MSGNN: Masked Schema based Graph Neural Networks

Hao Liu
Fudan University
Zhuhai Fudan Innovation Institute
Shanghai Key Laboratory of Data Science
liuhao20@fudan.edu.cn

Qianwen Yang
Fudan University
qwyang22@m.fudan.edu.cn

Taoyong Cui
Tsinghua University
cuitaoyong2022@gmail.com

Wei Wang
Fudan University
weiwang1@fudan.edu.cn

ABSTRACT

Heterogeneous graph representation learning aims to extract low-dimensional node representations from complex networks with different types of entities and relationships. With the prevalence of heterogeneous information networks (HINs) in real-world scenarios, it is of vital significance for a network embedding model to handle heterogeneity and capture as much semantic information as possible. Existing works can be roughly categorized into meta-path-based and adjacent matrix-based methods depending on their definition of node neighborhoods. Meta-path-based methods aim to capture semantic similarities but require manual design. Adjacent matrix-based methods focus on structural information but may risk losing semantic context. In this work, we propose using schema instances representing node minimal complete contexts to embed HINs, aiming to integrate the advantages of both methods and avoid their deficiencies. We introduce Masked Schema based Graph Neural Networks (MSGNN), which combines schema instances with bi-level self-supervised learning and mask technique to acquire effective context representations. Furthermore, we propose a decomposition-reconstruction schema instance retrieval strategy to ensure efficient instance searching. Comprehensive experiments demonstrate that MSGNN outperforms state-of-the-art models. In the link prediction task, the F1-score has improved by up to 16.08% compared to the suboptimal method.

PVLDB Reference Format:
Hao Liu, Qianwen Yang, Taoyong Cui, and Wei Wang. MSGNN: Masked Schema based Graph Neural Networks. PVLDB, 18(3): 571 - 584, 2024.
doi:10.14778/3712221.3712226

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/liuhao33/MSGNN.

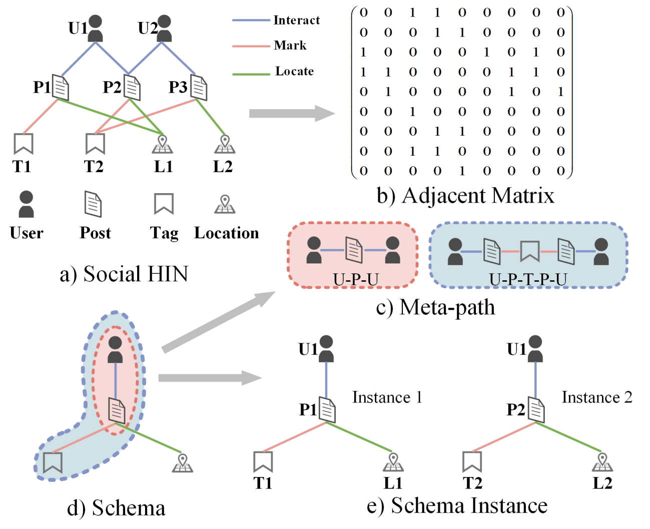


Figure 1: A toy example of HINs and relative illustrations of adjacency matrix, meta-path, schema, and schema instance.

1 INTRODUCTION

The graph is an intuitive data structure that effectively models complex relationships among numerous objects. However, graph data, being high-dimensional, needs to be transformed into low-dimensional representations through graph representation learning before being applied to downstream tasks. The emergence of graph neural networks [39] and their variants [11, 35] has significantly enhanced the performance of graph representation learning. While early network embedding methods focused on homogeneous graphs, the prevalence of heterogeneous information networks (HINs) [30] in real-world scenarios, such as citation networks [25], biomedical networks [31], and social networks [23], necessitates capturing rich semantic information and addressing the challenges posed by the interconnections of heterogeneous entities and relations in HINs. Given this, how to handle the heterogeneity of HINs to capture as much semantic information as possible has always been a top priority in HIN research and yet not properly solved.

In HIN representation learning research, the mainstream idea of obtaining node representations can be described as updating target nodes' embeddings with information aggregated from their neighboring nodes. Based on different approaches to defining node neighborhoods, we categorize current methods primarily into two

Proceedings of the VLDB Endowment, Vol. 18, No. 3 ISSN 2150-8097.
doi:10.14778/3712221.3712226

classes: meta-path-based methods [7, 29, 36] and adjacent matrix-based methods [21, 43]. Meta-path-based methods utilize meta-paths to capture semantic similarities between target nodes and thus identify meta-path-based neighborhoods. A meta-path is a specific path in HINs that connects two entities with a composite relation [32] and is considered to represent one specific semantic similarity. For example, Figure 1(a) illustrates a social HIN consisting of four types of nodes and three types of edges, and its two types of meta-paths are shown in Figure 1(c): UPU and UPTPU. The semantic similarities in UPU can be described as "two users interacted with the same post", indicating they might both be concerned about the topic so that both users should be neighbors based on meta-path UPU. Adjacent matrix-based methods refer to those that mainly focus on the structural information between nodes, as they utilize the adjacent matrices, as shown in Figure 1(b), or other sampling methods, to propagate node features and aggregate information from structural neighborhoods.

However, both types of methods have certain limitations. Meta-path-based methods face the challenge of designing appropriate meta-paths. According to the definition of meta-paths [32], any path in an HIN can be considered a meta-path since it represents composite relations between nodes. However, the meta-paths used to guide representation learning are not arbitrarily chosen. That is because: 1) Semantic similarities are implicit, making it difficult to determine which meta-path contributes to representation learning; 2) The search space for meta-path grows exponentially as the complexities of meta-path patterns increase, making it impractical for training purposes [26]. Typically, experts are needed to identify the most valuable meta-paths to ensure the abundance and diversity of captured semantic information. However, considering the complexities in real-world scenarios, it is still challenging to design appropriate meta-paths as semantic carriers for models to meet performance expectations. In contrast, adjacency matrix-based methods do not require manual designs. However, they focus solely on extracting information from structural neighborhoods, neglecting the rich semantic information presented in HINs. Although the HIN adjacency matrix can be regarded as a composition of 1-hop meta-paths, the adjacency matrix-based methods lack an effective semantic carrier such as meta-paths to help identify semantic neighborhoods, thus making it difficult to extract the implicit semantic information explicitly.

To address the aforementioned challenges, we propose to utilize **schema instances** [48] to guide HIN representation learning. Being the template for HIN [32], the network schema is often mentioned in data management and occasionally introduced as a high-order graph structure in graph neural networks (GNNs) [48]. Figure 1(d) provides an example schema. However, we observe the semantic property of schema for GNNs remains to be explored. Considering an HIN is a semantic network, and it is constructed according to the description of its schema, we assume that schema provides all necessary semantic information in an HIN. An instance of a schema is defined as the smallest subgraph in the HIN that matches all the types and relationships in the schema [48]. For example, Figure 1(e) shows two schema instances. Thus an HIN can be regarded as an outcome of concatenations of all schema instances. Since the schema instance encompasses all kinds of categories and relationships of nodes, providing comprehensive foundational information for nodes within the instance, we define the information represented by schema instances as the **minimal complete semantic** context. More specifically, because the schema describes how the HIN elements interact mutually, as the minimal structure that matches the schema, a schema instance is the smallest structure that contains all semantic information needed to define its inside nodes. Consequently, schema instances can be defined as **schema neighborhoods** of nodes inside of them, and the nodes inside an instance are schema neighbors to each other under the context represented by the instance.

In this work, we propose to use schema instances as a novel semantic carrier to aggregate semantic information for HIN embedding. Compared with meta-paths, experts or domain knowledge is no longer required to determine network schema because the schema is a unique structure for a certain HIN[48]. Moreover, schema instances contain more semantic information than meta-paths because meta-paths were initially defined as paths on the network schema [32]. More specifically, schema instances include all types of nodes and relations as well as complete node context, whereas one meta-path only incorporates a limited number of node and relation types and captures one kind of predefined semantic information only. Compared with adjacency matrices, schema instances contain rich semantic information explicitly. Meanwhile, schema neighbors are natural 1- to multi-hop graph structure neighbors depending on schema structure, implying that schema instances incorporate structural information as well. In summary, schema instances integrate the advantages of both meta-paths and adjacency matrices, thus exploring an approach to exploit schema instances for network embedding is non-trivial.

In this work, we propose the **M**asked **S**chema based **G**raph **N**eural **N**etworks (MSGNN) for graph representation learning based on schema instances in a self-supervised manner. The MSGNN consists of three main components: schema instance retrieval, node minimal complete context representation generation, and bi-level masked schema instances training. We prove the superiority of the proposed method on link prediction and node classification tasks in Section 5.

The contributions of this paper can be summarized as follows:

- To the best of our knowledge, we are the first to explore and define the semantic information represented by schema instances in HINs. In addressing the deficiencies of different approaches for heterogeneous networks, we elucidate the strength of exploiting schema instances and present a novel HIN embedding model MSGNN based on the schema.
- We devise a framework for schema based HIN representation learning, which includes schema induction, schema instance retrieval, and schema instance based training, among which an efficient strategy is proposed for schema instance retrieval.
- We conducted comprehensive comparative experiments to verify the effectiveness of MSGNN, and the results consistently demonstrate that our method outperforms all baseline models, achieving state-of-the-art performance.

2 RELATED WORKS

Heterogeneous Network Embedding. Many studies use meta-paths [4, 6, 8, 28, 29, 36] to learn node embeddings based on meta-path neighborhoods. Metapath2vec [4] leverages meta-path based random walks to produce node sequences and uses skip-gram for embedding. HERec [29] filters node sequences based on type constraints to capture information from homogeneous neighbors. HAN [36] combines attention mechanism with meta-paths in a bi-level architecture, but it discards all intermediate nodes along the meta-paths by only considering two end nodes, leading to information loss. MAGNN [8] solves this problem by encoding whole meta-path instances. These works make impressive performances, but they fail to answer the question about how to design meta-paths as mentioned in Section 1.

