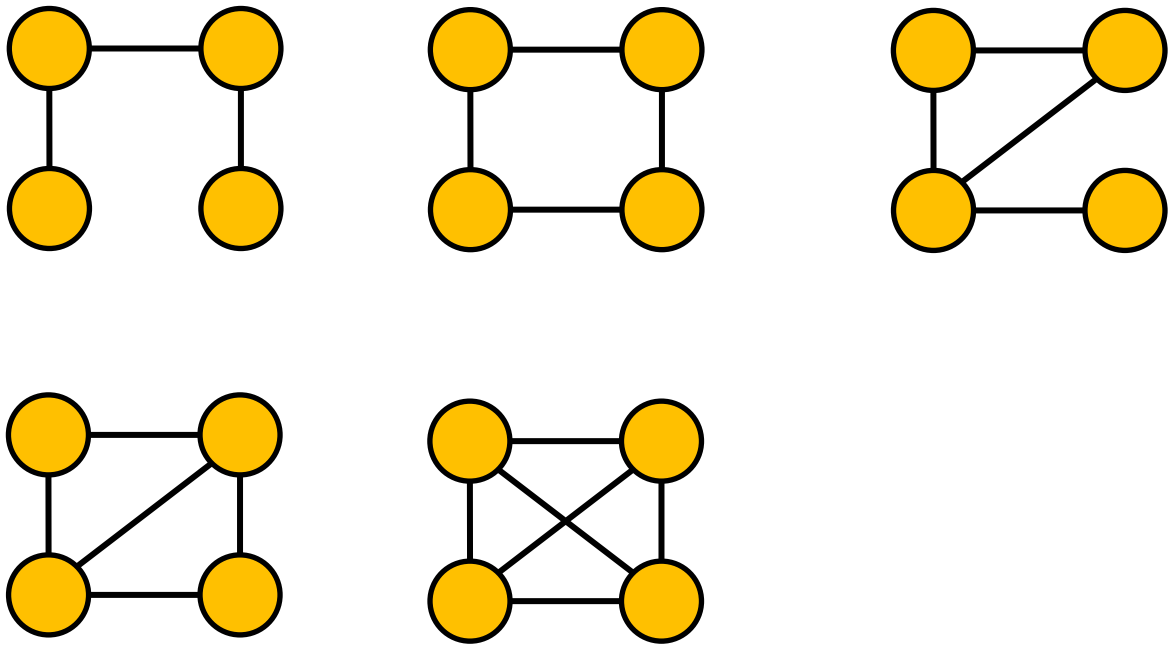
f:

V_1	V_2
A	X
B	Y
C	Z

A-B-C matches with X-Y-Z: There is a subgraph isomorphism between G1 and G2.

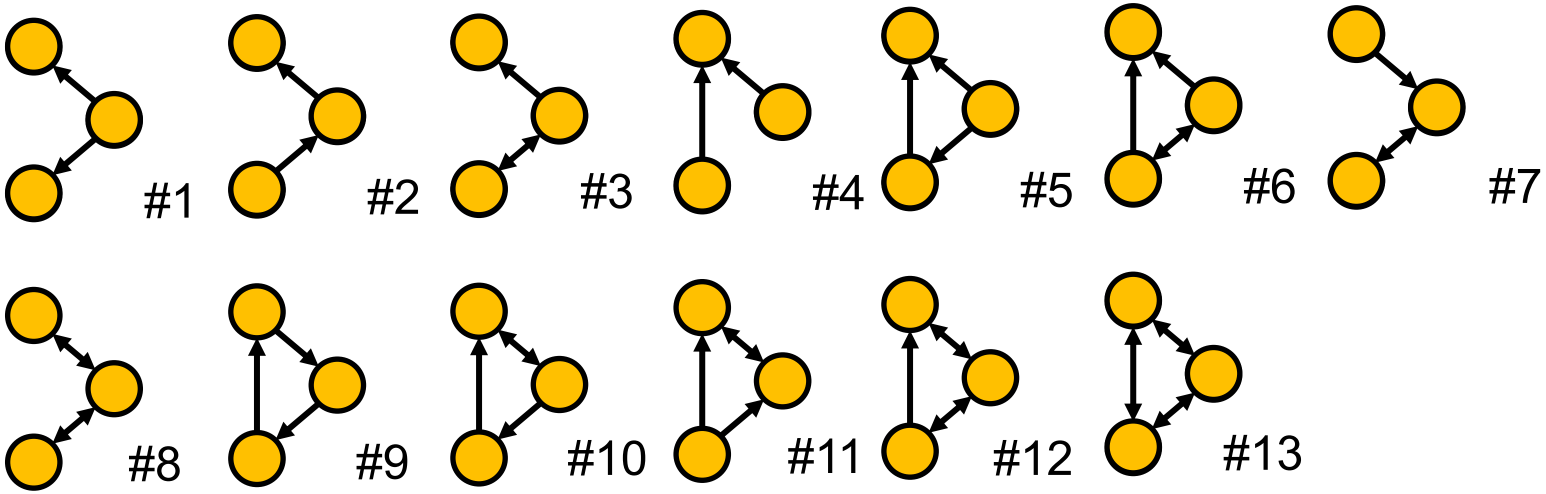
Case Example of Subgraphs (1)

- **All non-isomorphic, connected, undirected graphs of size 4**



Case Example of Subgraphs (2)

- **All non-isomorphic, connected, directed graphs of size 3**



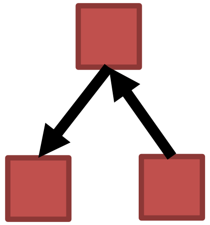
Network Motifs

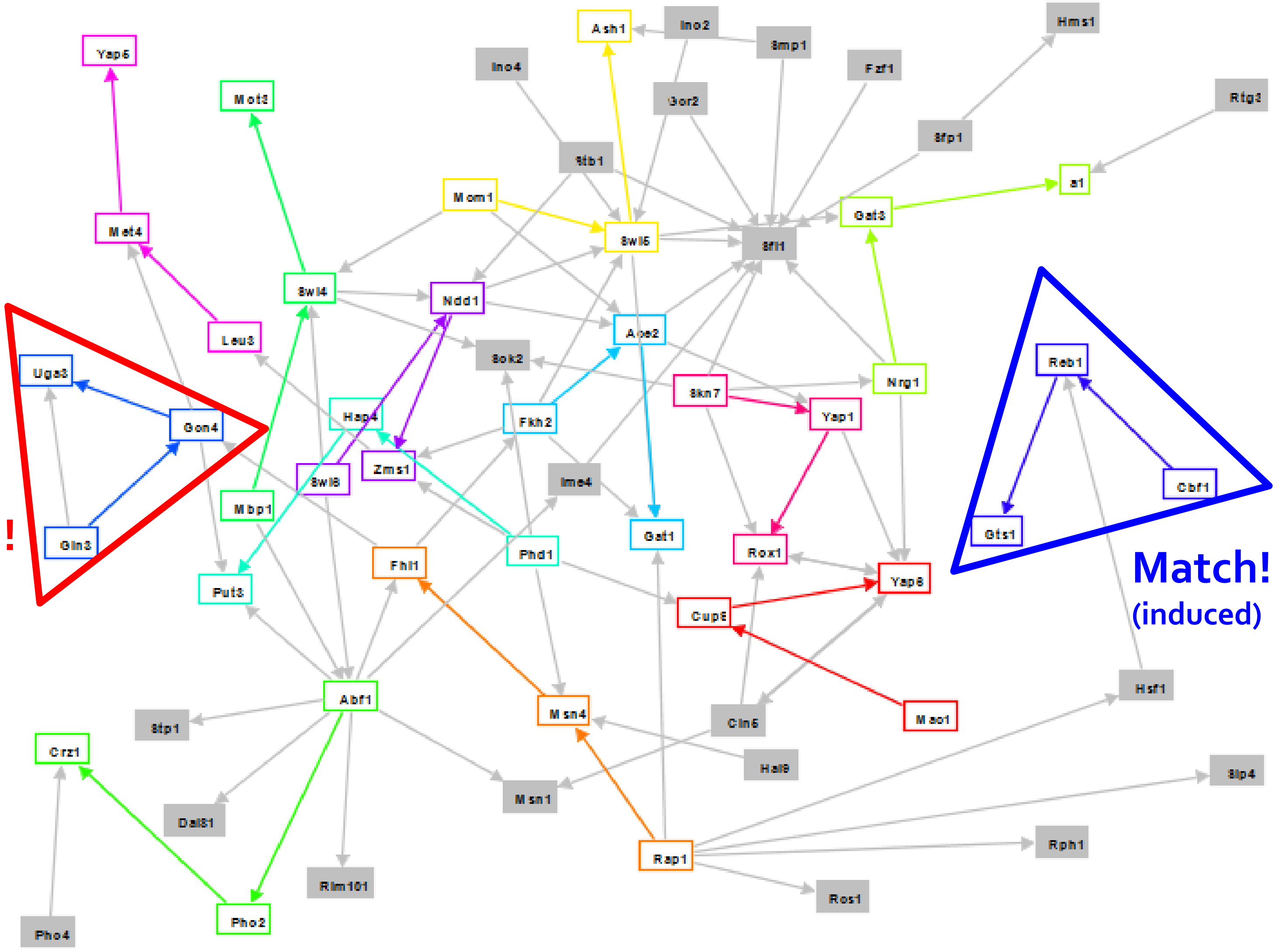
- **Network motifs:** “recurring, significant patterns of interconnections”

- **How to define a network motif:**
  - **Pattern:** Small (node-induced) subgraph
  - **Recurring:** Found many times, i.e., with high frequency **How to define frequency?**
  - **Significant:** More frequent than expected, i.e., in randomly generated graphs? **How to define random graphs?**

Motifs: Induced Subgraphs

Induced subgraph of interest (aka Motif):



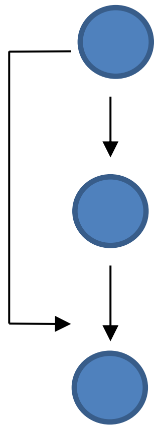


No match!
(not induced)

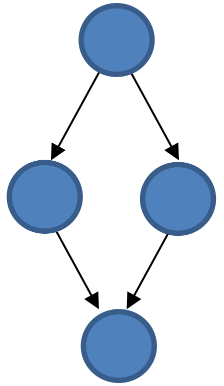
Match!
(induced)

Why Do We Need Motifs?

