Stanford CS224W: Identifying and Counting Motifs in Networks

CS224W: Machine Learning with Graphs

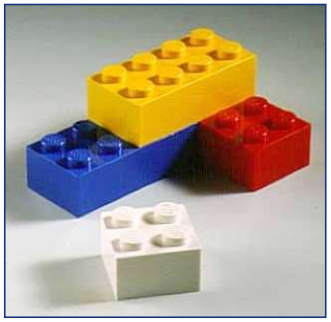
Jure Leskovec, Stanford University

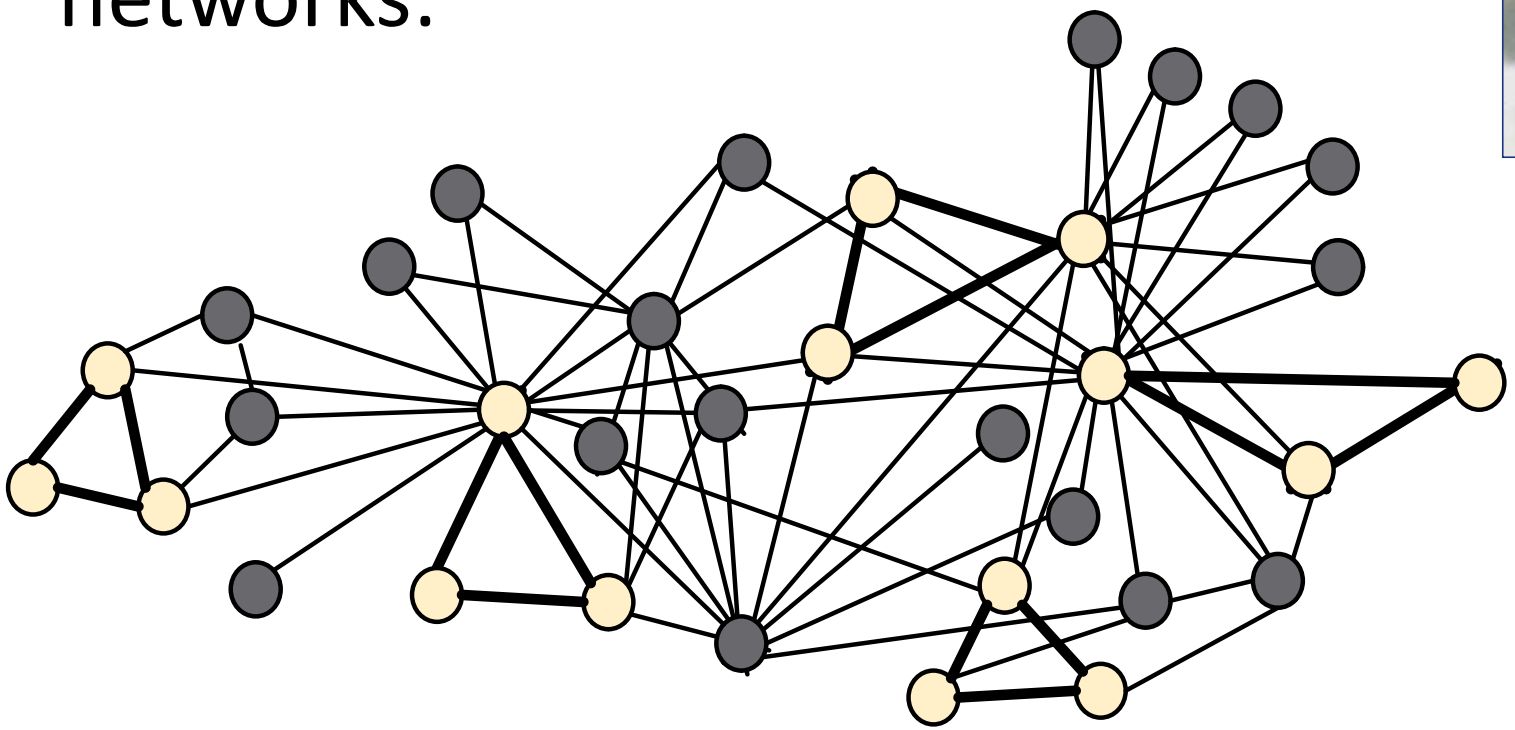
http://cs224w.stanford.edu

One of the classical problem in graph mining is identification of common building block that characterize the graph. With help of graph and node embedding we will try to solve this problem.

Subgraphs

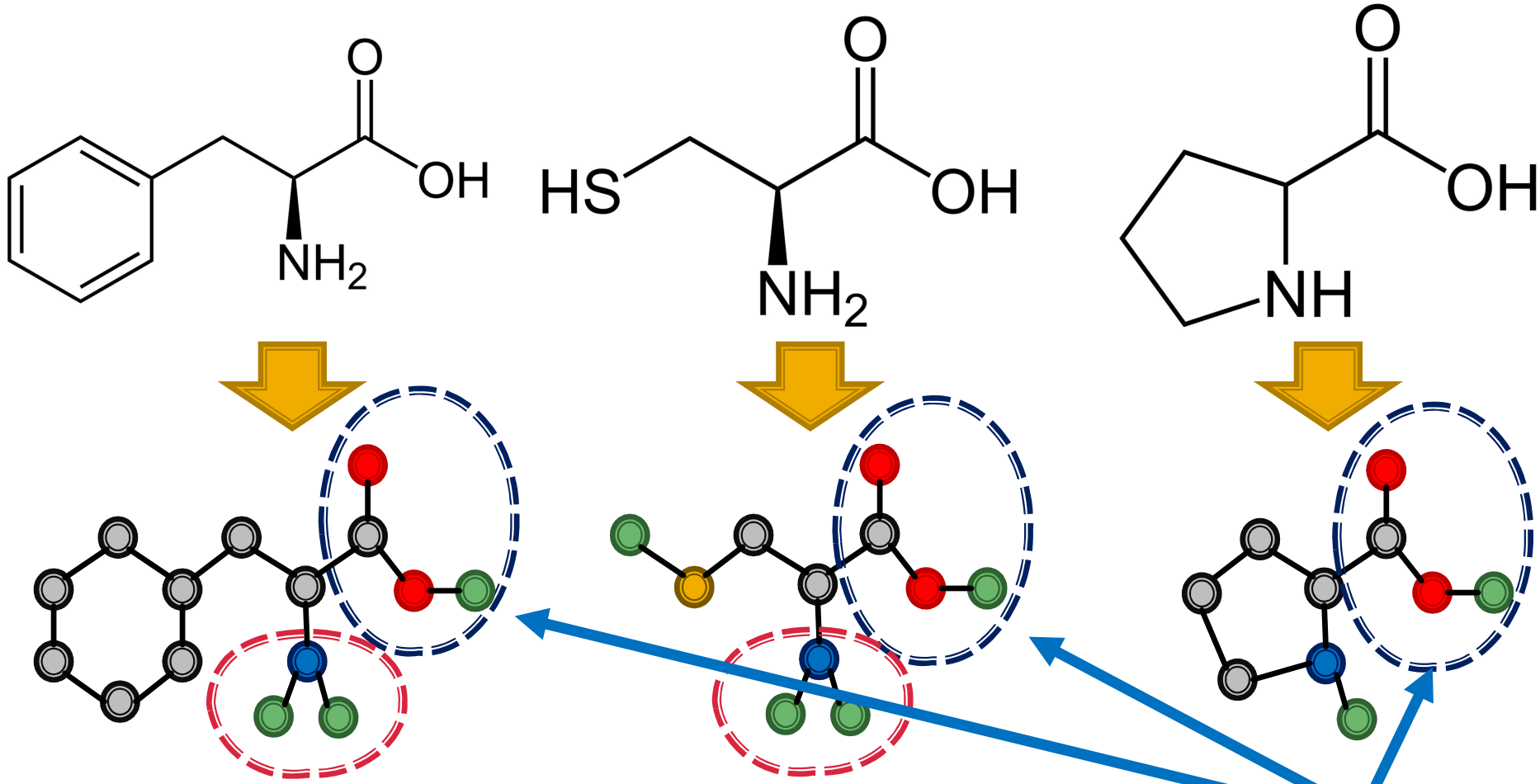
- **Subgraphs** are the building blocks of networks:





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

Building Blocks of Networks

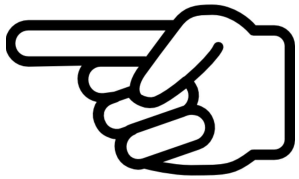


Carboxyl group = Acidic

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

Stanford CS224W: Subgraphs and Motifs

CS224W: Machine Learning with Graphs

Jure Leskovec, Stanford University

http://cs224w.stanford.edu

Define subgraph & motif

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
  - $V' \subseteq V$
  - $E' = \{(u, v) \in E \mid u, v \in V'\}$ - edges form set of E part of V'
  - G' is the subgraph of G **induced** by V'

- **Alternate terminology:** "induced subgraph" → Based on nodes.

In the induced subgraph edges are determine by subset of nodes that we take from set of nodes.

Definition: Subgraph (2)

Two ways to formalize "network building blocks"

less common

- 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
  - $E' \subseteq E$ → subset of edges that we select to create subgraph
  - $V' = \{v \in V \mid (v, u) \in E' \text{ for some } u\}$ v' comes from E'

- **Alternate terminology:** "non-induced subgraph" or just "subgraph" - Based on edges

This means when we say induced subgraph that means nodes selected and edges selected ~~from~~ automatically but when ~~we~~ we select edges and nodes get automatically determined then that is called ~~non~~ non-induced subgraph or subgraph

we often select node-induced subgraph as if we select edge-induced subgraph then number of possibility will explode.

Definition: Subgraph (3)

Two ways to formalize "network building blocks"

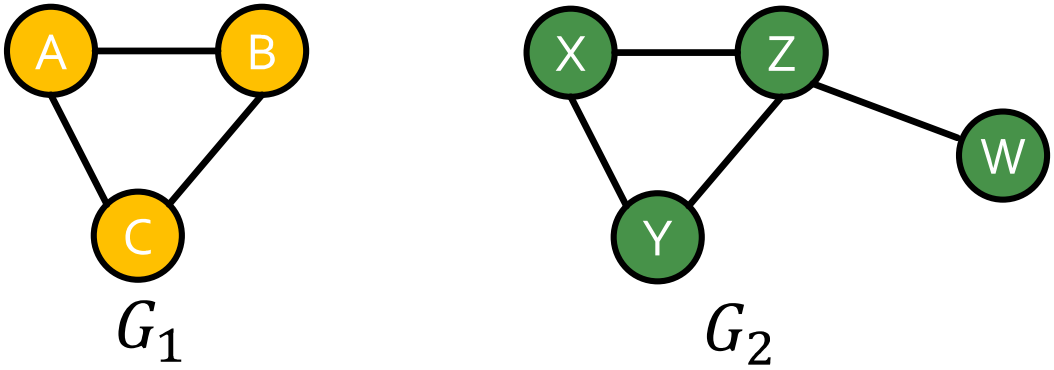
- The best definition depends on the domain! **Examples:**
  - **Chemistry**: node-induced (functional groups)
  - **Knowledge graphs**: Often edge-induced (focus is on edges representing logical relations)

Above: Two different way's of taking node & edge induced subgraph.

