SUREL+: Moving from Walks to Sets for Scalable Subgraph-based Graph Representation Learning

Haoteng Yin
Purdue University
yinht@purdue.edu

Muhan Zhang
Peking University
muhan@pku.edu.cn

Jianguo Wang
Purdue University
csjgwang@purdue.edu

Pan Li
Georgia Tech
panli@gatech.edu

ABSTRACT

Subgraph-based graph representation learning (SGRL) has recently emerged as a powerful tool in many prediction tasks on graphs due to its advantages in model expressiveness and generalization ability. Most previous SGRL models face computational issues related to the high cost of extracting subgraphs for each training or testing query. Recently, SUREL was proposed to accelerate SGRL, which samples random walks offline and joins these walks online as a proxy of subgraphs for prediction. Thanks to the reusability of sampled walks across different queries, SUREL achieves state-of-the-art performance in terms of scalability and prediction accuracy. However, SUREL still suffers from high computational overhead caused by node redundancy in sampled walks. In this work, we propose a novel framework SUREL+ that upgrades SUREL by using node sets instead of walks to represent subgraphs. By definition, such set-based representations avoid repeated nodes, but node sets can be irregular in size. To solve this issue, we design a dedicated sparse data structure to efficiently store and access node sets, and provide a specialized operator to join them in parallel batches. SUREL+ is modularized to support multiple types of set samplers, structural features, and neural encoders to complement the loss of structural information after the reduction from walks to sets. Extensive experiments have been performed to verify the effectiveness of SUREL+ in the prediction tasks of links, relation types, and higher-order patterns. SUREL+ achieves 3-11× speedups of SUREL while maintaining comparable or even better prediction performance; compared to other SGRL baselines, SUREL+ achieves ~20× speedups and significantly improves the prediction accuracy.

PVLDB Reference Format:
Haoteng Yin, Muhan Zhang, Jianguo Wang, and Pan Li. SUREL+: Moving from Walks to Sets for Scalable Subgraph-based Graph Representation Learning. PVLDB, 16(11): 2939 - 2948, 2023.
doi:10.14778/3611479.3611499

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/Graph-COM/SUREL_Plus.

1 INTRODUCTION

Graphs are widely used to model interactions in natural sciences and relationships in social life [20, 23]. Graph-structured data in the real world are highly irregular and often large-scale. To solve inference tasks on graphs, graph representation learning (GRL) that studies quantitative representations of graph-structured data has attracted much attention [15, 16, 45]. Recently, subgraph-based GRL (SGRL) has become an important research direction for researchers studying GRL algorithms and systems, as it achieves far better prediction performance than other approaches in many GRL tasks, especially those involving a set of nodes. Given a set of nodes of interest, namely a queried node set, SGRL models such as SEAL [50, 52], GraIL [39], and SubGNN [1] first extract a subgraph around the queried node set (called query-induced subgraph), and then use neural networks to encode extracted subgraphs for prediction. Extensive work shows that SGRL models are more robust [48] and more expressive [5, 12]; while canonical graph neural networks (GNNs) including GCN [22] and GraphSAGE [14] usually fail to make accurate predictions, due to their limited expressive power [9, 13, 52], incapability of capturing intra-node distance information [27, 37], and improper entanglement between receptive field size and model depth [18, 47, 48]. An example in Fig. 1 illustrates how SGRL works for link prediction and demonstrates its advantages over GNNs. Here, canonical GNNs generate and aggregate node representations to predict links, which would map structurally symmetric nodes into the same representation and lead to the ambiguity issue [46, 52]. So far, the advantages of SGRL methods have been proved in many applications, such as link and relation prediction [39, 50, 52], higher-order pattern prediction [29, 32], temporal network modeling [44], recommender systems [51], anomaly detection [1, 6], graph meta-learning [18], subgraph matching [28, 30], and molecular/protein research in life sciences [36, 43].

Albeit with multiple benefits of its algorithm, SGRL methods currently face two major computational challenges: (1) **Query Dependency**. A subgraph must be extracted for each queried node set, which is not reusable across different queries, and cannot be preprocessed if the query is unknown; (2) **Irregularity**. The extracted subgraphs are irregularly sized, resulting in poor batch processing and load-balancing performance. As shown in Fig. 3 (a), subgraph extraction in SEAL [50, 52] is prohibitively slow to be deployed in practice. This inspired recent work on dedicated hardware acceleration for subgraph extraction [10, 34]. However, how to fundamentally improve the scalability and efficiency of SGRL methods remains largely unexplored.

