Motif Prediction with Graph Neural Networks

Maciej Besta$^1$ Raphael Grob$^1$ Cesare Miglioli$^2$ Nicola Bernold$^1$ Grzegorz Kwasniewski$^1$

Gabriel Gjini$^1$ Raghavendra Kanakagiri$^3$ Saleh Ashkboos$^1$ Lukas Gianinazzi$^1$

Nikoli Dryden$^1$ Torsten Hoefler$^1$

$^1$Department of Computer Science, ETH Zurich
$^2$Research Center for Statistics, University of Geneva
$^3$Department of Computer Science, Indian Institute of Technology Tirupati

Abstract

Link prediction is one of the central problems in graph mining. However, recent studies highlight the importance of higher-order network analysis, where complex structures called motifs are the first-class citizens. We first show that existing link prediction schemes fail to effectively predict motifs. To alleviate this, we establish a general motif prediction problem and we propose several heuristics that assess the chances for a specified motif to appear. To make the scores realistic, our heuristics consider – among others – correlations between links, i.e., the potential impact of some arriving links on the appearance of other links in a given motif. Finally, for highest accuracy, we develop a graph neural network (GNN) architecture for motif prediction. Our architecture offers vertex features and sampling schemes that capture the rich structural properties of motifs. While our heuristics are fast and do not need any training, GNNs ensure highest accuracy of predicting motifs, both for dense (e.g., $k$-cliques) and for sparse ones (e.g., $k$-stars). We consistently outperform the best available competitor by more than 10% on average and up to 32% in area under the curve. Importantly, the advantages of our approach over schemes based on uncorrelated link prediction increase with the increasing motif size and complexity. We also successfully apply our architecture for predicting more arbitrary clusters and communities, illustrating its potential for graph mining beyond motif analysis.

1 Introduction and Motivation

One of the central problems in graph mining and learning is link prediction [5, 6, 59, 76, 93, 95], in which one is interested in predicting if a given pair of vertices will become connected. However, recent works argue the importance of higher-order graph organization [10], where one focuses on finding and analyzing small recurring subgraphs called motifs (sometimes referred to as graphlets or graph patterns) instead of individual links. Motifs are central to many graph mining problems in computational biology, chemistry, and a plethora of other fields [16, 22, 23, 28, 30, 41, 46]. Specifically, motifs are building blocks of different networks, including transcriptional regulation graphs, social networks, brain graphs, or air traffic patterns [10]. There exist many motifs, for example $k$-cliques, $k$-stars, $k$-clique-stars, $k$-cores, and others [20, 43, 54]. A huge number of works are dedicated to motif counting, listing (also called enumeration), or checking for the existence of a given motif [22, 30]. However, while a few recent schemes focus on predicting triangles [9, 63, 64], no works target the problem of general motif prediction, i.e., analyzing whether specified complex structures may appear in the data. As with link prediction, it would enable predicting the evolution
of data, but also finding missing structures in the available data. For example, one could use motif prediction to find probable missing clusters of interactions in biological (e.g., protein) networks, and use the outcomes to limit the number of expensive experiments conducted to find missing connections [59, 60].

In this paper, we first (Section 3) establish and formally describe a general motif prediction problem, going beyond link prediction and illustrating how to predict future (or missing from the data) higher-order network patterns. A key challenge is the appropriate problem formulation. Similarly to link prediction, one wants a score function that – for a given vertex set \( V_M \) – assesses the chances for a given motif to appear. However, the function must consider the increased complexity of the problem (compared to link prediction). In general, contrary to a single link, a motif may be formed by an arbitrary number \( |V_M| \) of vertices, and the number of potential edges between these vertices can be large, i.e., \( O(|V_M|^2) \). For example, one may be interested in analyzing whether a group of entities \( V_M \) may become a \( k \)-clique in the future, or whether a specific vertex \( v \in V_M \) will become a hub of a \( k \)-star, connecting \( v \) to \( k-1 \) other selected vertices from \( V_M \setminus \{v\} \). This leads to novel issues, not present in link prediction. For example, what if some edges, belonging to the motif being predicted, already exist? How should they be treated by a score function? Or, how to enable users to apply their domain knowledge? For example, when predicting whether the given vertices will form some chemical particle, a user may know that the presence of some link (e.g., some specific atomic bond) may increase (or decrease) the chances for forming another bond. Now, how could this knowledge be provided in the motif score function? We formally specify these and other aspects of the problem in a general theoretical framework, and we provide example motif score functions. We explicitly consider correlations between edges forming a motif, i.e., the fact that the appearance of some edges may increase or decrease the overall chances of a given motif to appear.

Then, we develop a learning architecture based on graph neural networks (GNNs) to further enhance motif prediction accuracy (Section 4). For this, we extend the state-of-the-art SEAL link prediction framework [93] to support arbitrary motifs. For a given motif \( M \) in question, we train our architecture on what is the “right motif surroundings” (i.e., nearby vertices and edges) that could result in the appearance of \( M \). Then, for a given set of vertices \( V_M \), the architecture infers the chances for \( M \) to appear. The key challenge is to be able to capture the richness of different motifs and their surroundings. We tackle this with an appropriate selection of negative samples, i.e., subgraphs that resemble the searched motifs but that are not identical to them. Moreover, when selecting the size of the “motif surroundings” we rely on an assumption also used in link prediction, which states that only the “close surroundings” (i.e., nearby vertices and edges, 1–2 hops away) of a link to be predicted have a significant impact on whether or not this link would appear [93, 95]. We use this assumption for motifs: as our evaluation shows, it ensures high accuracy while significantly reducing runtimes of training and inference (as only a small subgraph is used, instead of the whole input graph).

We call our GNN architecture SEAM: learning from Subgraphs, Embeddings and Attributes for Motif prediction\(^1\). Our evaluation (Section 5) illustrates the high accuracy of SEAM (often >90%), for a variety of graph datasets and motif sizes.

To motivate our work, we now compare SEAM and a proposed Jaccard-based heuristic that considers link correlations to two baselines that straightforwardly use link prediction independently for each motif link: a Jaccard-based score and the state-of-the-art SEAL scheme based on GNNs [93]. We show the results in Figure 1. The correlated Jaccard outperforms a simple Jaccard, while the proposed SEAM is better than SEAL. The benefits generalize to different graph datasets. Importantly, we observe that the larger the motif to predict becomes (larger \( k \)), the more advantageous our architecture delivers. This is because larger motifs provide more room for correlations between their associated edges. Straightforward link prediction based schemes do not consider this effect, while our methods

\(^1\) In analogy to SEAL [93, 95], which stands for “learning from Subgraphs, Embeddings, and Attributes for Link prediction”.

Figure 1: Motivating our work (SEAM): the accuracy (%) of predicting different motifs with SEAM compared to using a state-of-the-art SEAL link prediction scheme [93, 95] and a naive one that does not consider correlations between edges. The details of the experimental setup are in Section 5 (the dataset is USAir). Importantly: (1) SEAM outperforms all other methods, (2) the accuracy of SEAM increases with the size \( k \) of each motif, while in other methods it decreases.
do, which is why we offer more benefits for more complex motifs. The advantages of SEAM over the correlated Jaccard show that GNNs more robustly capture correlations and the structural richness of motifs than simple manual heuristics. Simultaneously, heuristics do not need any training.

Finally, SEAM also successfully predicts more arbitrary communities or clusters [16, 22, 39, 54]. They differ from motifs as they do not have a very specific fixed structure (such as a star) but simply have the edge density above a certain threshold. SEAM’s high accuracy in predicting such structures illustrates its potential for broader graph mining beyond motif analysis.

2 Background and Notation

We first describe the necessary background and notation.

Graph Model We model an undirected graph $G$ as a tuple $(V, E)$; $V$ and $E \subseteq V \times V$ are sets of nodes (vertices) and links (edges); $|V| = n$, $|E| = m$. Vertices are modeled with integers $1, ..., n$; $V = \{1, ..., n\}$. $N_v$ denotes the neighbors of $v \in V$; $d(v)$ denotes the degree of $v$.