Definition: Subgraph (4)

New ways of defining subgraph.

- 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**

Till now we have taken V' & E' from same graph, what if they are coming from different set. In above example can we say G_1 is "contained in" G_2. This is taking ~~two~~ toward's graph isomorphisam.

Below defining notion of graph isomorphisam.

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: $V_1 \rightarrow V_2$ such that $(u, v) \in E_1$ iff $\big(f(a), f(b)\big) \in E_2$
    - f is called the **isomorphism:**

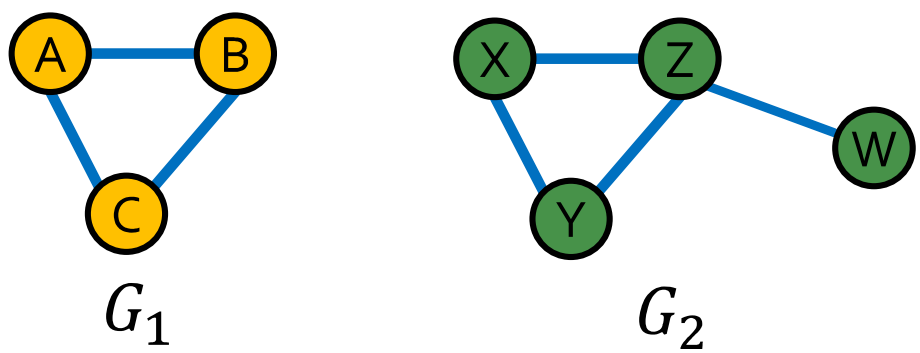
- We do not know if graph isomorphism is NP-hard, nor is any polynomial algorithm found for solving graph isomorphism.

bijection is for one-to-one mapping.

Now defining notion of subgraph 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:

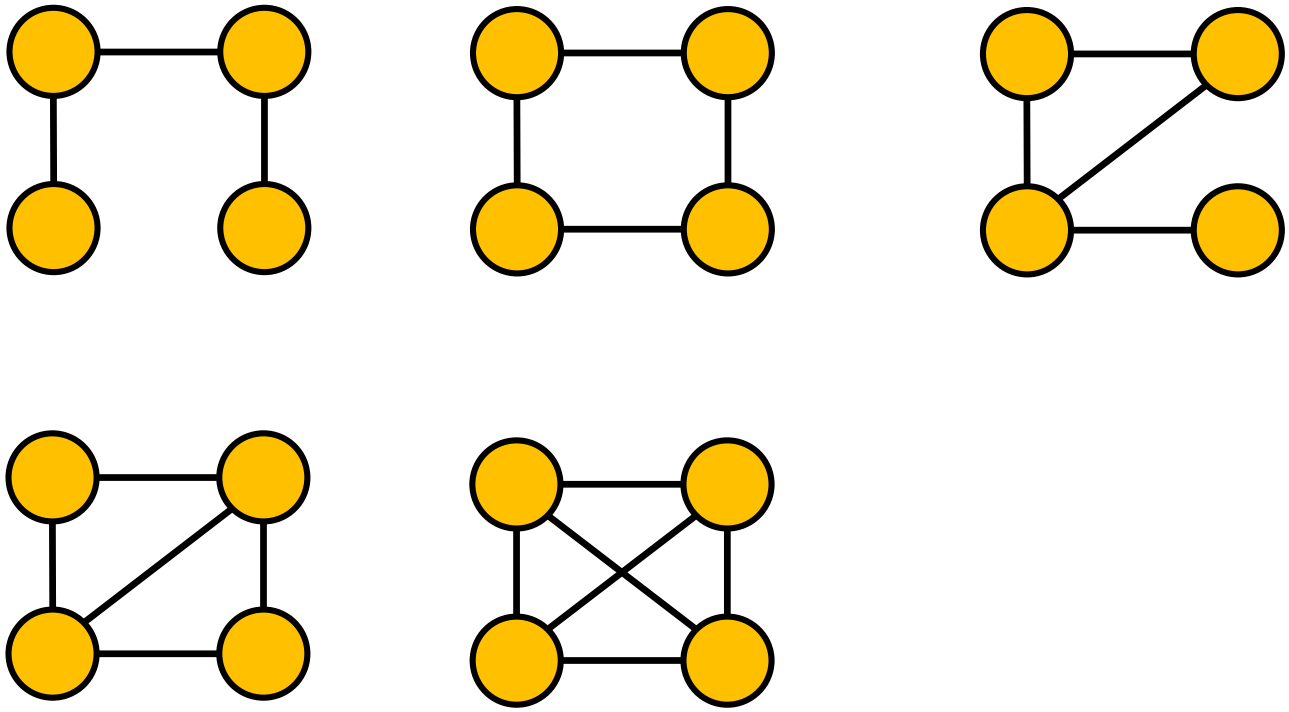
V_1	V_2
A	X
B	Y
C	Z

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

When any one subgraph of one graph graph is isomorphic with other graph then that is called subgraph isomorphism.

Case Example of Subgraphs (1)

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



when we say subgraph then all subgraph of given size. Here size reffers number of nodes and graph is connected.

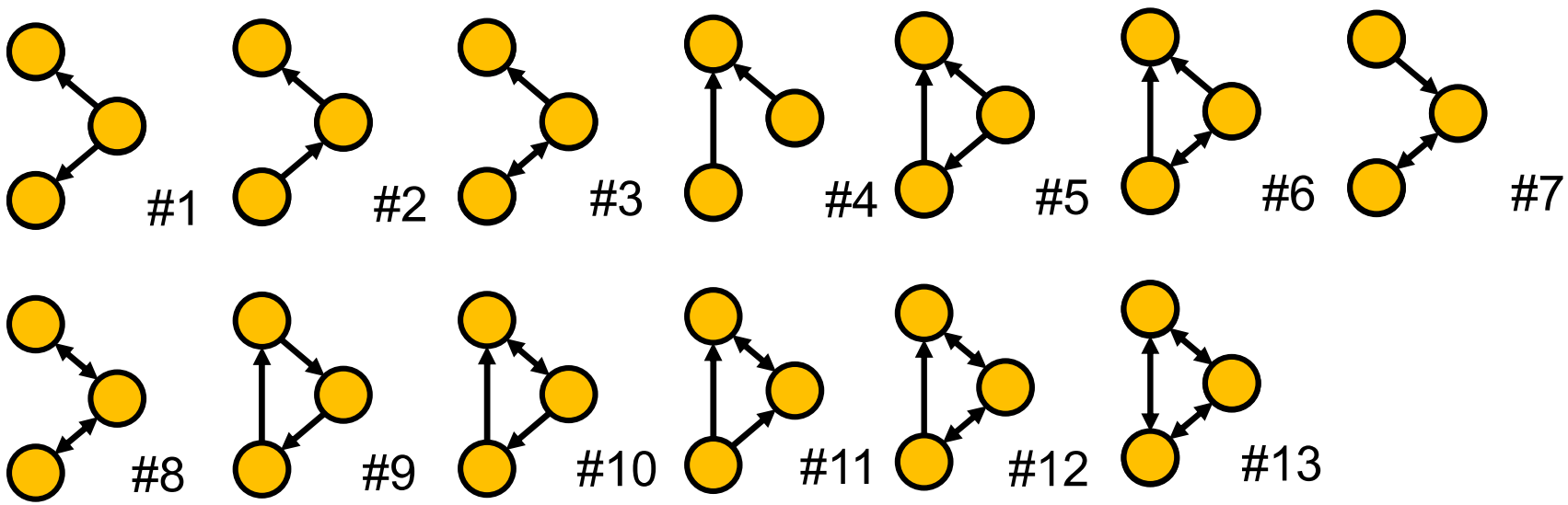
For given node we create all possible non-isomorphic subgraph. This is called size of subgraph from given number of node.

So with 4-node undirected connected non-isomorphic graph is 5

Similarly for directed node below is example.

Case Example of Subgraphs (2)

All non-isomorphic, connected, directed graphs of size 3 ⇒ 13 possible



13 possible subgraph.

with this building block we count frequency of each building block to understand which building block is basis of deciding character of that network.

with increasing number of nodes possible non-isomorphic subgraph increases exponentially so difficult to find subgraph with higher nodes.

Till now we have seen notion of subgraph. Now we understand what is network motifs.

Network Motifs

- **Network motifs:** "recurring, significant patterns of interconnections"

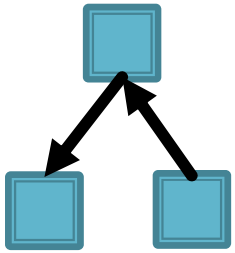
- **How to define a network motif:**
  - **Pattern:** Small (node-induced) subgraph
  - **Recurring:** Found many times, i.e., with high frequency **How to define frequency?**
  - **Significant:** More frequent than expected, i.e., in randomly generated graphs?