SUREL [47] is the state-of-the-art (SOTA) framework that applies algorithm and system co-design to implement SGRL. It employs the join of node-level sampled walks to represent query-specific subgraphs. Specifically, SUREL treats each node as a seed and runs multiple random walks from the seed on the graph offline. Given a queried node set, SUREL online joins and encodes the sampled walks for all queried nodes for prediction. The join operation builds

Proceedings of the VLDB Endowment, Vol. 16, No. 11 ISSN 2150-8097.
doi:10.14778/3611479.3611499

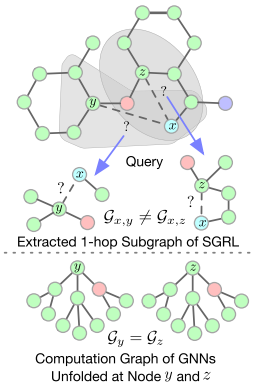


Figure 1: GNNs cannot correctly predict whether x is more likely linked with y or z, because y and z have the same node representations. While the representations based on one-hop neighbors (query-induced subgraph) are more expressive to tell these two node pairs apart.

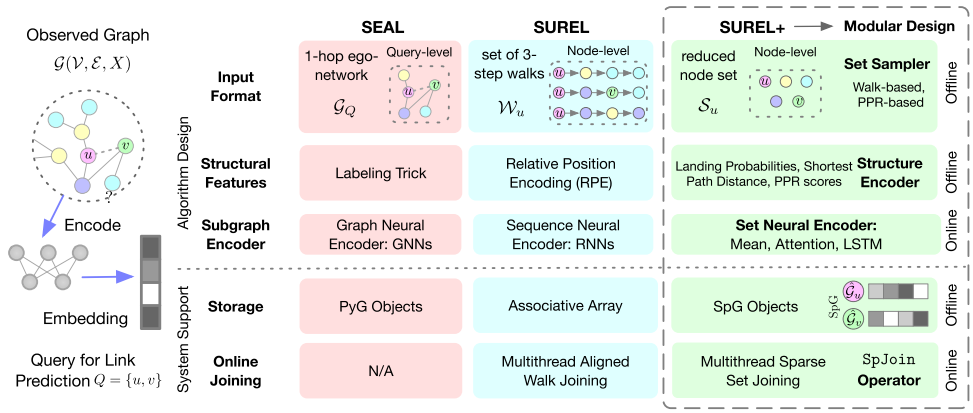


Figure 2: Overview of SUREL+: a side-by-side comparison with previous SGRL models. For a query, SGRL first prepares subgraphs paired with structural features, which are then fed into a neural encoder to obtain embeddings for prediction. SUREL+ samples node sets as input, while SEAL [50, 52] extracts induced subgraphs and SUREL [47] samples walks. To compensate for structure loss by reducing subgraphs to node sets, SUREL+ provides various types of set samplers, structure encoders, and set neural encoders. To better serve set-based representations, SUREL+ designs a customized sparse data structure `SpG` to store sampled node sets with fast access, and supports their online joining in parallel via a sparse join operator `SpJoin`.

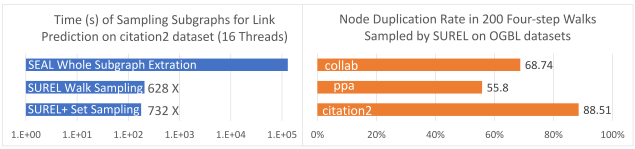


Figure 3: (a) Subgraph extraction in SEAL [50, 52] has much higher complexity than other simplified form samplers. The size of induced subgraphs is irregular, growing exponentially with the number of extracted hops [8, 48]. (b) Breaking subgraphs into reusable walks reduces complexity but faces the issue of heavy node duplicates.

the connection between sampled walks of queried nodes so that their joint walks can act as a proxy of the query-induced subgraph. To compensate for the structural loss by representing subgraphs in walks, a structural feature termed relative position encoding (RPE) is adopted, which records the relative position of each distinct node in sampled walks towards the seed. The RPE is precomputed offline, then attached to walks before being fed into neural networks (NNs) to make predictions. In SUREL, sampled walks from one seed can be reused for multiple queries whenever that seed node is involved. SUREL significantly improves the scalability of SGRL by this walk-sharing mechanism and benefits from the regularity of walks, which enables highly parallel walk sampling and online joining through a dedicated system design. However, SUREL still faces several inherent drawbacks of the walk-based representation, namely high node redundancy in sampled walks (over 55% duplication, see Fig. 3 (b)). This further raises the following issues: (1) **extra space** for hosting walks and positional encoding in memory, (2) **extra time** for operations on repeated nodes in walk joining and NN-based encoding in subsequent routines, and (3) **high workload** of data transfer between CPU and GPU.

In this work, we upgrade SUREL and develop a novel framework SUREL+ that once again benefits from algorithm-system co-design. The core concept of SUREL+ is simple: instead of using walks to represent subgraphs, we now employ node sets, thereby obviating node duplication. However, this new idea based on node sets introduces several difficulties for both algorithm and system design. From an algorithmic perspective, transitioning from induced subgraphs to walks and then to node sets, as explored in this study, results in a considerable loss of structural information. Therefore, a key challenge in algorithm design is to develop an approach that can compensate for this loss while maintaining performance. On the system side, SUREL [47] utilizes walks, which can be easily stored and processed in a regular alignment format by controlling the length and number of walk sampling. In contrast, node sets exhibit irregular sizes, posing a daunting task for efficient storage and access. Consequently, how to coordinate the designs of both sides constitutes the primary challenge of this work.

