Representation Learning for Frequent Subgraph Mining

Anonymous Authors-AAAI2021

Abstract

Identifying frequent subgraphs or *network motifs* has been crucial in analyzing and predicting properties of real-world networks. However, finding large commonly-occurring motifs remains an open problem due to its NP-hard subroutine of subgraph counting, and the combinatorial growth of the number of possible subgraphs. Here we present *Subgraph Pattern Miner (SPMiner)*, a novel neural approach to finding frequent subgraphs in a large target graph. SPMiner integrates graph neural networks, order embedding space, and an efficient search strategy to identify network subgraph patterns that appear most frequently in a target graph dataset. SPMiner first decomposes the target graph into many overlapping subgraphs and then encodes the subgraphs into order embeddings. SPMiner then uses a monotonic walk in the order embedding space to identify frequent motifs. Compared to existing approaches and possible neural alternatives, SPMiner is more accurate, faster, and more scalable. For 5- and 6-node motifs, we show that SPMiner can identify almost all of the most frequent motifs while being 100x faster than exact enumeration methods. In addition, SPMiner can also reliably identify frequent 10-node motifs, which is well beyond the size limit of exact enumeration approaches. And last, we show that SPMiner can find large 10+ node motifs that appear 10-100x more frequently than those found by current widely-used approximate methods.

1 Introduction

Finding subgraph patterns or *network motifs* that frequently recur in a graph dataset is important for identifying interpretable structural properties of complex networks (Benson, Gleich, and Leskovec 2016). Frequent subgraph mining is a challenging but important task in network science: given a target graph or a collection of graphs, it aims to discover subgraphs or motifs that occur frequently in this data (Kuramochi and Karypis 2004). In biology, subgraph counting is highly predictive for disease pathways, gene interaction and connectomes (Agrawal et al. 2018; Hočevar and Demšar 2014). In social science, subgraph patterns have been observed to be indicators of social balance and status (Leskovec, Huttenlocher, and Kleinberg 2010). In chemistry, common substructures are essential for predicting molecular properties (Cao, Jiang, and Girke 2008; Cereto-Massagué et al. 2015).

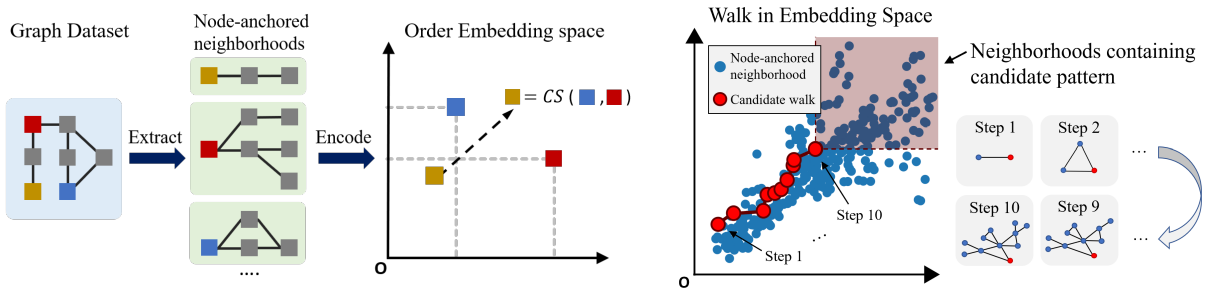
However, frequent subgraph mining has extremely high computational complexity, since it encompasses subgraph isomosphism, the NP-hard problem of determining if a query motif Q is a subgraph of a target graph G (Ribeiro et al. 2019). A traditional approach to motif mining is to enumerate all possible motifs Q of size up to k, usually $k \leq 5$, and then count appearances of each Q in a given dataset (Hočevar and Demšar 2014).

More recently, neural approaches to learning combinatorially-hard graph problems have been explored. Related tasks include prediction of edit distance (Bai et al. 2019), graph isomorphism (Fey et al. 2020; Guo et al. 2018; Li et al. 2019b; Xu et al. 2019a,b), pairwise maximum common subgraphs (Bai et al. 2020) and substructure counting (Chen et al. 2020). Most recently, NeuroMatch (Ying et al. 2020) focuses on subgraph isomorphism testing, which attempts to predict whether a single query motif Q is a subgraph of a large target graph G. While subgraph isomorphism is an important subroutine, frequent subgraph counting is much more challenging because it requires solving two intractable search problems: (1) counting the frequency of a given motif Q in G; (2) searching over all possible motifs Q to identify the frequent ones. Problem (1) is NP-hard, while Problem (2) is also hard because the number of possible graphs Q increases super-exponentially with their size.

Here we propose *Subgraph Pattern Miner (SPMiner)*, a general framework using graph representation learning for identifying frequent motifs in a large target graph. To the best of our knowledge, SPMiner is the first neural approach to mining frequent subgraphs. SPMiner consists of two steps: an encoder for embedding candidate subgraphs and a motif search procedure.

Encoder: Embedding Candidate Subgraphs. The encoder is an expressive graph neural network (GNN) with trainable dense skip layers. We decompose the input graph into overlapping node-anchored neighborhoods around each node. The encoder then maps these neighborhoods to points in an *order embedding* space. The order embedding space is trained to enforce the property that if one graph is a subgraph of another, then they are embedded to the "lower left" of each other (Figure 1 (a)). Hence the order embedding space captures the partial ordering of graphs under the subgraph relation.

Motif Search Procedure. SPMiner then reasons in the em-



(a) SPMiner Encoder

(b) SPMiner Decoder

Figure 1: SPMiner encoder (a) and motif search procedure (b). (a) The encoder decomposes a dataset into node-anchored neighborhoods, and maps them to points in an order embedding space, such that subgraphs are below and to the left of a given graph. Here yellow node-anchored neighborhood is a common subgraph (CS) of blue and red neighborhoods, so it is embedding to the lower left of both of them. (b) SPMiner iteratively adds nodes and edges to find frequent motifs. It performs a monotonic walk in the order embedding space to identify a motif that is a subgraph of many neighborhoods. The walk in red represents growing of a frequent motif. Points in the shaded region correspond to neighborhoods containing the final motif (Step 10) as a subgraph.

bedding space to identify frequent motifs of desired size k. SPMiner searches for a k-step walk in the embedding space that stays to the lower left of as many neighborhoods (blue dots) as possible (Figure 1 (b)). The walk is performed by iteratively adding nodes and edges to the current motif candidate, and tracking its embedding. The number of subgraphs to the top right in the embedding space gives the frequency of the motif.

Evaluation. We carefully design an evaluation framework to compare performance of SPMiner to its alternative methods. Current exact combinatorial frequent subgraph mining techniques only scale to motifs of up to 6 nodes. So, we show that for 5- and 6-node motifs (where their exact count can be obtained), SPMiner correctly identifies most of the top 10 most frequent motifs while being 100x faster than exact enumeration. As present exact enumeration methods do not scale beyond motifs of size 6, we generate synthetic graphs with planted frequent motifs of size 10, and again show that SPMiner is able to robustly identify them. Last, we also compare SPMiner to approximate methods for finding large motifs with over 10 nodes, and show that SPMiner is able to identify large motifs that are 10-100x more frequent than those identified by current methods.

Overall, there are several important benefits of our approach: robustness, accuracy and speed. In particular, **(a)** SPMiner avoids expensive combinatorial graph matching and counting by mapping the entire problem into an embedding space; **(b)** SPMiner allows for a neural model to estimate frequency of any motif Q directly in the order embedding space; **(c)** Training only needs to be performed once on a proposed large synthetic dataset, and then SPMiner can be applied to any graph dataset; **(d)** The embedding space can be efficiently navigated in order to identify a large motif with high frequency.

2 Related Work

Non-ML approaches to subgraph mining. There has been an extensive line of work on both exact and approximate methods for frequent subgraph mining that involve searching the possible subgraphs by pruning or compression. Exact methods often rely on canonical labelings (Inokuchi, Washio, and Motoda 2000; Kuramochi and Karypis 2004; Yan and Han 2002) to reduce the search space. Heuristic and sampling methods offer faster execution time while no longer guaranteeing that all frequent subgraphs will be found: greedy beam search (Ketkar, Holder, and Cook 2005), pattern contraction (Matsuda et al. 2000) and subgraph sampling (Wernicke 2006) have been used to reduce the size of the subgraph search space. However, these approaches scale poorly with increasing subgraph size due to combinatorial growth of the sample space; and the performance degrades significantly for larger motifs.

There have also been works in estimating the frequency of a query motif in a target graph. Hand-crafted combinatorial techniques have been successful for small motifs (up to 5 nodes) (Hočevar and Demšar 2014), and statistical sampling has been applied (Kashtan et al. 2004). However, these methods are unscalable for the subgraph mining task, since they would require intractable enumeration of all possible motifs of a given size.