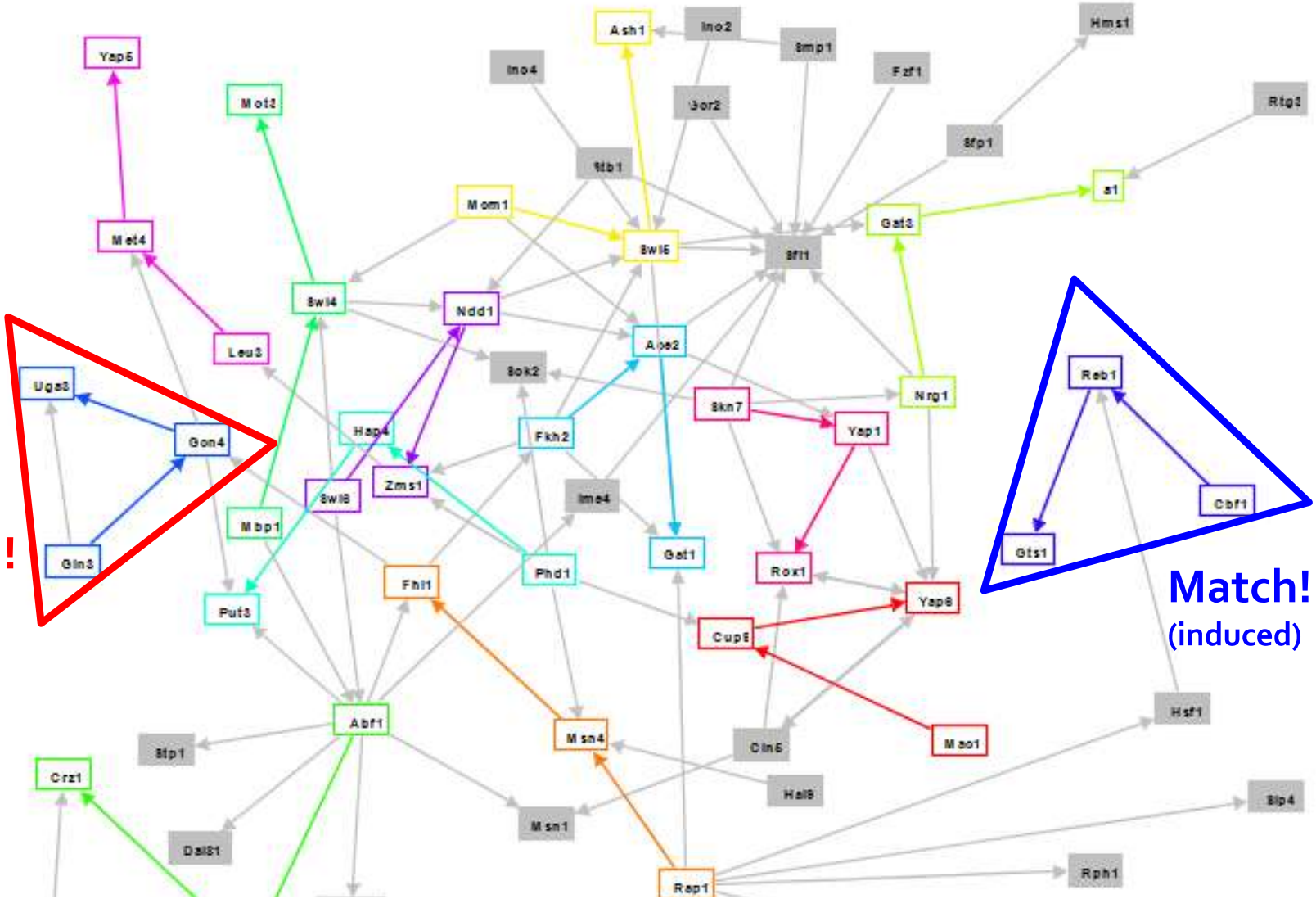
How to define random graphs?

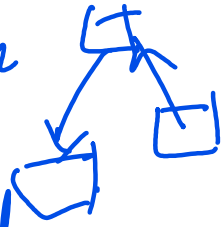
when induced subgraph frequency is more than expected based on null model (Random graph) then that is called motifs.

Motifs: Induced Subgraphs

Induced subgraph of interest (aka Motif):

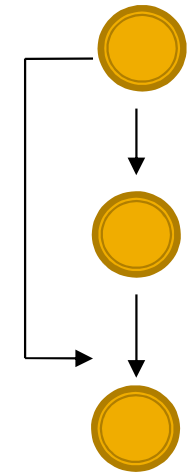




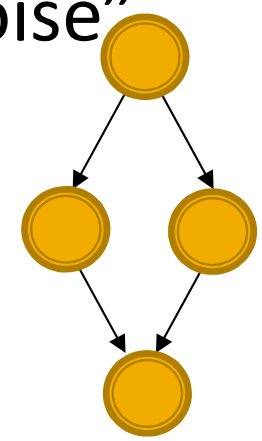
In above we want to find matching patter  . Red one is not match as its Δ out-blue is matching. Similarly many matching pattern.

Why Do We Need Motifs?

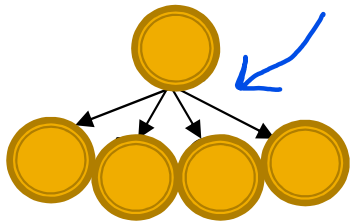
- **Motifs:**
  - Help us understand how graphs work
  - Help us make predictions based on presence or lack of presence in a graph dataset
- **Examples:**
  - **Feed-forward loops:** found in networks of neurons, where they neutralize "biological noise"
  - **Parallel loops:** found in food webs
  -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



Feed-forward loop



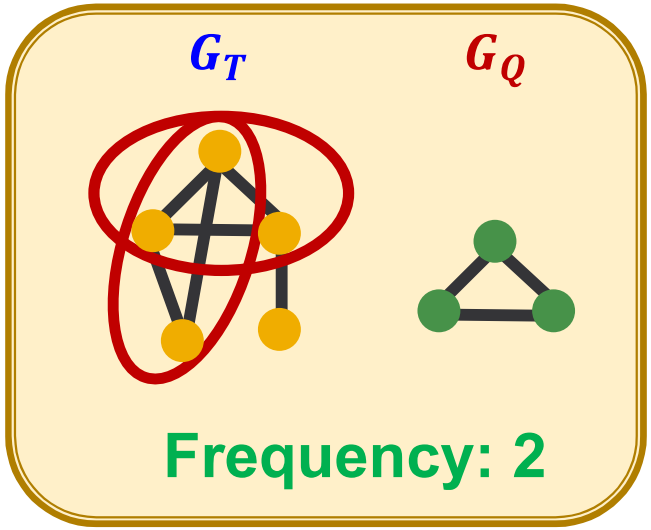
Parallel loop

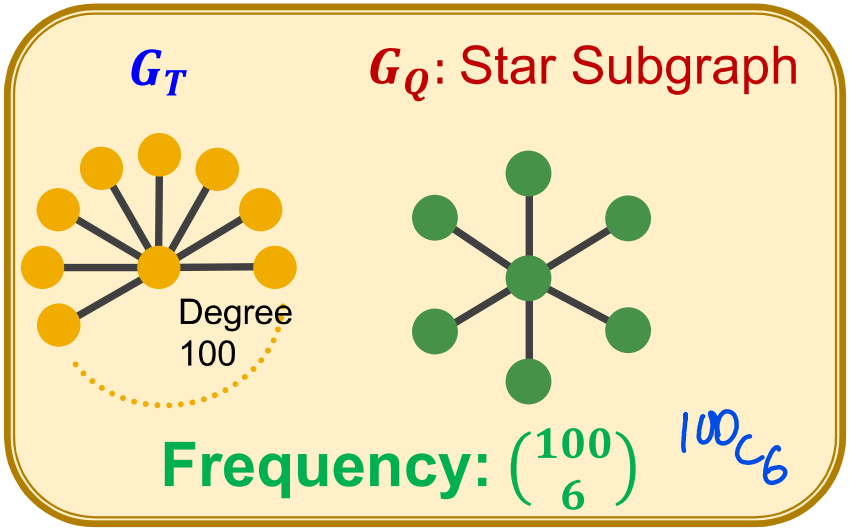


Single-input module

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





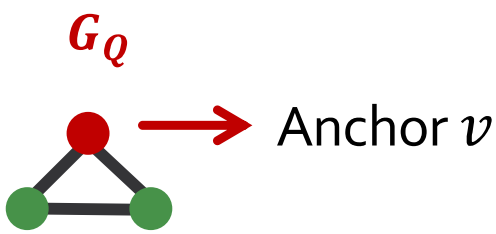
Subgraph frequency count can be done in two ways:

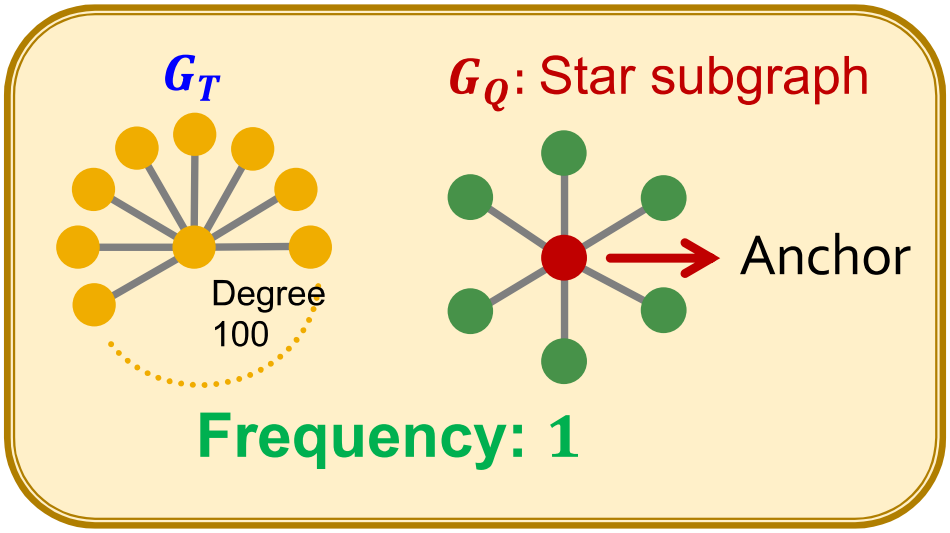
(1) Graph level subgraph frequency: In this we count freq^n of subgraph in main graph.

(2) Node level subgraph: In this we ~~we~~ make one node in subgraph as anchor node and then count how many ~~on~~ different ancor node present in main graph where we are doing matching: examples below.

Subgraph Frequency (2)

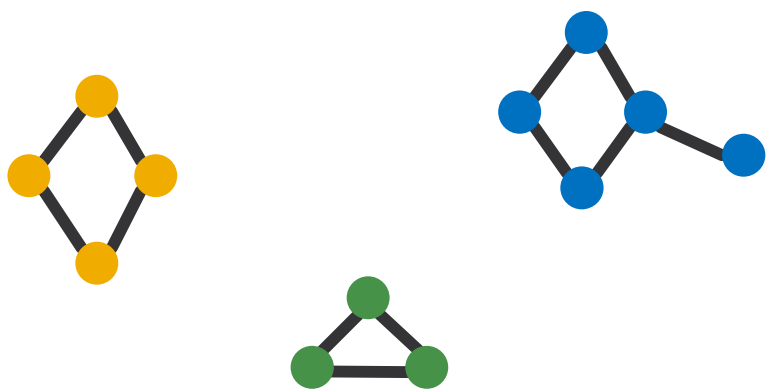
- Let G_Q be a small graph, v be a node in G_Q (the "anchor") and G_T be a target graph dataset.
- **Node-level Subgraph Frequency Definition:** The number of nodes u in G_T for which some subgraph of G_T is isomorphic to G_Q and the isomorphism maps u to v
- Let (G_Q, v) be called a **node-anchored**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





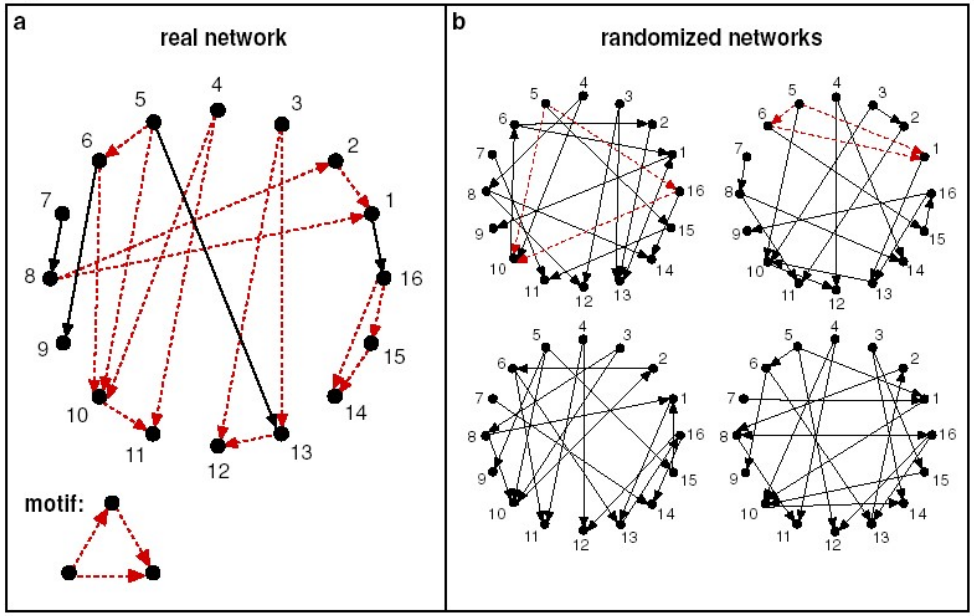
Subgraph Frequency (3)

- **What if the dataset contains multiple graphs, and we want to compute frequency of subgraphs in the dataset?**
- **Idea:** 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



Milo *et. al.*, Science 2002

Subgraph Significance: Null graph → Artificially generated

Real Graph → Real world graph.