SUREL+ tackles the above challenges throughout its entire pipeline. During preprocessing, SUREL+ samples a subset of unique nodes from each seed node's neighborhood to remove query dependency and avoid node duplication. To compensate for the loss of structural information, SUREL+ incorporates various types of set samplers and structure encoders to preserve local structures: *set samplers* employ different graph metrics to measure node importance and determine sampling rules; *structure encoders* support landing probabilities of random walks [26], shortest path distances, and personalized PageRank scores [19], covering most of the structural features used by previous SGRL models [27, 39, 47, 50, 52]. Furthermore, SUREL+ designs a dedicated sparse data structure, namely `SpG`, which can efficiently store sampled node sets and achieve fast access. During training and inference, SUREL+ implements a sparse operator `SpJoin` to perform join operations on the sampled node sets and their associated structural features of all nodes in a query to represent the query-induced subgraph for prediction. In addition, to capture diverse levels of interactions between node and structural features, SUREL+ supports multiple *set neural encoders*, such

as multi-linear perception + mean pooling, set attention [40] and LSTM [14] that ensure sufficient expressive power and consistent performance across various types of SGRL tasks.

Overall, our contributions can be summarized as follows:

- Algorithm: SUREL+ is a novel SGRL framework (*open source*), that utilizes reusable node sets and associates them with various types of structural features to represent query-induced subgraphs through their online joining. Adopting sets in SUREL+ greatly saves memory and computation but without degrading prediction performance compared with the SOTA baselines.
- System: SUREL+ designs a dedicated sparse data structure `SpG` and a sparse join operator `SpJoin` to support efficient storage and processing of node sets, which achieves much better efficiency and scalability than previous SGRL methods.
- We conduct extensive experiments on 9 real-world graphs, with millions/billions of nodes/edges, and demonstrate the advantages of SUREL+ in link/relation-type/motif prediction tasks. SUREL+ is 3-11× faster than the current SOTA SGRL method SUREL while maintaining comparable or even better prediction accuracy. SUREL+ also achieves ∼20× speedup with substantial prediction accuracy improvements over other SGRL baselines.

2 PRELIMINARIES

2.1 Notations and Relevant Definitions in SGRL

Let $\mathcal{G}(\mathcal{V}, \mathcal{E}, X)$ be an attributed graph with node set $\mathcal{V} = \{1, 2, ..., n\}$ and edge set $\mathcal{E}$, where $X \in \mathbb{R}^{n \times d}$ denotes node attributes with d-dimension. A query $Q \subset \mathcal{V}$ is a node set of interest for a certain type of task. We denote the subgraph induced by query Q as $\mathcal{G}_Q$ and the node-induced subgraph as $\mathcal{G}_u$, where induced subgraphs are typically within a small number of hops.

Definition 2.1 (Subgraph-based Graph Representation Learning (**SGRL**))**.** Given a query Q of node set over graph $\mathcal{G}$, SGRL aims to learn a representation of the query-induced subgraph $\mathcal{G}_Q$ to make prediction $f(\mathcal{G}_Q)$. $(\cdot)$ is usually a neural network. SGRL tasks come with some labeled queries $\{(Q_i, y_i)\}_{i=1}^{L}$ for supervision (positive samples) and other unlabeled queries $\{Q_i\}_{i=L+1}^{L+N}$ for inference.

Examples of SGRL Tasks *Link prediction* seeks to estimate the likelihood of a link between two endpoints in a given graph, where a query Q corresponds to a node pair. It can be further generalized to predict links with types over heterogeneous graphs [39] or to predict vascular access [33] and chemical bond [20] in domain-specific graphs. Tasks beyond pairwise relations are named *higher-order pattern prediction*, where a query Q consists of three or more nodes. In this work, we consider that given partially observed pairwise relations among queried nodes in Q, whether these queried nodes will establish certain full higher-order relation of interest [29, 38].

Review of SGRL Methods The current SGRL pipeline has three main parts, as shown in the *Algorithm Design* section of Fig. 2: preparation of subgraphs, construction of structural features, and neural encoder to obtain subgraph embeddings. Classical SGRL models often group query-dependent parts together, e.g., SEAL [50, 52] couples subgraph extraction and labeling trick [52], and then applies GNNs to encode extracted subgraphs attached with structural labels for prediction. However, such coupling is expensive and makes the computed intermediate results not reusable across queries, which motivates recent SGRL methods to decouple them. SUREL [47] substitutes explicit subgraph extraction with the online joining of multiple node-level walks presampled offline, with relative position encoding defined on sampled walks as structural features. Sampled walks and associated positional encoding can be shared to assemble subgraphs for multiple queries, which improves the reusability and scalability. Lastly, neural networks are applied to encode and aggregate the embedding of these walks for prediction.