Link Prediction We generalize the well-known link prediction problem. Consider two unconnected vertices $u$ and $v$. We assign a similarity score $s_{u,v}$ to them. All pairs of vertices that are not edges receive such a score and are ranked according to it. The higher a similarity score is, the “more likely” a given edge is to be missing in the data or to be created in the future. We stress that the link prediction scores are usually not based on any probabilistic notion (in the formal sense) and are only used to make comparisons between pairs of vertices in the same input graph dataset.

There are numerous known similarity scores. First, a large number of scores are called first order because they only consider the neighbors of $u$ and $v$ when computing $s_{u,v}$. Examples are the Common Neighbors scheme $s_{u,v}^{CN} = |N_u \cap N_v|$ or the Jaccard scheme $s_{u,v}^{J} = \frac{|N_u \cap N_v|}{|N_u \cup N_v|}$ [17].

These schemes assume that two vertices are more likely to be linked if they have many common neighbors. There also exist higher-order similarity schemes. Such schemes also consider vertices not directly attached to $u$ and $v$. All these schemes can be described using the same formalism of the $\gamma$-decaying heuristic proposed by [93]. Intuitively, for a given pair of vertices $(u, v)$, the $\gamma$-decaying heuristic for $(u, v)$ provides a sum of contributions into the link prediction score for $(u, v)$ from all other vertices, weighted in such a way that nearby vertices have more impact on the score.

Graph Neural Networks (GNNs) Graph neural networks (GNNs) are a recent class of neural networks for learning over irregular data such as graphs [26, 29, 72, 73, 78, 88, 90, 91, 96, 97]. There exists a plethora of models and methods for GNNs; most of them consist of two fundamental parts: (1) an aggregation layer that combines the features of the neighbors of each node, for all the nodes in the input graph, and (2) combining the scores into a new score. The input to a GNN is a tuple $G = (A, X)$. The input graph $G$ having $n$ vertices is modeled with an adjacency matrix $A \in \mathbb{R}^{n \times n}$. The features of vertices (with dimension $d$) are modeled with a matrix $X \in \mathbb{R}^{n \times d}$.

3 Motif Prediction: Formal Statement and Score Functions

We now formally establish the motif prediction problem. We define a motif as a pair $M = (V_M, E_M)$. $V_M$ is the set of existing vertices of $G$ that form a given motif ($V_M \subseteq V$). $E_M$ is the set of edges of $G$ that form the motif being predicted; some of these edges may already exist ($E_M \subset V_M \times V_M$).

We make the problem formulation (in § 3.1–§ 3.3) general: it can be applied to any graph generation process. Using this formulation, one can then devise specific heuristics that may assume some details on how the links are created, similarly as is done in link prediction. Here, we propose example motif prediction heuristics that harness the Jaccard, Common Neighbors, and Adamic-Adar link scores.

3.1 Motif Prediction vs. Link Prediction

We illustrate the motif prediction problem by discussing the differences between link and motif prediction. Figure 2 provides an overview of our work. We consider all these differences when proposing specific schemes for predicting motifs.

(M) There May Be Many Potential New Motifs for a Fixed Vertex Set Link prediction is a “binary” problem: for a given pair of vertices that are not connected, there can be only one link appearing. In
motif prediction, the situation is more complex. Consider vertices \( v_1, ..., v_k \). There are many possible motifs to appear between \( v_1, ..., v_k \). We now state a precise count; the proof is in the appendix.

**Observation 1.** Consider vertices \( v_1, ..., v_k \in V \). Assuming no edges already connecting \( v_1, ..., v_k \), there are \( 2^{(k)} - 1 \) motifs (with between 1 and \( \binom{k}{2} \) edges) that can appear to connect \( v_1, ..., v_k \).

**(E) Incoming Motifs May Have Existing Edges** A link can only appear between unconnected vertices. Contrarily, a motif can appear and connect vertices already with some edges between them.

**(D) There May Be “Deal-Breaker” Edges** There may be some edges, the appearance of which would make the appearance of a given motif unlikely or even impossible (e.g., existing chemical bonds could prevent other bonds). For example, consider a prediction query where one is interested whether a given vertex set can become connected with a star but in such a way that none of the non-central vertices are connected to one another. Now, if there is already some edge connecting these non-central vertices, this makes it impossible a given motif to appear while satisfying the query. We will refer to such edges as the “deal-breaker” edges.

**(L) Motif Prediction Query May Depend on Vertex Labeling** The query can depend on a specific vertex labeling. For example, when asking whether a 5-star will connect six given vertices \( v_1, ..., v_6 \), one may be interested in any 5-star connecting \( v_1, ..., v_6 \), or a 5-star connecting these vertices in a specific way, e.g., with its center being \( v_1 \). We enable the user to specify how edges in \( E_M \) should connect vertices in \( V_M \).

### 3.2 Types of Edges in Motifs

We first describe different forms of edges that are related to a motif. First, note that \( E_M = E_M, N \cup E_M,E \) where \( E_M, N \) are edges that do not exist in \( G \) at the moment of querying (\( \forall e \in E_M, N \ e \notin \mathcal{E} ; \mathcal{N} \) indicates “\( N \)-on-existing”) and \( E_M,E \) are edges that already exist, cf. (E) in § 3.1 (\( \forall e \in E_M,E \ e \in \mathcal{E} ; \mathcal{E} \) indicates “\( E \)-existing”). There may also be edges between vertices in \( V_M \) which do not belong to the motif \( M \) (i.e., they belong to \( E_{V_M} = \{ \{i, j\} : i, j \in V_M \land i \neq j \} \) but not \( E_M \)). We refer to such edges as \( E_M \) since \( E_{V_M} = E_M \cup E_{M,D} \) (i.e., a union of disjoints sets by definition). Some edges in \( E_M \) may be deal-breakers (cf. (D) in § 3.1), we denote them as \( E_{M,D} \) (\( D \) indicates “Deal-breaker”).

Non-deal-breakers that are in \( E_M \) are denoted with \( E_{M,I} \) (\( I \) indicates “\( I \)-nert”). Note that \( E_M = E_{M,D} \cup E_{M,I} \) and \( E_{M,D} = E_{V_M} \setminus E_M \). To conclude, as previously done for the set \( E_M \), we note that \( E_{M,D} = E_{M,D,I} \cup E_{M,D,E} \) where \( E_{M,D,I} \) are deal-breaker edges that do not exist in \( G \) at the moment of querying (\( \forall e \in E_{M,D,I} \ e \notin \mathcal{E} ; \mathcal{I} \) indicates “\( I \)-non-existing”) and \( E_{M,D,E} \) are deal-breaker edges that already exist, cf. (E) in § 3.1 (\( \forall e \in E_{M,D,E} \ e \in \mathcal{E} ; \mathcal{E} \) indicates “\( E \)-existing”). We explicitly consider \( E_{M,D,I} \) because – even if a given deal-breaker edge does not exist, but it does have a large chance of appearing – the overall motif score should become lower.

### 3.3 General Problem and Score Formulation

We now formulate a general motif prediction score. Analogously to link prediction, we assign scores to motifs, to be able to quantitatively assess which motifs are more likely to occur. Thus, one obtains a tool for analyzing future (or missing) graph structure, by being able to quantitatively compare different ways in which vertex sets may become (or already are) connected. Intuitively, we assume that a motif score should be high if the scores of participating edges are also high. This suggests one could reuse link prediction score functions.

Full extensive details of score functions, as well as more examples, are in the appendix.
A specific motif score function \( s(M) \) will heavily depend on a targeted problem. In general, we define \( s(M) \) as a function of \( V_M \) and \( E_M^* \); \( s(M) = s(V_M, E_M^*) \). Here, \( E_M^* = E_M \cup E_{M,D} \) are all the edges “that matter”: both edges in a motif (\( E_M \)) and the deal-breaker ones (\( E_{M,D} \)).