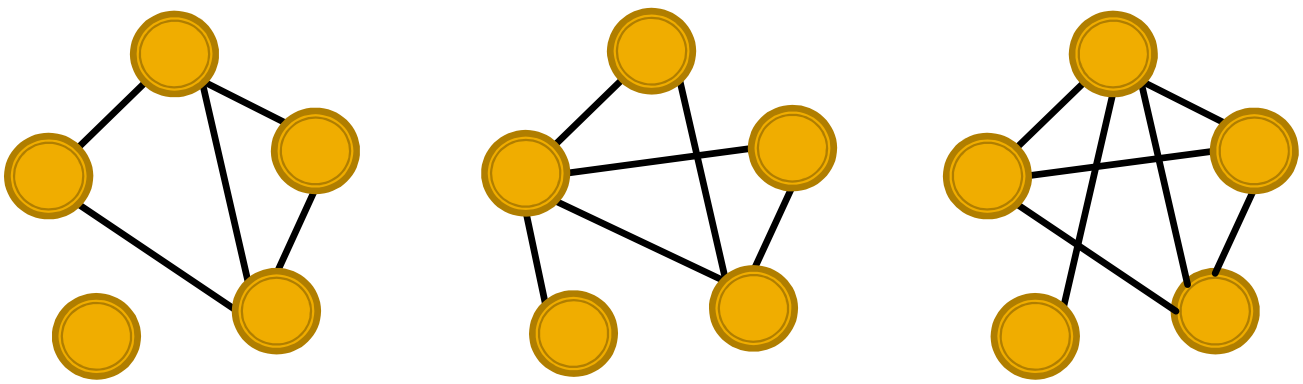
when frequency of subgraph pattern is more on real graph than null graph then that pattern has functional significance.

Defining Random Graphs (Null graph)

iid: Independent and identical distribution

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
- Can be disconnected:



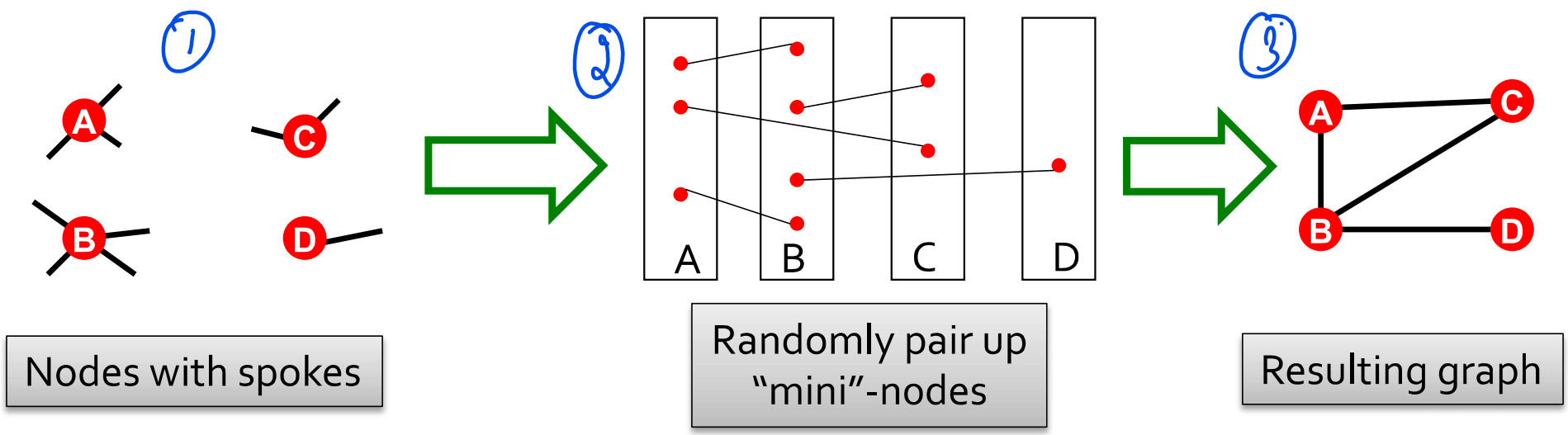
Three random graphs drawn from $G_{5,0.6}$

5 node with edge probability 0.6

For each edge biased coin of 0.6 tossed to get if edge exist.

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:**



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

In this degree sequence taken from real graph and random graph generated.

step: (1) Create node with spokes same as degree sequence.

(2) For each node create dots as shown in image-2

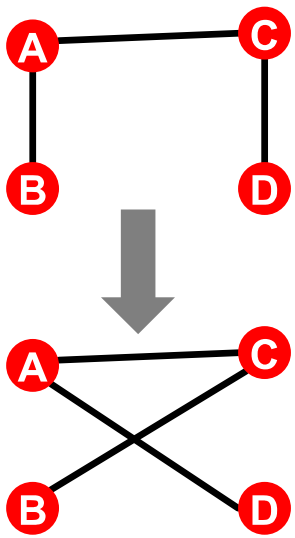
(3) Connect dots randomly.

(4) In case of parallel edge, only single edge will be counted during network generation.

(5) Generate the network as shown in image-3

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

Now we have seen ER null model and configuration model.
We normally preffere configuration model as this is more near to real one.

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**

We are finding statistical significance of presence of motif in both real and ~~star~~ null graph. If we find motif is over represented in real graph in comparison to null graph then we can say motif has functional significance.

To do this we 1st select some subgraph pattern in real graph and count the motif. Then generate lot of random graph with same statistics and count motifs in these graphs. Using Z-score we will try to understand statistical significance.

Now let us understand how Z-score is computed:

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

For Z-score computation following calculated.

1. N_i^{real} : count i^{th} patter occurance in real graph
2. $\bar{N}_i^{rand}$: For each randomly generated graph count pattern i then take avg.
3. N_i^{rand} : i^{th} patter count in random graph.
4. $std(N_i^{rand})$:- find std deviation of N_i^{rand}

$$Z_i = \left(N_i^{real} - \bar{N}_i^{rand}\right) / std\left(N_i^{rand}\right) \Rightarrow \text{Z score for } i^{th} \text{ motif.}$$

We find network significance profile for all possible motifs. For this we normalize each Z_i with all possible motif by $\sqrt{\sum_{j=1}^{K} Z_j^2}$ where j is the jth motifs and we take sum of square of all motifs. let K is total motifs then sum over j from 1 to K.

$$SP_i = \frac{Z_i}{\sqrt{\sum_{j=1}^{K} Z_j^2}}$$

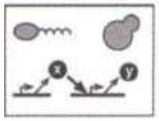
Significance Profile

- **For each subgraph:**
  - z-score metric is capable of classifying the subgraph "significance":
    - Negative values indicate **under-representation**
    - Positive values indicate **over-representation**
    - Zero – No significance
- 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