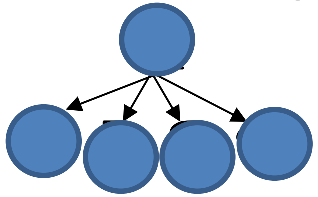
- **Motifs:**
  - Help us understand how graphs work
  - Help us make predictions based on presence or lack of presence in a graph dataset
- **Examples:**
  - **Feed-forward loops:** Found in networks of neurons, where they neutralize “biological noise”
  - **Parallel loops:** Found in food webs
  - **Single-input modules:** Found in gene control networks



Feed-forward loop



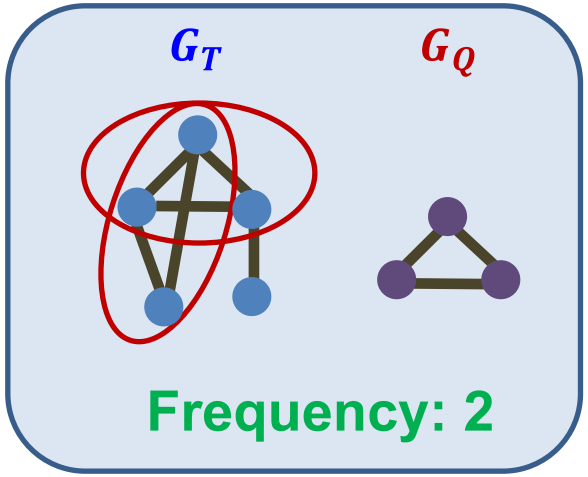
Parallel loop

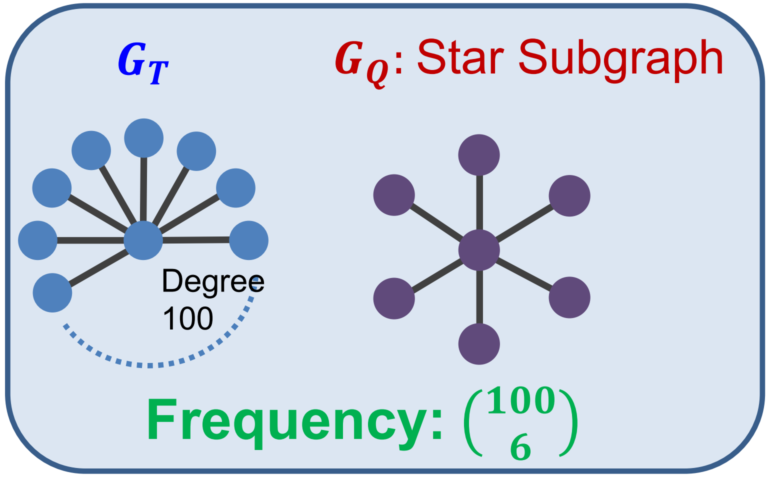


Single-input module

Subgraph Frequency (1)

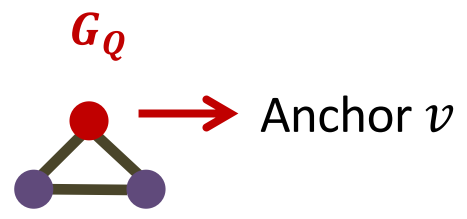
- Let G_Q be a small graph and G_T be a target graph dataset.
- **Graph-level Subgraph Frequency Definition**
  Frequency of G_Q in G_T: number of unique subsets of nodes V_T of G_T for which the subgraph of G_T induced by the nodes V_T is isomorphic to G_Q

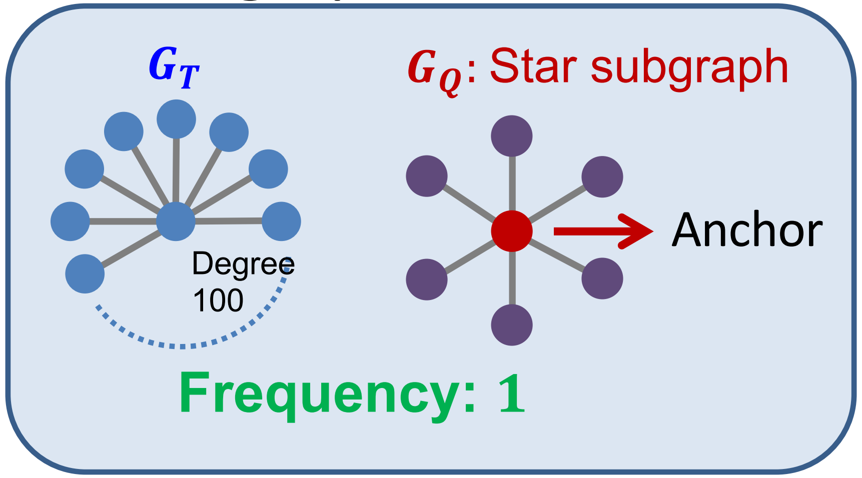




Subgraph Frequency (2)

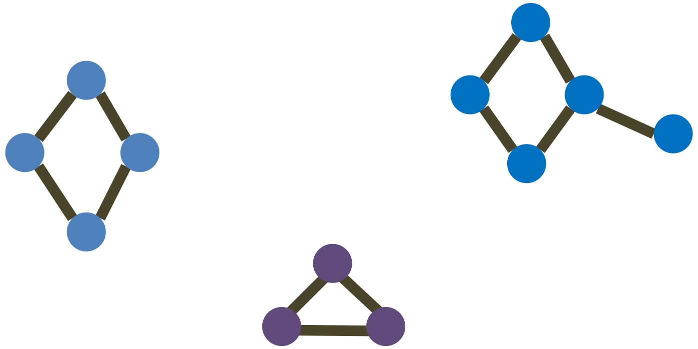
- Let G_Q be a small graph, v be a node in G_Q (the "anchor") and G_T be a target graph dataset.
- **Node-level Subgraph Frequency Definition:**
  The number of nodes u in G_T for which some subgraph of G_T is isomorphic to G_Q and the isomorphism maps node u to v
- Let (G_Q, v) be called a **node-anchored** subgraph
- Robust to outliers





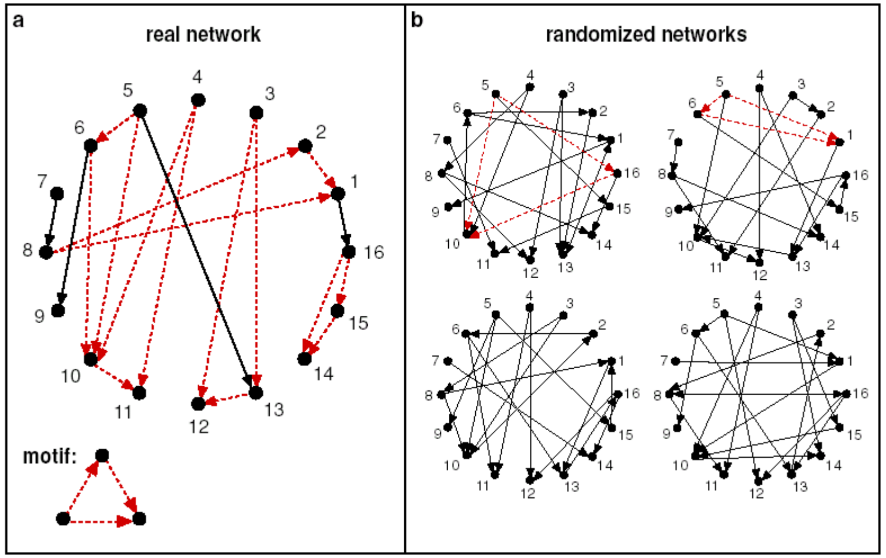
Subgraph Frequency (3)

- **What if the dataset contains multiple graphs, and we want to compute frequency of subgraphs in the dataset?**
- **Solution:** Treat the dataset as a giant graph G_T with disconnected components corresponding to individual graphs.



Defining Motif Significance

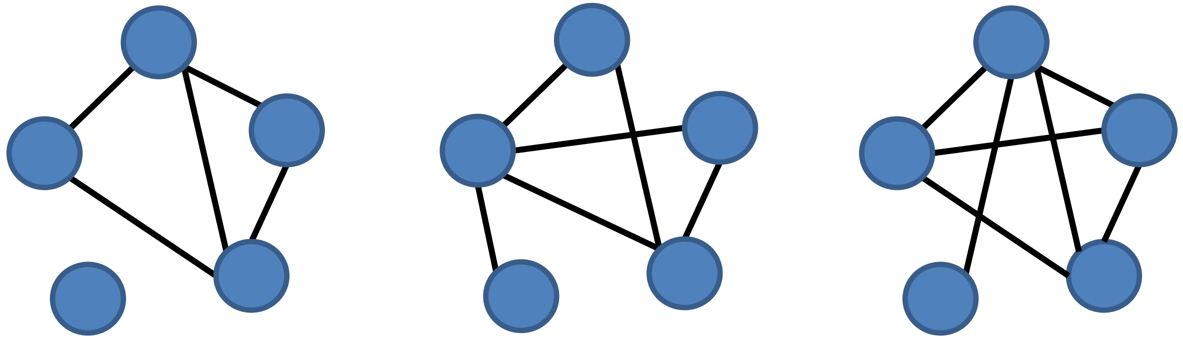
- **To define significance, we need to have a null-model (i.e., point of comparison).**
- **Key idea: Subgraphs that occur in a real network much more often than in a random network have functional significance.**



Defining Random Graphs

Erdős–Rényi (ER) random graphs:

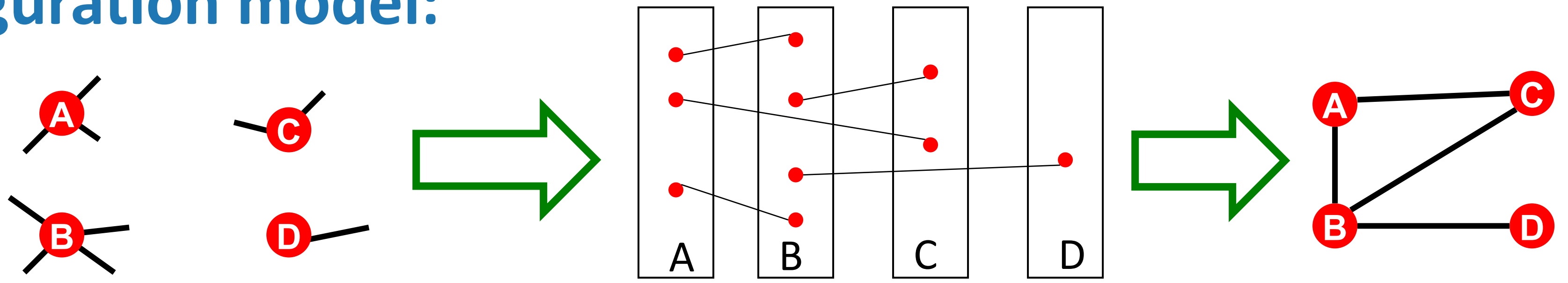
- $G_{n,p}$: undirected graph on n nodes where each edge (u, v) appears i.i.d. with probability p
  - **How to generate the graph:** Create n nodes, for each pair of nodes (u, v) flip a biased coin with bias p
- Generated graph is a result of a random process:



Three random graphs drawn from $G_{5,0.6}$

New Model: Configuration Model

- **Goal:** Generate a random graph with a given **degree sequence** $k_1, k_2, \ldots k_N$
- **Useful as a "null" model of networks:**
  - We can compare the real network G^{real} and a "random" G^{rand} which has the same degree sequence as G^{real}
- **Configuration model:**



Nodes with spokes

Randomly pair up "mini"-n0des

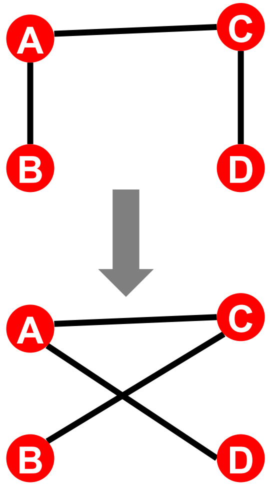
Resulting graph

We ignore double edges and self-loops when creating the final graph

Alternative for Spokes: Switching

- Start from a **given graph** G
- Repeat **the switching step** $Q \cdot |E|$ times:
  - Select a pair of edges A→B, C→D at random
  - **Exchange** the endpoints to give A→D, C→B
    - Exchange edges only if no multiple edges or self-edges are generated
- **Result:** A randomly rewired graph:
  - Same node degrees, randomly rewired edges
- Q is chosen large enough *(e.g.,* $Q = 100$*)* for the process to converge

Q is a constant parameter



Motif Significance Overview

- **Intuition:** Motifs are **overrepresented** in a network when compared to **random graphs:**
- **Step 1: Count motifs** in the given graph (G^{real})
- **Step 2: Generate random graphs** with similar statistics (e.g. number of nodes, edges, degree sequence), and count motifs in the random graphs
- **Step 3: Use statistical measures** to evaluate how significant is each motif
  - Use **Z-score**

Z-score for Statistical Significance

- Z_i captures **statistical significance of motif i**

$$Z_i = (N_i^{\text{real}} - \bar{N}_i^{\text{rand}})/\text{std}(N_i^{\text{rand}})$$

  - N_i^{real} is #(motif i) in graph G^{real}
  - $\bar{N}_i^{\text{rand}}$ is average #(motifs i) in random graph instances
- **Network significance profile (SP):**

$$SP_i = Z_i / \sqrt{\sum_j Z_j^2}$$

  - SP is a vector of **normalized Z-scores**
  - The dimension depends on number of motifs considered
  - SP emphasizes relative significance of subgraphs:
    - Important for comparison of networks of different sizes
    - Generally, larger graphs display higher Z-scores