To obtain the exact form of \( s(M) \), we harness existing link prediction scores for edges from \( E_M \), when deriving \( s(M) \) (details in § 3.4–§ 3.5). When using first-order link prediction methods (e.g., Jaccard), \( s(M) \) depends on \( V_M \) and potential direct neighbors. With higher-order methods (e.g., Katz [49] or Adamic-Adar [2]), a larger part of the graph that is “around \( V_M \)” is considered for computing \( s(M) \). Here, our evaluation (details in Section 5) shows that, similarly to link prediction schemes [93], it is enough to consider a small part of \( G \) (1-2 hops away from \( V_M \)) to achieve high prediction accuracy for motifs.

However, we observe that simply extending link prediction fails to account for possible correlations between edges forming the motif (i.e., edges in \( E_M \)). Specifically, the appearance of some edges may impact (positively or negatively) the chances of one or more other edges in \( E_M \). We provide score functions that consider such correlations in § 3.5.

### 3.4 Starting Simple: Motif Scores Based on Independent Links

There exist many score functions for link prediction [5, 6, 59, 76]. Similarly, one can develop motif prediction score functions with different applications in mind. As an example, we discuss score functions for a graph that models a set of people. An edge between two vertices indicates that two given persons know each other. For simplicity, let us first assume that there are no deal-breaker edges, thus \( E_M^* = E_M \). For a set of people \( V_M \), we set the score of a given specific motif \( M = (V_M, E_M) \) to be the product of the scores of the associated edges: \( s_\perp(M) = \prod_{e \in E_{M,N}} s(e) \) where \( \perp \) denotes the independent aggregation scheme. Here, \( s(e) \) is any link prediction score which outputs into \([0, 1]\) (e.g., Jaccard). Thus, also \( s_\perp(M) \in [0, 1] \) by construction. Moreover, this score implicitly states that \( \forall e \in E_{M,D} \) we set \( s(e) = 1 \). Clearly, this does not impact the motif score \( s_\perp(M) \) as the edges are already existing. Overall, we assume that a motif is more likely to appear if the edges that participate in that motif are also more likely. Now, when using the Jaccard Score for edges, the motif prediction score becomes \( s_\perp(M)^j = \prod_{e,u,v \in E_{M,N}} \frac{s(e)}{|N_u \cap N_v|} \).

To incorporate deal-breaker edges, we generalize the motif score defined previously as \( s^*_\perp(M) = \prod_{e \in E_M} s(e) \cdot \prod_{e \in E_{M,D}} (1 - s(e)) \), where the product over \( E_M \) includes partial scores from the edges that belong to the motif, while the product over \( E_{M,D} \) includes the scores from deal-breaker edges. Here, the larger the chance for a \( e \) to appear, the higher its score \( s(e) \) is. Thus, whenever \( e \) is a deal-breaker, using \( 1 - s(e) \) has the desired diminishing effect on the final motif score \( s^*_\perp(M) \).

### 3.5 Capturing Correlations between Links in a Motif

The main challenge is how to aggregate the link predictions taking into account the rich structural properties of motifs. Intuitively, using a plain product of scores implicitly assumes the independence of participating scores. However, arriving links may increase the chances of other links’ appearance in non-trivial ways. To capture such positive correlations, we propose heuristics based on the convex linear combination of link scores. To show that such schemes consider correlations, we first (Proposition 3.1) prove that the product \( P \) of any numbers in \([0, 1]\) is always bounded by the convex linear combination \( C \) of those numbers (the proof is in the appendix). Thus, our motif prediction scores based on the convex linear combination of link scores are always at least as large as the independent products of link scores (as we normalize them to be in \([0, 1]\), see § 3.6). The difference \( (C - P) \) is due to link correlations. Details are in § 3.5.1.

**Proposition 3.1.** Let \( \{x_1, \ldots, x_n\} \) be any finite collection of elements from \( U = \{x \in \mathbb{R} : 0 \leq x \leq 1\} \). Then, \( \forall n \in \mathbb{N} \) we have \( \prod_{i=1}^{n} x_i \leq \sum_{i=1}^{n} w_i x_i \), where \( w_i \geq 0 \forall i \in \{1, \ldots, n\} \) and subject to the constraint \( \sum_{i=1}^{n} w_i = 1 \).

For negative correlations caused by deal-breaker edges, i.e., correlations that lower the overall chances of some motif to appear, we introduce appropriately normalized scores with a negative sign in the weighted score sum. The validity of this approach follows from Proposition 3.1 by noting that \( \prod_{i=1}^{n} x_i \geq -\sum_{i=1}^{n} w_i x_i \) under the conditions specified in the proposition. This means
that any combination of such negatives scores is always lower than the product of scores $P$; the difference $|C - P|$ again indicates effects between links not captured by $P$. Details are in § 3.5.2.

### 3.5.1 Capturing Positive Correlation

In order to introduce positive correlation, we set the score of a given specific motif $M = (V_M, E_M)$ to be the convex linear combination of the vector of scores of the associated edges:

$$s(M) = f(s(e)) = \langle w, s(e) \rangle$$  \hspace{1cm} (1)

Here, $f(s(e)) : [0, 1]^{|E_M|} \rightarrow [0, 1]$ with $|E_M| = |E_{V_M} \setminus \overline{E}_M|$ (i.e., not considering either $\overline{E}$ or Deal-breaker edges). In the weight vector $w \in [0, 1]^{|E_M|}$, each component $w_i$ is larger than zero, subject to the constraint $\sum_{i=1}^{|E_M|} w_i = 1$. Thus, $s(M)$ is a convex linear combination of the vector of link prediction scores $s(e)$. Finally, we assign a unit score for each existing edge $e \in E_{M,e}$.

Now, to obtain a correlated Jaccard score for motifs, we set a score for each $\overline{E}$-non-existing edge $e_{(u,v)}$ as $\frac{|N_u \cap N_v|}{|N_u | + |N_v | - |N_u \cap N_v |}$, $\overline{E}$-existing edges each receive scores 1. Finally, we set the weights as $w = 1 \frac{1}{|E_M|}$, assigning the same importance to each link within the motif $M$. This gives $s(M)^J = \frac{1}{|E_M|} \left( \sum_{e_{(u,v)} \in E_{M,\overline{E}}} \frac{|N_u \cap N_v|}{|N_u | + |N_v | - |N_u \cap N_v |} + |E_{M,e}| \right)$. Any choice of $w_i > \frac{1}{|E_M|}$ places a larger weight on the $i$-th edge (and lower for others due to the constraint $\sum_{i=1}^{|E_M|} w_i = 1$). In this way we can incorporate domain knowledge for the motif of interest. For example, in the comparison of Figure 3, we set $w = 1 \frac{1}{|E_M|}$ because of the relevant presence of $E$-existing edges (each receiving a null score).

### 3.5.2 Capturing Negative Correlation from Deal-Breaker Links

To capture negative correlation potentially coming from deal-breaker edges, we assign negative signs to the respective link scores. Let $e \in E_M^* = E_M \cup \overline{E}_M$. Then we set $s^*_i(e) = -s_i(e)$ if $e \in E_{M,D,N}$, $\forall i \in \{1, ..., |E_M^*|\}$. Moreover, if there is an edge $e \in E_{M,D,E}$, we have $s^*_i(e) = 0$. Assigning a negative link prediction score to a potential Deal-breaker edge lowers the score of the motif. Setting $s^*(e) = 0$ when at least one Deal-breaker edge exists, allows us to rule out motifs which cannot arise. We now state a final motif prediction score:

$$s^*(M) = f(s^*(e)) = \max(0, \langle w, s^*(e) \rangle)$$  \hspace{1cm} (2)

Here $s^*(M) : [0, 1]^{|E_M^*|} \rightarrow [0, 1]$ with $|E_M^*| \leq \binom{|V_M|}{2}$. Furthermore, we apply a rectifier on the convex linear combination of the transformed scores vector (i.e., $\langle w, s^*(e) \rangle$) with the rationale that any negative motif score implies the same impossibility of the motif to appear. All other score elements are identical to those in Eq. (1).
3.6 Normalization of Scores for Meaningful Comparisons and General Applicability

The motif scores defined so far consider only link prediction scores \( s(e) \) with values in \([0, 1]\). Thus, popular heuristics such as Common Neighbors, Preferential Attachment, and the Adamic-Adar index do not fit into this framework. For this, we introduce a normalized score \( s(e)/c \) enforcing \( c \geq \|s(e)\|_\infty \) since the infinity norm of the vector of scores is the smallest value that ensures the desired mapping (the ceil function defines a proper generalization as \( \|s(e)\|_\infty = 1 \) for, e.g., Jaccard [17]). To conclude, normalization also enables the meaningful comparison of scores of different motifs which may differ in size or in their edge sets \( E_M \).