Neural approaches. Recently there are approaches that use graph neural networks (GNNs) to predict relations between graphs: models that learn to predict graph edit distance (Bai et al. 2019; Li et al. 2019b), graph isomorphism (Fey et al. 2020; Guo et al. 2018; Xu et al. 2019a,b), and substructure counting (Chen et al. 2020) have been developed. Neural models have also been proposed to find maximum common subgraphs (Bai et al. 2020) and generate prediction explanations (Duvenaud et al. 2015; Ying et al. 2019), but only in the context of a single prediction or graph pair rather than across an entire dataset. Our work here is inspired by the recent approach to neural subgraph matching called NeuroMatch (Ying et al. 2020). However, here we are solving a different and a much harder problem: our goal is not only to identify the frequency of a given motif Q in graph G but also identify motifs Q that have high frequency.

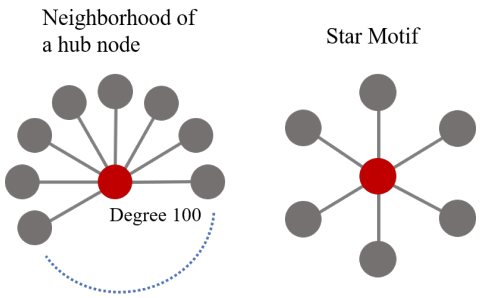


Figure 2: Distinction between Node-anchored and Graph-level subgraph frequency. Consider a hub node with degree 100 (left). We aim to determine frequency of the star motif (right). Definition 1 results in a count of 1. In contrast, Definition 2 counts in $\binom{100}{6}$ motif occurrences.

3 Proposed Method

We first introduce the problem of subgraph mining and its related subroutine of subgraph isomorphism, then define our objective of finding frequent motifs. We then introduce SPMiner, which has two components: an encoder that maps graphs into embeddings to capture subgraph relation, and a motif search procedure that identifies motifs that appear most frequently in the graph dataset.

3.1 Problem Setup

Let $G_T = (V_T, E_T)$ be a large *target graph* where we aim to identify common motifs, with vertices V_T and edges E_T. For notational simplicity, we consider the case of a single target graph. A dataset of multiple graphs could be considered as a large target graph with multiple disconnected components.

Analogously, let $G_Q = (V_Q, E_Q)$ be a *query motif*. The *subgraph isomorphism problem* is to determine whether an isomorphic copy of the query graph G_Q appears as a subgraph of the target graph G_T. Formally, G_Q is subgraph isomorphic to G_T if there exists

an injection $f : V_Q \mapsto V_T$ such that $(f(v), f(u)) \in E_T \iff (v, u) \in E_Q$ (all corresponding edges match). The function f is a *subgraph isomorphism mapping*. Following most previous motif mining literature, we focus on mining *node-induced* subgraphs, where subgraph edges are induced by the subset of nodes, but our method can also be applied to the variation of mining edge-induced subgraphs.

The problem of *frequent subgraph mining* is to identify subgraph patterns (i.e., motifs) that appear most frequently in a given dataset G_T. We focus on the case of finding *node-anchored* motifs (G_Q, v) (Benson, Gleich, and Leskovec 2016; Hočevar and Demšar 2014). Throughout the discussion, we say that G_Q anchored at $v \in V_Q$ is a subgraph of G_T anchored at $u \in V_T$ if there exists a subgraph isomorphism $f : V_Q \mapsto V_T$ satisfying $f(v) = u$. Our goal is to find most frequent motifs G_Q with associated *anchor nodes* $v \in V_Q$. We define the frequency of a node-anchored subgraph pattern as:

Definition 1. ***Node-anchored Subgraph Frequency.*** *Let (G_Q, v) be a node-anchored subgraph pattern. The frequency of motif G_Q in the graph dataset G_T, relative to anchor node v, is the number of nodes u in G_T for which there exists a subgraph isomorphism $f : V_Q \mapsto V_T$ such that $f(v) = u$.*

Although we focus on the Node-anchored frequency definition in designing methods, in experiments we additionally evaluate performance with the alternative graph-level frequency definition, and show that SPMiner works under both definitions.

Definition 2. ***Graph-level Subgraph Frequency.*** *Let G_Q be a subgraph pattern. The graph-level frequency of G_Q in G_T is the number of unique subsets of nodes $S \subset V_T$ for which there exists a subgraph isomorphism $f : V_Q \mapsto V_T$ whose image is S.*

It is important to note that in the node-anchored frequency definition, a pattern that occurs combinatorially many times in the target graph merely due to symmetry would only contribute a small count to the total (Fig. 2); Definition 1 is thus more robust to outliers than Definition 2.

We formulate the goal of SPMiner as optimization of the subgraph frequency:

Problem 1. ***Goal of SPMiner.*** *Given a target G_T, a size parameter k and desired number of results r, the goal of SPMiner is to identify, among all possible graphs of k nodes, the r graphs (i.e, motifs) with the highest frequency in G_T.*

3.2 SPMiner Encoder ϕ: Embedding Candidate Subgraphs

We first provide a high-level overview of subgraph encoding, consisting of two steps: (1) Given G_T, we decompose it by extracting neighborhoods G_v anchored at each node v. (2) The encoder ϕ uses a graph neural network (GNN) to map the neighborhoods G_v into an order embedding space (Figure 1).

Mapping node-anchored neighborhood to embedding space. Subgraph Frequency Definition 1 uses the concept of node-anchored subgraph isomorphism. We use a categorical node feature to represent whether a node is an anchor v of a neighborhood graph G_v, embed it by computing node embeddings of G_v through a GNN and aggregate into the neighborhood embedding by sum pooling.

Order embedding space. Order embedding (Vendrov et al. 2016) is a representation learning technique that uses the geometric relations of embeddings to model a partial ordering structure. Order embeddings are a natural way to model subgraph relations because subgraph isomorphism induces a partial ordering on the set of all graphs via its properties of *transitivity* and *antisymmetry*. Proofs are direct consequences of the subgraph definition and are given in Appendix.

Formally, we define a partial order $\preceq$ on the set of all graphs $\mathcal{G}$. Let $A, B \in \mathcal{G}$, and denote $A \preceq B$ if graph A is isomorphic to a subgraph of B. We learn the encoder $\phi : \mathcal{G} \mapsto \mathbb{R}^n$ that maps graphs to vectors, enforcing the *order embedding constraint* that $A \preceq B$ if and only if $\phi(A) \leq \phi(B)$ elementwise. In other words, embedding $\phi(A)$ is to the "lower left" of embedding $\phi(B)$. The *order embedding penalty* between two graphs can be defined as

$$E(A, B) = \|\max(0, \phi(A) - \phi(B))\|^2 . \quad (1)$$

We call $E(A, B)$ the *margin*. To enforce the order embedding constraint, we use this penalty in the following max-margin

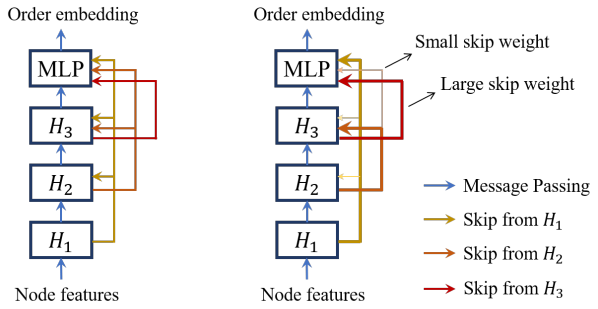


Figure 3: SPMiner Learnable skip layer. (a) Initially all skip connections are assigned equal weights. (b) After training the learnable skip GNN, the model learns the best skip connection configurations that encode subgraph relations. The architecture only requires $O(L^2)$ additional paramters.

loss:

$$\sum_{(A,B)\in P} E(A,B) + \sum_{(A',B')\in N} \max(0, \alpha - E(A',B')) \quad (2)$$

Here, P is the set of positive examples (pairs A, B where A is a subgraph of B) and N is the set of negative examples (pairs that do not satisfy the subgraph relation); α is a margin hyperparameter.

Observe that at the query time, given precomputed embeddings of A, B, we can use $E(A, B)$ to quickly determine if A is subgraph isomorphic to B, simply by checking if $E(A, B)$ is below a learned threshold. This is important as it allows us to test whether A is a subgraph of B in time linear in embedding dimension, independent of graph sizes.

SPMiner Model Architecture. It is essential for our GNN to be expressive in capturing neighborhood structures (Xu et al. 2019a). We achieve this with having a GNN of large depth. However, increasing depth can potentially degrade GNN model performance due to oversmoothing. We propose a new approach of *learnable skip layer*, based on the fully connected dense skip layers. Different from previous GNN skip layers (Hamilton, Ying, and Leskovec 2017; Li et al. 2019a; Xu et al. 2018), we use a fully connected skip layer analogous to DenseNet (Huang et al. 2017), and additionally assign a learnable scalar weight $w_{i,l}$ to each skip connection from layer i to l. Let $1 \le l \le L$ be the layer number, and H'_l be the embedding matrix at l-th layer after message passing. At each layer l, the node embedding matrix H_l is computed by:

$$H_l = \text{Concat}\left(\sum_{i=1}^{l-1} w_{i,l} H_i, H'_l\right), \ \forall l = 1, 2, \ldots, L \quad (3)$$

When learning embeddings for subgraph isomorphism tasks, we can view representations at layer i as describing the graph structural information for the i-th hop neighborhood (Hamilton, Ying, and Leskovec 2017). Learnable skip allows every layer of the model to easily access structural features of different sized neighborhoods. This ensures that only useful skip connections across layers are retained, and different sized subgraph components (at different layers) can be simultaneously considered.