2.2 Related Works

Scalable SGRL Design. Recent works on SGRL models have primarily focused on efficient subgraph extraction. Various techniques have been proposed, including PPR-based [4, 48] and random walk-based [47] subgraph samplers, and node neighborhood sampling through CUDA kernel (DGL, [10]) and tensor operations (PyG, [34]). Some frameworks have customized data structures to better support subgraph operations and gain higher throughput, such as associative arrays in SUREL [47], temporal-CSR in TGL [53], and GPU-orientated dictionary in NAT [31]. To achieve scalable modeling design, GDGNN [24] utilizes node representations along the geodesic path between queried nodes for prediction, partially decoupling structural feature construction from subgraph extraction. BUDDY [7] employs subgraph sketches to avoid explicitly extracting subgraphs for link prediction. However, these works either focus on specific aspects of SGRL's scalability issues, i.e., bottlenecks of extraction and storage or feature construction, or they are limited to specific tasks like link prediction. In contrast, SUREL+ provides a comprehensive co-design approach in scalable sampling, efficient storage, and expressive modeling, offering a general subgraph-based framework for scalable SGRL.

3 THE FRAMEWORK OF SUREL+

This section introduces SUREL+, whose key concept is to offline sample node sets around seed nodes in the graph, which can be joined online as a proxy of query-induced subgraphs for representation learning. This approach keeps only distinct nodes in the sampled set for reuse in different queries, effectively addressing memory and computation concerns of node duplication in existing walk-based representations adopted by SUREL [47]. SUREL+ features a modular design, supports various set samplers and structure encoders, and can flexibly select different set neural encoders to pair with diverse structural features to compensate for the structure information loss after reducing subgraphs to node sets. Furthermore, SUREL+ introduces a dedicated sparse data structure `SpG` and an arithmetic operator `SpJoin` to store node sets and perform their online joins efficiently. Fig. 2 compares SUREL+ and current SGRL models. The following subsections describe these modules in detail.

3.1 Set Samplers and Structure Encoders

SUREL+ uses set samplers to sample a set of nodes from the neighborhood of each seed node in the graph and calls structure encoders to construct structural features. Both of these operations are executed offline. The former is primarily for computational benefits, while the latter is performed to offset the loss of structural information due to the reduction from subgraphs (adopted by SEAL [50, 52]) or walks (adopted by SUREL [47]) to sets. Conceptually, SUREL+ represents the node-induced subgraph $\mathcal{G}_u$ via a combination of (1)

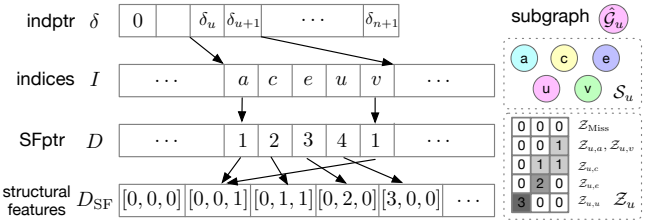


Figure 4: Node set $\mathcal{S}_u$ and its associated structural features $\mathcal{Z}_u$ stored in SpG. Here, D_{SF} shows the landing counts of nodes at different steps in sampled walks as an example, which can be normalized later to landing probabilities as structural features.

Table 1: Complexity comparison of GRL models. Suppose using $O(|\mathcal{E}|)$-many queries, SGRLs use partial edges ($q \ll |\mathcal{E}|$) for training. S and K denote the average size of extracted subgraphs and sampled node sets, respectively. L is the number of layers. d and k are respective dimensions of node and structural features. Assume d is fixed for all layers. Both SUREL and SUREL+ use the walk-based sampler for M-many m-step walks. c is the number of distinct k-dim structural features. δ_{n+1} is the size sum of all sampled node sets.

Methods	GNN [22]	SEAL [50, 52]	SUREL [47]	SUREL+
Structure	$O(\lvert\mathcal{V}\rvert + \lvert\mathcal{E}\rvert)$	$O(S\lvert\mathcal{E}\rvert)$	$O(mM\lvert\mathcal{V}\rvert)$	$O(\delta_{n+1})$
Feature	$O(d\lvert\mathcal{V}\rvert)$	$O(kS\lvert\mathcal{E}\rvert)$	$O(kmM\lvert\mathcal{V}\rvert)$	$O(\delta_{n+1} + c * k)$
Time	$O(\lvert\mathcal{E}\rvert Ld + \lvert\mathcal{E}\rvert Ld^2)$	$O(qS^L d^2)$	$O(qmMd^2)$	$O(qKd^2)$

a node set $\mathcal{S}_u$ comprising unique nodes sampled from the neighborhood of node u and (2) the associated structural features $\mathcal{Z}_u$ that reflects the position in $\mathcal{G}_u$ of each sampled node in $\mathcal{S}_u$.