4 The SEAM Architecture for Training and Inference with GNNs

We argue that one could also use neural networks to learn a heuristic for motif prediction. Following recent work on link prediction [93, 95], we use a GNN for this; a GNN may be able to learn link correlations better than a simple hand-designed heuristic. Simultaneously, heuristics are still important as they do not require expensive training. We now describe the architecture of our design called SEAM (learning from Subgraphs, Embeddings and Attributes for Motif prediction).

4.1 Overview

Let \( M = (V_M, E_M) \) be a motif to be predicted in \( G \). First, we extract the already existing instances of \( M \) in \( G \), denoted as \( G_p = (V_p, E_p) \); \( V_p \subseteq V \) and \( E_p \subseteq E \). We use these instances \( G_p \) to generate positive samples for training and validation. To generate negative samples (details in § 4.2), we find subgraphs \( G_n = (V_n, E_n) \) that do not form a motif \( M \) (i.e., \( V_n \subseteq V_M \) and \( E_M \nsubseteq E_n \) or \( E_M \cap E_n \neq \emptyset \)). Then, for each positive and negative sample, consisting of sets of vertices \( V_p \) and \( V_n \), we extract a “subgraph around this sample”, \( G_s = (V_s, E_s) \), with \( V_p \subseteq V_s \subseteq V \) and \( E_p \subseteq E_s \subseteq E \), or \( V_n \subseteq V_s \subseteq V \) and \( E_n \subseteq E_s \subseteq E \) (details in § 4.3). Here, we rely on the insights from SEAL [93] on their \( \gamma \)-decaying heuristic, i.e., it is \( G_s \), the “surroundings” of a given sample (be it positive or negative), that are important in determining whether \( M \) appears or not. The nodes of these subgraphs are then appropriately labeled to encode the structural information (details in § 4.5). With these labeled subgraphs, we train our GNN, which classifies each subgraph depending on whether or not vertices \( V_p \) or \( V_n \) form the motif \( M \). After training, we evaluate the real world accuracy of our GNN by using the validation dataset. The GNN architecture is detailed in § 4.6.

4.2 Positive and Negative Sampling

We need to provide a diverse set of samples to ensure that SEAM works reliably on a wide range of real data. For the positive samples, this is simple because the motif to be predicted \( (M) \) is specified. Negative samples are more challenging, because – for a given motif – there are many potential “false” motifs. In general, for each motif \( M \), we generate negative samples using three strategies. (1) We first select positive samples and then remove a few vertices, replacing them with other nearby vertices (i.e., only a small number of motif edges are missing or only a small number of deal-breaker edges are added). Such negative samples closely resemble the positive ones. (2) We randomly sample \( V_M \) vertices from the graph; such negative samples are usually sparsely connected and do not resemble the positive ones. (3) We select a random vertex \( r \) into an empty set, and then we keep adding randomly selected vertices from the neighborhood of vertices already in the set, growing a subgraph until reaching the size of \( V_M \); such negative samples may resemble the positive ones to a certain degree. The final set of negative samples usually contains about 80% samples generated by strategy (1) and 10% each of samples generated by (2) and (3). This distribution could be adjusted based on domain knowledge of the input graph (we also experiment with other ratios). Strategies (2) and (3) are primarily used to avoid overfitting of our model.

As an example, let our motif \( M \) be a closed triangle. More precisely, \( |V_M| = 3 \) and \( |E_M| = 3 \). Consider a simple approach of generating negative samples, in which one randomly samples 3 vertex indices and verifies if there is a closed triangle between them. If we use these samples, in our evaluation for considered real world graphs, this leads to a distribution of 90% unconnected samples \( |E_n| = 0 \), 9% samples with \( |E_n| = 1 \) and only about 1% of samples with \( |E_n| = 2 \). Thus, if we train our GNN with this dataset, it would hardly learn the difference between open triangles \( |E_n| = 2 \) and closed triangles \( |E_M| = 3 \). Therefore, we provide our negative samples by ensuring that a third of
samples are open triangles $|E_n| = 2$ and another third of samples have one edge $|E_M| = 1$. For the remaining third of samples, we use the randomly generated vertex indices described above, which are mostly unconnected vertices $|E_M| = 0$.

For dense subgraphs, the sampling is less straightforward. Overall, the goal is to find samples with edge density being either close to, or far away from, the density threshold of a dense subgraph to be predicted. If the edge density of the sampled subgraph is lower than the density threshold it becomes a negative sample and vice versa. The samples chosen further away from the density threshold are used to prevent overfitting similar to strategies (2) and (3) from above. For this, we grow a set of vertices $R$ (starting with a single random vertex), by iteratively adding selected neighbors of vertices in $R$ such that we approach the desired density.

Overall, we choose equally many positive and negative samples to ensure a balanced dataset. Furthermore, we limit the number of samples if there are too many, by taking a subset of the samples (selected uniformly at random). The positive and negative samples are split into a training dataset and a validation dataset. This split is typically done in a 9/1 ratio. To ensure an even distribution of all types of samples in these two datasets, we randomly permute the samples before splitting them.

### 4.3 Extracting Subgraphs Containing Positive and Negative Samples

To reduce the computational costs of our GNN, we do not use the entire graph $G$ as input in training or validation. Instead, we rely on recent insights on link prediction with GNNs [93, 95], which illustrate that it suffices to provide a subgraph capturing the “close surroundings” (i.e., 1–2 hops away) of the vertices we want to predict a link between, cf. Section 2. We take an analogous assumption for motifs (our evaluation confirms the validity of the assumption). For this, we define the “surroundings” of a vertex $v$ (selected uniformly at random). The positive and negative samples are split into a training dataset and a validation dataset. This split is typically done in a 9/1 ratio. To ensure an even distribution of all types of samples in these two datasets, we randomly permute the samples before splitting them.

### 4.4 Node Embeddings

In certain cases, the $h$-hop enclosing subgraph might miss some relevant information about the motif in question. To alleviate this, while simultaneously avoiding sampling a subgraph with large $h$, we also generate a node embedding $X_E \in \mathbb{R}^{n \times f}$ which encodes the information about more distant graph regions using random walks. For this, we employ the established node2vec [40] with the parameters from DeepWalk [67]. $f$ is the dimension of the low-dimensional vector representation of a node. We generate such a node embedding once and then only append the embedding vectors (corresponding to the nodes in the extracted subgraph) to the feature matrix of each extracted subgraph.