Training. To train the encoder, for each graph in a dataset, we sample a random subgraph anchored at a random node v as a target neighborhood G, and sample a smaller random subgraph as the query graph to be matched to the target neighborhood G_v. Negative examples are generated by perturbation or another random sample of subgraph. See Appendix for detailed parameters of positive and negative example generation. This binary classification and sampling setup allows us to circumvent exact computation of subgraph isomorphism or subgraph frequency, which would make training intractable.

3.3 SPMiner Decoder: Motif Search Procedure

Motif search procedure by walking in order embedding space. Given the encoder ϕ that maps node-anchored subgraphs to order embeddings, the goal of the search procedure is to identify node-anchored motifs that appear most frequently in the given graph dataset. SPMiner uses the approach of generating frequent motifs by iterative addition of nodes. Proposition 1 shows that the order embedding provides a well-behaved space that makes the search process efficient and effective:

Proposition 1. *Given an order embedding encoder ϕ, let a graph generation procedure be $\{G_0, G_1, \ldots, G_{k-1}\}$, where at any step i, G_i is generated by adding 1 node to G_{i-1}. Then $\{\phi(G_0), \phi(G_1), \ldots, \phi(G_{k-1})\}$ is a monotonic walk in the order embedding space.*

See Figure 1(b) for an example of monotonic walk in the order embedding space in 2D.

Hard frequency objective. Assuming a perfect order embedding, the *frequent motif objective* is then translated to finding a walk with destination embedding such that the number of neighborhoods G_v in dataset G_T satisfying $\phi(G_{k-1}) \preceq \phi(G_v)$ is maximized.

Soft frequency objective. Recall that the SPMiner encoder is trained with a max margin loss, where the margin $E(A, B) = \|\max(0, \phi(A) - \phi(B))\|^2$ can be interpreted as a measure of model confidence about A being a subgraph of B. To take model confidence into account, we define a continuous objective: find graph G_{k-1} of given size k that minimizes the total margin $m(G_{k-1})$. Under Definition 1, the predicted frequent subgraph G_{freq} is then:

$$\begin{aligned} &\phi(G_{\text{freq}}) = \arg\min_{G\in\mathcal{G}} m(G), \\ &\text{where } m(G) = \sum_{N\in\mathcal{N}} \|\max(0, \phi(G) - \phi(N)\|^2 . \end{aligned} \quad (4)$$

$\mathcal{G}$ is the set of all graphs of size k; $\mathcal{N}$ is the set of all neighborhood graphs in G_T. The total margin $m(G)$ is a *soft estimation* of the number of neighborhoods containing the anchored subgraph G.

Motif search. Directly finding the frequent motif is hard, so we design a special search procedure that iteratively grows the motif. In order to find a frequent motif of size k, SPMiner randomly samples a *seed node* from the dataset G_T, referred to as the trivial seed graph G_0 with size 1. Starting from the seed graph G_0, we iteratively generate the next graph by adding an adjacent node in G_T (and its corresponding edges). Figure 1(b) shows an example search process and

the corresponding walk in the embedding space. Proposition 1 guarantees that the corresponding embedding increases monotonically as more nodes are added. Throughout the walk/generation, we make use of both G_i and its embedding $\phi_i, \phi_i = \phi(G_i)$. In practice, to attain a robust estimate, we sample several seed nodes, run several walks, and select the resulting motifs of size k that we encountered the most times.

Search strategies: Greedy strategy, beam search and Monte Carlo Tree Search (MTCS). Our motif search procedure is general and different strategies can be implemented to navigate the space of motifs. Due to exponential growth, for identifying frequent motifs with sizes ≥ 9, exhaustive search over all possible walks is computationally infeasible. We propose a greedy strategy to improve scalability. At every step, SPMiner adds the node such that the total margin m in Eq. 4 is minimized. Let G' be chosen by adding 1 adjacent node to G_i in G_T. The greedy approximation simplifies Equation 4 into a step-wise minimization: $\forall i = 0, 1, \ldots, k-1$,

$$G_{\text{freq}} = G_{k-1}, \quad G_{i+1} = \arg\min_{G'} m(\phi(G')). \tag{5}$$

A beam search strategy strikes a balance between the greedy and exhaustive strategies, where instead of greedily adding the next node resulting in the largest margin, we explore a fixed number of options with the best margin scores to add nodes at each generation step.

The SPMiner framework also naturally lends itself to Monte Carlo Tree Search (MCTS) strategies (Coulom 2006) with neural value function. SPMiner MCTS runs multiple walks starting from multiple seed nodes and maintains a visit count $n(G_i)$ and a total value $f(G_i)$ for each step G_i in each walk explored so far. The visit counts $n(G_i)$ refer to the number of times where G_i is visited. Graphs that share an embedding also share a visit count, except for seed nodes; this setup allows SPMiner to revisit promising seeds and explore new ones. We design $f(G_i) = \sum_{G'}(1 - \log(\frac{m(G')}{|\mathcal{N}|} + 1))$, where G' ranges over all size-k graphs reached from any walk that visited G_i. We design the following objective based on the upper confidence bound criterion for trees (UCT) (Kocsis and Szepesvári 2006) to replace the greedy approach (Eq. 5), with an exploration constant c: $\forall i = 0, 1, \ldots, k-1$,

$$G_{i+1} = \arg\max_{G'} \left(\frac{f(G')}{n(G')} + c\sqrt{\frac{\log n(G_i)}{n(G')}} \right). \tag{6}$$

The use of f rather than the total margin $m(G')$ ensures that the first term has the same numerical scale as the second term. In the end, the motifs with highest visit count are selected.

3.4 SPMiner Expressive Power

Expressiveness of SPMiner encoder. To show the expressiveness of SPMiner encoder with order embedding, we first show the existence of a perfect order embedding, and then demonstrate that the SPMiner encoder architecture can capture information in the perfect order embedding.

In addition to satisfying the properties of subgraph relations, the following proposition demonstrates the full expressive power of the order embedding space in predicting subgraph relations.

Proposition 2. *Given a graph dataset of size n, we can find a perfect order embedding of dimension D, where D is the number of non-isomorphic node-anchored graphs of size no greater than n. The perfect order embeddings satisfy $z(G_Q) \preceq z(G)$ if and only if G_Q is a subgraph of G.*

See Appendix for the detailed proof, which constructs an order embedding by enumerating all possible non-isomorphic node-anchored graphs of size no greater than n, and placing the count of the i-th graph into the i-th dimension of the order embedding. The proposition indicates that order embeddings can achieve perfect expressiveness. Although D can be large in theory, in practice, the neural model can learn the counts for the most important substructures, omitting redundant counts.

4 Experiments

To our knowledge, SPMiner is the first method for learning neural models to perform frequent motif mining. Here, we propose the first benchmark suites of experimental settings, baselines and datasets to evaluate the efficacy of neural frequent motif mining models.

Experimental settings. we perform the following experiments: (1) **Small motifs.** We experiment with small motifs of 5 and 6 nodes, where exact enumeration methods (Hočevar and Demšar 2014; Cordella et al. 2004) are able to find ground-truth most frequent motifs. We show that SPMiner nearly perfectly finds these most frequent motifs. (2) **Large planted motifs.** To evaluate performance for mining larger motifs (computationally prohibitive for exact enumeration methods), we plant a large 10-node motif many times in a dataset (details in Appendix). Again, we show that SPMiner is able to identify the planted motif as one of the most frequent in the dataset. (3) **Large motifs in real-world datasets** We also compare SPMiner against approximate methods (Wernicke 2006; Kashtan et al. 2004) that scale to motifs with over 10 nodes. Here we find that SPMiner identifies large motifs that are 100x more frequent than the ones identified by approximate methods in real-world datasets. (4) **Comparison with exact methods.** We compare SPMiner against exact methods, gSpan and Gaston (Yan and Han 2002; Nijssen and Kok 2005), and show that the exact methods are substantially more expensive when mining larger subgraph patterns. Further **ablation** studies of the encoder architecture can be found in the appendix D.

Approximate baselines. We compare against two widely-used approximate sampling-based motif mining algorithms, RAND-ESU (Wernicke 2006) and MFINDER (Kashtan et al. 2004). For MFINDER, we omit the slow $O(n^{n+1})$ exact probability correction. We tune hyperparameters of these baselines and SPMiner so that they sample a comparable number of subgraphs and achieve comparable wall clock runtime (details in Appendix C).