Recent studies [5, 5, 14, 20, 40, 42, 44] attempt to address this problems posed by meta-paths. RHINE [20] distinguishes heterogeneous relations as either one-centered-by-another or peer-to-peer, and uses different models for two relations. SR-RSC [44] adopts multi-hop message passing to extract relation-based neighbor-graphs for HIN embeddings, thereby avoiding the use of meta-paths. GTN [42] learns meta-paths automatically and operates graph convolution on the learned meta-path graphs. MHGCN [40] proposes an automatic meta-path learning mechanism through a graph convolution module. Based on MHGCN, BPHGNN [5] defines combinations of relations as behavior patterns and learns representations from local and global perspectives. These works attempt to use partial graph structures to describe the semantic relationship between nodes and define node neighborhoods, and therefore avoid designing meta-paths explicitly. However, they either lack a clear definition of the semantic information represented by the proposed structures or cannot guarantee the completeness of captured semantic information.

There are some other approaches [2, 43, 47] for graph representation learning. HetGNN [43] samples a fixed number of neighbors and fuses their features using Bi-LSTMs. SimpleHGN [21] enhances GAT [35] by adding edge type vectors to the attention layer and aggregates structural neighbors. NSHE [48] learns representations by preserving node-level and schema-level proximity. SchemaWalk [26] uses schema to guide random walk to obtain more informative node sequences and employ skip-gram to embed the network. These methods mainly focus on graph structural information, neglecting semantic infomation.

Masked Modeling. Graph autoencoders (GAEs) [3, 16, 24] are a family of self-supervised learning models that take the graph input as self-supervision and learn to reconstruct the graph structure. MGAE [33] and GMAE [46] perform masking strategies on graph structure and node attributes as part of their self-supervised learning paradigm. MaskGAE [18] adopts masked graph modeling (MGM) to mask a portion of edges and aims to reconstruct the missing part with a partially unmasked graph structure. Instead of edge masking, SimSGT [19] introduces noise to the graph by random node masking which samples a random subset of nodes and replaces their features with a special token. GCMAE [37] combines masked autoencoder and contrastive learning to reconstruct the entire adjacency matrix, capturing global graph structures, rather than solely focusing on masked edges as in existing works. These works mainly focus on graph structure masking, and few of them attempt to mask networks from a semantic perspective. Considering the advanced performance achieved by this technique, we attempt to design a masked semantic model (MSM) for HIN representation learning to learn semantic information within the minimal complete context.

Algorithm 1: Schema Induction

```
Input  : An HIN $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, \phi, \psi\}$, $\phi : \mathcal{V} \to \mathcal{A}$, $\psi : \mathcal{E} \to \mathcal{R}$
         A threshold $t$
Output : Schema $O$
1  $Q \leftarrow \emptyset$
2  for each node type pair $(\mathcal{A}_i, \mathcal{A}_j)$, $\mathcal{A}_i \in \mathcal{A}, \mathcal{A}_j \in \mathcal{A}$ do
       // iterate all combinations of $\mathcal{A}$
3      $Cnt \leftarrow 0$
4      for each node pair $(u, v), \phi(u) = \mathcal{A}_i, \phi(v) = \mathcal{A}_j$ do
5          if $\exists$ edge $(u, v) \in \mathcal{G}$ then
6              $Cnt \leftarrow Cnt + 1$
               // accumulate connectivity of pair $(\mathcal{A}_i, \mathcal{A}_j)$
7              continue
8          $Connectivity \leftarrow \frac{Cnt}{|\mathcal{V}^{\mathcal{A}_i}|}$
9      if $Connectivity > t$ then
10         $Q.Append((\mathcal{A}_i, \mathcal{A}_j))$
11 $O \leftarrow CreateGraph(O|\mathcal{V}_O = \mathcal{A}, \mathcal{E}_O = \emptyset)$
12 for each edge $e \in Q$ do
13     $O.AddEdge(e)$
14 return $O$
```

3 PRELIMINARIES

Heterogeneous Information Network (HIN) [30] is a network model that contains multiple types of nodes and edges as well as rich semantic information. An HIN can be formally defined as $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, \phi, \psi\}$, where $\mathcal{V}$ denotes the set of nodes and $\mathcal{E}$ represents the set of edges. ϕ is a node type mapping function $\phi : \mathcal{V} \to \mathcal{A}$ and ψ is an edge type mapping function $\psi : \mathcal{E} \to \mathcal{R}$, where $\mathcal{A}$ and $\mathcal{R}$ denote the set of node types and edge types respectively. When $|\mathcal{A}| = |\mathcal{R}| = 1$, the network degenerates into a homogeneous graph.

Graph Neural Networks (GNNs) [39] are neural network models designed to handle graph-structured data, aiming to learn low-dimensional representations of nodes that capture structural information within the graph. The mechanism behind GNNs is to aggregate information from neighboring nodes and then update the information of target nodes. In the l-th layer, the representation of node u, denoted as $\mathbf{h}_v^{(l)} \in \mathbb{R}^{d_l}$, can be calculated as follows:

$$\mathbf{h}_u^{(l)} = \text{UPDATE}\left(\mathbf{h}_u^{(l-1)}, \text{AGGREGATE}\left(\{\mathbf{h}_v^{(l-1)} \mid v \in \mathcal{N}_u\}\right)\right). \quad (1)$$

where $\mathcal{N}_u$ denotes the set of neighboring nodes of u.

The Schema [9] is a formal representation of a set of concepts and their relationships within a domain. In an HIN, schema represents the meta-structure of the graph and records the connectivity of different types of nodes. Given the HIN $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, the schema is denoted as $O = \{\mathcal{A}, \mathcal{R}\}$, which preserves all the node types $\mathcal{A}$ and edge types $\mathcal{R}$ inside $\mathcal{G}$.

Schema instances, denoted as O_i, are the smallest subgraphs in the HIN that match all the types and relationships in the schema[48] and contain minimal complete semantic contexts of nodes inside. Given this, an HIN can also be represented as a complex network composed of plenty of schema instances, i.e., $\mathcal{G} = \bigcup_{i=1}^{N} O_i$. Here, The subscript i is the index of the instance and N denotes the total number of instances. We denote the set of schema instances as $O_{\text{sub}} = \{O_1, \ldots, O_N\}$, containing all schema instances of the HIN.

Algorithm 2: Main chain searching $mSearch(O, M)$

```
Input  : A schema graph $O = \{\mathcal{V}_O, \mathcal{E}_O\}$, a main chain $M$
         initialized $empty$ by default.
Output : All main chain candidates $Mains$
1  if $M$ is $empty$ then
2      $N \leftarrow O.nodes$
3  else
4      $N \leftarrow O.nodes \cap Set(M)$
5  $Mains, Paths \leftarrow \emptyset, \emptyset$ // $Mains$ contains main chain
       candidates
6  for each node $u \in N$ do
7      $path \leftarrow \emptyset$ // $path$ is a node sequence
8      $path.Extend(u)$
9      $Paths.Append(path)$
10 while $Paths$ is not $empty$ do
11     $Paths_{next} \leftarrow \emptyset$
12     for each $path$ in $Paths$ do
13         $\mathcal{N} \leftarrow O.GetNeighbors(path.tail)$
14         if $\mathcal{N} \subseteq Set(path.nodes)$ then
15             $Mains.Append(path)$
16         else
17             for each node $i \in \mathcal{N}/Set(path.nodes)$ do
18                 $path_{next} \leftarrow path.Extend(i)$
19                 $Paths_{next}.Append(path_{next})$
20     $Paths \leftarrow Paths_{next}$
21 $Mains \leftarrow Mains.Sort()$ descending by length
22 return $Mains$
```

4 METHODOLOGY

In this section, we described the details of Masked Schema based Graph Neural Networks. MSGNN consists of three main components: schema instance retrieval, node minimal context representations generation, and bi-level masked schema training. The overall framework of the algorithm is illustrated in Figure 2.

4.1 Schema Instance Retrieval

This section presents an efficient schema instance retrieval strategy, which obtains all possible instances by steps of schema decomposition, schema chain matching, and schema instance reconstruction. Details are illustrated in Figure 3. The goal of this section is to obtain the set of schema instances O_{sub} defined in Section 3.

4.1.1 Schema Decomposition.

As mentioned in Section 1, the schema is a unique structure for a certain HIN[48]. In most cases, schema is either prior knowledge or easy to recognize from graph data when the graph is simple. For cases where the schema is unknown, we prepare an effective approach to rapidly recognize the schema, shown in Algorithm 1. The core idea of the algorithm is to extract and keep type pairs that exhibit significant co-occurrence from the network, which can be regarded as the expected relationships in the schema. More specifically, this method examines the connectivity between every possible pair of node types by counting connected node pairs and then preserves the node type pair as an edge in the schema graph of which connectivity exceeds the predefined threshold. For a cleaned graph, the threshold can be set to zero.

Based on schema O, to achieve the purpose of learning minimal node context representations, finding out all schema instances contained in $\mathcal{G}$ is of utmost importance. However, due to the heterogeneity, a complex graph could include various types of nodes and edges, which makes the direct search of schema instances very challenging. In this work, we present a **decompose-reconstruct** strategy to narrow down the search space. We recursively decompose the schema into two types of chains and one residual component, namely main chain, subordinate chain, and residual. A chain is a simple path in schema, i.e. a sequence of node types without repeating. Chains are considered as middle targets of instance retrieval in the following steps to reduce search spaces. Below is a detailed explanation of the three components:

- **Main Chain:** Unique in each recursion layer, typically the longest path in the schema. A proper main chain should cover as many elements of O as possible to reduce the number of subordinate chains and residuals. Main chain instances serve as the foundation in the following reconstruction step, i.e. all other retrieved component instances are eventually appended to main chain instances. During each recursion, we should first identify the main chain.
- **Subordinate Chain:** Zero to multiple chains in each recursion layer. If the schema O has circles, we split a circle into a main chain and a subordinate chain during main chain identification, if necessary. Subordinate chains refer to 1-hop paths in O and are characterized by **both** ends intersecting the main chain. The head and tail of a subordinate chain is arbitrarily selected among two ends.
- **Residual:** Zero to multiple subgraphs in each recursion layer. After a main chain and subordinate chains are determined, the remaining elements form several disconnected subgraphs. Decomposing each subgraph recursively until no element remains.