We obtain (cf. §4.5)

$$X_s = (X_{s_1}, X_{s_E}, X_H, X_L, X_E) \in \mathbb{R}^{s \times (d+2f+2k)}.$$  

Here, we also extend the SEAL approach called negative injection for more effective embeddings [93, 95]. The authors of SEAL observe that if embeddings are constructed using the edge set containing positive training samples, the GNN would focus on fitting this part of information. Thus, SEAL generates embedding based on the edge set containing also negative training samples, which ultimately improves accuracy. In SEAM, we analogously include all potential motif and deal-breaker edges $E^*_M$ of all training samples to the input graph when generating the node embedding.
4.5 Node Labeling for Structural Features

In order to provide our GNN with as much structural information as possible, we introduce two node labeling schemes. These schemes serve as structural learning features, and we use them when constructing feature matrices of the extracted subgraphs, fed into a GNN. Let $s$ be the total number of vertices in the extracted subgraph $G_s$, and $k$ be the number of vertices forming the motif. We call the vertices in the respective samples ($V_p$ or $V_n$) the inner vertices since they form a motif sample. The rest of the nodes in the subgraph $G_s$ are called outer vertices.

The first label is simply an enumeration of all the inner vertices. We call this label the inner label. It enables ordering each vertex according to its role in the motif. For example, to predict a $k$-star, we always assign the inner label 1 to the star central vertex. This inner node label gets translated (e.g., with and without embeddings described in § 4.4).

After each epoch, to validate the accuracy, we simply generate predictions with the original information about the motifs. Ultimately, we expect our model to predict correct predictions; these results resemble the AUC ones, see the appendix.

The second label is called the outer label. The label assigns to each outer vertex its distances to each inner vertex. Thus, each of the $s-k$ outer vertices get $k$ labels. The first of these $k$ labels describes the distance to the vertex with inner label 1. All these outer labels form a labeling matrix $L \in \mathbb{N}^{(s-k) \times k}$, appended with a zero matrix $0_{(s-k)k} \in \mathbb{N}^{(s-k) \times k}$, becoming $X_L = (0_{kk} L)^T \in \mathbb{N}^{k \times k}$. The final feature matrix $X_s$ of the respective subgraph $G_s$ consists of $X_H$, $X_L$, the subgraph node embedding matrix $X_E$ and the subgraph input feature matrix $X_{s_1} \in \mathbb{R}^{s \times d}$; we have $X_s = (X_{s_1} X_E X_H X_L) \in \mathbb{R}^{s \times (d+f+2k)}$; $d$ is the dimension of the input feature vectors and $f$ is the dimension of the node embedding vectors.

4.6 Used Graph Neural Network Design

For our GNN model, we use the graph classification neural network DGCNN [94], used in SEAL [93, 95]. We now summarize its architecture. The first stage of this GNN consist of three graph convolution layers (GConv). Each layer distributes the vertex features of each vertex to its neighbors. Then, we feed the output of each of these GConv layers into a layer called k-sortpooling where all vertices are sorted based on their importance in the subgraph. After that, we apply a standard 1D convolution layer followed by a dense layer, followed by a softmax layer to get the prediction probabilities.

The input for our GNN model is the adjacency matrix of the selected $h$-hop enclosing subgraph $G^h_{V_p}$ together with the feature matrix $X_{s_1}$. With these inputs, we train our GNN model for 100 epochs. After each epoch, to validate the accuracy, we simply generate $G^h_{V_p}$ and $G^h_{V_n}$ as well as their feature matrix $X_{p}$ and $X_{n}$ from our samples in the validation dataset. We know for each set of vertices $V_p$ or $V_n$, if they form the motif $M$. Thus, we can analyse the accuracy of our model by comparing the predictions with the original information about the motifs. Ultimately, we expect our model to predict the set of vertices $V_p$ to form the motif $M$ and the set of vertices $V_n$ not to form the motif $M$.

5 Evaluation

We now illustrate the advantages of our correlated heuristics and of our learning architecture SEAM. We feature a representative set of results, extended results are in the appendix.

As comparison targets, we use motif prediction based on three link prediction heuristics (Jaccard, Common Neighbors, Adamic-Adar), and on the GNN based state-of-the-art SEAL link prediction scheme [93, 95]. Here, the motif score is derived using a product of link scores with no link correlation (“Mul”). We also consider our correlated heuristics, using link scores, where each score is assigned the same importance (“Avg”, $w = 1/|EM_{t,M}|$), or the smallest link score is assigned the highest importance (“Min”). This gives a total of 12 targets. We then consider different variants of SEAM (e.g., with and without embeddings described in § 4.4).

As the accuracy metric, we use AUC (Area Under the Curve), the standard metric to evaluate the accuracy of any classification model in machine learning. We also consider a plain fraction of all correct predictions; these results resemble the AUC ones, see the appendix.
We compare the accuracy of (1) our heuristics from Section 3, (2) a scheme using the SEAL link prediction, and (3) our proposed SEAM GNN architecture. The results are in Figure 3. First, we observe that the improvement in accuracy in SEAM almost always scales with the size of the motif. This shows that SEAM captures correlation between different edges (in larger motifs, there is more potential correlation between links). Importantly, the advantages and the capacity of SEAM to capture correlations, also hold in the presence of deal-breaker edges (“k-db-star”). Here, we assign links connecting pairs of star outer vertices as deal-breakers (e.g., 7-db-star is a 7-star with 15 deal-breaker edges connecting its arms with one another). We observe that the accuracy for k-stars with deal-breaker edges is lower than that for standard k-stars. However, SEAM is still the best baseline since it appropriately learns such edges and their impact on the motif appearance.

Second, our correlated heuristics (“Avg”, “Min”) improve the motif prediction accuracy over methods that assume link independence (“Mul”). Overall, in heuristics, we observe that $AUC_{Mul} < AUC_{Min} < AUC_{Avg}$ (except for k-db-stars). This shows that different aggregation schemes have different capacity in capturing the rich correlation structure of motifs. In particular, notice that “Min” is by definition (cf. Proposition 3.1) a lower bound of the score $s(M)$ defined in § 3.5.1. This implies that it is the smallest form of correlation that we can include in our motif score given the convex linear combination function proposed in § 3.5.1.

Interestingly, the Common Neighbors heuristic performs poorly. This is due to the similar neighborhoods of the edges that have to be predicted. The high similarity of these neighborhoods is caused by our subgraph extraction strategy discussed in Section 4.3, where we select the existing motif edges of the positive samples in such a way as to mimic the edge structure of the negative samples. These results show also that different heuristics do not perform equally with respect to the task of motif prediction and further studies are needed in this direction.

The accuracy benefits of SEAM over the best competitor (SEAL using the “Avg” way to compose link prediction scores into motif prediction scores) range from 12% to almost 32%. This difference is even larger for other methods; it is because there comparison targets cannot effectively capture link correlations in motifs. This result shows that the edge correlation in motifs is important to accurately predict a motif’s appearance, and that it benefits greatly from being learned by a neural network.

Finally, we also provide the results for more arbitrary structures (clusters, communities), see Figure 4. Here, “k-dense” indicates a cluster of $k$ vertices, with at least 90% of all possible edges present. The results follow similar trends to those in Figure 3; SEAM outperforms all other baselines. Its accuracy also increase with the increasing motif size.
5.2 Analysis of Different Aspects of Motif Prediction

We analyze the impact from embeddings. Interestingly, it consistently (by 0.2 – 4%) improves the accuracy while simultaneously reducing the variance in most cases by around 50% for cliques and dense clusters. We also consider other aspects, for example, we vary the number of existing edges, and even eliminate all such edges; the results follow similar patterns to those observed above. Finally, SEAM’s running times heavily depend on the used model parameters. A full SEAM execution on the Yeast graph dataset with 40,000 training samples and 100 training epochs does typically take between 15–75 minutes (depending heavily on the complexity of the motif, with stars and dense clusters being the fastest and slowest to process, respectively). The used hardware configuration includes an Intel 6130 @2.10GHz with 32 cores and an Nvidia V100 GPU; details are in the Appendix.