Set Samplers Two types of set samplers are adopted. The first type, named *Walk-based Sampler*, is to sample short-step random walks and eliminate repeated nodes during sampling. The second type, named *Metric-based Sampler*, is based on more principled graph metrics that measure the proximity between neighboring nodes and the seed node, such as personalized PageRank (PPR) scores [19] or short path distances. Specifically, the walk-based sampler runs M-many m-step random walks, starting from each seed u in parallel on the graph $\mathcal{G}$, and then puts only distinct nodes in these walks into the set $\mathcal{S}_u$. The metric-based sampler, taking PPR-based [4] as an example, first runs the push-flow algorithm [2] to obtain an approximation of the PPR vector for each seed u, and then selects the top-K nodes with the highest PPR scores into the set $\mathcal{S}_u$. Mathematically, PPR scores are convergent landing probabilities of seeded random walks that reach infinite steps. Therefore, these two samplers complement each other by leveraging either more local or global graph structures. We use hyper-parameters M, m to control random walks, and K to control metric-based sampler, which are all set as some constants in practice. The complexity of the above offline sampling procedures is $O(|\mathcal{V}|)$.

Structure Encoders The structure encoder is to construct structural features $\mathcal{Z}_{u,x} \in \mathbb{R}^k$ for each node x in the sampled node set $\mathcal{S}_u$. These features are crucial for inference tasks involving multiple nodes [52], and can be conceptually understood as defining the position of a node x relative to a seed node u within its neighborhood. One possible choice is landing probabilities of random walk [26, 27, 47]: each element $\mathcal{Z}_{u,x}[i]$ stores the counts of node x landed at step i of all walks sampled by the walk-based sampler starting at the seed u divided by the total number of walks. By definition, landing probabilities (LPs) can be obtained together with walk sampling. Another option is the shortest path distance (SPD) between x and u [27, 50, 52], which records their relative position in terms of quantitative reachability. PPR scores [19] is also a useful structural feature and can be computed along the running of a PPR-based sampler. Later, we denote the group of structural features for all nodes in $\mathcal{S}_u$ as $\mathcal{Z}_u = \{\mathcal{Z}_{u,x} | x \in \mathcal{S}_u\}$.

3.2 Set-based Storage - SpG

Node-set-based representations have advantages in terms of reusability and eliminating redundant nodes. However, the uneven sizes of sampled node sets pose great challenges to their storage and fast access. Note that these node sets must be frequently visited in subsequent online phases for inference. To overcome these obstacles, SUREL+ devises a specialized compressed sparse row (CSR) format called SpG, which reorganizes the storage of node sets and their structural features in a memory-efficient manner, as depicted in Fig. 4. Specifically, the node set $\mathcal{S}_u$ and its structural features $\mathcal{Z}_u$ are stored as a row of SpG, denoted as $\texttt{SpG}[u, :]$. Multiple node sets and their associated structural features are consolidated into three contiguous arrays:

- `indptr` $\delta \in \mathbb{Z}^{n+1}$, an integer array tracks the starting index of each stored node set (row). It records the cumulative sum of the sizes of all node sets $\mathcal{S}_u$, $\forall u \in \mathcal{V}$, e.g., $\delta_{u+1} = \delta_u + |\mathcal{S}_u|$, where $|\mathcal{S}_u|$ represents the size of the set $\mathcal{S}_u$. The total number of sampled nodes stored in SpG is δ_{n+1};
- `indices` $I \in \mathbb{Z}^{\delta_{n+1}}$, a coalesce array of all node sets $\mathcal{S}_u$, $\forall u \in \mathcal{V}$. The segment $I[\delta_u : \delta_{u+1}]$ corresponds to the indices of sampled nodes of $\mathcal{S}_u$ stored in sorted order. This ordering is particularly useful for speeding up the join operation discussed in Sec. 3.3.
- `SFptr` $D \in \mathbb{R}^{\delta_{n+1}}$, a pointer array contains the indices of the structural features $\mathcal{Z}_u$ stored in the array D_{SF}. The purpose of D_{SF} is to eliminate duplicate structural features, typically reside in GPU memory. This secondary index can further compress memory needs, especially when using LPs/SPDs that are likely to have many repeated values, but it is not necessary when using PPR scores since they tend to have distinct values.

Regarding the cost of SpG, `indptr` array is of size $|\mathcal{V}| + 1$, and the size of both `indices` and `SFptr` arrays is δ_{n+1}. The compressed encoding array D_{SF}, has a size of $c * k$, where c is the number of distinct structural features and k denotes feature dimension. The overall complexity of this data structure is $O(|\mathcal{V}| + \delta_{n+1} + c * k)$.

Comparison with Other Methods Table 1 summarizes the space and time complexity comparison of GRL methods. By adopting the walk-based sampler (sampling M-many m-step walks), δ_{n+1} amounts to approximately one-fifth of $mM|\mathcal{V}|$ used by SUREL, while the metric-based sampler (sampling top-K PPR scores) results in $\delta_{n+1} = K|\mathcal{V}|$ and $K < mM$ in general. Both values are substantially lower than $O(S|\mathcal{E}|)$ used by SEAL, where S is the average size of sampled subgraphs. For hosting structural features, SUREL+ employs the secondary index `SFptr` D and retains only distinct structural features in D_{SF} to further reduce the memory footprint. Since c typically remains independent of $|\mathcal{V}|$ in practice, SUREL+ equipped with SpG is highly suitable for handling large graphs.