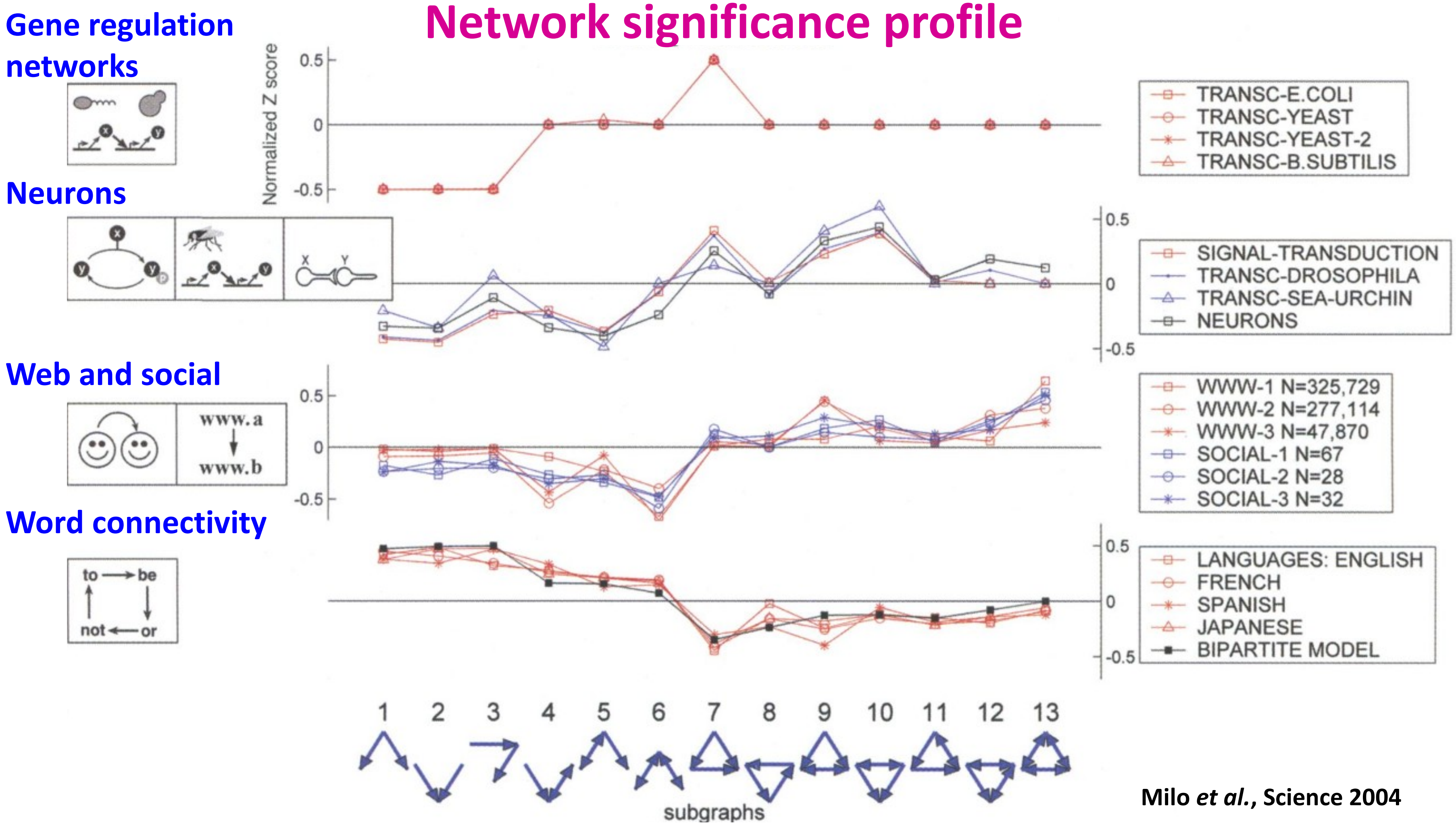
Significance Profile

- **For each subgraph:**
  - z-score metric is capable of classifying the subgraph “significance”:
    - Negative values indicate **under-representation**
    - Positive values indicate **over-representation**

- We create a **network significance profile:**
  - A feature vector with values for all subgraph types

- **Next:** Compare profiles of different graphs with **random graphs**:
    - Regulatory network (gene regulation)
    - Neuronal network (synaptic connections)
    - World Wide Web (hyperlinks between pages)
    - Social network (friendships)
    - Language networks (word adjacency)

Example Significance Profile



Milo *et al.*, Science 2004

Networks from the same domain have similar significance profiles

Summary: Detecting Motifs

- Count subgraphs i in G^{real}
- Count subgraphs i in random graphs G^{rand}:
  - **Null model:** Each G^{rand} has the same #(nodes), #(edges) and degree distribution as G^{real}

- Assign **Z-score** to motif i:
  - $Z_i = (N_i^{\text{real}} - \bar{N}_i^{\text{rand}})/\text{std}(N_i^{\text{rand}})$
  - **High Z-score:** Subgraph i is a **network motif of G**

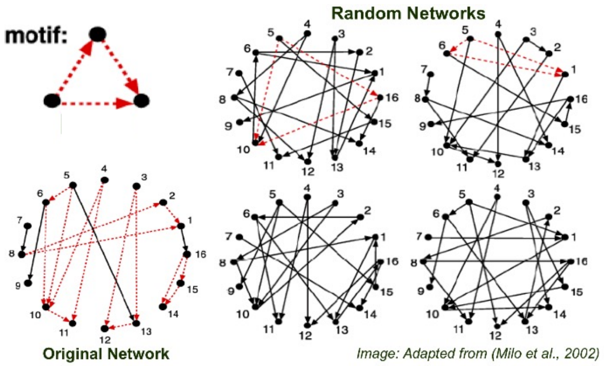
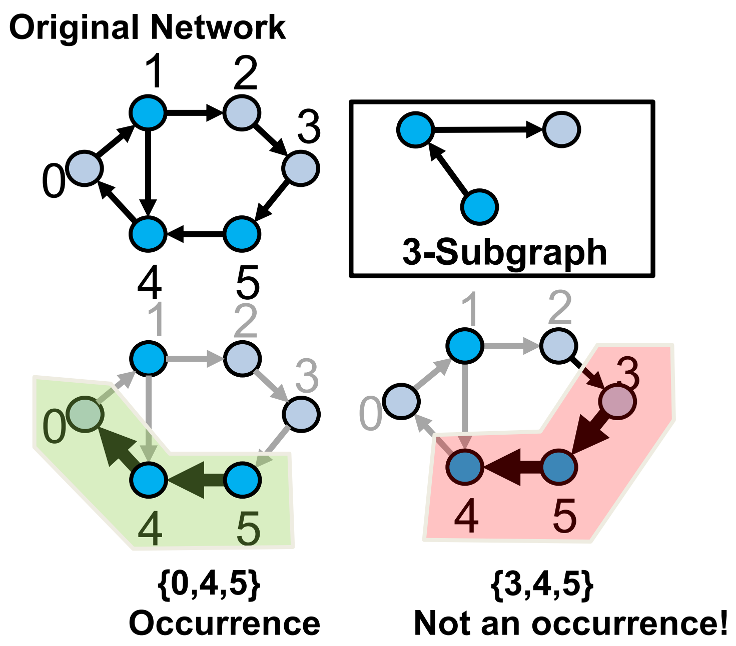


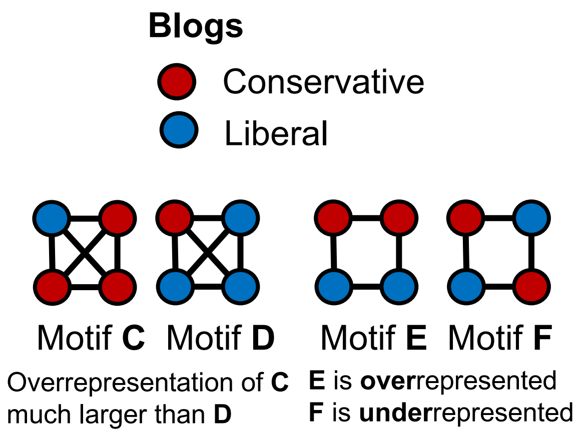
Image: Adapted from (Milo et al., 2002)

Variations on the Motif Concept

- **Extensions:**
  - Directed and undirected
  - Colored and uncolored
  - Temporal and static motifs

- **Variations on the concept:**
  - Different frequency concepts
  - Different significance metrics
  - Under-Representation (**anti-motifs**)
  - Different null models





Summary: Motifs

- Subgraphs and motifs are the **building blocks** of graphs
  - Subgraph isomorphism and counting are NP-hard
- Understanding which motifs are frequent or significant in a dataset gives insight into the unique characteristics of that domain
- Use **random graphs** as null model to evaluate the significance of motif via **Z-score**

GNN Applications: Graph Mining

Neural Subgraph Matching

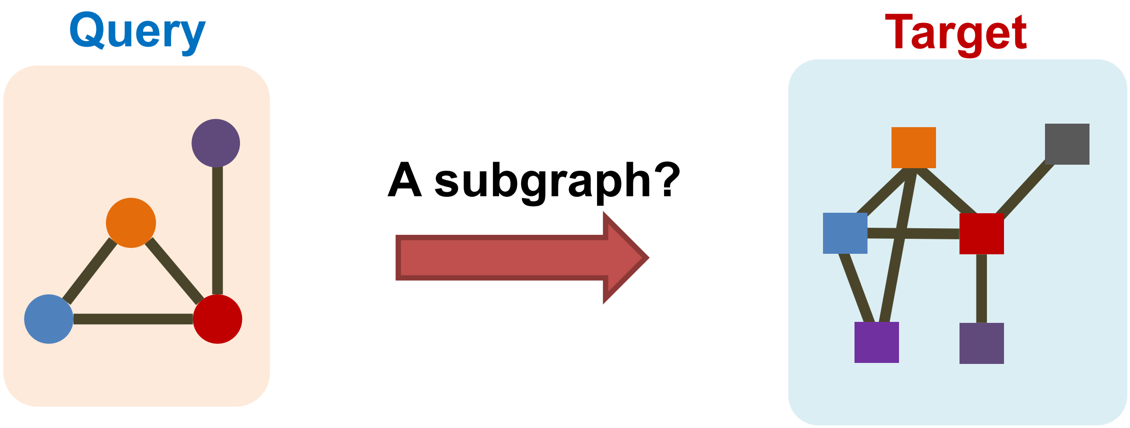
Subgraph Matching

Given:

- Large target graph (can be disconnected)
- Query graph (connected)

Decide:

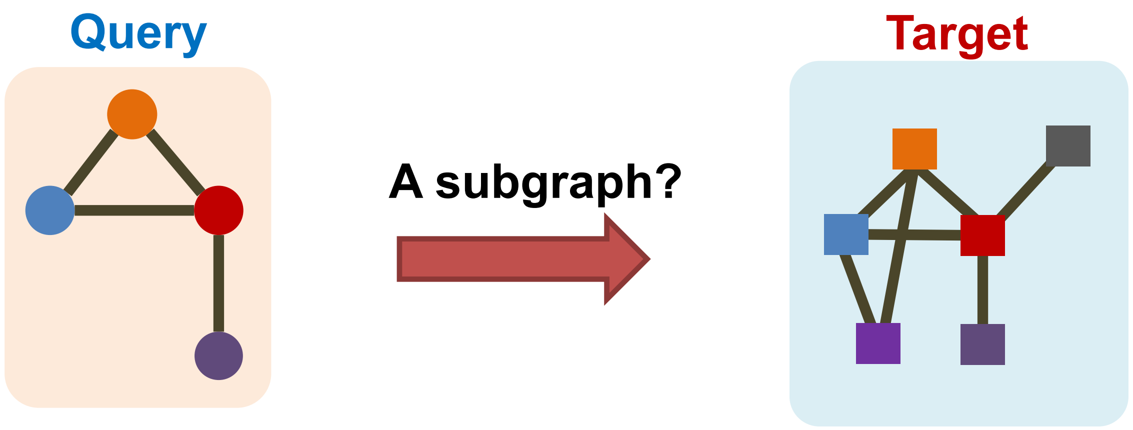
- Is a query graph a subgraph in the target graph?