Network significance profile

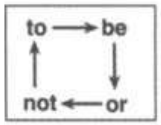
Gene regulation networks

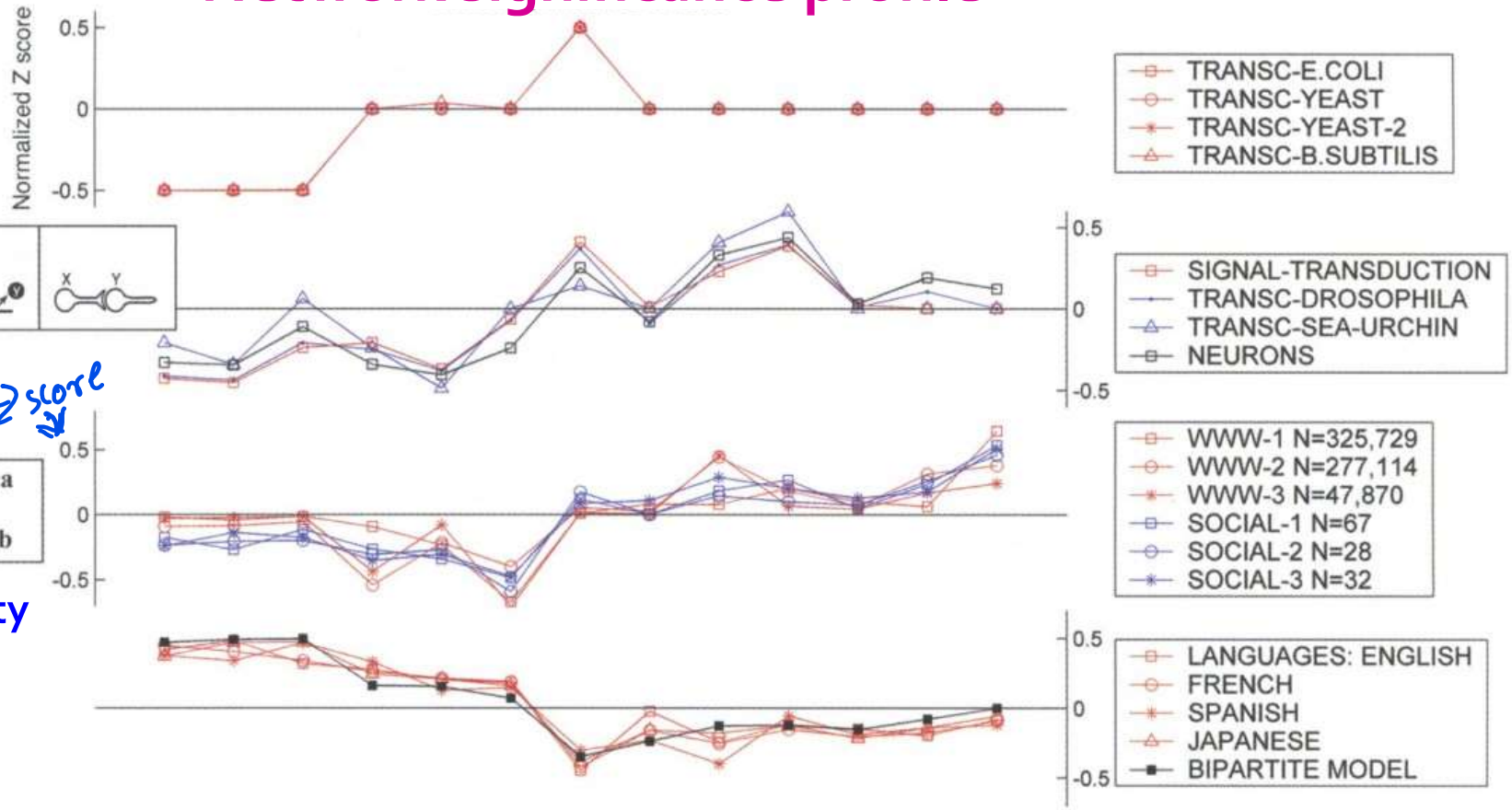


Neurons

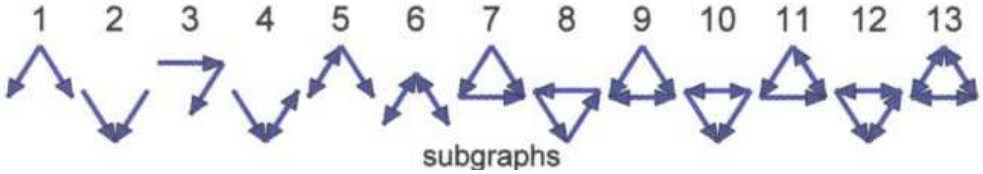
Web and social

Word connectivity





Z score



→ Motifs.

Milo *et al.*, Science 2004

Networks from the same domain have similar significance profiles

From above example we see different network have different motif significance

e.g in social network (WWW, SOCIAL) we motif is more significant because if i am connected with 2 friends ~~heavily~~ closly then they will also be connected. motif is not significate in this because maintaing very close relation with 2 different people who are not connected is very rare.

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**

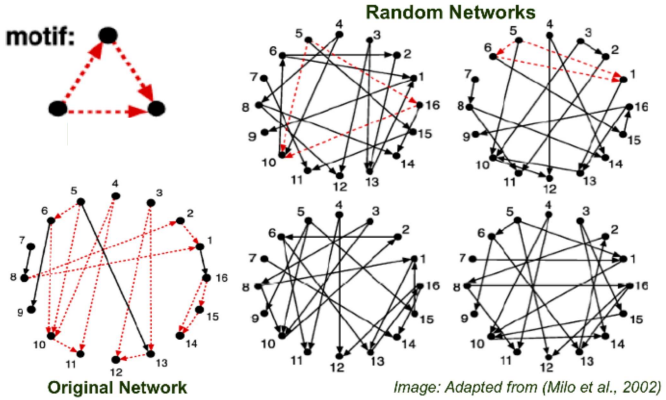
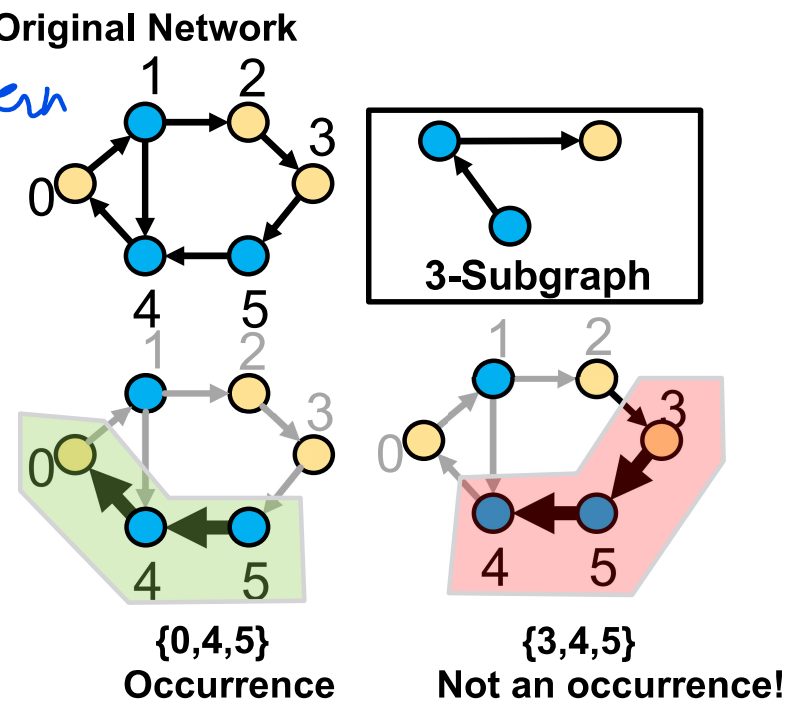


Image: Adapted from (Milo et al., 2002)

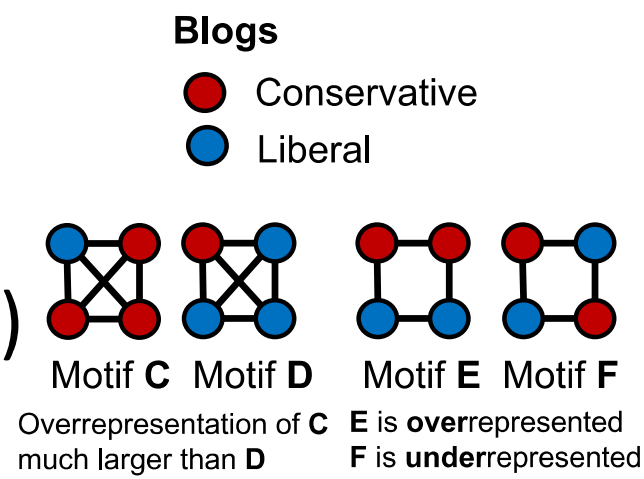
Variations on the Motif Concept

- **Extensions:** How to define motif pattern
  - 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

How to define



Many ways motifs can be defined as shown above, there is many variation in terms of extension and concept possible which open many research area in this.

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**

Stanford CS224W: Neural Subgraph Matching

CS224W: Machine Learning with Graphs

Jure Leskovec, Stanford University

http://cs224w.stanford.edu

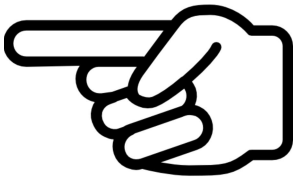
Till now we have seen the notion of subgraph and motif, how to count frequency and their significance. Now we will see the neural network method to quickly check if given graph is subgraph of network ~~of~~ or not.

Plan for Today

1) Subgraphs and Motifs

- Defining Subgraphs and Motifs
- Determining Motif Significance

2) Neural Subgraph Representations



3) Mining Frequent Motifs

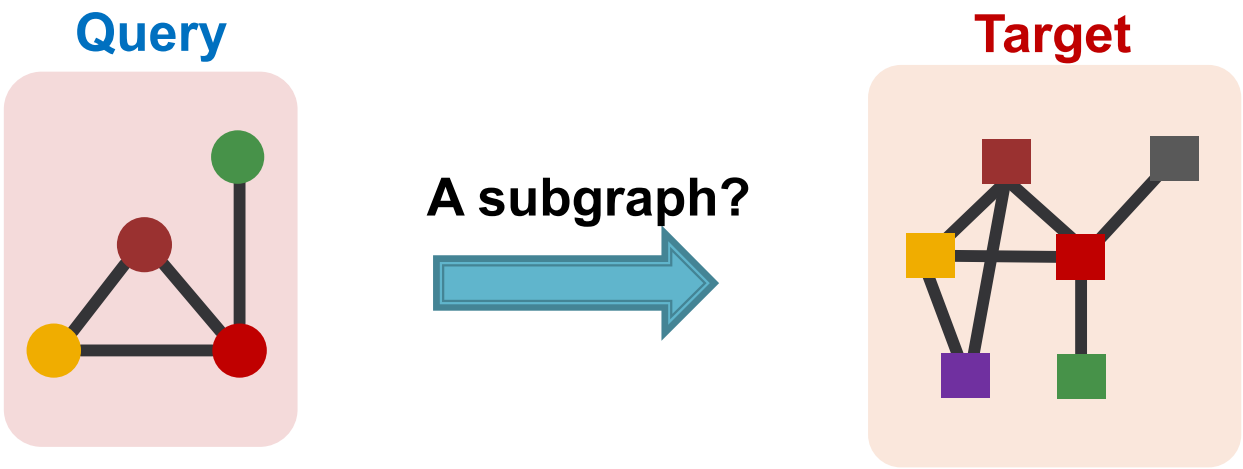
Subgraph Matching

Given:

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

Decide:

- Is query graph a subgraph in the target graph?

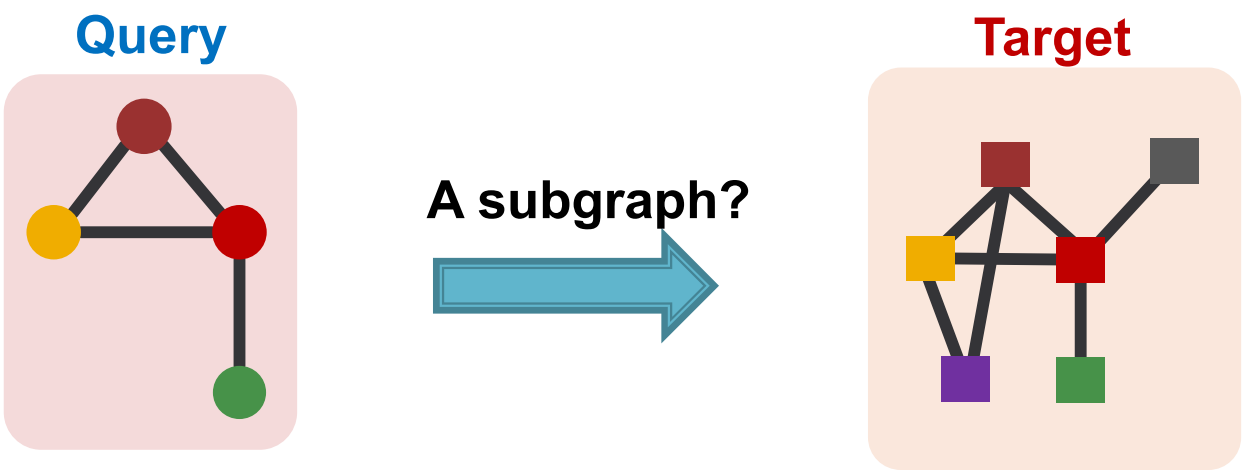


- Node colors indicate the correct mapping of the nodes

Let's define the problem like query graph and large target graph.
We need to find if query graph exist in large target graph or not.
Graph matching is isomorphism problem but we do not take combinatorial approach e.g matching edge by edge to determine the isomorphic or not but using neural network method as shown below
This problem will be developed as machine learning approach and develop this as prediction task.