3.3 Joining Node Sets via Sparse Operations

The goal of joining node sets is to connect queried nodes and construct query-induced subgraphs from presampled node sets to make predictions for SGRL queries. For a given query Q, we combine relevant node sets $\mathcal{S}_u, \forall u \in Q$ into $\mathcal{S}_Q = \bigcup_{u \in Q} \mathcal{S}_u$ and *join* their node-level structural features $\mathcal{Z}_u$ to the query level. In essence, query-level structural features $\mathcal{Z}_Q$ serve as an anchor that indicates the position of node $x \in \mathcal{S}_Q$ within the query-induced subgraph $\mathcal{G}_Q$. Specifically, for a node x in $\mathcal{S}_Q$, its query-level structural features $\mathcal{Z}_{Q,x}$ are obtained by concatenating the node-level structural features $\mathcal{Z}_{u,x}$ for all u in Q as

$$\mathcal{Z}_{Q,x} = ||_{u \in Q} \mathcal{Z}_{u,x} = [\dots \mathcal{Z}_{u,x} \dots] \in \mathbb{R}^{|Q| \times k}, \tag{1}$$

where $||$ denotes concatenation. In cases where $\mathcal{Z}_{u,x}$ does not exist as node $x \notin S_u$ while $u \in Q$, it is set to all zeros. For instance, in Fig. 5, node b is in $\mathcal{S}_v$ but not in $\mathcal{S}_u$, hence $\mathcal{Z}_{u,b}$ is set to zero. $\mathcal{Z}_Q$ is a collection of $\mathcal{Z}_{Q,x}, \forall x \in \mathcal{S}_Q$. Together, $\mathcal{S}_Q$ and $\mathcal{Z}_Q$ are substituted for the query-induced subgraph $\mathcal{G}_Q$ and later fed into the neural encoder to obtain embeddings for prediction.

The JOIN operator in databases is used to merge tables and establish connections. Concatenation in Eq. (1) requires matching among different node sets with varying sizes and arbitrary node orders, where an outer JOIN is well-suited for this task. In this case, the JOIN operator returns the associated value based on the node ids from target sets as the specified common field, regardless of its existence. Here, to obtain $\mathcal{Z}_{Q,x}$, node sets $\{\mathcal{S}_u\}_{u \in Q}$ are treated as tables: if the index of a node x matches one of the node indices in $\mathcal{S}_u$ for all u in Q, then the associated structural features $\mathcal{Z}_{u,x}$ are appended; otherwise, the field is filled with zeros. However, iterating over all $\mathcal{S}_u$'s to retrieve $\mathcal{Z}_{u,x}$ for each node $x \in \mathcal{S}_Q$ is highly inefficient, as its complexity can be $O(|Q| * |\mathcal{S}_Q|^2)$ per query. This becomes even more challenging when performing these operations for massive queries with sets $\mathcal{S}_u$ and $\mathcal{S}_Q$ of varying sizes.

To tackle this issue, we design an efficient sparse arithmetic operator `SpJoin` to perform joins in parallel on sparse data objects of `SpG`. This operator reduces the time complexity per query to $O(|Q| * |\mathcal{S}_Q|)$ by taking advantage of the fact that node indices of $\mathcal{S}_u$ stored in `SpG` are unique and in sorted order. The following demonstrates the use of `SpJoin` with a query $Q = \{u, v\}$.

Sparse Join Operator The operator `SpJoin` performs an outer JOIN on the sampled node sets from seeds u and v of query Q stored in `SpG` as $\texttt{SpG}[u,:]$ and $\texttt{SpG}[v,:]$ through

$$\texttt{SpJoin}(\texttt{SpG}[u,:], \texttt{SpG}[v,:]) =$$
$$\texttt{SpAdd}(\texttt{mask}, \texttt{SpG}[u,:]) - 1 \;||\; \texttt{SpAdd}(\texttt{mask}, \texttt{SpG}[v,:]) - 1$$

where $\texttt{mask} = \texttt{bool}(\texttt{SpAdd}(\texttt{SpG}[u,:], \texttt{SpG}[v,:]))$.