The schema O is usually a small graph, so we can limit the depth of recursion for acceleration. In this work, we set the maximum depth to 3. Moreover, considering all chain instances will be eventually concatenated together, we should choose a decomposition plan with the least number of chains to lower the number of concatenation operations and thus ensure efficiency. We search all possible main chains using Algorithm 2. In Algorithm 2, main chain searching starts from all nodes (line 6-9) and each route keeps visiting the neighboring nodes of its end. For each route, if all neighbors have been visited, stop walking and save this route as a main chain candidate (line 12-15). If not, adding the node to the end of the route to extend it and keep walking (line 17-19). Then, we iterate each

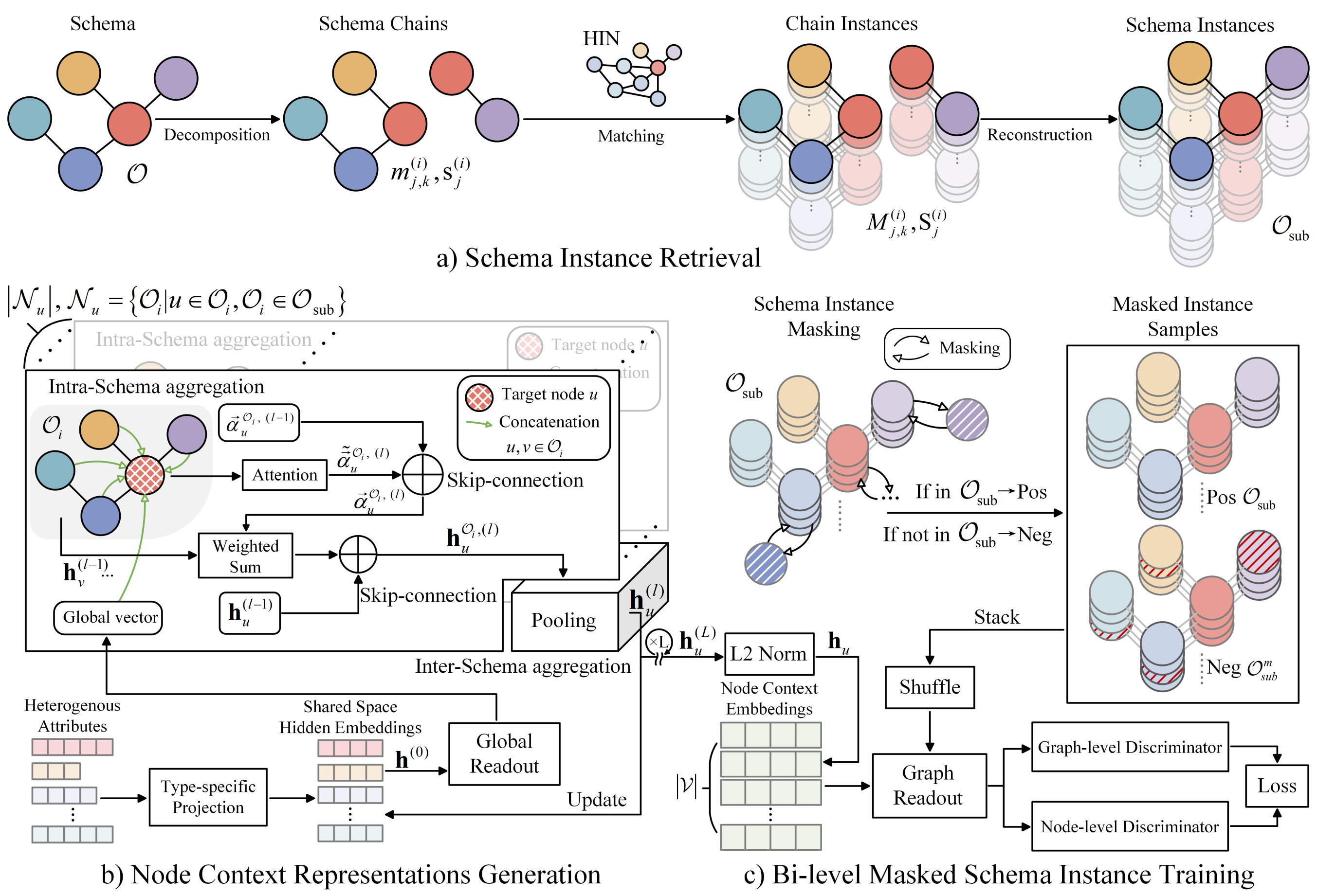


Figure 2: Overall framework of MSGNN. Including three parts: schema instance Retrieval, Node Context Representations Generating, and Bi-level Masked Schema Training.

main candidate to find a decomposition plan with the minimum number of chains using Algorithm 3. In Algorithm 3, a dictionary is defined to record and update the plan during the process. For each main chain candidate, we remove its edges to identify subordinate chains from the graph (line 5-10) then remove subordinate chains to identify residuals. If the graph is empty, finish decomposition and update the dictionary (line 13-18). If the residual graphs exist, decompose each subgraph recursively (line 20-25), and update the dictionary if a better plan (a plan with a smaller total number of chains) is found (line 27-34). Note that during recursion, the main chain candidates searching only starts from nodes that are contained in the iterated main chain candidate of the last recursion (Algorithm 2 line 3-4).

For better demonstration, we denote the main chain and its corresponding subordinate chain set as $m_{j,k}^{(i)}$ and $s_j^{(i)}$, respectively, where i refers to the recursive depth (or layer), j is the index of the main chain, i.e., the index of the corresponding residual graph (denoted as $res_j^{(i-1)}$) as the main chain is unique in each decomposition, and k stands for the index of the main chain in $(i-1)$-th recursion that produces $res_j^{(i-1)}$. Figure 3(a) illustrates an example of the decomposition on a subset of a large medical knowledge graph with seven node types. In Figure 3(a), a main chain $m_{0,0}^{(0)}$ in red is selected, and thus two subordinate chains $s_0^{(0)}$ in blue are split from circles. A residual graph in green is further decomposed as a main chain $m_{0,0}^{(1)}$ in the next recursion. Take $m_{0,0}^{(1)}$ as an example, the superscript 1 means $m_{0,0}^{(1)}$ is a main chain of the 2nd recursion, the first subscript 0 is the index of $m_{0,0}^{(1)}$, also indicating it is decomposed from $res_0^{(0)}$, and the second subscript 0 is the index of main chain that produce $res_0^{(0)}$ in last recursion, i.e., $m_{0,0}^{(0)}$.

4.1.2 *Schema Chain Matching.*

In this step, we query the HIN $\mathcal{G}$ with schema chains obtained in Section 4.1.1 for corresponding chain instances. A chain instance is a specific node sequence, aka. a path, whose type sequence matches the query. To further enhance the efficiency of chain instance matching, we break chains into node type triplet sequences $[(type_1 \rightarrow type_2), (type_2 \rightarrow type_3)...]$, see Figure 3(a) dotted boxes. The tail of a triplet is the head of the subsequent triplet.

Algorithm 3: Schema Decomposition $dcmp(O, d, M)$

```
Input  : A schema graph $O = \{\mathcal{V}_O, \mathcal{E}_O\}$, a depth limitation
         $depth$, a chain $M$ initialized as empty variable by default.
Output : A decomposition plan $D$ with four attributes
         $D.cnt, D.m, D.s, D.res$
1  $D.m, D.s, D.res \leftarrow \emptyset, \emptyset, \emptyset$
2  $D.cnt \leftarrow Count(O.edges)$
3  $Mains \leftarrow mSearch(O, M)$ // Algorithm 2
4  for each $m \in Mains$ do
5      $s, res \leftarrow \emptyset, \emptyset$
6      $\mathcal{G} \leftarrow O.Copy()$
7      $\mathcal{G}.RemoveEdges(m.edges)$
8      for each edge $(u, v) \in \mathcal{G}.edges$ do
9          if $u \in m$ and $v \in m$ then
10             $s.Append((u, v))$
11             $\mathcal{G}.RemoveEdges((u, v))$
12     $\mathcal{G}.RemoveIsolatedNodes()$
13     if $\mathcal{G}$ is empty then finish decomposition
14         $cnt \leftarrow Count(s) + 1$ // 1 is count of $m$
15         if $cnt < D.cnt$ then update $D$
16             $D.m, D.s, D.cnt, D.res \leftarrow m, s, cnt, res$
17         else
18             continue
19     else
20         if $depth > 0$ then
21             $depth \leftarrow depth - 1$
22             $\mathcal{G}_{res} \leftarrow GetDisconnectedComponents(\mathcal{G})$
23             for each $\mathcal{G}_{sub}$ in $\mathcal{G}_{res}$ do
24                 $D_{res} \leftarrow dcmp(\mathcal{G}_{sub}, depth, m)$ // recursion
25                 $res.Append(D_{res})$
26             $cnt_{res} \leftarrow 0$
27             if $Count(res) > 0$ then accumulate $D_{res}.cnt$
28                 for each $D_{res}$ in $res$ do
29                     $cnt_{res} \leftarrow cnt_{res} + D_{res}.cnt$
30             $cnt \leftarrow Count(s) + 1 + cnt_{res}$
31             if $cnt < D.cnt$ then update $D$
32                 $D.m, D.s, D.cnt, D.res \leftarrow m, s, cnt, res$
33             else
34                 continue
35         else
36             continue
37 return $D$
```

For each chain, we query all its type triplets and sequentially concatenate the returned node triplets to obtain chain instances, as shown in Figure 3(a)(b). Because each type triplet involves only a 1-hop relation, each type triplet query only requires a single lookup operation based on the adjacency matrix.