Isomorphism as an ML Task

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

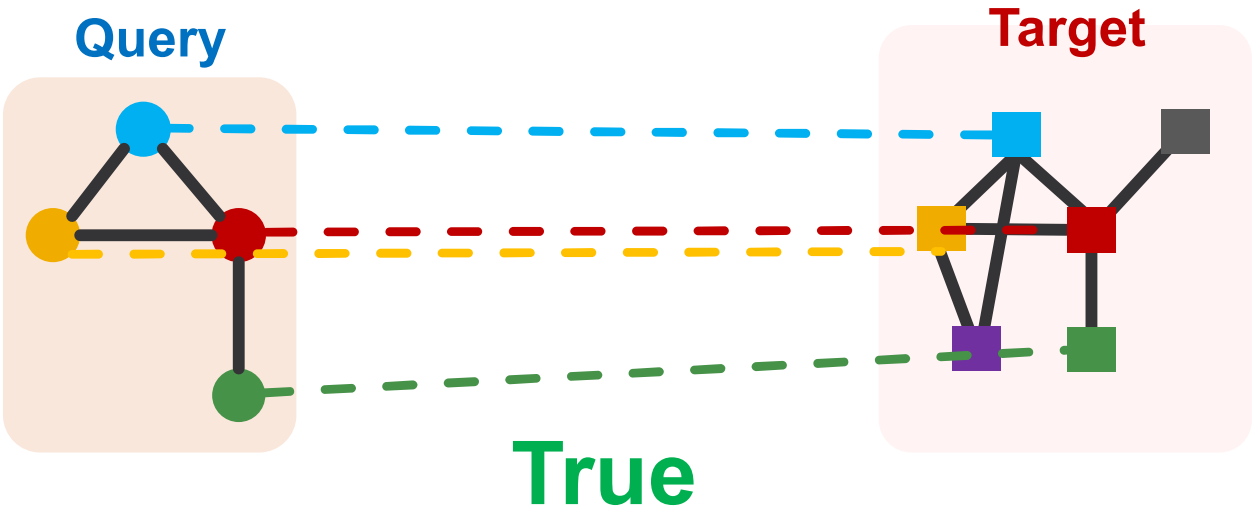


- **Intuition:** Exploit the **geometric shape** of embedding space to capture the properties of subgraph isomorphism

Here intuition is to exploit the geometric shape of embedding space to capture 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 all pairs of correspondences is another challenging problem, which will not be covered in this lecture.

only true or false

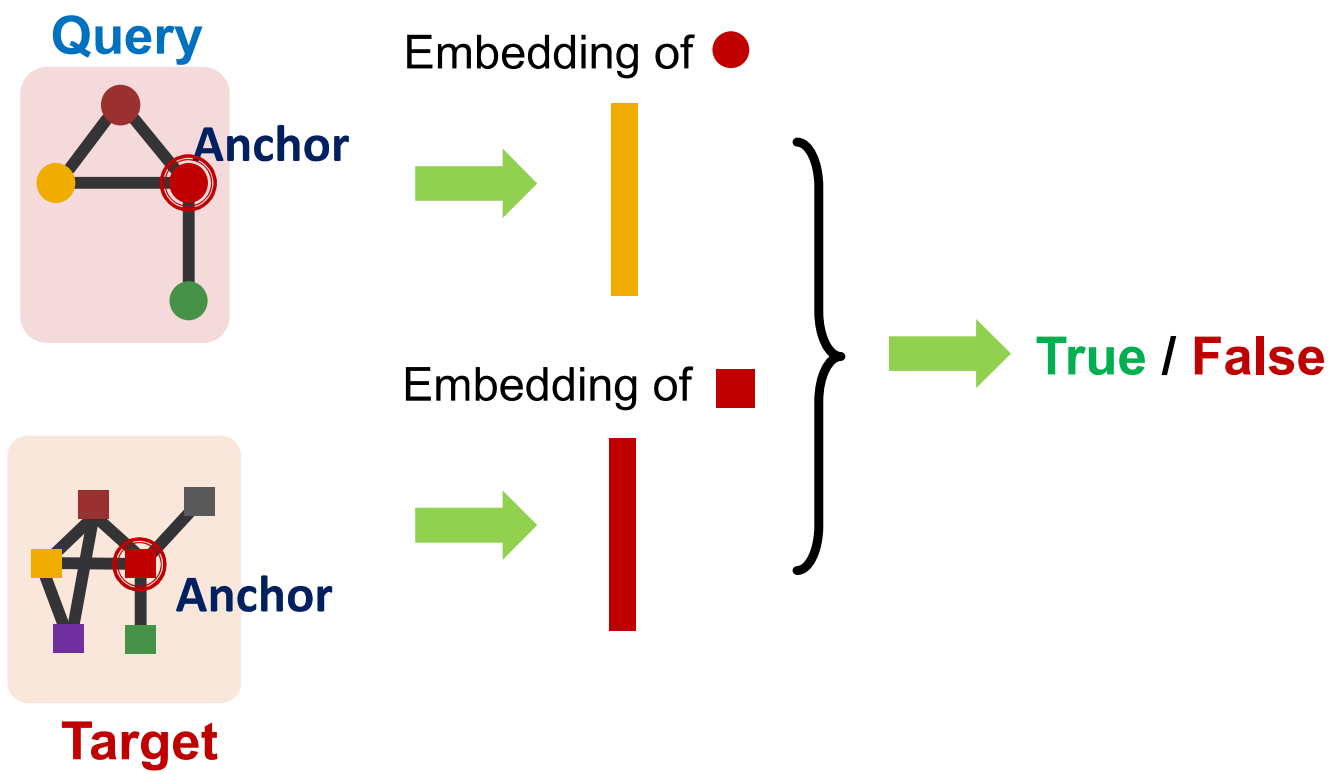
Overview of approach:

1. We decompose large graph into small set of subgraph called decomposition into neighbourhood.
2. Generate embedding for each subgraph
3. Generate embedding of query graph
4. Predict from generated embedding weather query graph is part of subgraph or not?

By this approach predicting TRUE or FALSE is superfast.

Neural Architecture for Subgraphs (1)

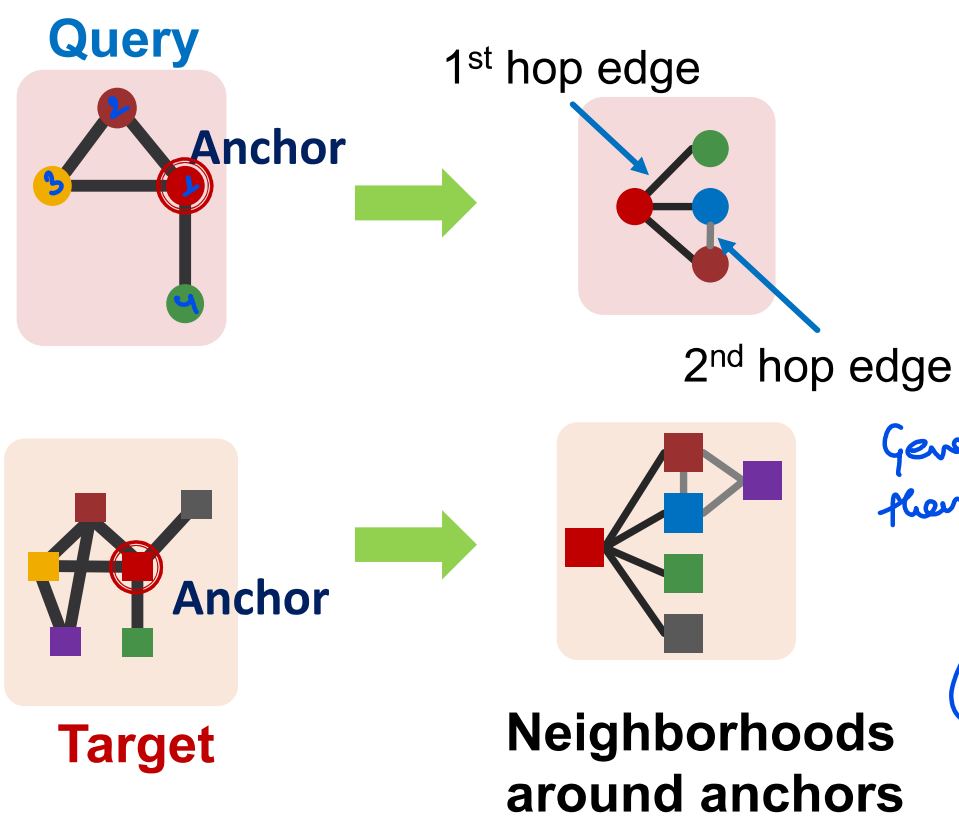
- **(1)** We are going to work with **node-anchored definitions**:



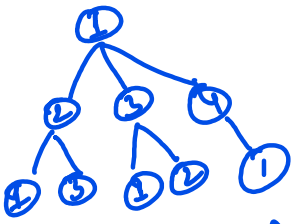
we are using node-anchored subgraph method to check subgraph. In this we take one anchored node in target-graph and take k-hop neighbourhood around that. Then from this neighbourhood we create embedding. Same embedding generated for query graph and then do prediction.

Neural Architecture for Subgraphs (2)

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



Computation graph.