Datasets. We mine frequent motifs in a variety of domains, including biological (ENZYMES), chemical (COX2) and image (MSRC) datasets (Borgwardt et al. 2005; Sutherland,

Hit rate (size-6 graphs)

Rank	SPMiner	MFinder	Rand-ESU
10	0.90	0.20	0.30
20	0.85	0.20	0.45
30	0.77	0.37	0.57
40	0.75	0.40	0.60
50	0.70	0.50	0.56

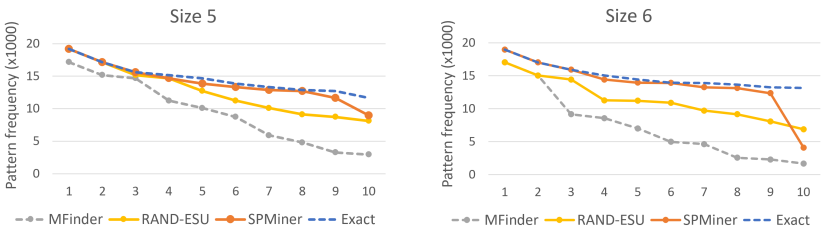


Figure 4: Among size-6 graphs, SPMiner is able to correctly identify the top K most frequent motifs more accurately than baselines (left). Furthermore, the top 10 motifs identified by SPMiner have higher frequency than those found by baselines, for size 5 (middle) and size 6 (right) graphs. The blue dotted line represents the frequency of the groundtruth most frequent motifs.

O'brien, and Weaver 2003; Neumann et al. 2016). We focus on the topological structure of these datasets, omitting node labels when identifying frequent motifs; incorporation of labels is an interesting avenue for further work.

Results. **(1) Small motifs**. First, we experiment with small motifs of 5 and 6 nodes, where exact enumeration methods (Hočevar and Demšar 2014; Cordella et al. 2004) are able to find true most frequent motifs. We pick an existing dataset, ENZYMES, and count the exact motif counts for all possible motifs of size 5 and 6. We use the metric *hit rate* at k, which measures the proportion of top-k frequent motifs identified by SPMiner and baselines, that are within the top-k most frequent motifs in groundtruth by exact enumeration. Figure 4 left shows that SPMiner consistently achieves higher hit rate compared to baselines for mining size 6 motifs.

We further compare the frequencies of the top 10 motifs found by exact enumeration, SPMiner and the baselines, MFINDER and RAND-ESU for size 5 and 6 motifs. We observe that SPMiner can consistently identify the size 5 and size 6 motifs whose frequencies are within 90% of that of the groundtruth. Furthermore, SPMiner runs in 5 minutes, versus 10 hours for exact enumeration with (exact) ESU (Wernicke 2006) using the same hardware (see Appendix).

(2) Large planted motifs. Currently exact methods for finding most frequent motifs are prohibitively expensive for larger motifs, and hence groundtruth frequency is hard to obtain. To evaluate identification of larger motifs, we randomly generate a large motif pattern Q of size $n_1 = 10$, and randomly attach an instance of the motif to base graphs of size $n_2 = 10$ generated using a synthetic generator. Each motif is attached to base graph by randomly connecting a nodes from the motif with a node from the base graph. This process ensures that this pattern is one of the most frequent in the dataset. We repeat this process to generate a dataset of 1000 graphs.

Then, we use SPMiner to identify frequent motifs. We expect SPMiner to output patterns that resemble the planted motif. Figure 5 shows three examples of the top 10 identified frequent patterns outputted by SPMiner. We observe that for all examples, the planted motif is identified as one of the most frequent by SPMiner[1].

(3) Identifying large motifs in real-world datasets. Lastly, we compare against several baselines for finding large frequent motifs in graph data. Our baselines are **MFINDER**, **RAND-ESU** and a neural baseline **MLP** that replaces order embeddings with an MLP and cross entropy loss. For varying motif sizes k, we take the top ten candidates for the most frequent motif for each method, and compute their true mean frequency via exact subgraph matching. Compared to approximate methods, SPMiner is able to identify motifs that appear 10-100x more frequently, as seen in Figure 6, particularly for graphs with size > 12. The MCTS search variant of SPMiner discovers frequent patterns of over 15 nodes in ENZYMES, while no baseline can find motifs of median frequency more than 3. Note that even though SPMiner's primary objective is to maximize anchored frequency (Definition 1), it outperforms all baselines on large motifs by 10-100x with the graph-level Definition 2 as well (Figure 6 right). We further present the results for other datasets in Appendix D.

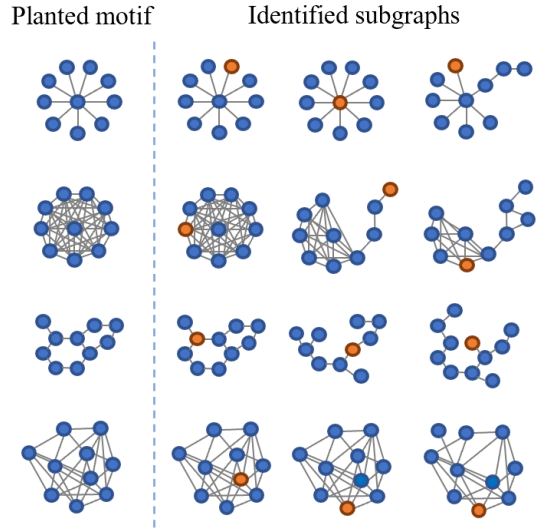


Figure 5: The frequent subgraphs identified by SPMiner closely match the planted motifs. Red nodes denote the anchor node of the motif identified in SPMiner.

(4) Comparison with exact methods. We consider exact motif mining methods, gSpan (Yan and Han 2002) and Gaston (Nijssen and Kok 2005), which are able to compute exactly the most frequent motifs in a dataset. For each dataset, we adapt their code in C++, and tune the support threshold

[1]See Appendix for more details on all identified motifs.

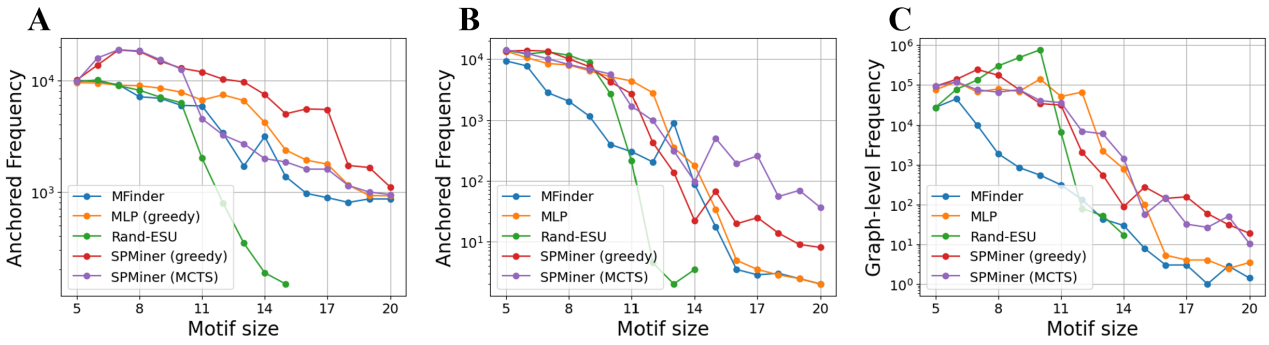


Figure 6: Comparison of median frequencies of motifs identified by different search strategies (greedy and MCTS) and baselines; higher is better. Across all motif sizes, SPMiner finds patterns with higher node-anchored frequency than the baseline, on both COX2 (**A**) and ENZYMES (**B**). The motifs found by SPMiner also have high graph-level frequency on ENZYMES (**C**).

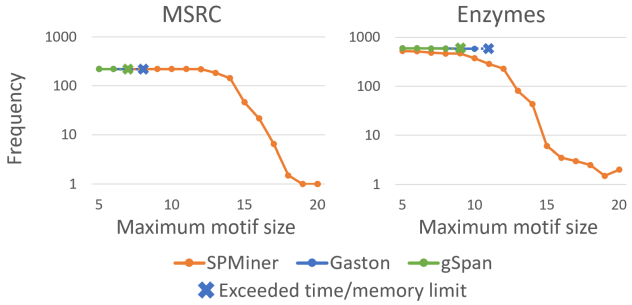


Figure 7: Frequency of motifs identified by the exact methods and SPMiner.

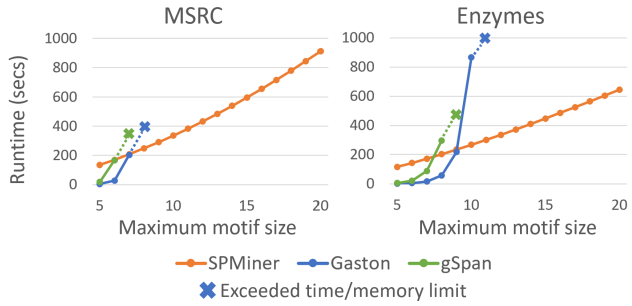


Figure 8: Runtime comparison between exact methods and SPMiner. The curves for gSpan and Gaston end early due to exceeding memory or time limit for larger motifs.

parameter (Yan and Han 2002) in order to obtain at least 10 frequent motifs of the specified size, without exceeding a runtime budget of 2 hours and memory budget of 50GB. Implementation details of these baselines are further explained in Appendix.