- Node colors indicate the correct mapping of the nodes

Isomorphism as an ML Task

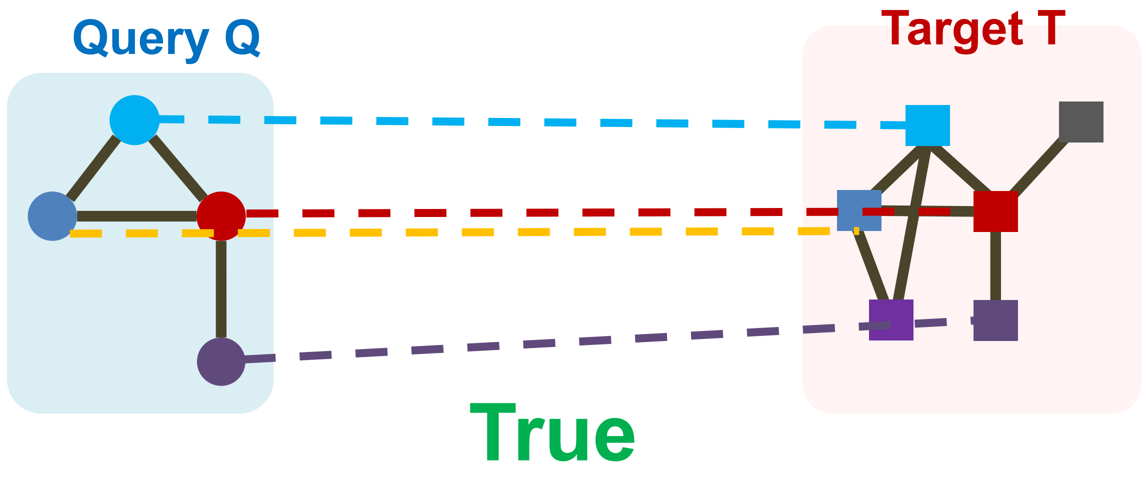
- Large target graph (can be disconnected)
- Query graph (has to be connected)
- **Use GNN to predict subgraph isomorphism:**



- **Intuition:** Exploit the **geometric shape** of **embedding space** to capture the properties of subgraph isomorphism

Task Setup

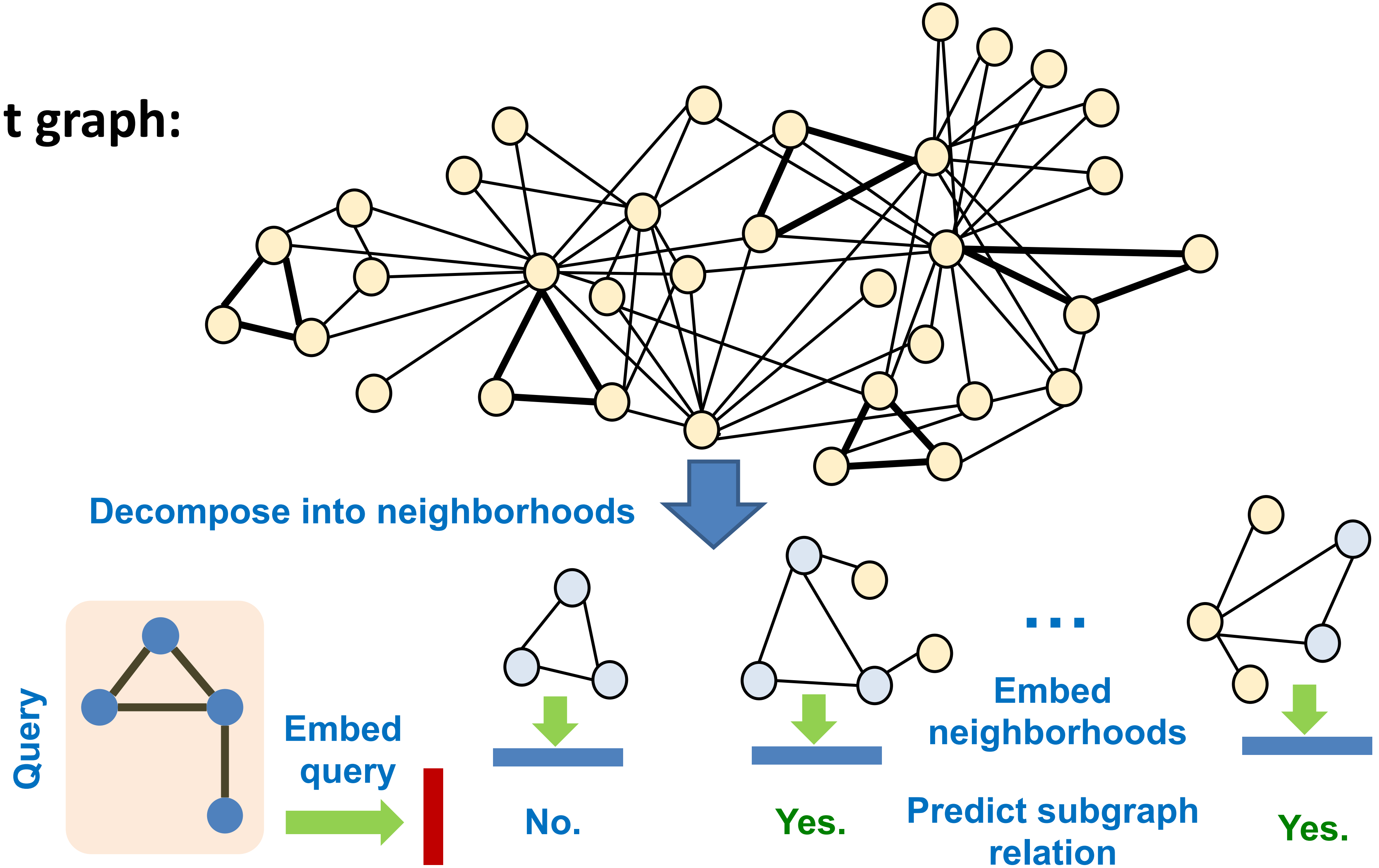
- **Consider a binary prediction:** Return **True** if query is isomorphic to a subgraph of the target graph, else return False



Finding node correspondences between Q and T is another challenging problem, which will not be covered in this lecture.

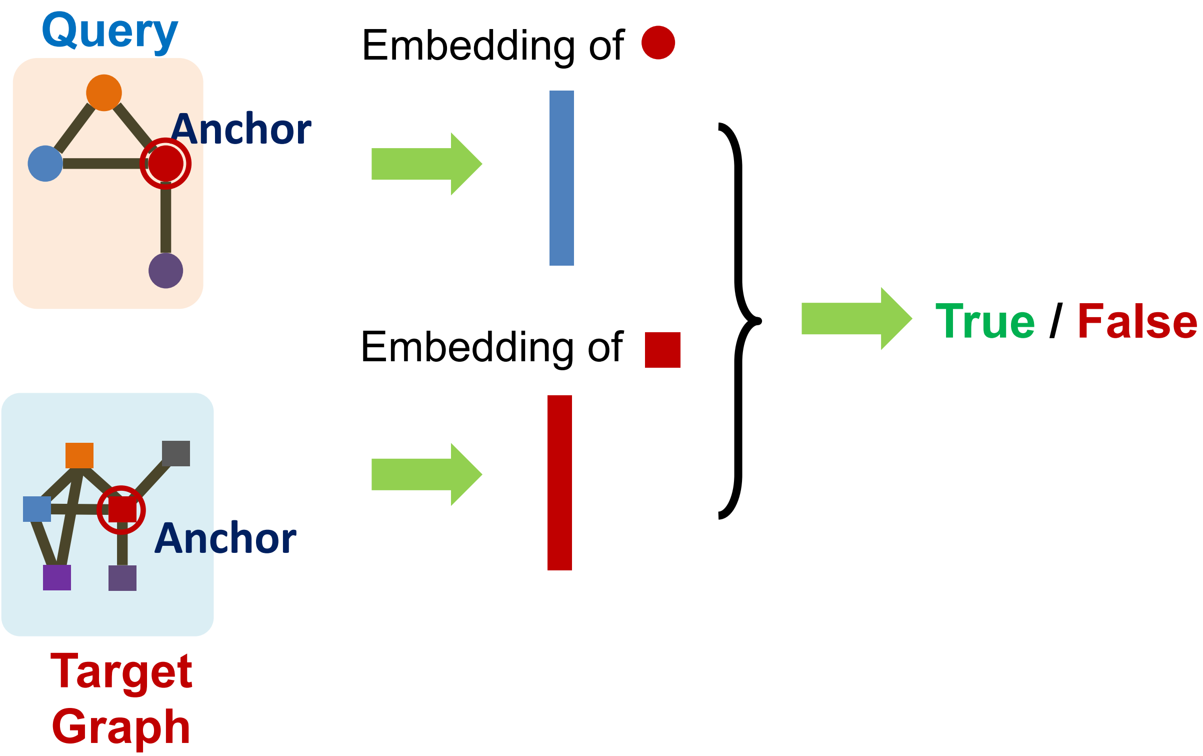
Overview of the Approach

- **Input graph:**



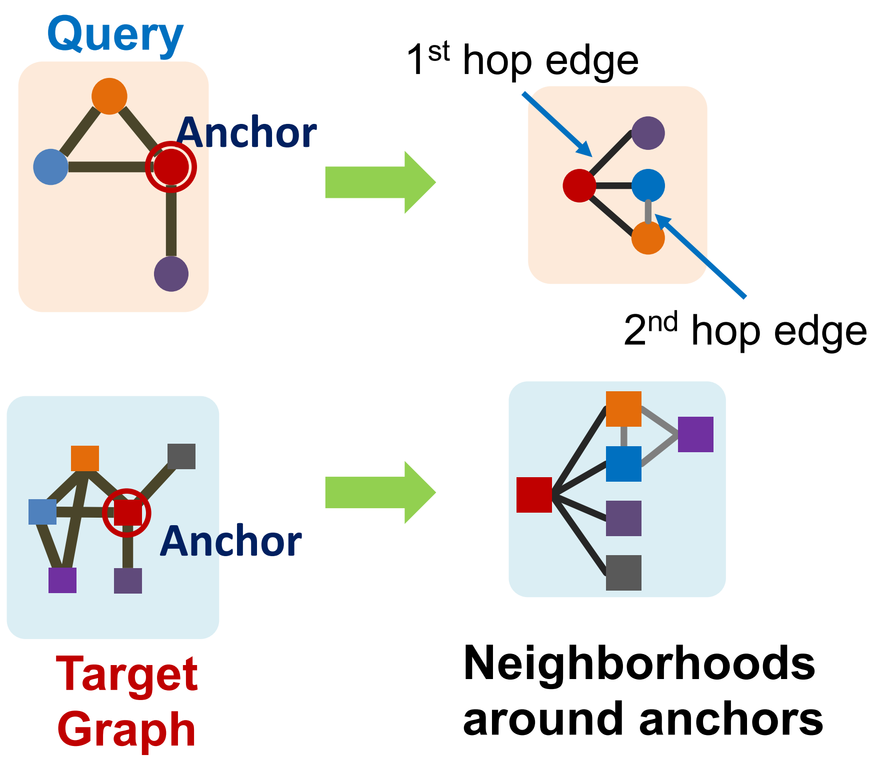
Neural Architecture for Subgraphs (1)

- **(1)** We are going to work with **node-anchored definitions**:



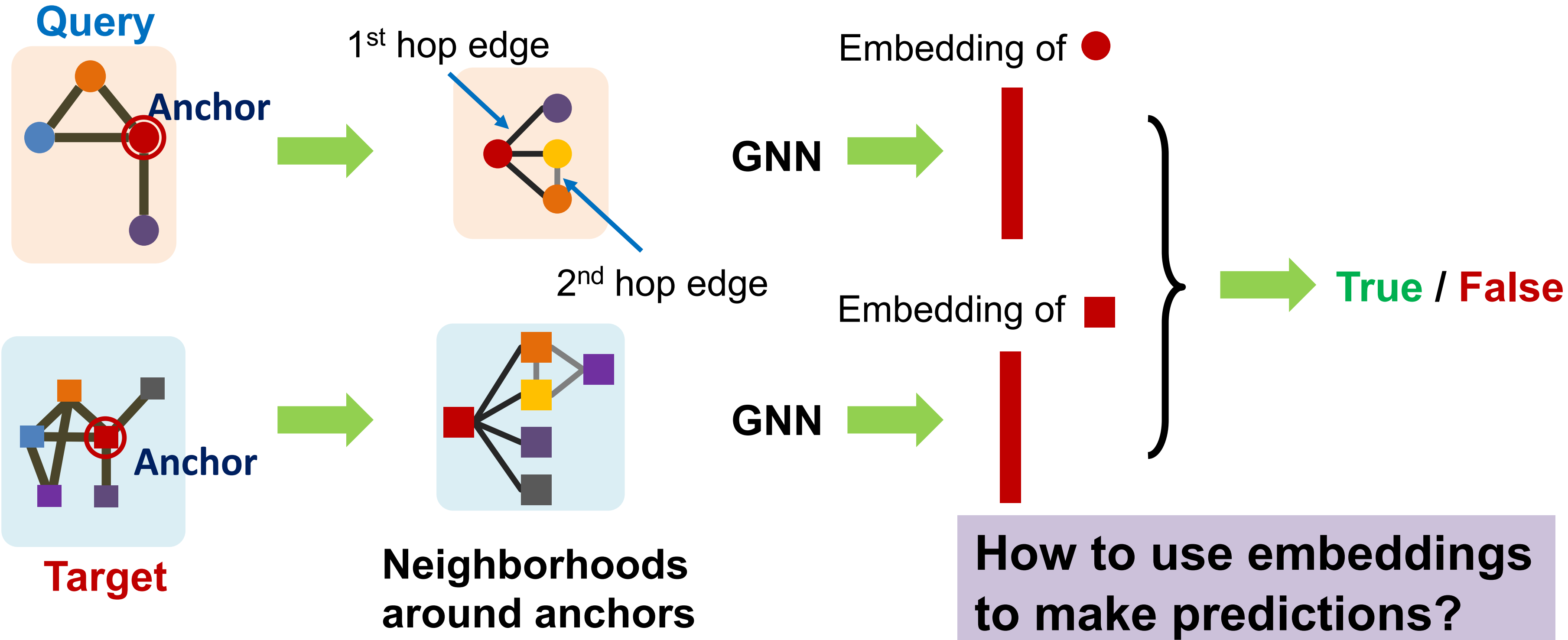
Neural Architecture for Subgraphs (2)

- **(2)** We are going to work with **node-anchored neighborhoods**:



Neural Architecture for Subgraphs (3)

- Use GNN to obtain representations of u and v
- **Predict if node u's neighborhood is isomorphic to node v's neighborhood:**



How to use embeddings to make predictions?

Why Anchor?

- **Recall node-level frequency definition:**
  The number of nodes u in G_T for which some subgraph of G_T is isomorphic to G_Q and the isomorphism maps u to v

- We can compute **embeddings** for u and v using GNN

- Use embeddings to decide if neighborhood of u is isomorphic to subgraph of neighborhood of v

- We not only predict if there exists a mapping, but also **a identify corresponding nodes** (u and v)!

Decomposing G_T into Neighborhoods

- **For each node in G_T:**
  - Obtain a **k-hop neighborhood** around the anchor
  - Can be performed using **breadth-first search** (BFS)
  - The depth k is a hyper-parameter (*e.g.* 3)
    - Larger depth results in more expensive model
- Same procedure applies to G_Q to obtain the neighborhoods

- We embed the neighborhoods using a GNN
  - By computing the **embeddings for the anchor** nodes in their respective neighborhoods

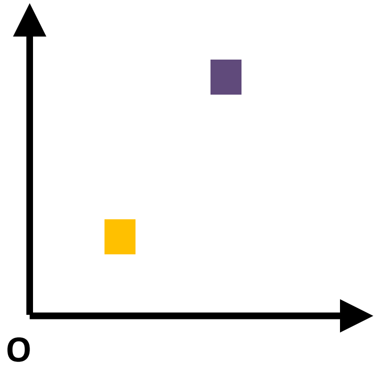
Idea: Order Embedding Space

Map graph A to a point z_A into a high-dimensional (e.g. 64-dim) embedding space, such that z_A is **non-negative in all dimensions**

Capture partial ordering (transitivity):

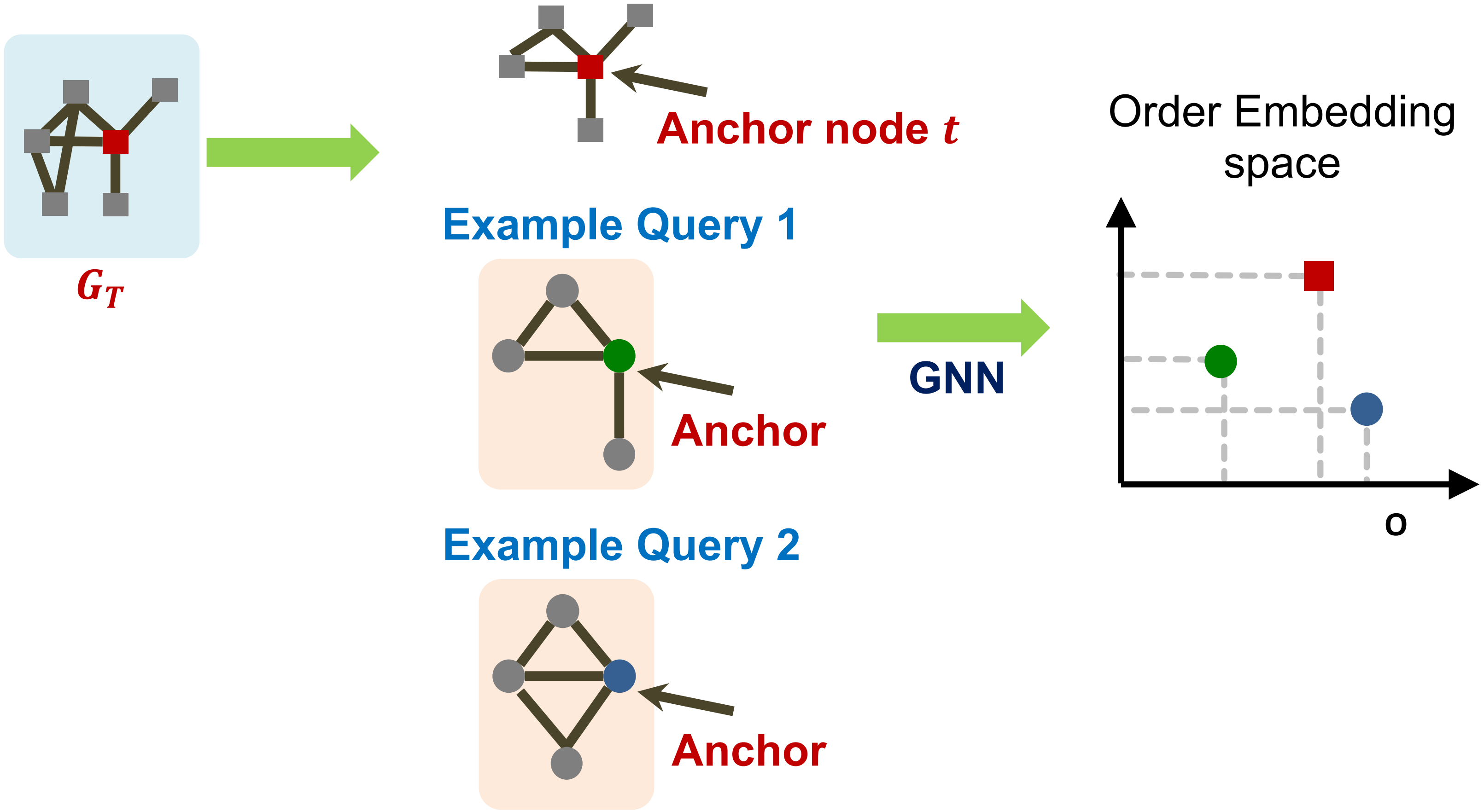
- We use ■ $\preccurlyeq$ ■ to denote that the embedding of ■ is less than or equal to ■ in **all of its coordinates**
- If ■ $\preccurlyeq$ ■, ■ $\preccurlyeq$ ■ then ■ $\preccurlyeq$ ■

Embedding space



Intuition: subgraph is to the lower-left of its supergraph (in 2D)

Subgraph Order Embedding Space



By comparing the embedding, we find that ● ≼ ■ but ● ⋠ ■,
Indicating that only query 1 is a subgraph of the neighborhood of t

Why Order Embedding Space?

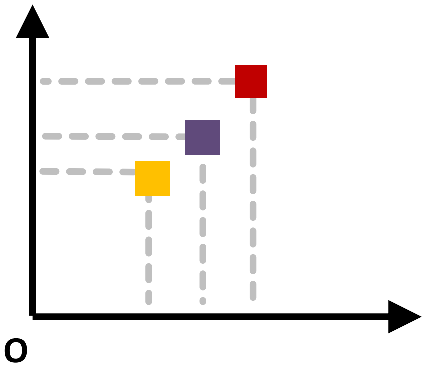
- **Subgraph isomorphism relationship** can be nicely encoded in **order embedding space**
  - **Transitivity**: If G_1 is a subgraph of G_2, G_2 is a subgraph of G_3, then G_1 is a subgraph of G_3
  - **Anti-symmetry**: If G_1 is a subgraph of G_2, and G_2 is a subgraph of G_1, then G_1 is isomorphic to G_2
  - **Closure under intersection**: The trivial graph of 1 node is a subgraph of any graph
  - **All properties have their counter-parts in the order embedding space**

Why Order Embedding Space?

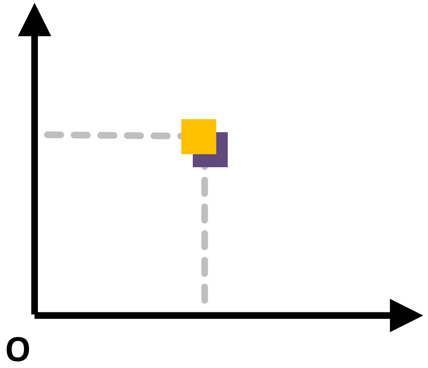
- **Subgraph isomorphism relationship** can be nicely encoded in order embedding space
  - **Transitivity**: If ■ ≼■, ■≼■ then ■≼■
  - **Anti-symmetry**: If ■ ≼ ■ and ■ ≼ ■ , then ■ = ■
  - **Closure under intersection**: The 0 embedding satisfies 0 ≼ ■ for any order embedding ■ since all dimensions of order embedding are non-negative
    - Corollary: If ■≼■ and ■≼■ then ■ has a valid embedding

0 embedding:
Trivial graph
with one node

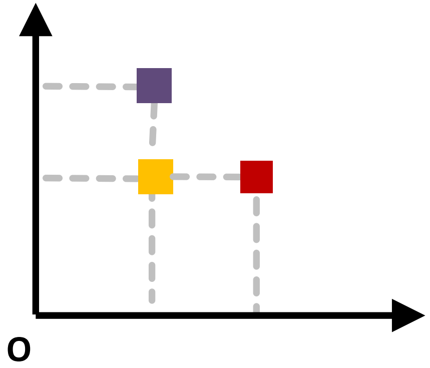
Transitivity



Anti-symmetry



Closure under intersection



Order Constraint (1)

- We use a GNN to learn to embed neighborhoods and preserve the **order embedding** structure
- **What loss function should we use, so that the learned order embedding reflects the subgraph relationship?**
- We design loss functions based on the **order constraint:**
  - Order constraint specifies the ideal order embedding property that reflects subgraph relationships

Order Constraint (2)

- **We specify the order constraint to ensure that the subgraph properties are preserved in the order embedding space**

