Subgraph and Motif

Yujia Zhu

Subgraphs and motifs are the building blocks of the graph. Analyzing subgraphs and motifs can help us understand how graphs work and predictions based on presence or lack of presence in a graph dataset.

Subgraph

We have two ways to define subgraph of a given graph $G = (V, E)$.

Def 1.(Node-induced subgraph): Take subset of the nodes and all edges induced by nodes:$G' = (V', E')$ is a node-induced subgraph if and only if:

1. $V' \subset V$
2. $E' = \{(u, v) \in E | u, v \in V'\}$

Def 2.(Edge-induced subgraph): Take subset of the edges and all corresponding nodes:$G' = (V', E')$ is a edge-induced subgraph if and only if:

1. $E' \subset E$
2. $V' = \{u \in V | (u, v) \in E' \text{ for some } u\}$

Two definitions have their own advantages and which to choose should depend on the domain knowledge.

Graph Isomorphism is a critical concept to show whether two graphs are identical. Suppose we have two graphs: $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. G_1 and G_2 are isomorphism if there exists a **bijection** function $f : V_1 \to V_2$ such that $(u, v) \in E_1 \iff (f(u), f(v)) \in E_2$. Currently, we do not know if graph isomorphism problem is NP-hard, nor is any polynomial algorithm found for solving graph isomorphism problem.

Subgraph Isomorphism: G_2 is subgraph-isomorphism to G_1 if some subgraph of G_2 is isomorphism to G_1. Commonly, we say G_1 *is a subgraph of* G_2. Note, subgraph here can be either node-induced subgraph or edge-induced subgraph. The problem of subgraph isomorphism is NP-hard.

Motif

Motif is another important concept in graph. Motif is *recurring and significant patterns of interconnections*. This means motif only care about interconnections but node type will not be taken into consideration. So, sometimes we call motif as node-induced subgraph. Suppose we have a graph $G : A - B - A - A - B$. Then the subgraph $A - B$ occurs three times in the graph but motif $X - Y$ emerges at every edges in the graph so the motif occurs five times.

As is defined previously, motif has two properties: recurring and significant. Recurring means the motif occurs in the graph with high frequency. But frequency gives the absolute measurement (Big graphs have large number and small graphs have small number). Significant means the motif is more frequent than expected. So, significance is a relative measurement and we need to have a null model/ basis.

We usually use random graph to portray the null model. This means motif is the subgraph that occurs in a real network much more often than in a random network. But, we want the simulated random graph is much similar to the real graph. Besides having the same number of nodes and edges, a way to imporve similarity is to ensure the real graph and random graph have the same degree sequence. Simulated random graphs

with given degree sequence are called configuration model. Suppose we have simulate k graphs, then we can compute the significance of the motif using **Z-score**:

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

Z_i captures the statistical significance of motif i. N_i^{real} is the number of motif i in graph G^{real},$\bar{N}_i^{\text{rand}}$ and $std(N_i^{\text{rand}})$ are respectively the mean and standard deviation of the number of motif i in graph set $\{G^{\text{rand}}\}$. Positive Z-score means the motif is overrepresented, while negative Z-score means underrepresentation. **Network significance profile**(SP) is the vector of the normalized Z-score, which is defined as $SP_i = Z_i/\sqrt{\sum_j Z_j^2}$.

Subgraph Matching

Subgraph Matching is to query whether a graph is a subgraph of another graph. We are given a large *target graph* (which can be disconnected) and a *query graph* (which is connected). We need to decide whether query graph is a subgraph in the target graph or not.

GNN can be used to solve this problem. For the node u in the query graph, we can use GNN to get the information of the node and its neighbours. The embedding of node u will be get (so, GNN here is a multi-layer node embedding). For each nodes in the target graph, we can also get their embeddings. With embeddings, we can decide the existence of subgraph.

The GNN maps the node to the embedding. But, we can use a better embedding space to quickly solve subgraph matching problem. **Order Embedding Space** directly embeds the graph into the embedding space. Namely, order embedding space maps graph A into a high-dimensional (e.g. 64-dim) embedding vector z_A. The embedding space has several special properties:

1. z_A is non-negative in all dimensions
2. If z_A is less than or equal to z_B in all of its coordinates, then we donate $z_A \leq z_B$. Order Embedding Space has the property that A is a subgraph of B if and only if $z_A \leq z_B$

Note that subgraph isomorphism relationship can be nicely encoded in order embedding space. Subgraph isomorphism has three main properties:

1. **Transitivity**: if G_1 is a subgraph of G_2 and G_2 is a subgraph of G_3, then G_1 is a subgraph of G_3
2. **Anti-symmetry**: if G_1 is a subgraph of G_2 and G_2 is a subgraph of G_1, then G_1 is isomorphism to G_2
3. **Closure under intersection**: the trivial graph of 1 node is a subgraph of any graphs