In Figure 7, we plot the frequency of the motif identified by SPMiner and the exact baseline methods, against the size of the motifs to be identified. SPMiner identifies small frequent motifs (of size less than 10) whose frequency is at least 90% of the groundtruth most frequent motifs (identified by exact methods). For both datasets, gSpan and Gaston exceed the resource budget when identifying motifs of size larger than 10, while SPMiner can still identify them efficiently. Note that although we only record the runtime of SPMiner inference step, the training cost can be alleviated via pretraining on a large synthetic datasets (see Appendix C for details).

We further show that the runtime of the exact methods grows exponentially, whereas SPMiner has a linear trend as the size of motif grows. In Figure 8, we plot the runtime required against the size of frequent motifs identified by SPMiner and the exact methods. We observe that even with more computational budget, baseline methods quickly become intractable due to exponential increase in runtime.

5 Limitations

SPMiner is the first approach to mine large frequent motifs in graph datasets. However, there are still limitations to this pioneering approach. The algorithm does not directly optimize for the graph-level frequency definition (Definition 2), although we observe that in practice the subgraphs identified by SPMiner are still frequent (see Figure 6(c)). Additionally, SPMiner only provides a list of frequent subgraph patterns via search, but not an accurate count prediction for a given subgraph pattern. Future work in approximating the # P problem of subgraph counting is needed. Finally, although in experiments, we find that distinguishing between the anchor node and other nodes in the neighborhood via node features results in more expressive GNNs beyond the WL test (illustrated in Appendix F), further work on more expressive GNNs can further improve SPMiner. We hope that SPMiner opens a new direction in graph representation learning and embedding space search to solve graph mining problems.

6 Conclusion

We propose SPMiner, the first neural framework for identifying frequent motifs in a large target graph using graph representation learning. SPMiner learns to encode subgraphs into an order embedding space, and uses a novel search procedure on the learned order embedding space to identify frequent motifs of a dataset. SPMiner advances the state-of-the-art, and is able to identify frequent motifs that are 10-100x more frequent than those identified by existing methods.

References

Agrawal, M.; Zitnik, M.; Leskovec, J.; et al. 2018. Large-scale analysis of disease pathways in the human interactome. In *PSB*. World Scientific.

Albert, R.; and Barabási, A.-L. 2000. Topology of evolving networks: local events and universality. *Physical Review Letters* .

Bai, Y.; Ding, H.; Bian, S.; Chen, T.; Sun, Y.; and Wang, W. 2019. Simgnn: A neural network approach to fast graph similarity computation. In *WSDM*.

Bai, Y.; Xu, D.; Gu, K.; Wu, X.; Marinovic, A.; Ro, C.; Sun, Y.; and Wang, W. 2020. Neural Maximum Common Subgraph Detection with Guided Subgraph Extraction.

Benson, A. R.; Gleich, D. F.; and Leskovec, J. 2016. Higher-order organization of complex networks. *Science* .

Borgwardt, K. M.; Ong, C. S.; Schönauer, S.; Vishwanathan, S.; Smola, A. J.; and Kriegel, H.-P. 2005. Protein function prediction via graph kernels. *Bioinformatics* .

Cao, Y.; Jiang, T.; and Girke, T. 2008. A maximum common substructure-based algorithm for searching and predicting drug-like compounds. *Bioinformatics* .

Cereto-Massagué, A.; Ojeda, M. J.; Valls, C.; Mulero, M.; Garcia-Vallvé, S.; and Pujadas, G. 2015. Molecular fingerprint similarity search in virtual screening. *Methods* .

Chen, Z.; Chen, L.; Villar, S.; and Bruna, J. 2020. Can graph neural networks count substructures? *arXiv preprint arXiv:2002.04025* .

Chen, Z.; Villar, S.; Chen, L.; and Bruna, J. 2019. On the equivalence between graph isomorphism testing and function approximation with gnns. In *NeurIPS*.

Cordella, L. P.; Foggia, P.; Sansone, C.; and Vento, M. 2004. A (sub) graph isomorphism algorithm for matching large graphs. *PAMI* .

Coulom, R. 2006. Efficient selectivity and backup operators in Monte-Carlo tree search. In *International conference on computers and games*. Springer.

Duvenaud, D. K.; Maclaurin, D.; Iparraguirre, J.; Bombarell, R.; Hirzel, T.; Aspuru-Guzik, A.; and Adams, R. P. 2015. Convolutional networks on graphs for learning molecular fingerprints. In *NeurIPS*.

Erdos, P.; and Renyi, A. 1959. On random graphs. I. *Publ. Math. Debrecen* .

Fey, M.; and Lenssen, J. E. 2019. Fast Graph Representation Learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*.

Fey, M.; Lenssen, J. E.; Morris, C.; Masci, J.; and Kriege, N. M. 2020. Deep graph matching consensus. In *ICLR*.

Guo, M.; Chou, E.; Huang, D.-A.; Song, S.; Yeung, S.; and Fei-Fei, L. 2018. Neural graph matching networks for few-shot 3d action recognition. In *ECCV*.

Hamilton, W.; Ying, Z.; and Leskovec, J. 2017. Inductive representation learning on large graphs. In *NeurIPS*.

Hočevar, T.; and Demšar, J. 2014. A combinatorial approach to graphlet counting. *Bioinformatics* .

Holme, P.; and Kim, B. J. 2002. Growing scale-free networks with tunable clustering. *Physical review E* .

Huang, G.; Liu, Z.; Van Der Maaten, L.; and Weinberger, K. Q. 2017. Densely connected convolutional networks. In *CVPR*.

Inokuchi, A.; Washio, T.; and Motoda, H. 2000. An Apriori-Based Algorithm for Mining Frequent Substructures from Graph Data. In Zighed, D. A.; Komorowski, J.; and Żytkow, J., eds., *Principles of Data Mining and Knowledge Discovery*. Springer Berlin Heidelberg.

Kashtan, N.; Itzkovitz, S.; Milo, R.; and Alon, U. 2004. Efficient sampling algorithm for estimating subgraph concentrations and detecting network motifs. *Bioinformatics* .

Kersting, K.; Kriege, N. M.; Morris, C.; Mutzel, P.; and Neumann, M. 2016. Benchmark Data Sets for Graph Kernels. URL http://graphkernels.cs.tu-dortmund.de.

Ketkar, N. S.; Holder, L. B.; and Cook, D. J. 2005. Subdue: Compression-based frequent pattern discovery in graph data. In *Proceedings of the 1st international workshop on open source data mining: frequent pattern mining implementations*.

Kipf, T. N.; and Welling, M. 2017. Semi-supervised classification with graph convolutional networks. In *ICLR*.

Kocsis, L.; and Szepesvári, C. 2006. Bandit based monte-carlo planning. In *European conference on machine learning*. Springer.

Kuramochi, M.; and Karypis, G. 2004. An efficient algorithm for discovering frequent subgraphs. *IEEE TKDE* .

Leskovec, J.; Huttenlocher, D.; and Kleinberg, J. 2010. Signed networks in social media. In *SIGCHI*.

Li, G.; Muller, M.; Thabet, A.; and Ghanem, B. 2019a. Deepgcns: Can gcns go as deep as cnns? In *ICCV*.

Li, Y.; Gu, C.; Dullien, T.; Vinyals, O.; and Kohli, P. 2019b. Graph matching networks for learning the similarity of graph structured objects. In *ICML*.

Matsuda, T.; Horiuchi, T.; Motoda, H.; and Washio, T. 2000. Extension of graph-based induction for general graph structured data. In *PAKDD*.

Morris, C.; Ritzert, M.; Fey, M.; Hamilton, W. L.; Lenssen, J. E.; Rattan, G.; and Grohe, M. 2019. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Neumann, M.; Garnett, R.; Bauckhage, C.; and Kersting, K. 2016. Propagation kernels: efficient graph kernels from propagated information. *Machine Learning* .

Nijssen, S.; and Kok, J. N. 2005. The gaston tool for frequent subgraph mining. *Electronic Notes in Theoretical Computer Science* .

Ribeiro, P.; Paredes, P.; Silva, M. E. P.; Aparício, D.; and Silva, F. 2019. A Survey on Subgraph Counting: Concepts, Algorithms and Applications to Network Motifs and

Graphlets. *CoRR* abs/1910.13011. URL http://arxiv.org/abs/1910.13011.

Riesen, K.; and Bunke, H. 2008. IAM graph database repository for graph based pattern recognition and machine learning. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, 287–297. Springer.

Rossi, R. A.; and Ahmed, N. K. 2015. The Network Data Repository with Interactive Graph Analytics and Visualization. In *AAAI*. URL http://networkrepository.com.