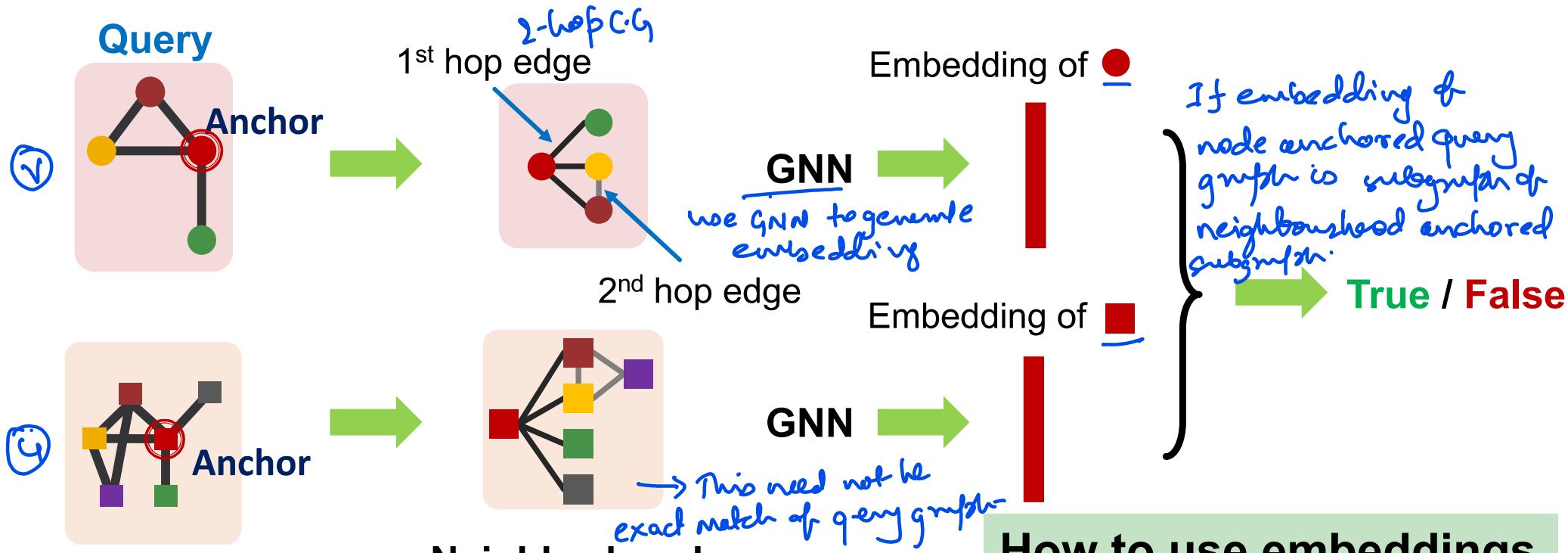
This covers all edges.

Generate all possible 2-hop C.G and then do prediction.

(generate many node-anchored neighborhood subgraphs)

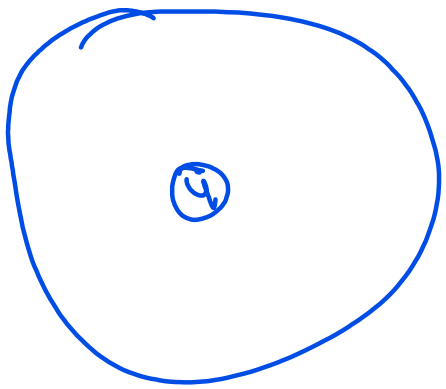
Neural Architecture for Subgraphs (3)

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



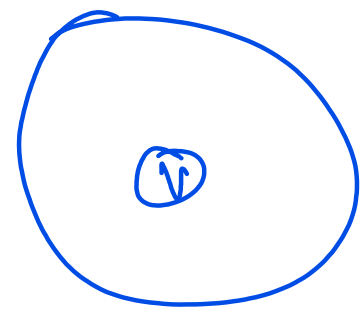
How to use embeddings to make predictions?

Large graph



G_T

Query Graph



G_Q

Q The subgraph around anchored node u match with graph around node v

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 Anchor and remaining edges has to map.
- 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)!

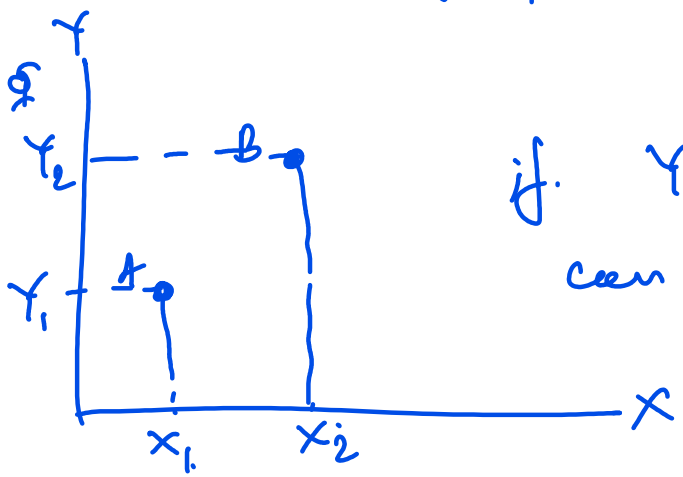
We use anchore node because in GNN embedding can be created. This embedding will be used to decide if neighbourhood of node u and neighbourhood of node v is isomorphic.

Now see how to decompose G_T into neighborhood:

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

After generation of embedding there is cost method to compare if embedding of G_Q is subgraph of G_T (Node anchored subgraph) using order embedding space.



if. $Y_1 < Y_2$ and $X_1 < X_2$ then in graph we can place as shown in graph called lower left.

Incase of higher dimension e.g 64, 128 then all corresponding co-ordinate of one node is less than other then that node can be placed on lower left. This is called node ordering.

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**

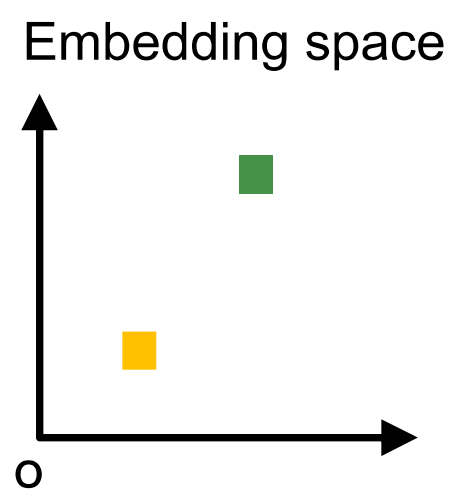
Capture partial ordering (transitivity):

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

  (64-dim) 64-dim

- If A ■ $\preccurlyeq$ B ■, B ■ $\preccurlyeq$ C ■ then A ■ $\preccurlyeq$ C ■ (Transitive)

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

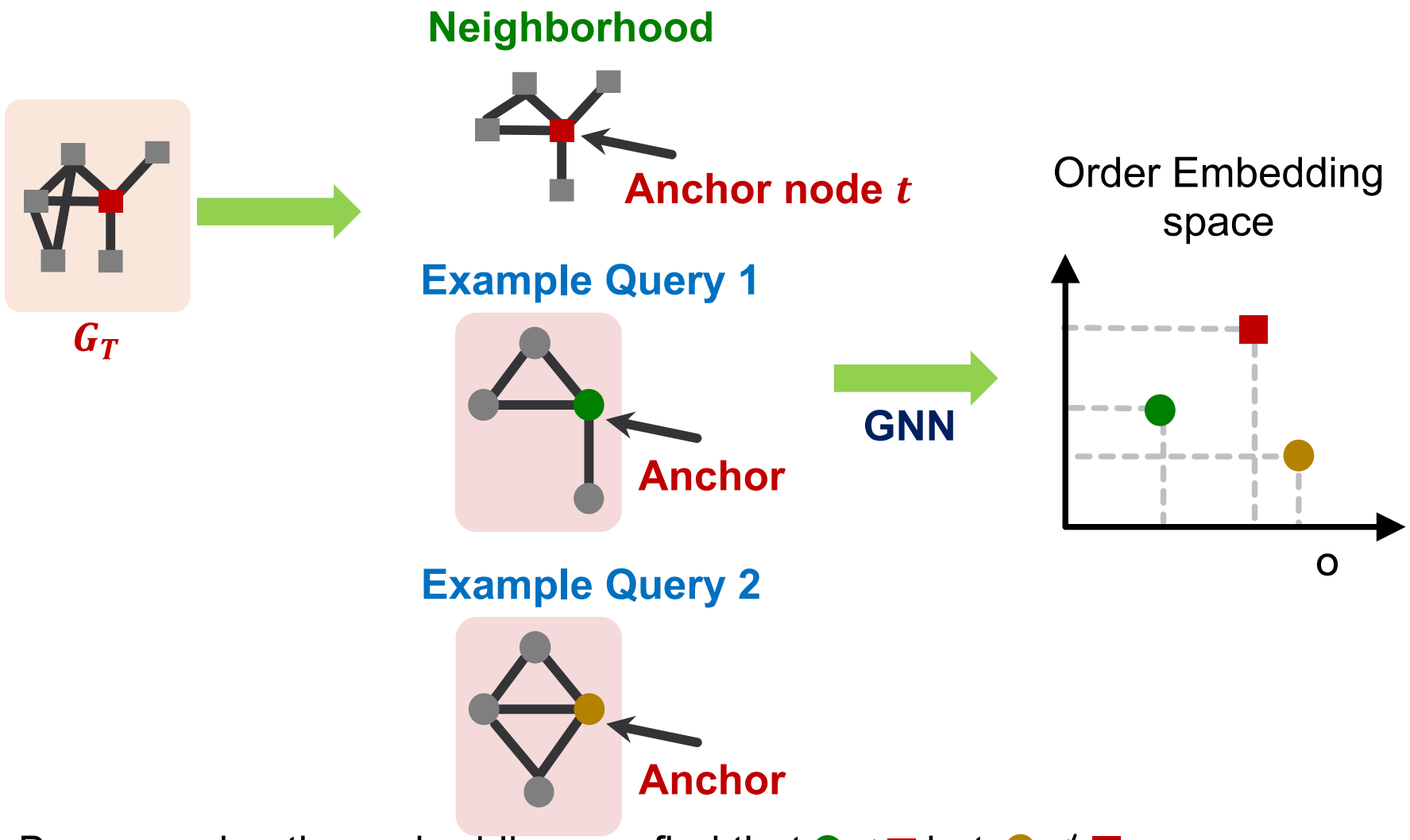


example : 3-dim embedding space

$$\left.\begin{aligned} A &= [10, 15, 20] \\ B &= [15, 20, 25] \\ C &= [20, 25, 30] \end{aligned}\right\}$$

then C is lower left of B and B is lower left of A. C is also lower left A. ~~so~~ This is because of transitivity.

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

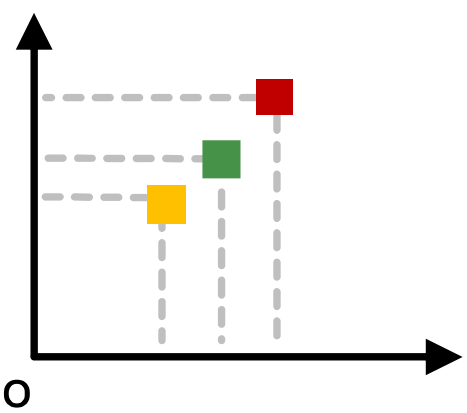
Why Order Embedding Space?

- **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**

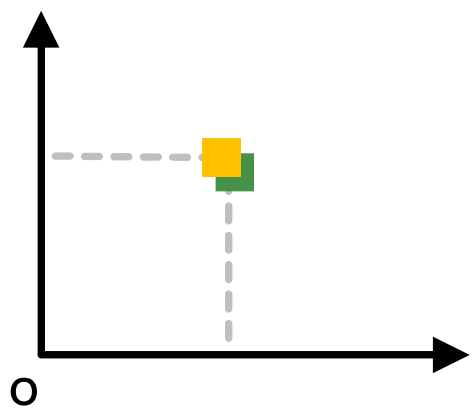
Why Order Embedding Space?