This step disassembles the process of schema chain instance matching, which is a path query, into multiple 1-hop queries, significantly reducing the complexity and time consumption for searching. Moreover, this approach allows parallel querying of all chains for more flexible and efficient matching.

4.1.3 *Schema Instance Reconstruction.*

Similar to Section 4.1.1, we denote main chain instances and corresponding subordinate chain instances set as $M_{j,k}^{(i)}$, and $S_j^{(i)}$, respectively. Once obtaining all instances of the main chain and subordinate chains, we reconstruct the schema instances through two steps:

(1) For each subordinate chain instance in $S_j^{(i)}$, we first connect it to corresponding main chain instances in $M_{j,k}^{(i)}$ according to the head nodes and tail nodes of the subordinate chain instances.
(2) For each main chain instance in $M_{j,\mathbf{k}}^{(\mathbf{i})}$, we connect it to its corresponding upper layer main chain instance in $M_{\mathbf{k},l}^{(\mathbf{i-1})}$ sequentially until $i - 1 = 0$.

After the 2 steps, schema instances are reconstructed in $M_{0,0}^{(0)}$. Thus we obtain O_{sub} by creating graph for each instance in $M_{0,0}^{(0)}$ based on schema O. Algorithm 4 provides the details of the whole process of schema instance retrieval (Section 4.1.2 is described in line 2-17 and Section 4.1.3 is described in line 18-22). An example of one schema instance reconstruction is provided in Figure 3(b)(c). We notice that in some datasets, the obtained instances are incomplete, i.e., certain nodes are missing. This phenomenon may be due to a lack of data, meaning the graph is incomplete. Based on our analysis in Section 1, schema instances represent the minimal complete contexts for involved nodes. Therefore, an incomplete schema instance indicates abnormal semantic contexts for all nodes in this subgraph. In this case, we recommend discarding those incomplete subgraphs and learning node representations based on complete semantic context only, to prevent introducing noise during the learning process. Alternatively, we could also padding those incomplete subgraphs if necessary.

Our proposed strategy synchronizes the search of all instances in a largely reduced search space, with only a few lookup operations to graph, and it is parallel-friendly. In the implementation, we could use the table join operation to perform the concatenation operations mentioned in Section 4.1.2 and Section 4.1.3 to further accelerate the retrieval. Overall, this staged retrieval strategy significantly reduces the search space and enhances search efficiency.

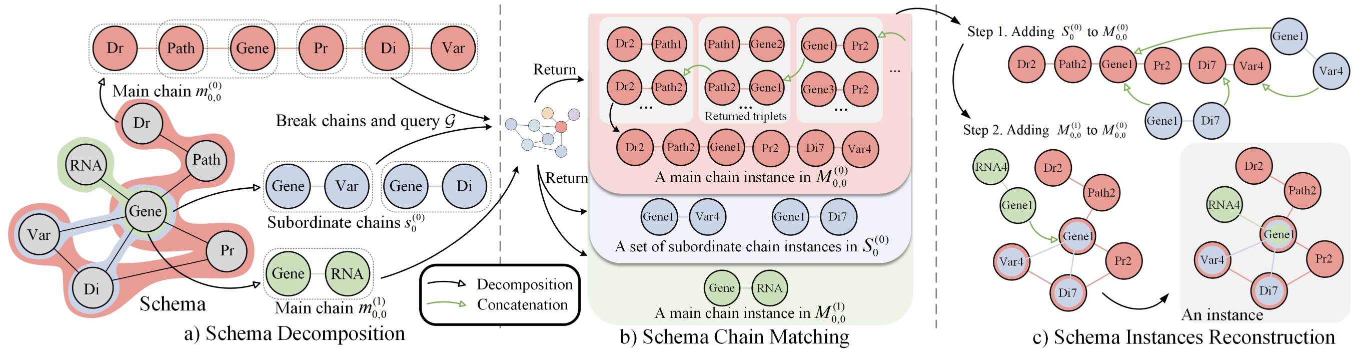
4.2 Node Context Representations Generation

Since the schema instances O_{sub} have been obtained, the purpose of this section is to generate minimal complete context representations of nodes based on O_{sub}. The process could be divided into two setps: intra-schema aggregation and inter-schema aggregation. For each node, intra-schema aggregation integrates the semantic information within each context represented by schema instance, while inter-schema aggregation fuses the information from all contexts. Details are shown in Figure 2(b).

4.2.1 *Intra-schema Aggregation.*

Given that heterogeneous nodes in an HIN reside in different feature spaces, we conduct type-specific projection to map all node features into the same latent vector space. For a node $u \in \mathcal{V}$ of type $A \in \mathcal{A}$, we have:

$$\mathbf{h}_u^{(0)} = f(\mathbf{x}_u^A, \mathcal{A}), \qquad (2)$$



Dr
Path
Gene
Pr
Di
Var
RNA
Main chain $m_{0,0}^{(0)}$
Break chains and query $\mathcal{G}$
Subordinate chains $s_0^{(0)}$
Main chain $m_{0,0}^{(1)}$
Schema
Decomposition
Concatenation
a) Schema Decomposition
Return
Return
Returned triplets
A main chain instance in $M_{0,0}^{(0)}$
A set of subordinate chain instances in $S_0^{(0)}$
A main chain instance in $M_{0,0}^{(1)}$
b) Schema Chain Matching
Step 1. Adding $S_0^{(0)}$ to $M_{0,0}^{(0)}$
Step 2. Adding $M_{0,0}^{(1)}$ to $M_{0,0}^{(0)}$
An instance
c) Schema Instances Reconstruction

Figure 3: Schema Instance Retrieval. Including Schema decomposition, Schema chain matching, and Schema Instance reconstruction.

Algorithm 4: Schema Instances Retrieval

```
Input  : An HIN $\mathcal{G}$ with schema $O = \{\mathcal{A}, \mathcal{R}\}$, the decomposition
         plan $D$
Output : All $O$ instances $O_{sub}$
1  $d \leftarrow GetRecursionDepth(D)$
2  $Main \leftarrow \{m_{j,k}^{(i)} | m_{j,k}^{(i)} \in D, i \in [0, d]\}$
3  $Sub \leftarrow \{s_j^{(i)} | s_j^{(i)} \in D, i \in [0, d]\}$
4  $M, S \leftarrow \emptyset, \emptyset$ // $M, S$ contain chain instances
5  for each $m_{j,k}^{(i)} \in Main$ do
6      $head \leftarrow m_{j,k}^{(i)}.Pop()$
7      $tail \leftarrow m_{j,k}^{(i)}.Pop()$
8      $M_{j,k}^{(i)} \leftarrow Query(\mathcal{G}|edge(head \rightarrow tail))$
9      while $m_{j,k}^{(i)} \neq \emptyset$ do
10         $head \leftarrow tail$
11         $tail \leftarrow m_{j,k}^{(i)}.Pop()$
12         $Q \leftarrow Query(\mathcal{G}|edge(head \rightarrow tail))$
13         $M_{j,k}^{(i)} \leftarrow InnerJoin(M_{j,k}^{(i)}, Q)$ on head of $Q$
14     $M.Append(M_{j,k}^{(i)})$
15 for each $s_j^{(i)} \in Sub$ do
16     $S_j^{(i)} \leftarrow Query(\mathcal{G}|edge(s_j^{(i)}.heads \rightarrow s_j^{(i)}.tails))$
17     $S.Append(S_j^{(i)})$
18 for each $(M_{j,k}^{(i)}, S_j^{(i)}) \in Zip(M, S)$ do
19     $M_{j,k}^{(i)} \leftarrow InnerJoin(M_{j,k}^{(i)}, S_j^{(i)})$ on head and tail of $S_j^{(i)}$
20 while $d > 0$ do
21     $M_{k,l}^{(d-1)} \leftarrow InnerJoin(M_{k,l}^{(d-1)}, M_{j,k}^{(d)})$ on head of $M_{j,k}^{(d)}$
22     $d \leftarrow d - 1$
23 $O_{sub} \leftarrow \emptyset$
24 for each instance $o_i \in M_{0,0}^{(0)}$ do
25     $O_i \leftarrow CreateGrpah(o_i|\mathcal{A}, \mathcal{R})$
26     $O_{sub}.Append(O_i)$
27 return $O_{sub}$
```

where $\mathbf{h}_u^{(0)}$ represents the type-specific projection vector of node u, $\mathbf{x}_u$ is the original node feature, and the function $f(\cdot)$ represents a linear transformation with a nonlinear activation.

The type-specific projection aligns heterogeneous nodes into the same space, and for the subsequent aggregation process, we will utilize the projected vectors $\mathbf{h}_u^{(0)}$ as input for the first layer.

Next, we adopt the attention mechanism to perform the intra-schema aggregation. For each target node $u \in \mathcal{V}$, its schema neighborhood is denoted as $\mathcal{N}_u = \{O_i | u \in O_i, O_i \in O_{sub}\}$, while each O_i represents a context of u. The attention coefficients of the l-th layer between node u and its neighbor v within a certain schema instance O_i can be calculated by:

$$\widetilde{\alpha}_{uv}^{O_i,(l)} = \frac{\exp\left(\sigma\left(\vec{a}^T\left[\mathbf{h}_u^{(l-1)} \| \mathbf{h}_v^{(l-1)}\right]\right)\right)}{\sum_{w \in O_i} \exp\left(\sigma\left(\vec{a}^T\left[\mathbf{h}_u^{(l-1)} \| \mathbf{h}_w^{(l-1)}\right]\right)\right)}, \tag{3}$$

where $\vec{a}$ is attention vector, $\|$ denotes the vector concatenation operator, $\sigma(\cdot)$ is the activation function.