These three properties can be well explained in the order embedding space (note that $\leq$ symbol means less than or equal to in all of its coordinates):

1. **Transitivity**: if $z_A \leq z_B$ and $z_B \leq z_C$ then $z_A \leq z_C$
2. **Anti-symmetry**: if $z_A \leq z_B$ and $z_B \leq z_A$ then $z_A = z_B$
3. **Closure under intersection**: the embedding of 1-node graph is 0, then $0 \leq z_A, \forall A$ for all graph A since all dimensions of order embedding are non-negative

We can use a GNN to learn to embed neighborhoods and preserve the order embedding structure by a specified loss function. The design of loss function should be based on order constraints. Order constraint specifies the ideal order embedding property that reflects subgraph relations.

Suppose the dimension of embedding space is D, then graph G_Q is a subgraph of G_T ($G_Q \subset G_T$) if and only if $z_Q[i] \leq z_T[i], \forall i = 1, 2, ..., D$. So,we can use a max-margin loss to train the GNN. The loss function is as follows:

$$E(G_q, G_t) = \sum_{i=1}^{D} [\max(0, z_q[i] - z_t[i])]^2$$

To learn the correct order embeddings, we want to learn z_q, z_t such that:

- $E(G_q, G_t) = 0$ if G_q is a subgraph of G_t
- $E(G_q, G_t) > 0$ if G_q is not a subgraph of G_t

This is for positive samples. Minimizing $E(G_q, G_t)$ is required for positive samples. But we need negative samples as well. For negative samples, we should minimize $\max(0, \alpha - E(G_q, G_t))$.

Sample construction is another important procedure for order embedding space learning. We should construct samples $\{(G_q, G_t)\}$ where half the time G_q is a subgraph of G_t and the other half, it is not. The generation of G_q and G_t is from the dataset G. Suppose G_t is in the dataset G, we can use **BFS Sampling** to get positive sample G_q which is subgraph of G_t. BFS Sampling has following procedures:

1. Random choose a node v form G_t and initialize $S = \{v\}, V = \emptyset$
2. Let $N(S)$ be all the neighbouring nodes of nodes in S. At each time, sample 10% of nodes in $N(S) - V$ and place them in S. Place the remaining nodes in V
3. After K steps, take the subgraph of G_t induced by nodes S as G_q

Positive samples can be got through BFS sampling. Negative samples can be corrupted from positive samples by adding or removing edges/nodes. BFS sampling will be done throughout the learning process. At every iteration, new positive and negative samples will be provided. This means the model will see different subgraph examples to improve performance and avoid overfitting. K is a hyperparameter in BFS sampling which decides the depth of the sampling. It is a hyperparameter that trades off running time and performance. Usually, we set K as 3 to 5, but we should also consider the size of dataset when making decisions.

At the test time, we can decide whether G_Q is a subgraph of G_T by the criterion $E(G_Q, G_T) < \epsilon$, where ϵ is a hyperparameter. Embedding graphs within an order embedding space allows subgraph isomorphism to be efficiently represented and tested by the relative positions of graph embeddings.

Find Frequent Motifs/Subgraphs

Now, we are going to find the most frequent k motifs in a given graph. To solve this problem, we need to overcome two challenges. The first is to enumerate all size-k connected subgraphs and the second is to count the number of occurrences of each subgraph type. Both two challenges are hard to handle because they are computionally hard.

SPMiner is a neural model to identify frequent motifs. Suppose we have a graph G_T and size parameter k. We want to identify the r graphs with the highest frequency in G_T among all possible graphs of k nodes. The key idea is to decompose input graph G_T into neighbourhoods and embed neighborhoods into an order embedding space. The benefit of order embedding is to quickly find out the frequency of a given subgraph.

Suppose we have a set subgraphs G_{N_i} of G_T, then we can estimate frequency of motif G_Q by counting the number of G_{N_i} such that their embeddings z_{N_i} satisfy $z_Q \leq z_{N_i}$. The step of SPMiner is as follows:

1. Start by randomly choose a random node u in the target graph G_T. Set $S = \{u\}$. Since now S(the motif) only contains one node, so its location in order embedding space is near the left-bottom corner.
2. Iteratively grow a motif by choosing a neighbour of the node in S and add the node to S. We choose node by the criterion of maximizing the number of neighbourhoods in the right-top area of order embedding space.
3. Terminate until reaching a desired motif size (the number of neighbourhood can not be larger). Take the node-induced subgraph by S.

When choosing the neighbour, we have different methods. One is the greedy strategy. This means we directly add the node which gives the largest number of neighbourhood. Other strategies like Monte Carlo Tree Search are also acceptable.

Summary

Subgraphs and motifs are important concepts that provide insights into the structure of graphs. Their counts can be used as features for nodes and graphs. Order embedding space is useful in subgraph matching problem and frequent motif finding problem.