Sutherland, J. J.; O'brien, L. A.; and Weaver, D. F. 2003. Spline-fitting with a genetic algorithm: A method for developing classification structure- activity relationships. *Journal of chemical information and computer sciences* .

Vendrov, I.; Kiros, R.; Fidler, S.; and Urtasun, R. 2016. Order-Embeddings of Images and Language. In Bengio, Y.; and LeCun, Y., eds., *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. URL http://arxiv.org/abs/1511.06361.

Watts, D. J.; and Strogatz, S. H. 1998. Collective dynamics of 'small-world'networks. *Nature* .

Wernicke, S. 2006. Efficient detection of network motifs. *IEEE/ACM transactions on computational biology and bioinformatics* .

Xu, K.; Hu, W.; Leskovec, J.; and Jegelka, S. 2019a. How Powerful are Graph Neural Networks? In *ICLR*.

Xu, K.; Li, C.; Tian, Y.; Sonobe, T.; Kawarabayashi, K.-i.; and Jegelka, S. 2018. Representation Learning on Graphs with Jumping Knowledge Networks. In *ICML*.

Xu, K.; Wang, L.; Yu, M.; Feng, Y.; Song, Y.; Wang, Z.; and Yu, D. 2019b. Cross-lingual Knowledge Graph Alignment via Graph Matching Neural Network.

Yan, X.; and Han, J. 2002. gspan: Graph-based substructure pattern mining. In *ICDM*. IEEE.

Ying, R.; Lou, Z.; You, J.; Wen, C.; Canedo, A.; and Leskovec, J. 2020. Neural Subgraph Matching. *arXiv preprint* .

Ying, Z.; Bourgeois, D.; You, J.; Zitnik, M.; and Leskovec, J. 2019. Gnnexplainer: Generating explanations for graph neural networks. In *NeurIPS*.

A Proofs of Subgraph Properties

NeuroMatch (Ying et al. 2020) introduces the following observations that demonstrate the natural use of order embeddings in subgraph matching. The arguments hold for both the problem of subgraph matching, and motif mining. We list the correspondences between order embedding and subgraph relation in this section for completeness.

Note, however, that these subgraph relations are not able to account for the appearances of multiple of the same subgraph in the target graph, as required by motif mining. Therefore SPMiner solves this challenge by encoding node neighborhoods in the target graph, and performing a walk in the order embedding space.

Transitivity. Suppose that G_1 is a subgraph of G_2 with bijection f mapping all nodes from G_1 to a subset of nodes in G_2, and G_2 is a subgraph of G_3 with bijection g. Let v_1, v_2, v_3 be anchor nodes of G_1, G_2, G_3 respectively. By definition of anchored subgraph, $f(v_1) = v2$ and $g(v_2) = v_3$. Then the composition $g \circ f$ is a bijection. Moreover, $g \circ f(v_1) = g(v_2) = v_3$, where Therefore G_1 is a subgraph of G_3, and thus the transitivity property.

This corresponds to the transitivity of order embedding.

Anti-symmetry. Suppose that G_1 is a subgraph of G_2 with bijection f, and G_2 is a subgraph of G_1 with bijection g. Let $|V_1|$ and $|V_2|$ be the number of nodes in G_1 and G_2 respectively. By definition of subgraph isomorphism, G_1 is a subgraph of G_2 implies that $|V_1| \leq |V_2|$. Similarly, G_2 is a subgraph of G_1 implies $|V_2| \leq |V_1|$. Hence $|V_1| = |V_2|$. The mapping between all nodes in G_1 and G_2 is bijective. By definition of isomorphism, G_1 and G_2 are graph-isomorphic.

This corresponds to the anti-symmetry of order embedding.

Intersection. By definition, if G_3 is a common subgraph of G_1, G_2, the G_3 is a subgraph of both G_1 and G_2. Since a trivial node is a subgraph of any graph, there is always a non-empty intersection set between two graphs.

Correspondingly, if $z_3 \preceq z_1$ and $z_3 \preceq z_2$, then $z_3 \preceq \min\{z_1, z_2\}$. Here $\min$ denotes the element-wise minimum of two embeddings. Note that the order embedding z_1 and z_2 are positive, and therefore $\min\{z_1, z_2\}$ is another valid order embedding, corresponding to the non-empty intersection set between two graphs.

Note that this paper assumes the frequent motifs are connected graphs. And thus it also assumes that all neighborhoods in a given datasets are connected and contain at least 2 nodes (an edge). This is a reasonable assumption since we can remove isolated nodes from the datasets, as connected motifs of size k ($k > 1$) can never contain isolated nodes. In this case, the trivial intersection corresponds to a graph of 2 nodes and 1 edge.

B Model Analysis

B.1 Proof of order embedding expressiveness

Proposition 3. *Given a graph dataset of size n, we can find a perfect order embedding of dimension D, where D is the number of non-isomorphic node-anchored graphs of size no greater than n. The perfect order embeddings satisfy $z(G_Q) \preceq z(G)$ if and only if G_Q is a subgraph of G.*

Dataset	E-R	COX2	DD	MSRC_21	FIRSTMMDB
MSE	8.4	7.1	8.9	10.5	12.7
REL ERR	9.4%	12.5%	11.8%	13.4%	11.5%

Table 1: The MSE and relative MSE for predicting log of motif counts.

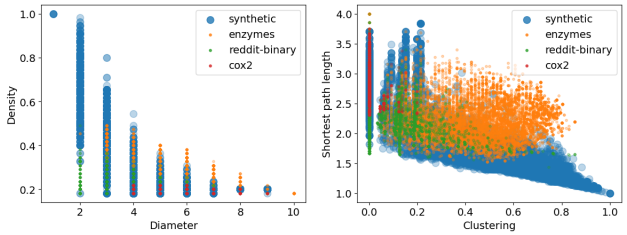


Figure 9: Graph statistics of the synthetic and real-world graph datasets. Each point represents the graph statistics of one graph; the color represents the dataset that the point belongs to.

Proof. This perfect order embedding can be simply constructed by enumerating all possible non-isomorphic node-anchored graphs of size no greater than n, and place the count of the i-th graph into the i-th dimension of the order embedding.

If graph G_Q is a subgraph of graph G, then by definition there exists an isomorphism f between G_Q and a subgraph of G. Therefore each motif in G_Q can be mapped to the corresponding motif in G by f. Hence the motif count of G_Q for any motif is no less than the corresponding motif count of G.

Conversely, if graph G_Q is not a subgraph of graph G, then the motif isomorphic to G_Q has count at least 1 for G_Q, and 0 for graph G, violating the order constraint. □

B.2 Motif Counts Prediction

As an experiment to demonstrate the capability of GNNs in predicting motif counts, we randomly generate graphs using synthetic and real datasets (Kersting et al. 2016) and train a GNN to predict (log) counts of small motifs, as computed by exact enumeration. We find that GNNs are able to accurately estimate these counts, with relative mean squared error of around 10% across all datasets. Together with Proposition 1, this finding confirms their capacity to learn order embeddings that capture the subgraph relation.

C Further Implementation Details

Synthetic dataset. Synthetic datasets with graph generator can be used to train a general subgraph relation encoder agnostic to the dataset domain, and the model can benefit from large data by sampling from the generator. In contrast to the previous works, we use a combination of graph generators, to ensure that the model can be trained on a diverse set of graphs in terms of graph properties.

We design a generator that randomly chooses one of the following generators: Erdős-Rényi (E-R) (Erdos and Renyi 1959), Extended Barabási–Albert graphs (Albert and Barabási 2000), Power Law Cluster graphs (Holme and Kim 2002) and Watts-Strogatz graph (Watts and Strogatz 1998). We design prior distributions for the parameters for each of the generators. For the Erdős-Rényi generator, the probability of an edge is $p \sim \text{Beta}(1.3, 1.3n/\log_2(n) - 1.3)$. For the Extended Barabási–Albert generator, the number of attachment edges per node is $m \sim \text{Unif}(1, 2\log_2 n)$, the probability of adding edges is $p \sim \text{Exp}(20)$ (capped at 0.2) and the probability of rewiring edges is $q \sim \text{Exp}(20)$ (capped at 0.2). For the Power Law Cluster generator, the number of attachment edges per node is $m \sim \text{Unif}(1, 2\log_2 n)$ and the probability of adding a triangle after each edge is $p \sim \text{Unif}(0, 0.5)$. For the Watts-Strogatz generator, each node is connected to $k \sim n\text{Beta}(1.3, 1.3n/\log_2(n) - 1.3)$ neighbors (minimum 2) and the rewiring probability is $p \sim \text{Beta}(2, 2)$. Here n is the desired graph size.