Since the schema instances only represent local information, we introduce a global representation, denoted as $\mathbf{h}_g$, via vector concatenation. In this way, both local and global information can be taken into account when calculating the attention coefficients. Therefore, Eq. 3 is updated to:

$$\widetilde{\alpha}_{uv}^{O_i,(l)} = \frac{\exp\left(\sigma\left(\vec{a}^T\left[\mathbf{h}_u^{(l-1)} \| \mathbf{h}_v^{(l-1)} \| \mathbf{h}_g^{(l-1)}\right]\right)\right)}{\sum_{w \in O_i} \exp\left(\sigma\left(\vec{a}^T\left[\mathbf{h}_u^{(l-1)} \| \mathbf{h}_w^{(l-1)} \| \mathbf{h}_g^{(l-1)}\right]\right)\right)}, \tag{4}$$

where $\mathbf{h}_g^{(l-1)}$ is obtained by a readout function:

$$\mathbf{h}_g^{(l-1)} = \text{ReadOut}_{\text{global}}(\mathbf{H}^{(l-1)}), \tag{5}$$

where $\mathbf{H}^{(l-1)}$ is the embedding matrix of $l-1$-th layer and the average pooling is employed as the global readout function in this work.

Then, we update the attention coefficients via the skip-connection [17], which makes the updating process more stable and improves the generalization ability of the model:

$$\alpha_{uv}^{O_i,(l)} = (1-\beta)\widetilde{\alpha}_{uv}^{O_i,(l)} + \beta\alpha_{uv}^{O_i,(l-1)}, \tag{6}$$

where $\beta \in [0, 1]$ is a hyperparameter.

Next, we further perform skip-connection to update the representation of u:

$$\mathbf{h}_u^{O_i,}(l) = \sigma\left(\sum_{v\in O_i} \alpha_{uv}^{O_i,(l)} \cdot \mathbf{h}_v^{(l-1)} + \mathbf{W}_{res}^{(l-1)} \cdot \mathbf{h}_u^{(l-1)}\right), \quad (7)$$

where $\mathbf{h}_u^{O_i,(l)}$ is the intra-aggregation of the target node u with respect to O_i of l-th layer, and $\mathbf{W}_{res}^{(l-1)}$ is the learnable matrix. The whole process is shown in Figure 2(b).

4.2.2 *Inter-schema Aggregation.*

Since each schema instance O_i of the target node u generates an intra-aggregation representation, we need to further fuse all semantic information provided by each schema instance in $\mathcal{N}$ through the inter-schema aggregation. Generally, a dual-layer attention mechanism is commonly used in such scenarios; however, we observed its effectiveness to be unsatisfactory. According to [15], the dual-layer attention structure fails to distinguish the importance of different semantic sources, which leads to severe overfitting. Therefore, we turn to use the max pooling function to aggregate the semantic information between schema instances:

$$\mathbf{h}_u^{(l)} = \text{Pooling}(\{\mathbf{h}_u^{O_i,(l)} \mid \forall O_i \in O_{\text{sub}}, u \in O_i\}) \quad (8)$$

We adopt multi-head attention following GAT [35], the independent attention mechanisms are executed via Eq. 4, and then the results of multiple attention layers are concatenated as the representation:

$$\mathbf{h}_u^{(l)} = \text{Concat}(\sigma(\mathbf{h}_u^{k,(l)})), \quad (9)$$

where k is the number of attention heads, Concat$(\cdot)$ denotes the concatenation of vectors.

We regard Equation 3 to 9 as an MSGNN layer. The model extracts deeper information by stacking layers. Notice in the last layer, the representation is obtained by averaging operation:

$$\mathbf{h}_u^{(L)} = \frac{1}{K}\sum_k \mathbf{h}_u^{k,(L)}. \quad (10)$$

Finally, through a linear layer followed by L2 normalization [21], we obtain the inter-aggregation representation of node u, denoted as $\mathbf{h}_u$:

$$\mathbf{h}_u = \frac{\mathbf{W}\mathbf{h}_u^{k,(L)}}{\left\|\mathbf{W}\mathbf{h}_u^{k,(L)}\right\|} \quad (11)$$

where $\mathbf{h}_u$ is the context representation of u.

4.3 Bi-level Masked Schema Training

To strengthen the capability of capturing schema intrinsic semantic information, we adopt the masking technique for schema instance masking and designed two specific tasks to distinguish masked instances from both node-level and graph-level. The bi-level masked schema training is shown in Figure 2(c).

4.3.1 *Schema Instance Masking.*

In this section, we improve the masking operation on schema instances to generate negative samples required for self-supervised tasks. Initially, we attempt the widely used all-zero mask, which substitutes node embeddings with zero vectors, but the effect is not satisfactory, as shown in Table 4. Inspired by [15] using random graphs as noise distributions, we turn to employ a random mask by sampling nodes in the graph to corrupt the instance by replacement, which leads to some improvement. Considering the large difference of information carried by nodes of different types, using random nodes for replacement will make the corrupted samples more different from the positive samples, which in turn reduces the difficulty of the task and lowers model performance. Therefore, we further improve the strategy by replacing nodes with homogeneous nodes to construct difficult negative samples to strengthen the model performance on minimal contexts learning. The experiment and discussion on the different masking strategies can be found in section 5.5.

As shown in Figure 2(c), we take schema instances O_{sub} obtained in Section 4.1 as positive samples and randomly replace nodes in O_{sub} with those of the same type as masks to corrupt the instance meanwhile preserve a certain level of semantics similarity. In Figure 2 (c), the masked nodes are marked with diagonal lines. If the masked instances are not included in O_{sub}, it is labeled as a negative sample (otherwise it is still a positive sample) and denoted as O_i^m. All negative samples are denoted as O_{sub}^m.

Next, we stack all positive and negative samples and shuffle, then readout each sample to obtain graph-level representations of O_j:

$$\mathbf{h}_G^{O_j} = \text{ReadOut}_{\text{graph}}(\{\mathbf{h}_u \mid \forall u \in O_j, O_j \in O_{\text{sub}} \cup O_{\text{sub}}^m\}) \quad (12)$$

Here, we employ a max pooling with a linear transformation as the readout function.

4.3.2 *Graph-level Discrimination.*

For graph-level training, we design a graph discriminator to determine whether the subgraph has been masked:

$$\mathbf{y}_{\text{pred},G} = \text{Discriminator}_G\left(\mathbf{h}_G^{O_j}\right) \quad (13)$$

where $\mathbf{y}_{\text{pred},G}$ stands for the predicted labels of the graph-level task, indicating whether the instance is corrupted. In this work, we use a full connection layer as the graph discriminator.

Then we calculate the cross-entropy loss:

$$\mathcal{L}_G = \sum_{O_j} \text{CrossEntropy}\left(\mathbf{y}_{\text{pred},G}, \mathbf{y}_{\text{true},G}\right), \quad (14)$$

4.3.3 *Node-level Discrimination.*

For node-level training, we design a node discriminator to predict which nodes in a subgraph are masked:

$$\mathbf{y}_{\text{pred},N} = \text{Discriminator}_N\left(\mathbf{h}_G^{O_j}\right) \quad (15)$$

where $\mathbf{y}_{\text{pred},N}$ is the predicted labels of the node-level task, indicating which node is corrupted. In this work, the node discriminator is implemented as a reshape layer, which reshapes the embedding $\mathbf{h}_G^{O_j}$ to a matrix with $|\mathcal{A}|$ rows, with a full connection layer.

Considering the label of node-level task is usually unbalanced because in most cases the number of masked nodes is smaller than that of all nodes, i.e., most node labels are negative, we employ focal loss to alleviate the potential influence:

$$\mathcal{L}_N = \sum_{O_j} \text{FocalLoss}\left(\mathbf{y}_{\text{pred},N}, \mathbf{y}_{\text{true},N}\right), \quad (16)$$

Finally, we conduct joint training of both tasks, enabling our model to learn minimal context semantics from both graph-level and node-level. We optimize our model by minimizing the final objective function:

$$\mathcal{L} = \gamma \cdot \mathcal{L}_N + (1 - \gamma) \cdot \mathcal{L}_G, \tag{17}$$

where $\gamma \in [0, 1]$ is a balance scalar.

5 EXPERIMENTS

In this section, we conduct an extensive set of experiments to evaluate the effectiveness of our proposed method, MSGNN, on node classification and link prediction tasks by comparing it with existing state-of-the-art (SOTA) methods. In particular, we perform additional experiments to verify the efficiency of the schema instance retrieval strategy and the effectiveness of the masking approaches. Furthermore, we provide a comprehensive model analysis, including an ablation study and parametric experiments, to gain insights into the key components and hyperparameters of MSGNN.

5.1 Experimental Setups

5.1.1 Datasets.

For the task of node classification, we evaluate our approach on four widely-used benchmark datasets, including two academic citation networks: *DBLP* [21] and *AMiner* [34]; and two movie rating dataset: *IMDB-L* [22] and *Freebase* [1]. These datasets cover diverse domains and provide a comprehensive evaluation of our method's performance on different types of heterogeneous information networks.

In addition to the above datasets, we also employ *Yelp* [12] and *IMDB* [42] for the task of link prediction. By including these datasets, we aim to demonstrate the broad applicability and generalizability of the proposed approach to diverse domains beyond academic and knowledge graph datasets.

The summary statistics of these datasets are presented in Table 1.

Table 1: Summary of datasets (nTypes: node types, eTypes: edge types, Target: target node, and Classes: Target classes).