Embedding dimension

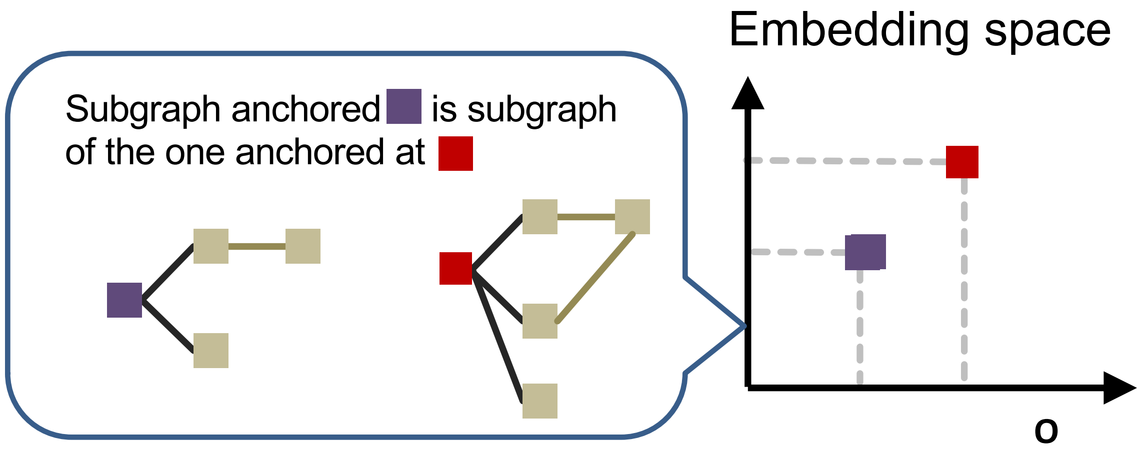
$$\forall_{i=1}^{D} z_q[i] \leq z_t[i] \quad \text{iff} \quad G_Q \subseteq G_T$$

trained with max-margin loss

Query embedding

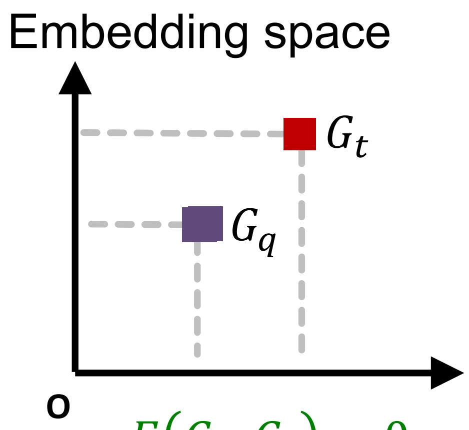
Target embedding

Subgraph Relation

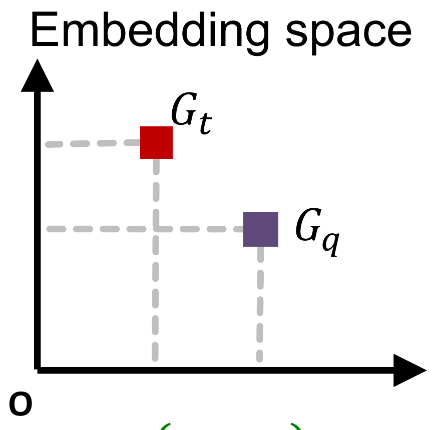


Loss Function: Order Constraint

- GNN Embeddings are learned by minimizing a **max-margin** loss
- Define $E(G_q, G_t) = \sum_{i=1}^{D}(\max(0, z_q[i] - z_t[i]))^2$ as the "margin" between graphs G_q and G_t



$E(G_q, G_t) = 0$
According to the order embedding, G_q is a subgraph of G_t!



$E(G_q, G_t) > 0$
According to the order embedding, G_q is **not** a subgraph of G_t!

Loss Function

- Embeddings are learned by minimizing a **max-margin** loss
- Let $E(G_q, G_t) = \sum_{i=1}^{D}(\max(0, z_q[i] - z_t[i]))^2$ be the "margin" between graphs G_q and G_t
- **To learn the correct order embeddings, we want to learn $\boldsymbol{z_q}, \boldsymbol{z_t}$ such that**
  - $E(G_q, G_t) = 0$ when G_q **is** a subgraph of G_t
  - $E(G_q, G_t) > 0$ when G_q **is not** a subgraph of G_t

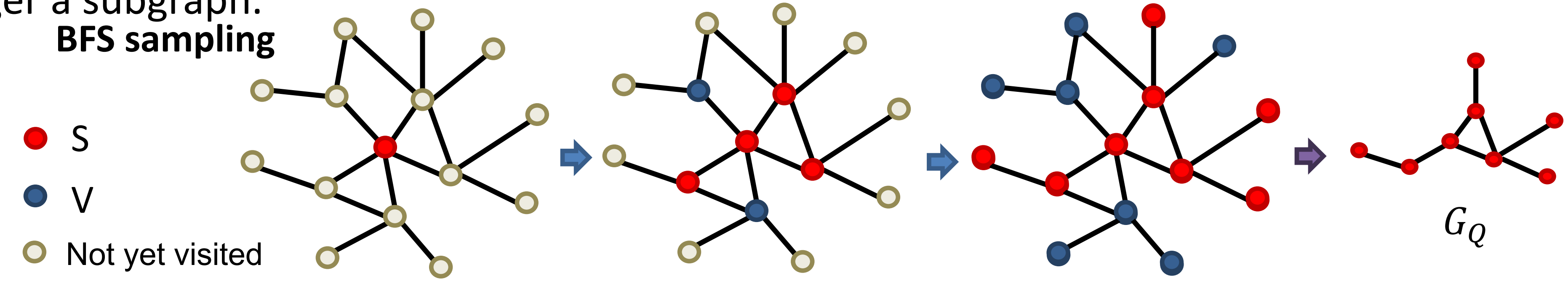
Training Neural Subgraph Matching

- To learn such embeddings, **construct training examples** (G_q, G_t) where half the time, G_q is a subgraph of G_t, and the other half, it is not
- Train on these examples by minimizing the following **max-margin loss**:
  - **For positive examples:** Minimize $E(G_q, G_t)$ when G_q is a subgraph of G_t
  - **For negative examples:**
    Minimize $\max(0, \alpha - E(G_q, G_t))$
    - Max-margin loss prevents the model from learning the degenerate strategy of moving embeddings further and further apart forever

Training Example Construction

- Need to generate training queries G_Q and targets G_T from the dataset G
- Get G_T by choosing a random anchor v and taking all nodes in G within distance K from v to be in G_T
- **Positive examples:** Sample induced subgraph G_Q of G_T. Use **BFS sampling:**
  - Initialize $S = \{v\},\ V = \emptyset$
  - Let $N(S)$ be all neighbors of nodes in S. At every step, sample 10% of the nodes in $N(S) \setminus V$, put them in S. Put the remaining nodes of $N(S)$ in V.
  - After K steps, take the subgraph of G induced by S anchored at q
- **Negative examples** (G_Q not subgraph of G_T): "corrupt" G_Q by adding/removing nodes/edges so it's no longer a subgraph.

BFS sampling



Training Details

- **How many training examples to sample?**
  - At every iteration, we sample new training pairs
  - **Benefit**: Every iteration, the model sees different subgraph examples
  - Improves performance and avoids **overfitting** – since there are exponential number of possible subgraphs to sample from
- **How deep is the BFS sampling?**
  - A hyper-parameter that trades off **runtime and performance**
  - Usually use 3-5, depending on size of the dataset

Subgraph Predictions on New Graphs

- **Given**: query graph G_q anchored at node q, target graph G_t anchored at node t
- **Goal**: output whether the query is a node-anchored subgraph of the target
- **Procedure**:
  - If $E\left(G_q, G_t\right) < \epsilon$, predict "True"; else "False"
  - ϵ is a hyper-parameter
- To check if G_Q is isomorphic to a subgraph of G_T , repeat this procedure for all $q \in G_Q$, $t \in G_T$. Here G_q is the neighborhood around node $q \in G_Q$.

Summary: Neural Subgraph Matching

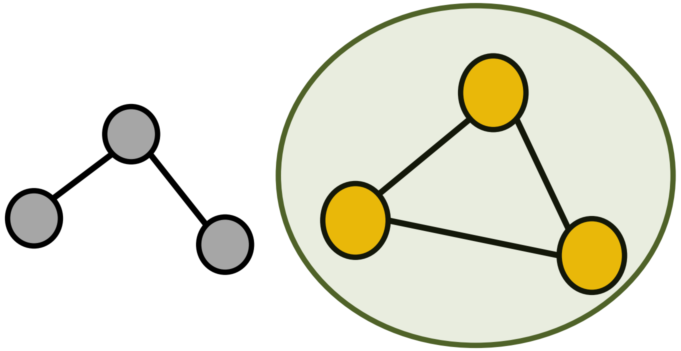
- Neural subgraph matching uses a **machine learning-based approach** to learn the NP-hard problem of subgraph isomorphism
  - Given query and target graph, it embeds both graphs into an order embedding space
  - Using these embeddings, it then computes $E(G_q, G_t)$ to determine whether query is a subgraph of the target
- Embedding graphs within an **order embedding space** allows subgraph isomorphism to be efficiently represented and tested by the relative positions of graph embeddings

GNN Applications: Graph Mining

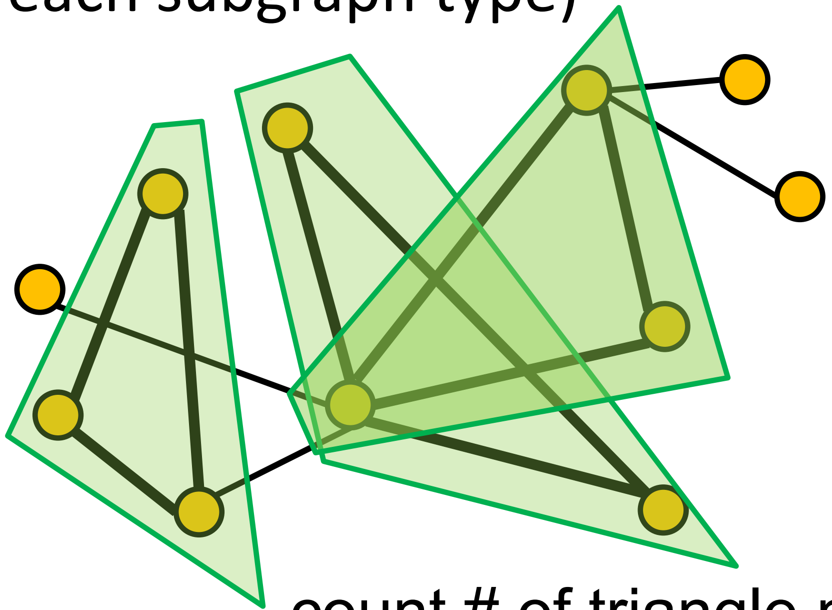
Finding Frequent Subgraphs

Intro: Finding Frequent Subgraphs

- Generally, finding the most frequent size-k motifs requires solving two challenges:
  - **1) Enumerating** all size-k connected subgraphs
  - **2) Counting** #(occurrences of each subgraph type)



Possible size-3 motifs



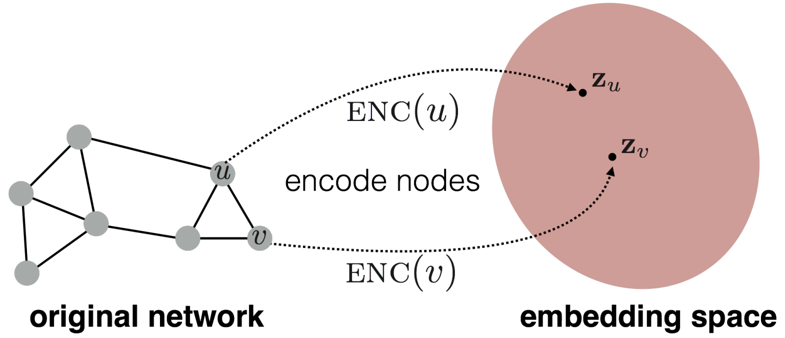
count # of triangle motifs

Why is it Hard?