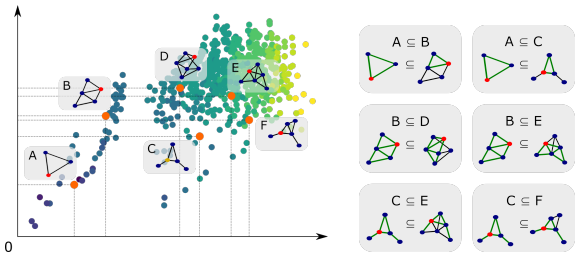
Using this graph generator, we create a balanced dataset of graph pairs (A, B) in which a pair is positive if B is a subgraph of A. To create positive pairs, we sample a graph A from the generator and sample a subgraph B using the sampling procedure of MFINDER (Kashtan et al. 2004). To create negative pairs, we sample a graph A from the generator. With 50% probability, we sample a subgraph B as in the positive case, then randomly add up to 5 edges to B so that it is unlikely to be a subgraph of A. Otherwise, we sample an auxilliary subgraph X of the same size as A, then let B be a sampled subgraph of X, making it unlikely to be a subgraph of A. We sample A of size uniform from 6 to 29 and B of size uniform from 5 to $|A| - 1$.

Dataset statistics. We demonstrate that our synthetic data generation scheme is capable of generating graphs with high variety. Figure 9 shows the statistics of the synthetic graphs, compared to real-world datasets. In terms of graph statistics, including density, diameter, average shortest path length, and average clustering coefficient, the synthetic dataset (large blue dots) covers most of the real-world datasets, including those in the domains of chemistry (COX2), biology (ENZYMES) and social networks (REDDIT-BINARY) (Fey and Lenssen 2019).

The high coverage of statistics suggests that the synthetic dataset can serve well as a application-agnostic dataset that allows our model SPMiner to learn order embedding models in the most generic sense. In practice, given a very small real-world dataset, one can train on this synthetic dataset, and transfer the model (either directly or with fine-tuning) to apply to this real-world dataset.

Table 2 shows the graph statistics of the datasets used in our experiments.

Training details. Our encoder model architecture consists of a single hidden-layer feedforward module, followed by 8 graph convolutions with ReLU activation and learned dense skip connections, followed by a 4-layer perceptron with 64-dimensional output. We use SAGE graph convolutions (Hamilton, Ying, and Leskovec 2017) with sum aggregation and no neighbor sampling. We train the network with the Adam optimizer using learning rate 10^{-4} for 1 million batches of synthetic data, with batch size 64 and balanced class distribution.



(a) Order embedding space

	Synthetic		ENZYMES		COX2		FIRSTMMDB	
	Acc	AUPR	Acc	AUPR	Acc	AUPR	Acc	AUPR
GCN+MLP	95.8	37.2	94.7	31.5	91.6	47.2	94.3	34.1
GIN+MLP	96.1	41.0	95.4	34.8	89.0	37.3	94.7	35.9
SAGE+MLP	**96.3**	43.9	96.2	40.2	91.3	45.8	94.7	38.8
No skip	95.9	45.7	96.5	57.3	92.1	70.5	95.1	53.4
SPMiner	96.2	**46.8**	**96.6**	**60.9**	**92.6**	**71.2**	**95.6**	**59.9**

(b) Model comparison (Accuracy and Area Under Precision-Recall Curve with unit 0.01)

Figure 10: (a) Two dimensions of the embedding space. Each point corresponds to a graph; color range from blue to yellow corresponds to small-to-large graphs. Subgraphs are embedded to the lower left of their supergraphs. (b) The order embedding performance: given a pair of graphs the task is to predict whether they are subgraphs or not.

Dataset	Number of graphs	Number of nodes	Number of edges
ENZYMES	600	19.6K	37.3K
COX2	467	19.3K	20.3K
MSRC_9	221	9.0K	21.6K
MNROADS	1	2.6K	3.3K
COIL-DEL	3900	84.0K	211.5K
PLANT-STAR	1000	20K	30.2K
PLANT-CLIQUE	1000	20K	66.2K
PLANT-MOLECULE	1000	20K	32.2K
PLANT-RANDOM	1000	20K	49.3K

Table 2: Graph statistics of datasets used in experiments. The PLANT datasets are those used in Experiment (2).

Decoder configuration. To sample node-anchored neighborhoods, we follow the iterative procedure of MFINDER (Kashtan et al. 2004): after picking a random node as the anchor, we maintain a search tree, in each step randomly picking a node from the frontier with probability weighted by its number of edges with nodes in the search tree. The procedure terminates when the search tree reaches N nodes, and we take the subgraph induced by these nodes as the neighborhood. We select N uniformly randomly from 20 to 29 for each neighborhood (except for Experiment **(2)**, where the maximum size is 25), sampling 10000 neighborhoods in total.

For a given maximum motif size k, SPMiner generates all motifs up to size k through a single run of the search procedure. For the greedy procedure, we sample 1000 seed nodes and expand each corresponding candidate motif up to size k, recording all candidates for intermediate sizes. For the MCTS procedure, we run a total of 1000 simulations, divided equally among each motif size up to k, reusing the existing search tree with each new iteration; this procedure creates a useful prior for each successive motif size. We use exploration constant $c = 0.7$ throughout.

Baseline configuration. For MFINDER (Kashtan et al. 2004), we sample 10000 neighborhoods per motif size. We omit the slow $O(n^{n+1})$ exact probability correction, simply weighting each sampled neighborhood equally. We take the first sampled node as the anchor node in node-anchored experiments.

For RAND-ESU (Wernicke 2006), we set the expansion probabilities for each search tree level i according to $p_i = (1-i/(k+1))^\tau$ where k is the maximum motif size and τ can be tuned for runtime depending on the dataset; we use $\tau = 5$ for ENZYMES (except $\tau = 2.3$ in Experiment **(1)**), $\tau = 2$ for COX2, $\tau = 2$ for MNROADS and $\tau = 9$ for COIL-DEL. We use the suggested variance reduction technique of sampling a fixed proportion of children to expand at each level.

For both methods, we use the WL-isomorphism test to count motif frequencies.

D Further Experimental Results

Order embedding space. Figure 10(a) demonstrates the structure of our order embedding space (here in just 2 dimensions). Notice how subgraphs are embedded to the lower left of their supergraphs. Furthermore, Figure 10(b) evaluates the faithfulness of the order embedding space: given a pair of graphs, the task is to predict whether one is a subgraph of the other. We use 2 metrics, the accuracy and area under PR curve (AUPR) to evaluate the performance. SPMiner achieves 95% accuracy in determining whether one graph is a subgraph of the other.

Ablation study. We further conduct ablation studies. A random model that outputs labels according to the class distribution would receive 3.5 AUPR in our task. The following architectures are considered: (1) GCN+MLP: uses MLP with cross entropy loss to replace order embedding; uses the GCN (Kipf and Welling 2017) architecture; (2) GIN+MLP: same as (1) but with GIN (Xu et al. 2018) architecture; (3) SAGE+MLP: same as (1) but with SAGE (Hamilton, Ying, and Leskovec 2017) architecture[1]; (4) No skip: same as SPMiner (order embedding loss and the SAGE (Hamilton, Ying, and Leskovec 2017) architecture), but does not use the proposed learnable skip layer. Results in Table 10 (b) demonstrate that both order embedding and the learnable skip layer are crucial components in performance gains, together reaching over 60 AUPR on real-world datasets.

Extended planted motif results. Figure 11 shows the full set of top ten subgraphs identified by SPMiner for each planted motif in Experiment **(2)**. In three of the four cases, the planted motif is the top-ranked motif identified by the model. Note that the model tends to identify subgraphs similar to the planted motif, though often with additional "attachment edges"; motifs with these attachments tend to be frequent as well due to the dataset construction.

[1]We use the SAGE architecture with sum aggregation, but without neighbor sampling.

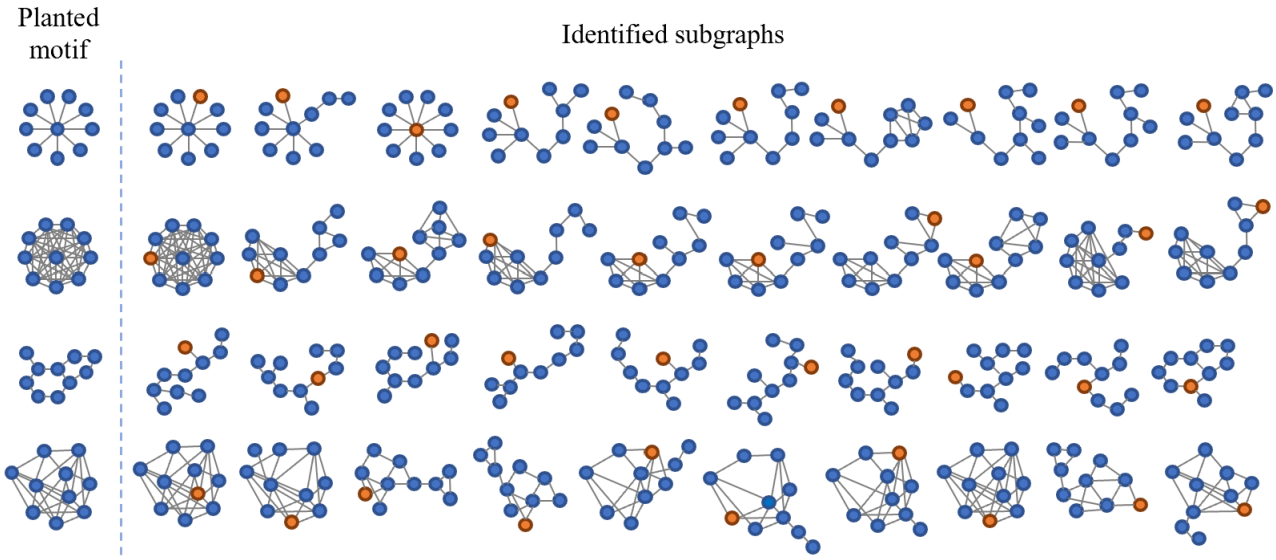


Figure 11: Full set of top ten subgraphs identified by SPMiner for each planted motif, ordered by increasing rank from left to right. In three of the four cases, the planted motif is the top-ranked motif identified by the model. Anchor nodes are shown in orange.