	# Nodes	# nTypes	# Edges	# eTypes	Target	# Classes	Task
AMiner	55,783	3	153,676	4	paper	4	LP&NC
DBLP	26,128	4	239,566	6	author	4	LP&NC
Freebase	43,854	4	151034	6	movie	3	NC
IMDB-L	21,420	4	86,642	6	movie	4	NC
IMDB	12,772	3	18,644	2	-	-	LP
Yelp	3,913	5	38,680	4	-	-	LP

5.1.2 Baselines.

To comprehensively evaluate the proposed MSGNN against the SOTA approaches, we compare SOTA methods like Homogeneous network embedding methods:

- **node2vec** [10] - node2vec is a representative method for graph representation by leveraging the random walk to generate node sequences over graphs.
- **SGC** [38] - SGC proposes to simplify the graph convolutional networks by removing the non-linear projection during the information propagation between graph layers.

Meta-path-based Heterogeneous network embedding methods:

- **MHGCN** [41]-MHGCN can automatically learn useful heterogeneous meta-path interactions of different lengths in multiplexed heterogeneous networks through multilayer convolutional aggregation.
- **BPHGNN** [5]- BPHGNN learns multiplexed heterogeneous network embedding through deep behavioral pattern aggregation and wide behavioral pattern aggregation of multiplexed heterogeneous networks
- **RGCN** [27] - RGCN uses weight sharing and coefficient constraints that take into account the effects of different edge types on nodes.
- **HetGNN** [43] - HetGNN samples the fixed-length meta-paths via recurrent neural networks.
- **HAN** [36] - HAN applies graph attention network on multiplex network considering the inter- and intra-network interactions, which exploit manually selected meta-paths to learn node embedding.
- **MAGNN** [7] - MAGNN is a meta-path-based graph neural network approach that improves results by integrating intermediate semantic nodes and information from multiple meta-paths.

Meta-path-free heterogeneous network embedding methods:

- **SR-RSC** [45] - SR-RSC proposes a meta-path-free Framework based on self-supervised subgraph contrastive learning, where the model stacks multiple coding layers to drive multi-hop message passing.
- **simpleHGN** [21] - simpleHGN experimentally verifies that Meta-path is not necessary on most heterogeneous graph datasets and proposes a simple optimal model based on GAT [35].
- **RSHN** [49] - RSHN constructs a Coarsened Line Graph Neural Network (CL-GNN) to embed nodes and edges without meta-path based on relational structure perception.
- **HGT** [13] - HGT automatically learns the importance of implicit meta-paths by decomposing the interaction matrix.
- **HINormer** [22] - HINormer utilizes Graph Transformers (GTs) to capture the local structure and heterogeneous relationships.

5.1.3 Experimental Settings.

For the task of link prediction, we classify node pairs present in graphs as positive pairs and those not as negative pairs. We predict all types of relation in graphs. For baselines, the datasets are split into training, validating, and testing sets in a proportion of 85%, 5%, and 10%, respectively. For the proposed method, we split O_{sub} into training, validating, and testing sets in a proportion of around 85%, 5%, and 10%, respectively. For a fair comparison, we ensure that 10% of edges exist exclusively in the testing set. We employ the F1 score, PR-AUC (precision-recall area under the curve), and R-AUC (area under the ROC curve) as metrics to evaluate the performance of the link prediction task. For datasets without node features, we use the one-hot vectors instead. For all baselines, we use their official implementations to guarantee their performance and search the learning rate within {0.0001, 0.0005, 0.001, 0.005, 0.01}, hidden dimensions within {64, 128, 256, 512}, output dimension within {32, 64, 128, 256}, number of attention heads, if applicable, within {1, 2, 4, 6, 8}, and number of layers within {2, 3, 4, 5} for best hyper-parameter settings. The early stop technique is applied to all baselines. For the task of node classification, we randomly divide the nodes into training, validation, and test sets following the standard split from [22]. Micro-F1 and macro-F1 are employed as metrics to evaluate the classification performance. All involved experiments in both tasks are repeated ten times, and we report the averaged results with standard deviations. The experiments are performed on a platform of Intel i9-14900k CPU with 64 GB memory and an RTX 4090 GPU.

5.2 Link Prediction

We evaluate the model performance by comparing our MSGNN with SOTA baselines on unsupervised task link prediction. We predict all types of links and the inner product is adopted. The results are reported in Table 2. The proposed MSGNN achieves optimal results on all datasets. Experimental results demonstrate that our MSGNN achieves a maximum improvement of 9.59%, 9.86%, and 16.08% compared to the suboptimal results in terms of ROC-AUC, PR-AUC, and F1-score.

Specifically, we make the following observations: Firstly, the two homogeneous embedding methods (node2vec and SGC) ignore the rich heterogeneous information and thus are not as effective as the heterogeneous methods. Among heterogeneous methods, MSGNN aggregates information from schema instances that contain richer and more complete data, thereby outperforming all other competitors. While the node2vec demonstrates high ROC-AUC and PR-AUC scores on the *Aminer* dataset, its performance in terms of F1-Score and Accuracy (34.20%, not listed in the table) is notably poor. This discrepancy is attributed to its predicted probabilities being close to 1. Secondly, across most datasets, simpleHGN demonstrates competitive performance due to enhanced graph attention networks, highlighting the efficacy of attention mechanisms in representation learning. In contrast, MSGNN employs the same mechanism for intra-schema aggregation but diverges by applying the attention mechanism to schema instances rather than structural neighbors. This approach enables MSGNN to capture deeper semantic information, resulting in significant performance gains: up to 8.10% in ROC-AUC, 7.00% in PR-AUC, and 8.00% in F1-score compared to the performance of simpleHGN on the *DBLP* dataset. Finally, as a SOTA model, BPHGNN achieves suboptimal results by effectively utilizing predefined behavioral patterns to aggregate information from both local and global perspectives. In contrast, our approach demonstrates significant advantages by searching schema instances to provide structural and semantic information. In particular, MSGNN achieves an impressive F1-score performance of over 98% on the *IMDB* dataset, surpassing the suboptimal model BPHGNN by 15.87%.

5.3 Node Classification

For the node classification task, we evaluate the node context representations learned by MSGNN to verify its effectiveness in improving the performance of downstream models. The results are shown in Table 3. The first four baselines are meta-path-based heterogeneous network methods, the last four are meta-path-free heterogeneous network methods. We employ MSGNN to generate node presentations with rich semantic information and utilize HINormer as a classifier. Experimental results demonstrate that

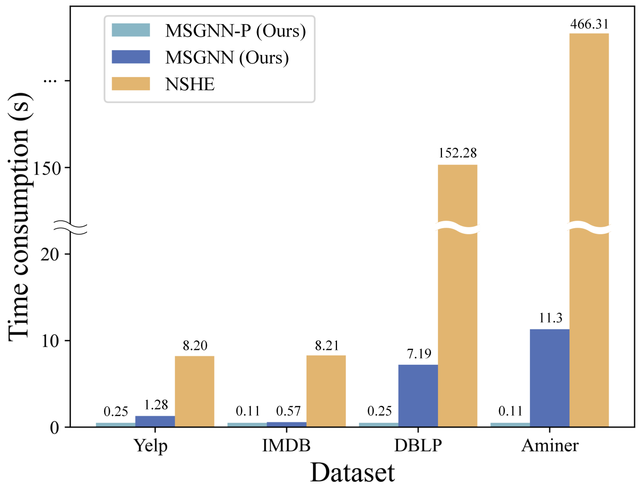


Figure 4: Time consumption comparison with NSHE. The larger the dataset, the more significant the improvement in search efficiency achieved by our proposed MSGNN

.

MSGNN+HINormer significantly outperforms all the baseline models. Notably, our MSGNN achieves a maximum improvement of 4.55% and 10.82% in terms of Micro-F1 and Macro-F1, respectively, compared to the SOTA model HINormer. The higher Macro-F1 score indicates that our model performs well across all classes of target nodes, which is particularly advantageous when dealing with imbalanced class distributions, such as in the case of *Freebase* dataset, where certain classes have significantly more nodes than others. The improved performance demonstrates that the node context representations obtained by MSGNN are highly effective for downstream node classification tasks, indicating the significance of aggregating information from schema neighborhoods.

5.4 Experiment on Efficiency of Instance Search

In this section, we compare the time-consumption of our proposed instance retrieval strategy with that of NSHE [48], which is a HIN embedding method using schema as a high-order structure to supplement structure information. Considering NSHE samples only one instance for each target node during each training epoch, we remove the sampling operation in NSHE for a fair comparison. The experimental results are shown in Figure 4, where MSGNN-P refers to parallel computation-enabled MSGNN. For subgraph querying, our approach efficiently retrieves schema instances in various datasets compared to that of NSHE. NHSE searches the instances in a depth first manner, while our approach significantly reduces the search space by employing the decomposition and reconstruction introduced in Section 4.1, remarkably improving the search efficiency. In particular, on complex datasets such as *DBLP* and *Aminer* datasets, NSHE's search methods are explosively time-consuming, taking *10* times longer than ours, and even *1000* times longer than the parallel computing one. The schema instance retrieval strategy of MSGNN has great potential in dealing with large-scale datasets.

Table 2: Model performance comparison for the task of link prediction on different datasets.

In this table, tabular results are in percent; the best result is **bolded** and the runner-up is underlined. A dash (-) denotes out of memory.