- **Subgraph isomorphism relationship** can be nicely encoded in order embedding space
  - **Transitivity**: if ■ $\preccurlyeq$ ■, ■ $\preccurlyeq$ ■ then ■ $\preccurlyeq$ ■
  - **Anti-symmetry**: if ■ $\preccurlyeq$ ■ and ■ $\preccurlyeq$ ■ , then ■ $=$ ■
  - **Closure under intersection**: the 0 embedding satisfies 0 $\preccurlyeq$ ■ for any order embedding ■ since all dimensions of order embedding are non-negative
    - Corollary: if ■ $\preccurlyeq$ ■ and ■ $\preccurlyeq$ ■ then ■ has a valid embedding But ■ $\not\preccurlyeq$ ■

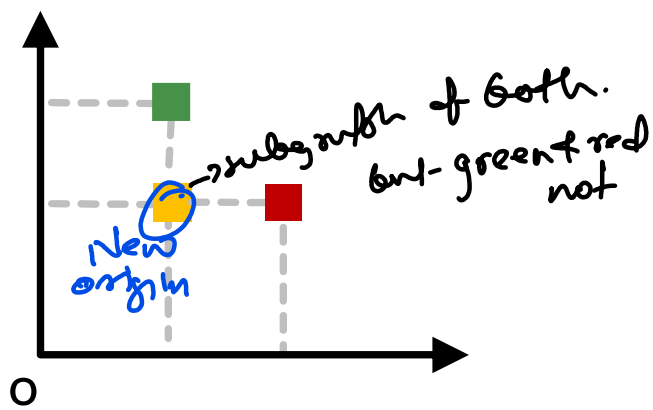
Transitivity



Anti-symmetry



Closure under intersection



Transitivity or lower left order embedding relation shows that the upper graph embedding will contain same patter as subgraph on lower left containing.

Closer under intersection checks which subgraph is common between two graph.

We need to learn the anchored node embedding such that order embedding is preserved. what should be loss function which reflects subgraph structure. We see these in below slide.

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

[lower left order]

we use order constraint in defining loss function.

Order Constraint (2)

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

Embedding dimension

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

Query embedding

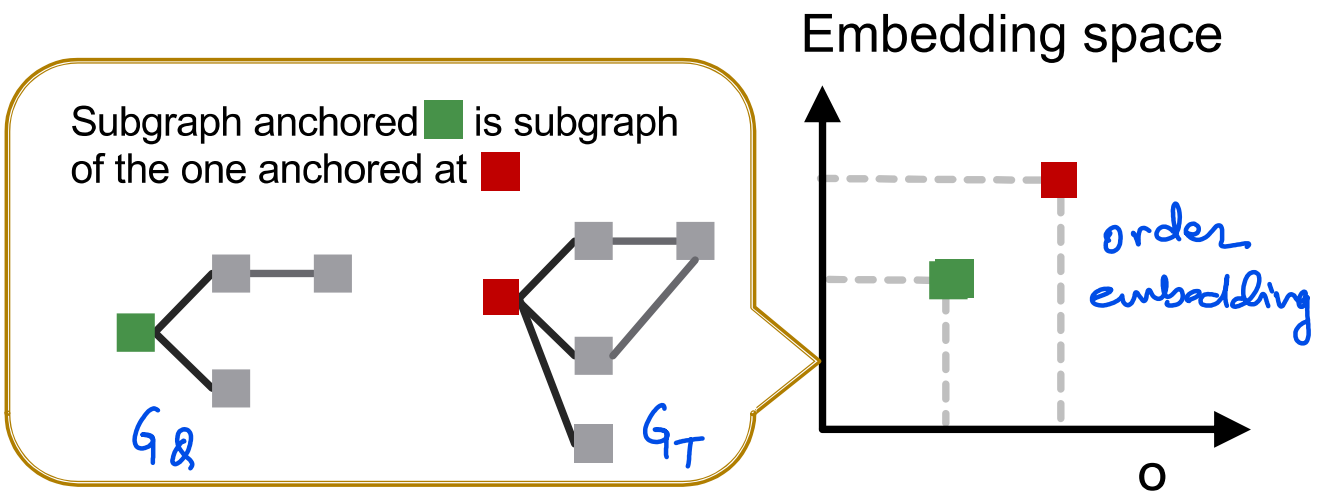
Target embedding

Subgraph Relation

trained with max-margin loss

Explained in next slide

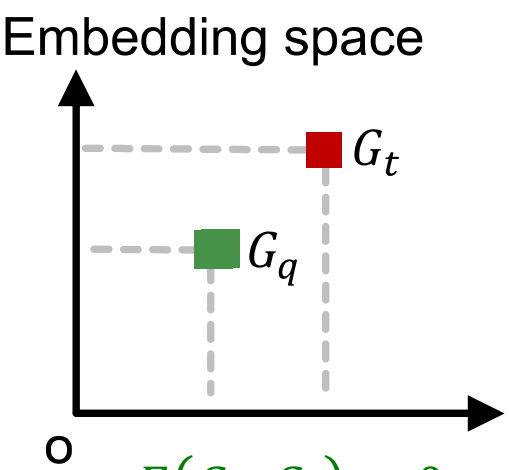
If G_Q is subgraph of G_T then all coordinate $z_q[i]$ should be less than or equal to all co-ordinate of $z_u[i]$



To quantify the violation of order constraint, we define loss function ~~as~~ called max-margin loss. For this we take maximum of 0 or difference between all pair co-ordinate of two embedding as shown in next slide.

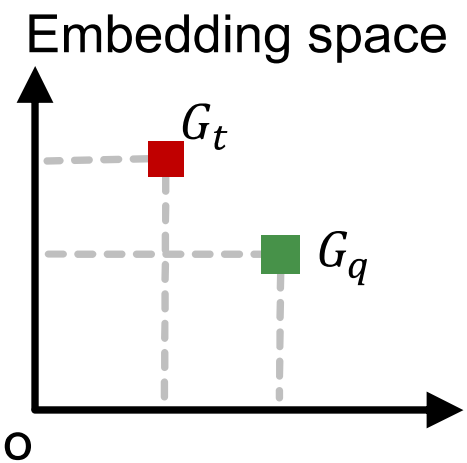
Loss Function: Order Constraint

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



$E\left(G_q, G_t\right) = 0$

According to the order embedding, G_q is a subgraph of G_t!



$E\left(G_q, G_t\right) > 0$

According to the order embedding, G_q is **not** a subgraph of G_t!

$\max(0, Z_q[i] - Z_t[i])$ will be zero if Z_q will be in lower left. This happens because $Z_q[i] \leq Z_t[i]$ if Z_q is lower left. Since this is lower left then $Z_q[i] - Z_t[i]$ is negative so $\max[0, (-ve) \text{ value}] = 0$

But if Z_q is not lower left then some value will be positive and margine $E(G_d, G_T)$ will be positive. We do this for all training set. since $E(G_d, G_T)$ is differentiable so we can back propogate and update the parameter to minimize loss.

Loss Function

- Embeddings are learned by minimizing a **max-margin** loss
- Let $E\left(G_q, G_t\right) = \sum_{i=1}^{D}(\max(0, z_q[i] - z_t[i]))^2$ be the "margin" between graphs G_q and G_t
- **To learn the correct order embeddings, we want to learn z_q, z_t such that**
  - $E\left(G_q, G_t\right) = 0$ when G_q **is** a subgraph of G_t
  - $E\left(G_q, G_t\right) > 0$ 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

In training 50% positive example where Z_q is subgraph of Z_t so minimize $E(Z_q, Z_t)$ and 50% negative training where we maximize penalty $E(Z_q, Z_t)$ because Z_q is not subgraph of Z_t

For this we use hing-loss kind of function to convert maximization to minimization.

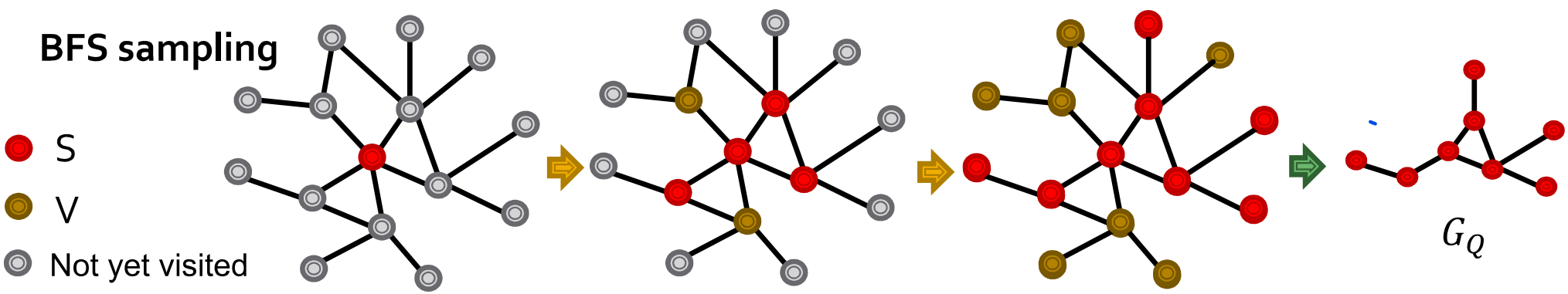
$$E = \text{Minimum}\left[\text{Max}(0, \alpha - E(G_q, G_t))\right]$$

where α is margine.

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
- Use **BFS sampling** to get G_Q. Sample induced subgraph of G_T:
  - Initialize $S = \{v\},\ V = \emptyset$ N(S) = Neighbor of node v in S
  - Let $N(S)$ be all neighbors of nodes in S. At every step, sample 10% of the nodes in $N(S) \setminus V$ and place them in S. Place the remaining nodes of $N(S)$ in V. → 90 %
    Probablistic
  - After K steps, take the subgraph of G induced by S anchored at q
- For 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



Process intuition: From Graph G we create many G_T (Neighbourhood subgraph). This is done by randomly choosing node v in G and keeping k-hop nodes.

From this subgraph G_T we generate G_Q. This is done as follows

1. Initialize $S = \{v\}$ (Any random node in G_T) and set $V = \{\phi\}$: Empty set.
2. Create set N which will have neighbours of all node present in S
3. Take 10% sample from S & V set
4. Put 10% sample node in S and remaining in V set
5. Again take neighbours of all node present in S
6. Repeate step 3, 4, 5 upto k-steps
7. Create subgraph from set S and anchor at node q in that.

Negative sample is just corrupting G_Q by adding or removing nodes and edges.

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 $E(G_q, G_t) < \epsilon$, 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