As illustrated in Fig. 5, `SpJoin` consists of three steps: (1) The `SpJoin` operator utilizes sparse arithmetic operations from SciPy [41]: `SpAdd` performs an element-wise addition ($X \oplus Y$) of the non-zero elements in X and Y; the resulting values are converted to binary via the `bool` operator and saved in the `mask`, which contains the node indices of the union set $\mathcal{S}_Q$. (2) `SpAdd` are then applied between `mask` and each $\texttt{SpG}[u,:]$, $\forall u \in Q$ following by the reduction '-1', which explicitly adds missing values (all zeros at default) to structural features $\mathcal{Z}_{u,x}$ for all x if $x \notin \mathcal{S}_u$ while $x \in \mathcal{S}_v$, for some $v \in Q \backslash \{u\}$. (3) When the secondary index `SFptr` D is enabled, the results of `SpJoin` consist of concatenated pointers of `SFptr`, which

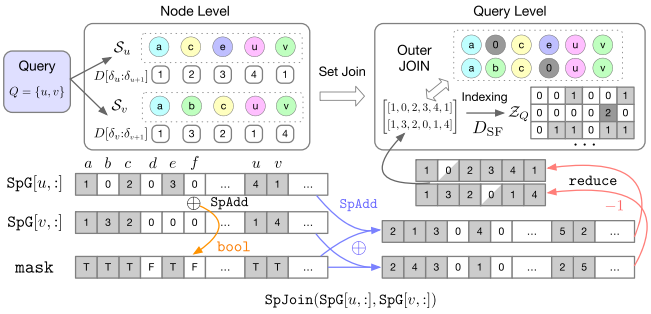


Figure 5: An illustration of joining structural features from node-level to query-level (Eq. (1)) via the `SpJoin` operator on node sets stored in `SpG`. Note that `SpG` objects do not physically carry 0 as above shown in $\texttt{SpG}[u,:]$ and $\texttt{SpG}[v,:]$. Only non-zero elements in grey blocks are stored in `SpG` and performed arithmetic operations by `SpJoin`. The half-grey blocks correspond to added missing values.

can be used to obtain joined structural features $\mathcal{Z}_Q$ by indexing to gather their values from the array D_{SF}.

To optimize performance, multithreading is employed to leverage the single program multiple data pattern in these sparse operations. Since the processing time of each query linearly depends on the size of $\mathcal{S}_Q$, we further divide queries of each training batch into groups with nearly balanced sums of $|\mathcal{S}_Q|$'s, and assign one thread process a group to mitigate potential delays caused by uneven workloads.

Comparison with SUREL [47] Despite SUREL adopting hash-based join operators instead to build query-level structural features, its overall computation and memory consumption are much higher compared to SUREL+. This is due to the presence of numerous repeated nodes in walks (over half as Fig. 3 (b) shown). The set-based input of SUREL+ substantially reduces the workload of transferring data from CPU to GPU and also requires fewer per-node operations on GPU to process transmitted $\mathcal{Z}_Q$'s. As Table 1 shows, SUREL+ reduces time complexity from $O(mM)$ to $O(K)$, where $K < mM$ is the average size of sampled node sets. These advantages ultimately boost SUREL+ for better efficiency and scalability.

3.4 Set Neural Encoders

After joining sampled node sets for each query Q, the resulting $(\mathcal{S}_Q, \mathcal{Z}_Q)$ is utilized as a proxy for the query-induced subgraph $\mathcal{G}_Q$ and then fed into neural encoders for prediction. The mini-batch training procedure of multiple Q's is summarized in Algorithm 1. Next, we introduce the neural encoders supported by SUREL+.

The adopted neural encoders are simple. For each $(\mathcal{S}_Q, \mathcal{Z}_Q)$,

$$h_Q = \texttt{AGGR}\left(\{enc(\mathcal{Z}_{Q,x}) | x \in \mathcal{S}_Q\}\right) \in \mathbb{R}^d. \tag{2}$$

Here, $enc(\cdot)$ encodes query-level structural features $\mathcal{Z}_{Q,x}$ using a multi-linear perception (MLP). If node attributes are present, they can be appended to the structural features as $\mathcal{Z}_{Q,u}||X_u$. `AGGR` is used to aggregate the encoded features, which can be any neural encoders applied to sets such as mean/sum/max pooling or set transformers. Currently, SUREL+ supports the implementations of `AGGR` in mean pooling, LSTM [14], and attention [40]. Note that the LSTM applies random permutations to the elements in the set before encoding them as a sequence; while the attention first computes soft attention scores based on the output of $enc(\cdot)$ for each set element and then performs attention-score-weighted pooling. Sec.

Algorithm 1: The mini-batch training pipeline of SUREL+

Input: Given a graph $\mathcal{G}(\mathcal{V}, \mathcal{E}, X)$, a group of queries $\{(Q, y_Q)\}$ for training, batch size B, a set SAMPLER, a structure ENCODER, and a set AGGR

Output: A neural network for encoding subgraphs $enc(\cdot)$

1 Preprocessing: SAMPLER and ENCODER $\rightarrow (\mathcal{S}_u, \mathcal{Z}_u)$ for all $u \in \mathcal{V}$; convert and save $(\mathcal{S}_u, \mathcal{Z}_u)$'s as SpG objects.
2 **for** *each mini-batch* $\mathcal{Q}_B = \{..., Q, ...\}$ **do**
3 Generate negative training queries (if not given) $\{\bar{Q}_i\}$ by random sampling and put them into $\mathcal{Q}_B$;
4 Call SpJoin operator to perform joining on SpG objects $\{(\mathcal{S}_u, \mathcal{Z}_u) | u \in Q\}$ for all queries $Q \in \mathcal{Q}_B$ in parallel;
5 Encode the joined results $(\mathcal{S}_Q, \mathcal{Z}_Q)$ as proxy of subgraphs via Eq. (2) with specified AGGR and get the prediction $\hat{y}_Q$ from readout h_Q by multithreads;
6 Backward propagation based on the loss $\mathcal{L}(\hat{y}_Q, y_Q)$.
7 **end**