Method	DBLP			IMDB			Yelp			Aminer		
	R-AUC	PR-AUC	F1	R-AUC	PR-AUC	F1	R-AUC	PR-AUC	F1	R-AUC	PR-AUC	F1
node2vec	74.61	76.49	68.60	52.83	50.54	48.37	50.62	49.74	50.25	91.87	89.11	50.84
SGC	77.32	79.17	73.00	78.63	81.51	71.13	82.40	83.20	77.98	57.93	60.00	54.50
MAGNN	82.97	80.57	78.91	70.36	72.08	61.52	61.83	67.81	67.72	67.65	67.65	58.64
simpleHGN	84.90	83.32	77.92	87.72	87.51	79.74	80.83	77.63	74.79	86.73	89.47	81.33
SR-RSC	77.47	72.85	70.99	88.94	82.69	80.88	67.57	60.19	64.89	-	-	-
MHGCN	71.80	72.20	70.30	90.30	90.35	85.84	76.86	77.30	75.84	77.10	75.30	73.00
BPHGNN	84.27	84.18	71.38	92.55	90.44	85.28	77.36	78.40	79.86	-	-	-
Ours (MSGNN)	**93.04**	**90.26**	**85.88**	**99.81**	**99.36**	**98.81**	**86.42**	**85.36**	**80.48**	**97.75**	**94.14**	**94.41**
Std.	0.12	0.10	0.10	0.01	0.01	0.01	0.36	0.29	0.30	0.26	0.22	0.24

Table 3: Performance evaluation on node classification, with MSGNN as a self-supervised pre-training strategy.

In this table, tabular results are in percent; the best result is **bolded** and the runner-up is underlined.

Methods	DBLP		IMDB-L		Freebase		AMiner	
	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1
RGCN	92.07	91.52	62.95	58.85	60.82	59.08	81.58	62.53
HetGNN	92.33	91.76	51.16	48.25	62.99	58.44	72.34	55.42
HAN	92.05	91.67	64.63	57.74	61.42	57.05	81.90	64.67
MAGNN	93.76	93.28	64.67	56.49	64.43	58.18	82.64	68.60
RSHN	93.81	93.34	64.22	59.85	61.43	57.37	73.33	51.48
HGT	93.49	93.01	67.20	63.00	66.43	60.03	85.74	74.98
SimpleHGN	94.46	94.01	67.36	63.53	67.49	62.49	86.44	75.73
HINormer	94.94	94.57	67.83	64.65	69.42	63.93	88.04	79.88
Ours (HINormer+MSGNN)	**95.74**	**96.06**	**68.23**	**66.25**	**72.58**	**70.85**	**89.57**	**83.08**
Std.	0.23	0.45	0.24	0.29	0.29	0.56	0.32	0.38

Table 4: Different mask strategy.

In this table, tabular results are in percent; the best result is **bolded**.

Methods	DBLP			IMDB			Yelp			AMiner		
	R-AUC	PR-AUC	F1	R-AUC	PR-AUC	F1	R-AUC	PR-AUC	F1	R-AUC	PR-AUC	F1
Random mask	79.87	82.81	82.78	99.72	99.00	98.65	70.21	71.55	74.70	97.14	94.02	94.04
Zero mask	50.00	75.00	0.00	50.00	75.00	0.00	50.00	75.00	0.00	24.78	35.88	30.88
Ours	**93.04**	**90.26**	**85.88**	**99.81**	**99.36**	**98.81**	**86.42**	**85.36**	**80.48**	**97.75**	**94.14**	**94.41**

Table 5: Ablation study.

In this table, tabular results are in percent; the best result is **bolded** and the runner-up is underlined.

Methods	DBLP			IMDB			Yelp			AMiner		
	R-AUC	PR-AUC	F1	R-AUC	PR-AUC	F1	R-AUC	PR-AUC	F1	R-AUC	PR-AUC	F1
w/o graph task	77.68	72.67	70.46	96.80	96.57	90.74	62.80	64.09	64.93	57.81	54.52	54.63
w/o node task	92.62	89.57	85.53	99.67	99.06	98.07	79.52	73.26	73.26	92.28	92.91	94.18
w/o instance	62.52	59.27	58.76	99.01	98.29	95.44	70.21	63.21	68.78	95.45	92.32	89.50
w/o global	92.52	89.63	85.15	99.66	99.06	98.01	79.40	73.22	73.23	96.40	93.25	93.11
all	**93.04**	**90.26**	**85.88**	**99.81**	**99.36**	**98.81**	**86.42**	**85.36**	**80.48**	**97.75**	**94.14**	**94.41**

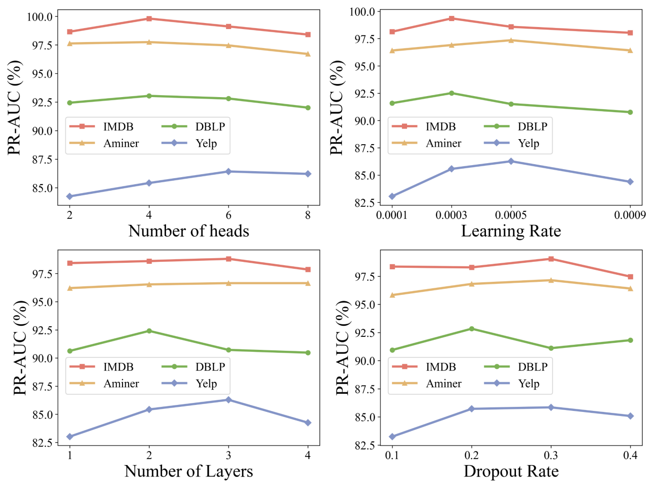


Figure 5: Parameters sensitivity. PR-AUC with the number of heads in the attention layers, the learning rate, the number of layers, and the dropout rate.

5.5 Experiment on Mask Strategy

In this section, we evaluate the masking strategy in MSGNN on link prediction task, and the results in Table 4 verify its effectiveness. For the selected schema instance, MSGNN employs a masking strategy of replacing partial nodes with those of the same type to generate corrupted samples. For the compared two strategies, one replaces nodes with random nodes, while the other replaces node features with zero vectors. The result shows that the performance of random masking (i.e., random mask in Table 4) is significantly lower than that of the same-type masking in the *DBLP* and *Yelp* datasets, and slightly lower in *IMDB* and *Aminer* datasets. This is because, by random masking, the features of replacement nodes may differ significantly from the original nodes, leading the model to converge prematurely due to the oversimplified task, thus failing to learn useful information. On the other hand, our strategy preserves certain semantic similarities, therefore effective node embeddings can be learned. Moreover, the discriminative ability of the zero mask strategy (i.e., zero mask in Table 4) for positive and negative samples is almost negligible and thus fails to learn any effective information as it destroys the semantic context of the schema instances.

5.6 Ablation Study

To validate the effectiveness of each component of the proposed MSGNN, we further conduct experiments on the following variations:

w/o graph task: This variant considers only node-level training while ignoring graph-level for self-supervised learning.

w/o node task: In contrast, this variant removes the node-level training while retaining the graph-level training.

w/o instance: This variant does not employ schema instance to aggregate node minimal context semantics and the HIN is directly fed into the model.

w/o global: This variant does not take into account the global information when computing the attention coefficients.

all: This is the original MSGNN model.

Experimental results of the ablation study are presented in Table 5. The observations and conclusions are as follows:

- The model trained with bi-level tasks outperforms variants trained with either graph-level task or node-level task. Specifically, the variant without the node-level task yields suboptimal results in most cases, while the results without the graph task are substantially lower than the original model on all four datasets. In particular, the R-AUC, PR-AUC, and F1 scores are almost halved without the graph-level task in the *Aminer* dataset, highlighting the importance of the subgraph semantics.
- Compared to the model variant without schema instances, introducing schema instances leads to significant improvements. Specifically, the R-AUC, PR-AUC, and F1 scores have increased by an average of 11.66%, 11.96%, and 11.02% in the four datasets, respectively.
- Global concatenate also plays an important role, with the R-AUC, PR-AUC, and F1 scores increasing by an average of 1.46%, 1.44%, and 1.77% across all datasets, respectively.

5.7 Parameters Sensitivity

We conduct a thorough evaluation of the sensitivity of several important hyperparameters in MSGNN, and the impact of these hyperparameters is illustrated in Figure 5. In terms of the number of attention heads, we observe that a moderate range, such as [4, 6], generally leads to better overall performance. Furthermore, we find that moderate values for learning rates, such as [0.0005, 0.0009], tend to yield optimal performance. When examining the optimal number of layers in MSGNN, we have empirically observed that employing three layers tends to yield improved performance levels. This can be attributed to the fact that a higher number of layers enhances the capacity of MSGNN to capture the deeper and more diverse semantics arising from graph heterogeneity. Additionally, we investigate the impact of the dropout rate, which determines the probability of deactivating neurons in MSGNN. When the dropout rate is set between 0.2 and 0.3, the model achieves better results.

Through systematic testing and careful analysis of these hyperparameters, we have determined that MSGNN demonstrates a high degree of robustness overall. These findings provide valuable insights into the optimal configuration of hyperparameters for MSGNN, contributing to its effectiveness in various graph-related downstream tasks.

6 CONCLUSION

We have demonstrated that schema instance represents a minimal complete semantic context for nodes in HINs. By combining schema instances with self-supervised learning and the masking technique to enhance the capability of extracting semantic information from node minimal complete context, our proposed method, MSGNN, generates informative node representations for downstream tasks. In future work, we plan to explore the potential of leveraging meta-paths as pathways to connect different schema instances.

REFERENCES

[1] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. 1247–1250.

[2] Taoyong Cui and Yuhan Dong. 2024. Simple Orthogonal Graph Representation Learning (Student Abstract). In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38. 23462–23464.

[3] Taoyong Cui, Chenyu Tang, Mao Su, Shufei Zhang, Yuqiang Li, Lei Bai, Yuhan Dong, Xingao Gong, and Wanli Ouyang. 2024. Geometry-enhanced pretraining on interatomic potentials. *Nature Machine Intelligence* (2024), 1–9.

[4] Yuxiao Dong, Nitesh V Chawla, and Ananthram Swami. 2017. metapath2vec: Scalable Representation Learning for Heterogeneous Networks. In *KDD '17*. ACM, 135–144.