6 Related Work

We mention schemes related to prediction in Section 1. Next, many works exist on listing, counting, or finding different patterns [4, 6, 16, 20, 22, 24, 27, 28, 33, 34, 37, 43, 46, 54, 55, 57, 59, 68–70, 75, 81, 86]. Yet, they do not focus on prediction. Third, different works analyze the temporal aspects of motifs [48, 52, 53, 58, 66, 82], or use motif features for predicting links [1]. However, none of them considers prediction of general motifs. Moreover, there exists an extensive body of work on graph processing and algorithms, both static and dynamic (also called temporal, time-evolving, or streaming) [11–14, 18, 19, 25, 38, 50, 71, 74]. Still, they do not consider prediction of motifs. Finally, we use GNNs [73, 90, 97] for making predictions about motif appearance. An interesting venue of future work would be harnessing GNNs for other graph related tasks, such as compression [15, 21, 23].

There exist several recent frameworks for computations on GNNs [35, 42, 44, 45, 51, 56, 80, 85, 89, 92, 98]. While we use Pytorch Geometric, motif prediction can be integrated into any of these frameworks to enhance their processing capabilities. An interesting line of work would be to implement motif prediction using the serverless paradigm [7, 31, 47, 61], for example within one of recent dedicated serverless engines [79].

7 Conclusion & Discussion

Higher-order network analysis is an important approach for mining irregular data. Yet, it lacks methods and tools for predicting the evolution of the associated datasets. For this, we establish a problem of predicting general complex graph structures called motifs, such as cliques or stars. We illustrate its differences to simple link prediction, and then we propose heuristics for motif prediction that are invariant to the motif size and capture potential correlations between links forming a motif. Our analysis enables incorporating domain knowledge, and thus – similarly to link prediction – it can be a foundation for developing motif prediction schemes within specific domains.

While being fast, manual heuristics leave some space for improvements in prediction accuracy. To address this, we develop a graph neural network (GNN) architecture for predicting motifs. We show that it outperforms the state of the art, offering excellent accuracy, which improves with the growing size and complexity of the predicted motif. We also successfully apply our architecture to predicting more arbitrarily structured clusters, indicating its broader potential in mining irregular data.

**Discussion: Societal Impact** After extensively reviewing the available link prediction surveys [5, 6, 59, 76], we conclude that our work does not have any direct negative impact. Issues from more effective methods for mining data should be tackled using established privacy methods [3, 65, 77, 84].

**Discussion: Limitations & Future Work** We currently do not yet support certain motifs (e.g., k-clubs) and input graphs with weighted edges and vertices. Moreover, SEAM does not use distributed training for more scale. All these aspects are future work.

**Acknowledgments and Disclosure of Funding**

We thank Hussein Harake, Colin McMurtrie, Mark Klein, Angelo Mangili, and the whole CSCS team granting access to the Ault and Daint machines, and for their excellent technical support. We thank Timo Schneider for immense help with computing infrastructure at SPCL.
References


Appendix A: Proofs

We recall the statement of Observation 1 in Section 3.1:

Consider vertices $v_1, ..., v_k \in V$. Assuming no edges already connecting $v_1, ..., v_k$, there are $2^\binom{k}{2} - 1$ motifs (with between 1 and $\binom{k}{2}$ edges) that can appear to connect $v_1, ..., v_k$.

Proof. We denote as $E_k = \{\{i, j\} : i, j \in V_k \land i \neq j\}$ the edge set of the undirected subgraph $(V_k, E_k)$ with $V_k \subseteq V$. The number of all possible edges between $k$ vertices is $|E_k| = \binom{k}{2}$. Any subset of $E_k$, with the exception of the empty set, defines a motif. Thus the set of all possible subsets (i.e., the power set $\mathcal{P}$) of $E_k$ is the set of motifs. Then, since $|\mathcal{P}(E_k)| = 2^\binom{k}{2}$, we subtract the empty set (which we consider as an invalid motif) from the total count to obtain the desired result. \qed
We recall the statement of Proposition 3.1 in Section 3.5:

Let \( \{ x_1, \ldots, x_n \} \) be any finite collection of elements from \( U = \{ x \in \mathbb{R} : 0 \leq x \leq 1 \} \). Then, \( \forall n \in \mathbb{N} \) we have \( \prod_{i=1}^{n} x_i \leq \sum_{i=1}^{n} w_i x_i \), where \( w_i \geq 0 \) \( \forall i \in \{1, \ldots, n\} \) and subject to the constraint \( \sum_{i=1}^{n} w_i = 1 \).

**Proof.** We start by noticing that \( \prod_{i=1}^{n} x_i \leq \min \{ x_1, \ldots, x_n \} \). This is trivial to verify if \( \exists i \) such that \( x_i = 0 \) for \( i \in \{1, \ldots, n\} \). Otherwise, it can be shown by contradiction: imagine that \( \prod_{i=1}^{n} x_i > \min \{ x_1, \ldots, x_n \} \).

We know that \( U \) is closed with respect to the product (i.e., \( \prod_{i=1}^{n} x_i \in U \) \( \forall n \in \mathbb{N} \)). Then, we can divide both sides by \( \min \{ x_1, \ldots, x_n \} \), since we ruled out the division by zero, to obtain \( \prod_{i=1}^{n-1} x_i > 1 \).

This implies \( \prod_{i=1}^{n-1} x_i \notin U \), which contradicts that \( U \) is closed to the product. For the right side of the original statement, we know by definition that \( x_i \geq \min \{ x_1, \ldots, x_n \} \) \( \forall i \in \{1, \ldots, n\} \).

Since \( w_i \geq 0 \), we can also write that \( w_i x_i \geq \min \{ x_1, \ldots, x_n \} \) \( \forall i \in \{1, \ldots, n\} \).

Thus, since \( U \) is an ordered set, we can state that \( \sum_{i=1}^{n} w_i x_i \geq \sum_{i=1}^{n} w_i \min \{ x_1, \ldots, x_n \} \).

But then, since \( \sum_{i=1}^{n} w_i \min \{ x_1, \ldots, x_n \} = \min \{ x_1, \ldots, x_n \} \), we conclude that \( \min \{ x_1, \ldots, x_n \} \leq \sum_{i=1}^{n} w_i x_i \).

This ends the proof thanks to the transitive property. \( \square \)

**Appendix B: Details of Datasets**

In this section, we provide additional details on the various datasets that we used. We selected networks of different origins (biological, engineering, transportation), with different structural properties (different sparsities and skews in degree distributions).

**USAir** [8] is a graph with 332 vertices and 2,126 edges representing US cities and the airline connections between them. The vertex degrees range from 1 to 19 with an average of 9.8. Power [87] is the electrical grid of the Western US with 4,941 vertices and 6,594 edges. The vertex degrees range from 1 to 19 with an average degree of 2.7.

**Appendix C: Analysis and Decision Process of SEAM Model Parameters**

We now discuss in more detail the selection of the SEAM model parameters.

**Choosing learning rate and number of epochs**

We first describe the process of tuning the hyperparameters for our motif prediction framework. In order to find the optimal learning rate for SEAM we try different learning rates as shown in Figures 5, 6 and 7. The associated hyperparameters are highly dependent on the specific motif to be predicted and on the used dataset. As an example, we analyze the hyperparameters for \( k \)-stars and \( k \)-cliques on the USAir graph dataset. The plots show that there is a sweet spot for the learning rate at 0.001-0.002. Any value below that rate is too small and our model cannot train its neural network effectively, while for the values above that, the model is unable to learn the often subtle differences between hard negative samples and positive samples. The number of epochs of the learning process can be chosen according to the available computational power of the user.