- **Just knowing if a certain subgraph exists in a graph, is a hard computational problem!**
  - Subgraph isomorphism is NP-complete
- **Computation time grows exponentially as the size of the subgraphs increases**
  - Feasible motif size for traditional methods is relatively small (3 to 7)

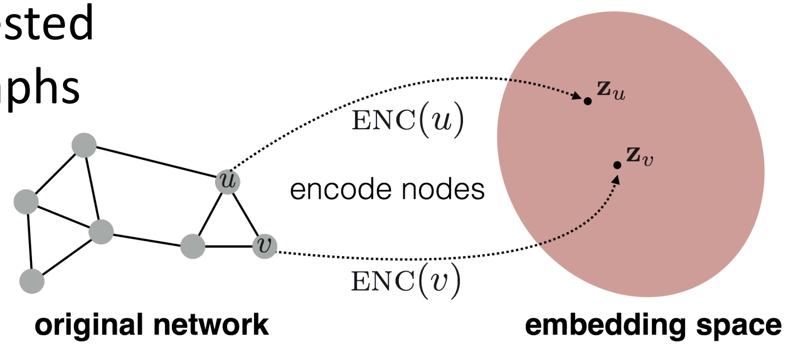
Solution with Representation Learning

- Finding frequent subgraph patterns is **computationally hard**
  - **Combinatorial explosion** of number of possible patterns
  - Counting **subgraph frequency** is NP-hard
- **Representation learning** can tackle these challenges:
  - **Combinatorial explosion** → organize the search space
  - **Subgraph isomorphism** → prediction using GNN



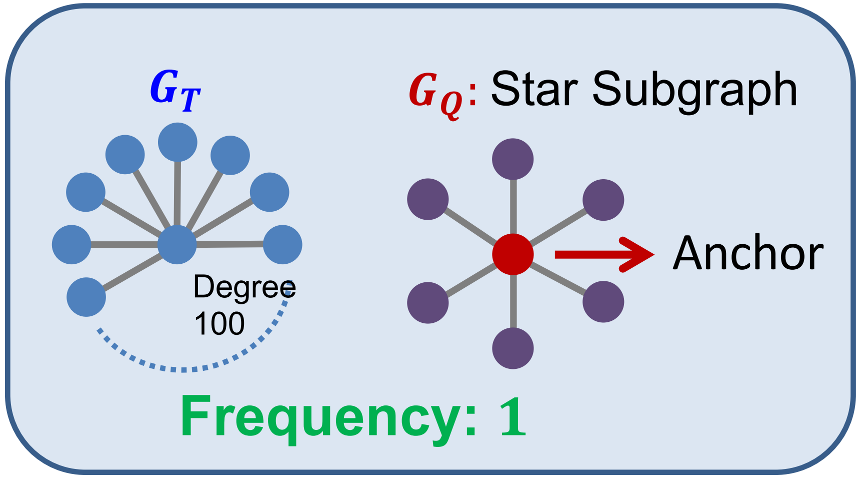
Solution with Representation Learning

- **Representation learning** can tackle these challenges:
  - **1) Counting** #(occurrences of each subgraph type)
    - **Solution:** Use GNN to "predict" the frequency of the subgraph.
  - **2) Enumerating** all size-k connected subgraphs
    - **Solution:** Don't enumerate subgraphs but construct a size-k subgraph incrementally
      - Note: We are only interested in high frequency subgraphs



Problem Setup: Frequent Motif Mining

- Target graph (dataset) G_T, size parameter k
- Desired number of results r
- **Goal**: Identify, among *all* possible graphs of k nodes, the r graphs with the highest frequency in G_T.
- **We use the node-level definition:**
  The number of nodes u in G_T for which some subgraph of G_T is isomorphic to G_Q and the isomorphism maps u to v.



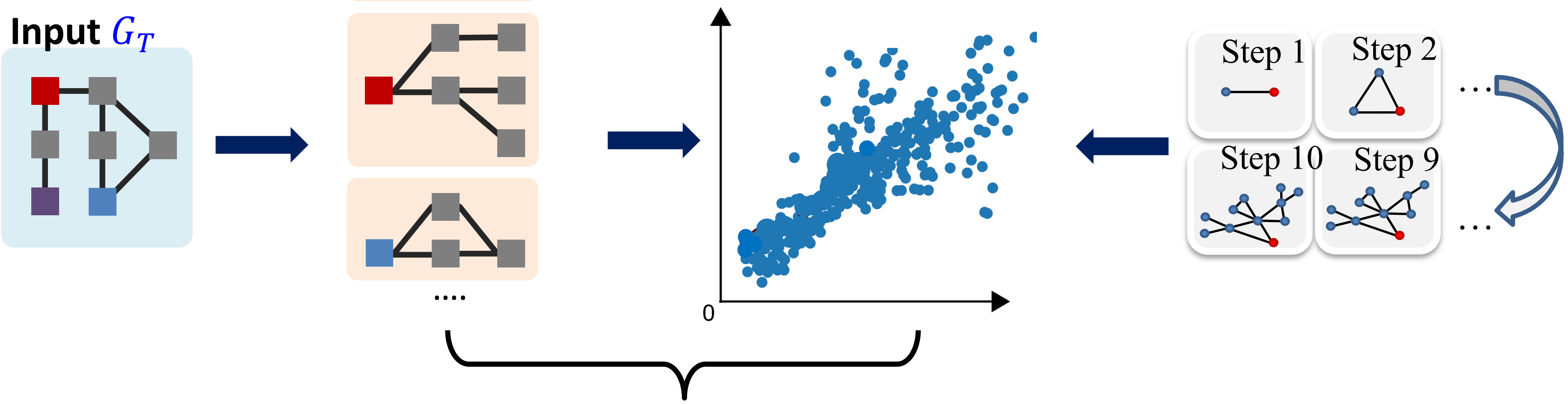
SPMiner: Overview

SPMiner: A neural model to identify frequent motifs

Decompose: Overlapping *node-anchored* neighborhoods

Encoder: Embed subgraphs into *order embedding space*

Search Procedure: Find frequent subgraphs by *growing* patterns



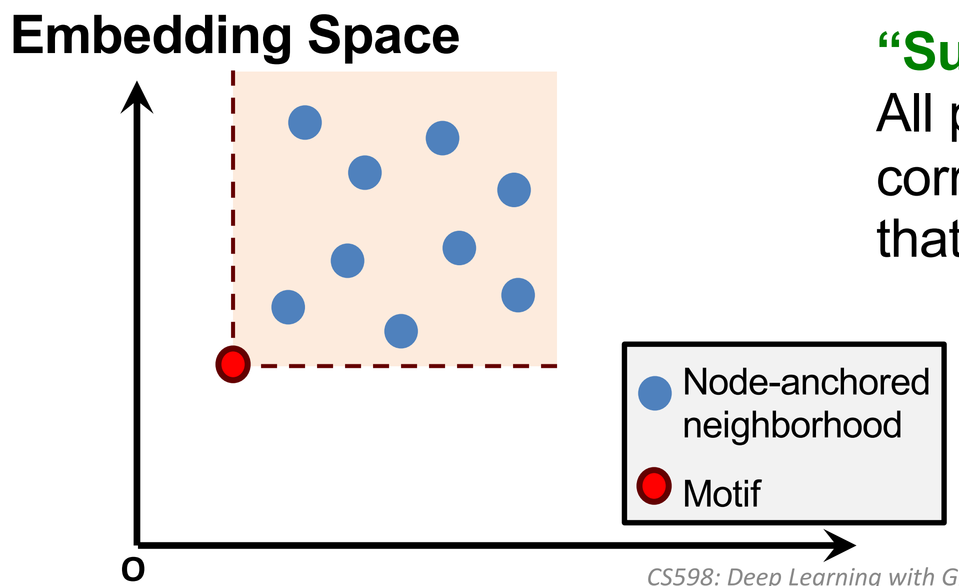
Same as neural subgraph matching

SPMiner: Key Idea

- Decompose input graph G_T into neighborhoods
- Embed neighborhoods into an order embedding space
- **Key benefit of order embedding:**
  We can quickly "predict" the frequency of a given subgraph G_Q

Motif Frequency Estimation

- **Given:** Set of subgraphs ("node-anchored neighborhoods") G_{N_i} of G_T (sampled randomly)
- **Key idea:** Estimate frequency of G_Q by counting the number of G_{N_i} such that their embeddings z_{N_i} satisfy $z_Q \leq z_{N_i}$
  - This is a consequence of the **order embedding space property**

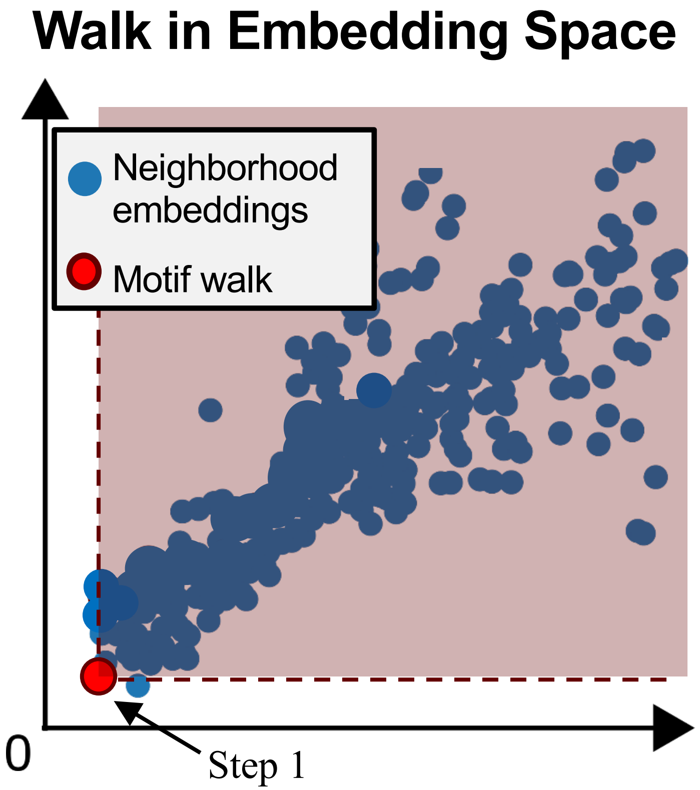


"Super-graph" region:
All points in the red shaded region correspond to neighborhoods in G_T that contain G_Q

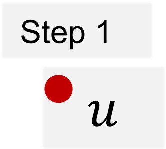
Benefit: Super-fast subgraph frequency counting!

SPMiner Search Procedure (1)

- **Initial step**: Start by randomly picking a starting node u in the target graph G_T. Set $S = \{u\}$.



Each point in the shaded region represents a neighborhood in target graph that contains the motif pattern

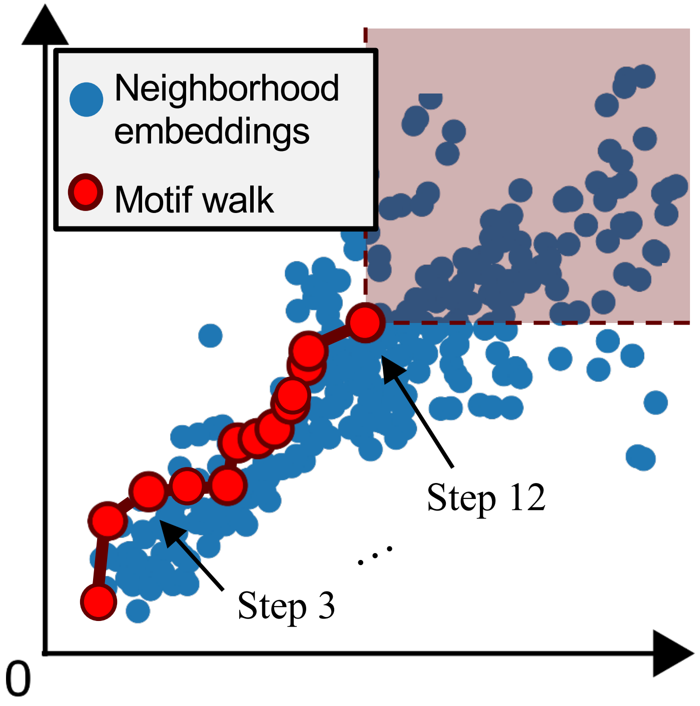


Initially, all neighborhoods contain the trivial subgraph

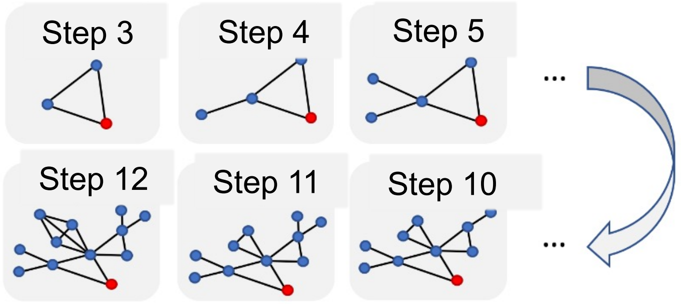
SPMiner Search Procedure (2)

- **Iteratively**: Grow a motif by iteratively choosing a neighbor in G_T of a node in S and add that node to S. We grow the motif S to find **larger frequent** motifs!