[5] Chaofan Fu, Guanjie Zheng, Chao Huang, Yanwei Yu, and Junyu Dong. 2023. Multiplex Heterogeneous Graph Neural Network with Behavior Pattern Modeling. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (, Long Beach, CA, USA,) *(KDD '23)*. Association for Computing Machinery, New York, NY, USA, 482–494. https://doi.org/10.1145/3580305.3599441

[6] Tao-yang Fu, Wang-Chien Lee, and Zhen Lei. 2017. HIN2Vec: Explore Meta-paths in Heterogeneous Information Networks for Representation Learning. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management* (Singapore, Singapore) *(CIKM '17)*. Association for Computing Machinery, New York, NY, USA, 1797–1806.

[7] Xinyu Fu, Jiani Zhang, Ziqiao Meng, and Irwin King. 2020. Magnn: Metapath aggregated graph neural network for heterogeneous graph embedding. In *Proceedings of The Web Conference 2020*. 2331–2341.

[8] Xinyu Fu, Jiani Zhang, Ziqiao Meng, and Irwin King. 2020. MAGNN: Metapath Aggregated Graph Neural Network for Heterogeneous Graph Embedding. In *Proceedings of The Web Conference 2020 (WWW '20)*. ACM.

[9] Kaushal Giri. 2011. Role of ontology in semantic web. *DESIDOC Journal of Library & Information Technology* 31, 2 (2011).

[10] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable Feature Learning for Networks. arXiv:1607.00653 [cs.SI]

[11] William L. Hamilton, Rex Ying, and Jure Leskovec. 2018. Inductive Representation Learning on Large Graphs. arXiv:1706.02216 [cs.SI]

[12] Binbin Hu, Yuan Fang, and Chuan Shi. 2019. Adversarial learning on heterogeneous information networks. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 120–129.

[13] Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. 2020. Heterogeneous graph transformer. In *Proceedings of the web conference 2020*. 2704–2710.

[14] Houye Ji, Xiao Wang, Chuan Shi, Bai Wang, and S Yu Philip. 2021. Heterogeneous graph propagation network. *IEEE Transactions on Knowledge and Data Engineering* 35, 1 (2021), 521–532.

[15] Di Jin, Zhizhi Yu, Dongxiao He, Carl Yang, S Yu Philip, and Jiawei Han. 2021. GCN for HIN via implicit utilization of attention and meta-paths. *IEEE Transactions on Knowledge and Data Engineering* 35, 4 (2021), 3925–3937.

[16] Thomas N Kipf and Max Welling. 2016. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308* (2016).

[17] Guohao Li, Chenxin Xiong, Ali Thabet, and Bernard Ghanem. 2020. Deepergcn: All you need to train deeper gcns. *arXiv preprint arXiv:2006.07739* (2020).

[18] Jintang Li, Ruofan Wu, Wangbin Sun, Liang Chen, Sheng Tian, Liang Zhu, Changhua Meng, Zibin Zheng, and Weiqiang Wang. 2023. What's Behind the Mask: Understanding Masked Graph Modeling for Graph Autoencoders. arXiv:2205.10053 [cs.LG]

[19] Zhiyuan Liu, Yaorui Shi, An Zhang, Enzhi Zhang, Kenji Kawaguchi, Xiang Wang, and Tat-Seng Chua. 2024. Rethinking Tokenizer and Decoder in Masked Graph Modeling for Molecules. arXiv:2310.14753 [cs.LG]

[20] Yuanfu Lu, Chuan Shi, Linmei Hu, and Zhiyuan Liu. 2019. Relation Structure-Aware Heterogeneous Information Network Embedding. arXiv:1905.08027 [cs.SI]

[21] Qingsong Lv, Ming Ding, Qiang Liu, Yuxiang Chen, Wenzheng Feng, Siming He, Chang Zhou, Jianguo Jiang, Yuxiao Dong, and Jie Tang. 2021. Are we really making much progress? Revisiting, benchmarking and refining heterogeneous graph neural networks. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 1150–1160.

[22] Qiheng Mao, Zemin Liu, Chenghao Liu, and Jianling Sun. 2023. HINormer: Representation Learning On Heterogeneous Information Networks with Graph Transformer. arXiv:2302.11329 [cs.LG]

[23] J Clyde Mitchell. 1974. Social networks. *Annual review of anthropology* 3, 1 (1974), 279–299.

[24] Shirui Pan, Ruiqi Hu, Guodong Long, Jing Jiang, Lina Yao, and Chengqi Zhang. 2019. Adversarially Regularized Graph Autoencoder for Graph Embedding. arXiv:1802.04407 [cs.LG]

[25] Filippo Radicchi, Santo Fortunato, and Alessandro Vespignani. 2011. Citation networks. *Models of science dynamics: Encounters between complexity theory and information sciences* (2011), 233–257.

[26] Ahmed E. Samy, Lodovico Giaretta, Zekarias T. Kefato, and Šarūnas Girdzijauskas. 2022. SchemaWalk: Schema Aware Random Walks for Heterogeneous Graph Embedding. In *Companion Proceedings of the Web Conference 2022* (Virtual Event, Lyon, France) *(WWW '22)*. Association for Computing Machinery, New York, NY, USA, 1157–1166. https://doi.org/10.1145/3487553.3524728

[27] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. 2018. Modeling relational data with graph convolutional networks. In *The semantic web: 15th international conference, ESWC 2018, Heraklion, Crete, Greece, June 3–7, 2018, proceedings 15*. Springer, 593–607.

[28] Jingbo Shang, Meng Qu, Jialu Liu, Lance M Kaplan, Jiawei Han, and Jian Peng. 2016. Meta-path guided embedding for similarity search in large-scale heterogeneous information networks. *arXiv preprint arXiv:1610.09769* (2016).

[29] Chuan Shi, Binbin Hu, Wayne Xin Zhao, and Philip S. Yu. 2017. Heterogeneous Information Network Embedding for Recommendation. arXiv:1711.10730 [cs.SI]

[30] Chuan Shi, Yitong Li, Jiawei Zhang, Yizhou Sun, and S Yu Philip. 2016. A survey of heterogeneous information network analysis. *IEEE Transactions on Knowledge and Data Engineering* 29, 1 (2016), 17–37.

[31] Chang Su, Jie Tong, Yongjun Zhu, Peng Cui, and Fei Wang. 2020. Network embedding in biomedical data science. *Briefings in bioinformatics* 21, 1 (2020), 182–197.

[32] Yizhou Sun and Jiawei Han. 2012. *Mining heterogeneous information networks: principles and methodologies*. Morgan & Claypool Publishers.

[33] Qiaoyu Tan, Ninghao Liu, Xiao Huang, Rui Chen, Soo-Hyun Choi, and Xia Hu. 2022. MGAE: Masked Autoencoders for Self-Supervised Learning on Graphs. arXiv:2201.02534 [cs.LG]

[34] Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. 2008. ArnetMiner: Extraction and Mining of Academic Social Networks. In *KDD'08*. 990–998.

[35] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. arXiv:1710.10903 [stat.ML]

[36] Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S Yu. 2019. Heterogeneous graph attention network. In *The world wide web conference*. 2022–2032.

[37] Yuxiang Wang, Xiao Yan, Chuang Hu, Fangcheng Fu, Wentao Zhang, Hao Wang, Shuo Shang, and Jiawei Jiang. 2023. Generative and Contrastive Paradigms Are Complementary for Graph Self-Supervised Learning. *arXiv preprint arXiv:2310.15523* (2023).

[38] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, et al. 2019. Simplifying Graph Convolutional Networks. In *ICML*. 6861–6871.

[39] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. 2020. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems* 32, 1 (2020), 4–24.

[40] Pengyang Yu, Chaofan Fu, Yanwei Yu, Chao Huang, Zhongying Zhao, and Junyu Dong. 2022. Multiplex Heterogeneous Graph Convolutional Network. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '22)*. ACM. https://doi.org/10.1145/3534678.3539482

[41] Pengyang Yu, Chaofan Fu, Yanwei Yu, Chao Huang, Zhongying Zhao, and Junyu Dong. 2022. Multiplex heterogeneous graph convolutional network. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 2377–2387.

[42] Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J Kim. 2019. Graph transformer networks. *Advances in neural information processing systems* 32 (2019).

[43] Chuxu Zhang, Dongjin Song, Chao Huang, Ananthram Swami, and Nitesh V Chawla. 2019. Heterogeneous graph neural network. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 793–803.

[44] Rui Zhang, Arthur Zimek, and Peter Schneider-Kamp. 2022. A Simple Meta-path-free Framework for Heterogeneous Network Embedding. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management* (Atlanta, GA, USA) *(CIKM '22)*. Association for Computing Machinery, New York, NY, USA, 2600–2609. https://doi.org/10.1145/3511808.3557223

[45] Rui Zhang, Arthur Zimek, and Peter Schneider-Kamp. 2022. A simple meta-path-free framework for heterogeneous network embedding. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*. 2600–2609.

[46] Sixiao Zhang, Hongxu Chen, Haoran Yang, Xiangguo Sun, Philip S. Yu, and Guandong Xu. 2022. Graph Masked Autoencoders with Transformers. arXiv:2202.08391 [cs.LG]

[47] Jianan Zhao, Xiao Wang, Chuan Shi, Binbin Hu, Guojie Song, and Yanfang Ye. 2021. Heterogeneous graph structure learning for graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 35. 4697–4705.

[48] Jianan Zhao, Xiao Wang, Chuan Shi, Zekuan Liu, and Yanfang Ye. 2021. Network schema preserving heterogeneous information network embedding. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence* (Yokohama, Yokohama, Japan) *(IJCAI'20)*. Article 190, 7 pages.

[49] Shichao Zhu, Chuan Zhou, Shirui Pan, Xingquan Zhu, and Bin Wang. 2019. Relation structure-aware heterogeneous graph neural network. In *2019 IEEE*

international conference on data mining (ICDM). IEEE, 1534–1539.