<table>
<thead>
<tr>
<th>Learning Rate</th>
<th>AUC Score</th>
<th>3-star</th>
<th>5-star</th>
<th>7-star</th>
<th>3-clique</th>
<th>5-clique</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.000125</td>
<td>84.84 ± 0.91</td>
<td>82.04 ± 0.64</td>
<td>85.35 ± 0.01</td>
<td>84.07 ± 0.95</td>
<td>85.08 ± 1.91</td>
<td>87.96 ± 1.18</td>
</tr>
<tr>
<td>0.00025</td>
<td>84.58 ± 0.91</td>
<td>82.04 ± 0.64</td>
<td>85.35 ± 0.01</td>
<td>84.07 ± 0.95</td>
<td>85.08 ± 1.91</td>
<td>87.96 ± 1.18</td>
</tr>
<tr>
<td>0.0005</td>
<td>84.64 ± 0.91</td>
<td>82.04 ± 0.64</td>
<td>85.35 ± 0.01</td>
<td>84.07 ± 0.95</td>
<td>85.08 ± 1.91</td>
<td>87.96 ± 1.18</td>
</tr>
<tr>
<td>0.001</td>
<td>84.68 ± 0.91</td>
<td>82.04 ± 0.64</td>
<td>85.35 ± 0.01</td>
<td>84.07 ± 0.95</td>
<td>85.08 ± 1.91</td>
<td>87.96 ± 1.18</td>
</tr>
<tr>
<td>0.002</td>
<td>84.72 ± 0.91</td>
<td>82.04 ± 0.64</td>
<td>85.35 ± 0.01</td>
<td>84.07 ± 0.95</td>
<td>85.08 ± 1.91</td>
<td>87.96 ± 1.18</td>
</tr>
<tr>
<td>0.004</td>
<td>84.76 ± 0.91</td>
<td>82.04 ± 0.64</td>
<td>85.35 ± 0.01</td>
<td>84.07 ± 0.95</td>
<td>85.08 ± 1.91</td>
<td>87.96 ± 1.18</td>
</tr>
<tr>
<td>0.008</td>
<td>84.80 ± 0.91</td>
<td>82.04 ± 0.64</td>
<td>85.35 ± 0.01</td>
<td>84.07 ± 0.95</td>
<td>85.08 ± 1.91</td>
<td>87.96 ± 1.18</td>
</tr>
<tr>
<td>0.016</td>
<td>84.84 ± 0.91</td>
<td>82.04 ± 0.64</td>
<td>85.35 ± 0.01</td>
<td>84.07 ± 0.95</td>
<td>85.08 ± 1.91</td>
<td>87.96 ± 1.18</td>
</tr>
<tr>
<td>0.032</td>
<td>84.88 ± 0.91</td>
<td>82.04 ± 0.64</td>
<td>85.35 ± 0.01</td>
<td>84.07 ± 0.95</td>
<td>85.08 ± 1.91</td>
<td>87.96 ± 1.18</td>
</tr>
</tbody>
</table>

Figure 5: AUC-Score comparison for different learning rates and a training duration of 50 epochs on USAir graph.

SEAM parameters: proposed labels enabled, proposed embedding disabled, number of epochs = 50, training dataset size = 100,000
We want to assess the smallest number of samples that still ensures an effective learning process. We also analyze the effect of different training dataset sizes on the prediction strength of SEAM. Here, we analyze the effects and contributions from different variants of SEAM. First, we investigate the accuracy improvements due to our proposed labeling scheme in Section 4.5. Then, we empirically justify our approach to only sample the $h$-hop enclosing subgraph for small $h$ (1–2). Finally, we evaluate the performance of every prediction method if there are no motif edges already present.

### Analysis of different training dataset sizes

Figure 8 shows the different accuracy results of SEAM, for different motifs and training dataset sizes. We observe that the accuracy strongly depends on the motif to be predicted. For example, a dense subgraph can be predicted with high accuracy with only 100 training samples. On the other hand, prediction accuracy of the 5-star motif improves proportionally to the amount of training samples while still requiring more samples (than plain dense subgraphs) for a high accuracy score. For all motifs, we set our minimal amount of training samples to 20,000 for positive and for negative ones.

### Appendix D: Analysis of Different Variants of Motif Prediction in SEAM

Here, we analyze the effects and contributions from different variants of SEAM.
Effect on accuracy of our proposed labeling scheme

Figure 9 shows that our proposed labeling scheme generally has a positive impact on the accuracy of SEAM. The exception is the $k$-star motif. For $k = 3$, the labeling scheme significantly improves the accuracy. On the other hand, using $k > 3$ reduces the accuracy while simultaneously increasing the variance of test results. This effect can be explained with the implementation details of our labeling scheme. We remove every edges between all the motif vertices to calculate our $k$-dimensional distance labels. This procedure seems to misrepresent the structure of $k$-stars for $k > 3$. There are possible improvements to be gained in future work by further optimizing our labeling scheme.

<table>
<thead>
<tr>
<th>Motif Type</th>
<th>SEAM no labels</th>
<th>SEAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-star</td>
<td>82.75 ± 0.75</td>
<td>94.71 ± 0.32</td>
</tr>
<tr>
<td>5-star</td>
<td>94.71 ± 0.32</td>
<td>98.86 ± 0.12</td>
</tr>
<tr>
<td>7-star</td>
<td>98.86 ± 0.12</td>
<td>92.21 ± 0.99</td>
</tr>
<tr>
<td>3-clique</td>
<td>92.21 ± 0.99</td>
<td>97.51 ± 0.35</td>
</tr>
<tr>
<td>5-clique</td>
<td>97.51 ± 0.35</td>
<td>98.88 ± 0.27</td>
</tr>
<tr>
<td>7-clique</td>
<td>98.88 ± 0.27</td>
<td>81.41 ± 0.95</td>
</tr>
<tr>
<td>3-dbstar</td>
<td>81.41 ± 0.95</td>
<td>85.39 ± 0.64</td>
</tr>
<tr>
<td>5-dbstar</td>
<td>85.39 ± 0.64</td>
<td>87.58 ± 0.64</td>
</tr>
<tr>
<td>7-dbstar</td>
<td>87.58 ± 0.64</td>
<td>88.78 ± 1.49</td>
</tr>
</tbody>
</table>

Figure 9: Effect of our proposed labeling scheme on USAir graph.

SEAM parameters: h-hop = 1, proposed embedding enabled, learning rate = 0.002, number of epochs = 100, training dataset size = 100,000

Effect on accuracy of different $h$-hop enclosing subgraphs

Zhang et al. [93] motivated the use of small $h$-hop neighborhoods for their SEAL framework with the $\gamma$-decaying heuristic. We now provide additional data to backup this decision in SEAM. Figures 11 and 10 show that in most cases there is not much performance to be gained by sampling an $h$-hop enclosing subgraph with $h > 2$. This effect is especially striking for sparse graph datasets like the Power shown in Figure 11. The accuracy starts to drop significantly for $h > 2$. The only outlier in our little test was the 5-star motif shown in Figure 10. This effect was most likely caused by the specifics of this particular dataset and it does reflect a trend for other graphs. An additional explanation could also be the non-optimal labeling implementation for the 5-star motif. These special cases do not justify to increase the neighborhood size of the motif in a general case.

<table>
<thead>
<tr>
<th>Motif Type</th>
<th>1-hop</th>
<th>2-hop</th>
<th>3-hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-star</td>
<td>89.98 ± 1.28</td>
<td>92.72 ± 0.71</td>
<td>93.88 ± 0.71</td>
</tr>
<tr>
<td>5-star</td>
<td>92.72 ± 0.71</td>
<td>94.90 ± 0.56</td>
<td>73.86 ± 3.61</td>
</tr>
<tr>
<td>7-star</td>
<td>93.88 ± 0.71</td>
<td>73.86 ± 3.61</td>
<td>73.42 ± 4.96</td>
</tr>
<tr>
<td>3-clique</td>
<td>90.18 ± 0.90</td>
<td>93.65 ± 0.62</td>
<td>72.19 ± 3.64</td>
</tr>
<tr>
<td>5-clique</td>
<td>93.65 ± 0.62</td>
<td>73.86 ± 3.61</td>
<td>73.42 ± 4.96</td>
</tr>
<tr>
<td>7-clique</td>
<td>90.37 ± 1.03</td>
<td>90.47 ± 2.12</td>
<td>73.42 ± 4.96</td>
</tr>
</tbody>
</table>

Figure 10: Comparison of different $h$-hop enclosing subgraphs used for our SEAM framework on USAir graph

SEAM parameters: proposed labels enabled, proposed embedding disabled, learning rate = 0.002, number of epochs = 100, training dataset size = 100,000

Figure 11: Comparison of different $h$-hop enclosing subgraphs used for our SEAM framework on Power graph

SEAM parameters: proposed labels enabled, proposed embedding disabled, learning rate = 0.002, number of epochs = 100, training dataset size = 100,000. The Power graph does not contain enough 5-cliques and 7-cliques due to the sparsity of the graph.
Figure 12: Comparison of the prediction accuracy of SEAM for different already present motif edges on Power graph

SEAM parameters: h-hop = 1, proposed labels enabled, proposed embedding enabled, learning rate = 0.002, number of epochs = 100, training dataset size = 100,000. The Power graph does not contain enough 5-cliques and 7-cliques due to the sparsity of the graph.