Walk in Embedding Space



- Small motifs grow by adding neighbors
- Their embeddings correspond to red points on the left

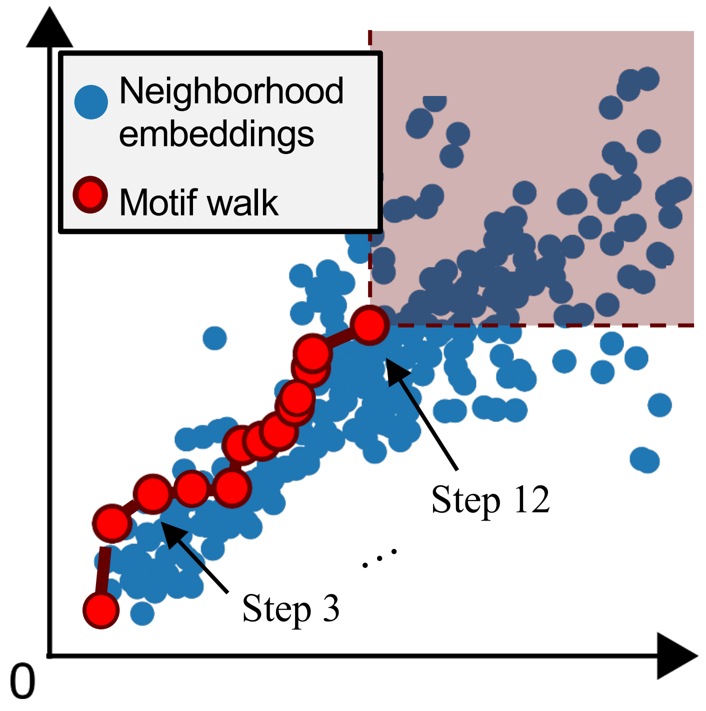


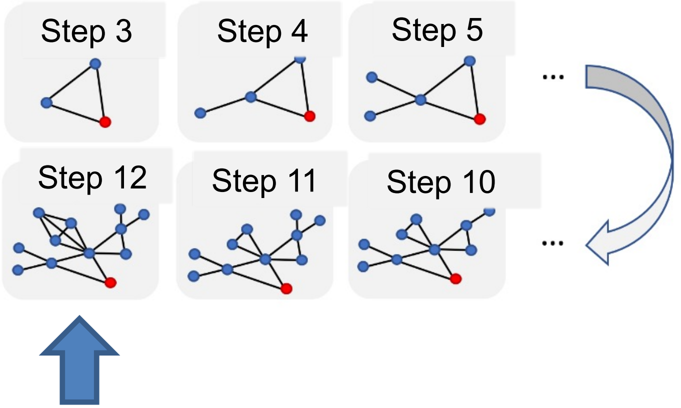
Goal: maximize number of neighborhoods in red shaded area after k step!

SPMiner Search Procedure (3)

- **Termination**: Upon reaching a desired motif size, take the subgraph of the target graph induced by S.

Walk in Embedding Space

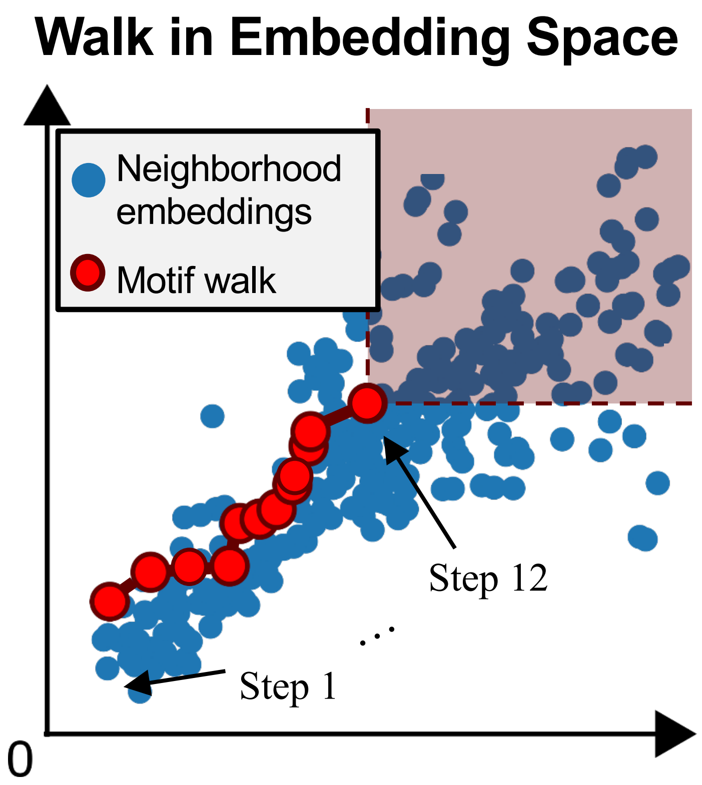




Identified frequent motif of size 12:
It has the largest number of blue points in super-graph region, among all embeddings of possible subgraphs of size 12

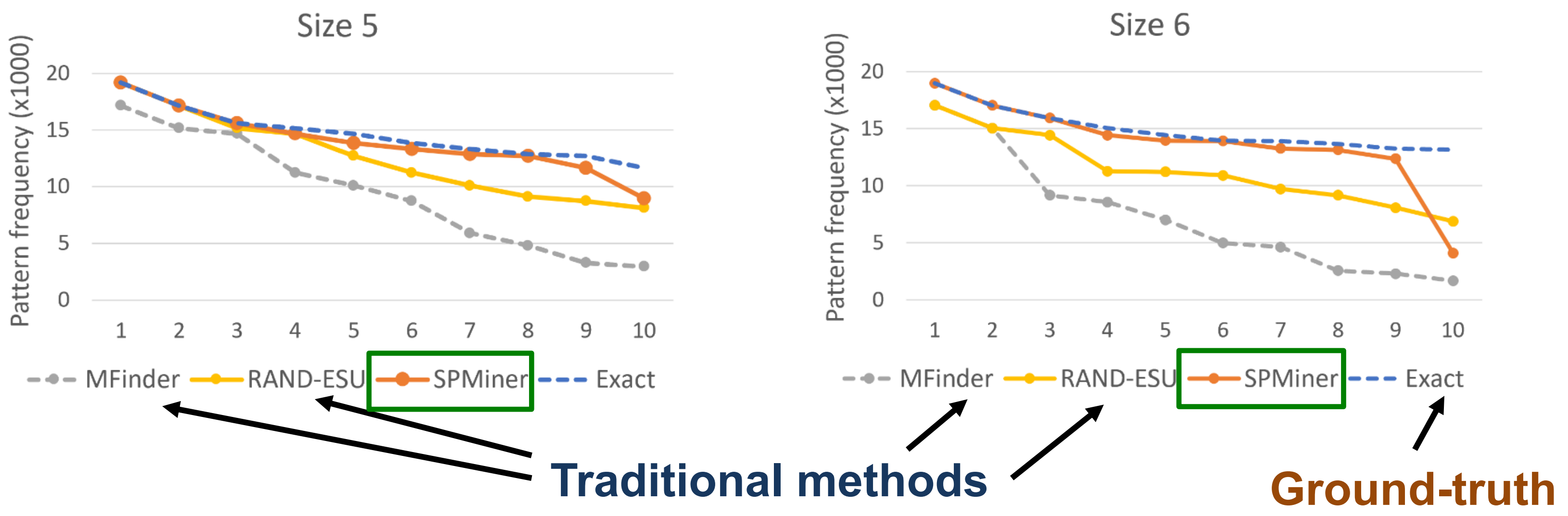
SPMiner Search Procedure (4)

- **How to pick which node to add at each step?**
- **Def: Total violation** of a subgraph G: the number of neighborhoods that do not contain G.
  - The number of neighborhoods G_{N_i} that do **not** satisfy $z_Q \preccurlyeq z_{N_i}$
  - Minimizing total violation = maximizing frequency
- **Greedy strategy (heuristic)**: At every step, add the node that results in the **smallest total violation**



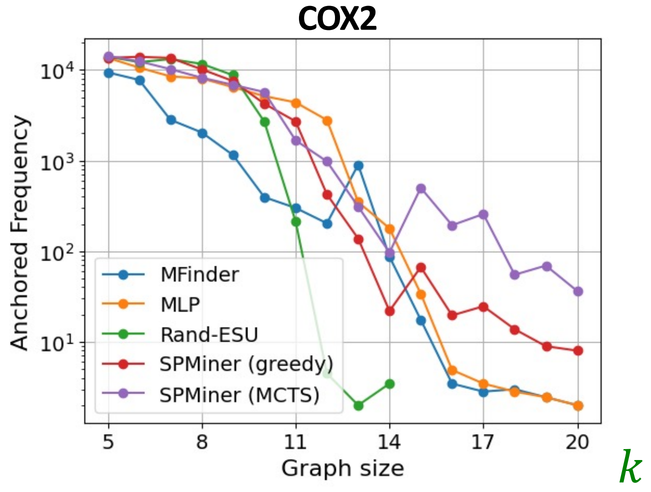
Results: Small Motifs

- **Ground-truth:** Find most frequent 10 motifs in dataset by brute-force exact enumeration (expensive)
- **Question**: Can the model identify frequent motifs?
- **Result**: The model identifies 9 and 8 of the top 10 motifs, respectively.

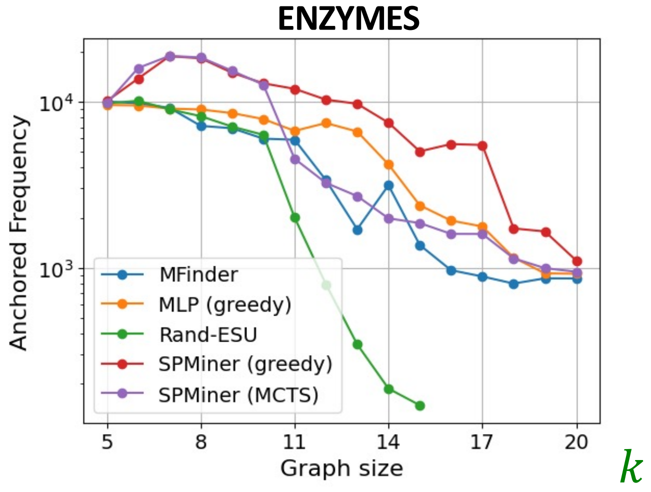


Experiments: Large motifs

- **Question**: How do the frequencies of the identified motif compare?
- **Result**: SPMiner identifies motifs that appear 10-100x more frequently than the baselines



Molecule dataset



Protein dataset

Summary

- **Subgraphs** and **motifs** are important concepts that provide insights into the structure of graphs. Their frequency can be used as features for nodes/graphs.

- We covered **neural approaches** to prediction subgraph isomorphism relationship.

- **Order embeddings** have desirable properties and can be used to encode subgraph relations

- **Neural embedding-guided search** in order embedding space can enable ML model to identify motifs much more frequent than existing methods