Frequency comparison of identified motifs. We additionally run Experiment **(3)** on two more datasets, COIL-DEL (Riesen and Bunke 2008), a dataset of 3D objects derived from COIL-100, and a graph of the Minnesota road network (Rossi and Ahmed 2015), abbreviated MNROADS. We find again that SPMiner consistently identifies more frequent motifs than the baselines, especially for large graphs (Figure 12).

E Runtime Comparison

Runtime comparison with approximate methods. We tune the hyperparameters so that all methods are comparable in runtime and sample a comparable number of subgraphs. All methods are single-process and implemented with Python; the neural methods are implemented with PyTorch Geometric (Fey and Lenssen 2019). We run all methods on a single Xeon Gold 6148 core; the neural methods additionally use a single Nvidia 2080 Ti RTX GPU. Table 3 shows that SPMiner both runs in less time and requires fewer subgraph samples than the baselines.

Runtime comparison with exact methods. In experiments, we compare with exact methods gSpan (Yan and Han 2002) and Gaston (Nijssen and Kok 2005). For gSpan, we use a highly optimized C++ implementation[1], which we modify to prune the search once the identified motif reaches the specified maximum motif size, in order to increase its efficiency in our setting. We use the single-threaded version. For Gaston, we use the official C++ implementation[2], using the variant with occurrence lists and specifying the desired maximum motif size through a command-line argument. We impose a resource limit of two hours of runtime and 50GB of memory

[1]https://github.com/Jokeren/gBolt

[2]http://liacs.leidenuniv.nl/~nijssensgr/gaston/download.html

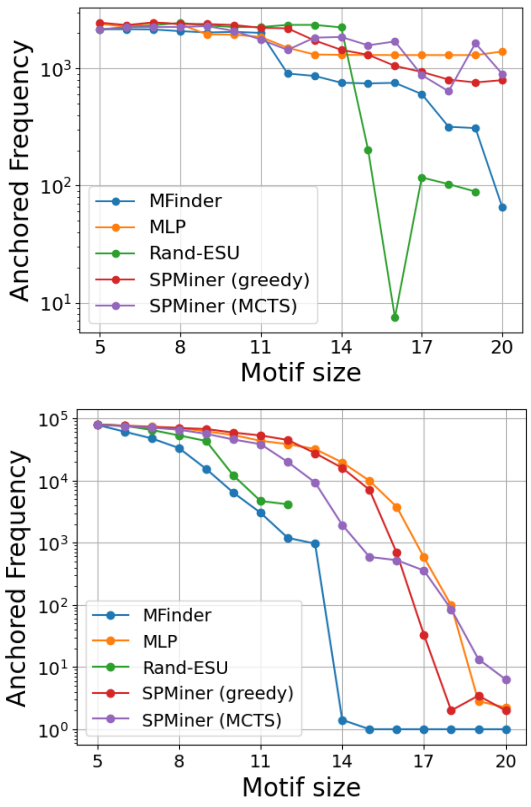


Figure 12: Median frequencies of motifs identified by SPMiner and baselines for each graph size; higher is better. Left: MNROADS; right: COIL-DEL.

for both methods. We run both methods on a single Xeon Gold 6148 core.

F GNN expressive power

Previous works (Xu et al. 2019a; Morris et al. 2019) have identified limitations of a class of GNNs. More specifically, GNNs face difficulties when asked to distinguish regular graphs. In this work, we circumvent the problem by distinguishing the anchor node and other nodes in the neighborhood via one-hot encoding (See Section 3.2). The idea is explored in a concurrent work "Identity-aware Graph Neural Networks" (ID-GNNs). It uses Figure 13 to demonstrate the expressive power of ID-GNN, which distinguishes anchor node from other nodes. For example, while d-regular graphs such as 3-cycle and 4-cycle graphs have the same GNN computational graphs, their ID-GNN computational graphs are different, due to identification of anchor nodes via node features. Such modification enables better expressive power than message-passing GNNs such as GIN.

A future direction is to investigate the performance of recently proposed more expressive GNNs (Chen et al. 2019) in the context of subgraph mining. The SPMiner framework is general and any GNN can be used in its decoder component, and could benefit from more expressive GNNs.

Dataset	ENZYMES	COX2	COIL-DEL	MNROADS
RAND-ESU	13:59	13:56	19:22	4:47
MFINDER	17:29	17:13	20:57	16:24
SPMiner	9:26	8:13	11:15	9:41

(a) Runtime comparison with approximate methods

Dataset	ENZYMES	COX2	COIL-DEL	MNROADS
RAND-ESU	39235	29368	85149	7605
MFINDER	10000	10000	10000	10000
SPMiner	1500	1500	1500	1500

(b) Average number of subgraphs sampled

Table 3: Comparison of runtimes and number of subgraphs sampled (averaged over all motif sizes) for each method in Experiment **(3)**. SPMINER has the lowest runtimes and requires fewer samples to identify frequent motifs. We compare against the greedy variant of SPMiner; the MCTS variant samples the same number of subgraphs and incurs small computational overhead.

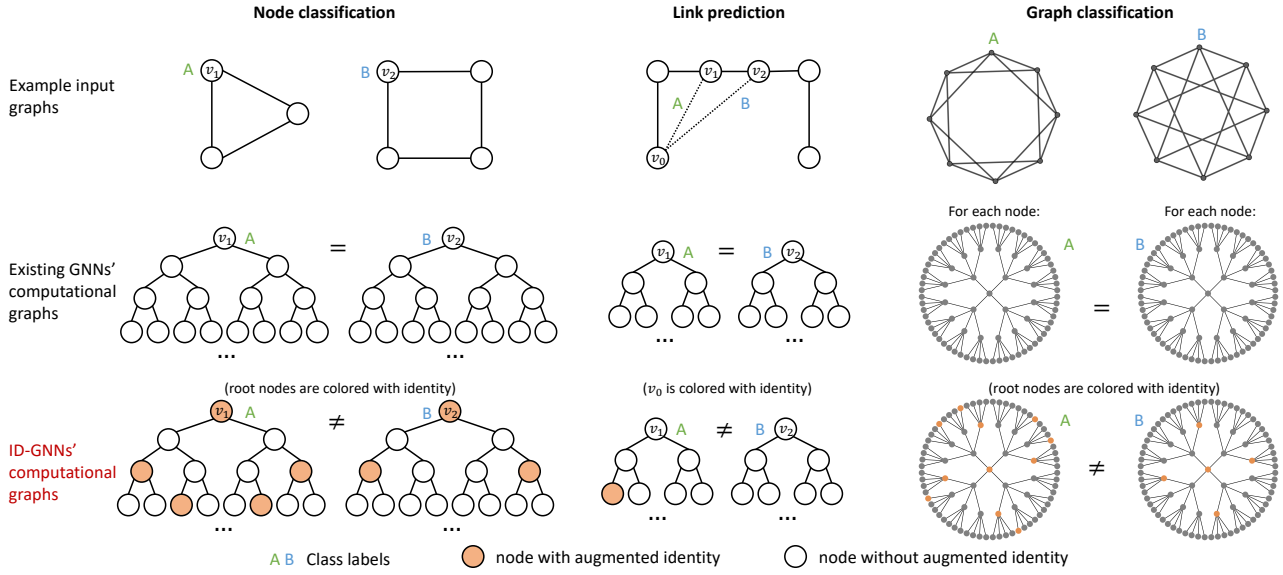


Figure 13: An overview of the proposed ID-GNN model. We consider node, edge and graph level tasks, and assume nodes do not have additional features. Across all examples, the task requires an embedding that allows for the differentiation of the label A vs. B nodes in their respective graphs. However, across all tasks, existing GNNs, regardless of depth, will *always* assign the same embedding to both classes of nodes, because for all tasks the computational graphs are identical. In contrast, the colored computation graphs provided by ID-GNNs allows for clear differentiation between the nodes of class A and class B, as the colored computation graph are no longer identical across all tasks.