Effect on accuracy if no motif edges are previously known

We now illustrate that SEAM also ensures high accuracy when no or very few motif edges are already present, see Figures 12 and 13. Thus, we can conclude that SEAM’s prediction strength relies mostly on the structure of the neighborhood subgraph, embeddings, vertex attributes, and our proposed labeling scheme, and not necessarily on whether a given motif is already partially present. Outliers in this experiment are the 3-clique in the Power graph, the $k$-star motif with $k > 3$ in the USAir graph, and the 3-star motif in general. Still, there is no general tendency indicating that SEAM would profit greatly from the presence of most motif edges.

Appendix E: Analysis of Additional Datasets

We now analyze additional datasets, similarly to Section 5. An interesting result is the difference in accuracy for the Power graph dataset shown in Figure 14. This graph dataset is very sparse with very low average vertex degree. This result clearly shows very low accuracy of SEAL and other motif scores if there are just a few vertices in the neighborhood of the motif. The prediction accuracy for $k$-stars with deal-breaker edges is significantly better. This is caused by the properties of the positive samples discussed in Section 4.2. The prediction task of these positive samples boils down to predicting one motif edge, which has to be added, and several deal-breaker edges, that cannot appear. Due to the sparsity of the motif neighborhood, these deal-breaker edges are often predicted correctly to not appear, which significantly increases the prediction strength of SEAL and all the other motif scores.

In Figure 15, we use the Yeast graph dataset. The results in this dataset match the general trend of all our previous results. The exception in this dataset is the slight drop in accuracy for bigger stars and stars with deal-breaker edges. We conjecture that accuracy drop is caused by, for example (1) this dataset having a lot challenging negative samples or negative ones (4.2) for bigger motifs, (2) the neighborhoods of negative and positive samples being almost indistinguishable, or (3) limitations of our model.
<table>
<thead>
<tr>
<th>Method</th>
<th>3-star</th>
<th>5-star</th>
<th>7-star</th>
<th>3-clique</th>
<th>5-clique</th>
<th>7-clique</th>
<th>3-dbstar</th>
<th>5-dbstar</th>
<th>7-dbstar</th>
</tr>
</thead>
<tbody>
<tr>
<td>CN (Mul)</td>
<td>46.15</td>
<td>54.60</td>
<td>60.40</td>
<td>44.84</td>
<td>60.80</td>
<td>46.91</td>
<td>46.60</td>
<td>50.10</td>
<td>50.50</td>
</tr>
<tr>
<td>CN (Min)</td>
<td>46.37</td>
<td>50.50</td>
<td>50.50</td>
<td>44.70</td>
<td>50.77</td>
<td>45.92</td>
<td>45.50</td>
<td>50.72</td>
<td>50.92</td>
</tr>
<tr>
<td>CN (Avg)</td>
<td>46.27</td>
<td>50.15</td>
<td>50.15</td>
<td>44.36</td>
<td>50.82</td>
<td>45.24</td>
<td>45.00</td>
<td>50.18</td>
<td>50.42</td>
</tr>
<tr>
<td>AA (Mul)</td>
<td>57.03</td>
<td>50.50</td>
<td>50.50</td>
<td>54.44</td>
<td>50.47</td>
<td>50.70</td>
<td>50.50</td>
<td>50.42</td>
<td>50.42</td>
</tr>
<tr>
<td>AA (Min)</td>
<td>57.41</td>
<td>50.15</td>
<td>50.15</td>
<td>54.26</td>
<td>50.41</td>
<td>50.67</td>
<td>50.45</td>
<td>50.45</td>
<td>50.45</td>
</tr>
<tr>
<td>AA (Avg)</td>
<td>57.76</td>
<td>50.50</td>
<td>50.50</td>
<td>54.36</td>
<td>50.26</td>
<td>50.76</td>
<td>50.50</td>
<td>51.23</td>
<td>51.23</td>
</tr>
<tr>
<td>jaccard (Mul)</td>
<td>57.49</td>
<td>50.50</td>
<td>50.50</td>
<td>54.36</td>
<td>50.26</td>
<td>50.76</td>
<td>50.50</td>
<td>51.23</td>
<td>51.23</td>
</tr>
<tr>
<td>jaccard (Min)</td>
<td>58.97</td>
<td>50.50</td>
<td>50.50</td>
<td>54.36</td>
<td>50.26</td>
<td>50.76</td>
<td>50.50</td>
<td>51.23</td>
<td>51.23</td>
</tr>
<tr>
<td>jaccard (Avg)</td>
<td>57.92</td>
<td>50.50</td>
<td>50.50</td>
<td>54.36</td>
<td>50.26</td>
<td>50.76</td>
<td>50.50</td>
<td>51.23</td>
<td>51.23</td>
</tr>
<tr>
<td>SEAL (Mul)</td>
<td>71.84</td>
<td>50.50</td>
<td>50.50</td>
<td>54.36</td>
<td>50.26</td>
<td>50.76</td>
<td>50.50</td>
<td>51.23</td>
<td>51.23</td>
</tr>
<tr>
<td>SEAL (Min)</td>
<td>71.51</td>
<td>50.50</td>
<td>50.50</td>
<td>54.36</td>
<td>50.26</td>
<td>50.76</td>
<td>50.50</td>
<td>51.23</td>
<td>51.23</td>
</tr>
<tr>
<td>SEAL (Avg)</td>
<td>71.51</td>
<td>50.50</td>
<td>50.50</td>
<td>54.36</td>
<td>50.26</td>
<td>50.76</td>
<td>50.50</td>
<td>51.23</td>
<td>51.23</td>
</tr>
<tr>
<td>SEAM, no embedding</td>
<td>89.81</td>
<td>50.50</td>
<td>50.50</td>
<td>54.36</td>
<td>50.26</td>
<td>50.76</td>
<td>50.50</td>
<td>51.23</td>
<td>51.23</td>
</tr>
</tbody>
</table>

Figure 14: Comparison of different motif prediction schemes on Power graph.

Figure 15: Comparison of different motif prediction schemes on Yeast graph.

Appendix F: Details of Implementation & Used Hardware

Our implementation of SEAM and SEAL use the PyTorch Geometric Library [36]. We employ Ray [62] for distributed sampling and preprocessing, and RaySGD for distributed training and inference.

To run our experiments, we used the AULT cluster and the Piz Daint cluster at CSCS [32]. For smaller tasks, we used nodes from the AULT cluster such as AULT9/10 (64 AMD EPYC 7501 @ 2GHz processors, 512 GB memory and 4 Nvidia V100 GPUs), AULT23/24 (32 Intel Xeon 6130 @ 2.10GHz processors, 1.5TB memory and 4 Nvidia V100 GPUs), and AULT25 (128 AMD EPYC 7742 @ 2.25GHz processors, 512 GB memory and 4 Nvidia A100 GPUs). For larger, tasks we used our distributed implementation on the Piz Daint cluster (5704 compute nodes, each with 12 Intel Xeon E5-2690 v3 @ 2.60GHz processors, 64 GB memory and a Nvidia Tesla P100 GPU).