4.4 demonstrates empirically that the choice of AGGR has non-trivial effects on prediction performance. Lastly, a fully-connected layer takes the readout h_Q as input to make the final prediction $\hat{y}_Q$. In our experiments, all SGRL tasks are formulated as binary classification, and thus Binary Cross Entropy is used as the loss function $\mathcal{L}$.

4 EVALUATION

In this section, we aim to evaluate the following questions:

- Regarding space and time complexity, how much improvement can SUREL+ achieve by adopting sets instead of walks compared to the SOTA SGRL framework SUREL?
- Can SUREL+ provide prediction performance comparable to all baselines using or not using SGRL methods?
- How sensitive is SUREL+ to different choices of set samplers, structure encoders, and set neural encoders?
- How do sparse storage SpG and parallelism in SpJoin operator benefit the runtime and scaling performance of SUREL+?

4.1 Experiment Setup

Extensive experiments have been performed to evaluate SUREL+ using homogeneous, heterogeneous, and higher-order homogeneous graphs on three types of tasks: link prediction, relation type prediction, and higher-order pattern prediction. A homogeneous graph is a graph that does not include node/link types, while a heterogeneous graph includes various node/link types. Higher-order graphs are hypergraphs in our setting that contain hyperedges connecting two or more nodes.

Datasets Table 2 summarizes the statistics of the datasets used to benchmark SGRL methods. Five datasets are selected from the Open Graph Benchmark (OGB, [17]) for link and relation type prediction, including social networks of citation - citation2 and collaboration - collab; biological network of protein interaction - ppa and vascular - vessel; and one heterogeneous academic network ogb-mag, which contains node types of paper (P), author (A) and their relations extracted from MAG [42]. The vessel dataset has unique significance as a very recent large (>3M nodes), sparse, biological graph extracted from mouse brains [33] for examining GRL in scientific discovery. Two hypergraph datasets collected by [3] are used for higher-order pattern prediction: DBLP-coauthor is a temporal hypergraph, where each hyperedge denotes a time-stamped paper connecting all its authors. tags-math contains sets of tags applied to questions on the website math.stackexchange.com, represented as hyperedges. For higher-order pattern prediction tasks, the number of hyperedges is the main computation bottleneck, in which one may connect more than two nodes. Two industrial graphs, criteo-click [11] with *16.5M* records of online banner ads clicking and twitter [25] with *1.5B* following relations of users are used to examine the model scalability for real-world applications.

Table 2: Summary Statistics for Evaluation Datasets.

Dataset	Type	#Nodes	#Edges	Split(%)
criteo-click	Homo./Bipartite	Campaign: 675 User: 6,142,256	16,468,027	97/1.5/1.5
twitter	Homo./Social.	41,652,230	1,468,364,884	99.98/0.01/0.01
citation2	Homo./Social.	2,927,963	30,561,187	98/1/1
collab	Homo./Social.	235,868	1,285,465	92/4/4
ppa	Homo./Bio.	576,289	30,326,273	70/20/10
vessel	Homo./Bio.	3,538,495	5,345,897	80/10/10
ogb-mag	Hetero.	(P): 736,389 (A): 1,134,649	P-A: 7,145,660 P-P: 5,416,271	99/0.5/0.5
tags-math	Higher.	1,629	projected: 91,685 hyperedges: 822,059	60/20/20
DBLP-coauthor	Higher.	1,924,991	projected: 7,904,336 hyperedges: 3,700,067	60/20/20

Settings For link prediction, OGB's standard data split is used to isolate validation and test links (queries) from the input graph. For prediction tasks of relation type and higher-order pattern, the same procedure to prepare graph data is adopted as in SUREL [47]: the relations of paper-author (P-A, "written by") and paper-paper (P-P, "cited by") are selected; higher-order queries in hypergraph datasets are node triplets, where the goal is to predict whether it will foster in a hyperedge given two of them have observed pairwise connections; to learn the representation on hypergraphs, we project hyperedges into cliques and treat the projection results as ordinary graphs. All experiments are run 10 times independently, and we report the mean performance and standard deviation.

Baselines We consider two classes of baselines. *Canonical GNNs*: GCN [22], GraphSAGE [14], GraphSAINT [49], and their variants with the prefix 'H*' that are directly applied for heterogeneous graphs with node types and for hypergraphs through clique expansion. R-GCN [35] performs relational message passing on heterogeneous graphs; SGRL models: SEAL [50, 52], GDGNN [24], and SUREL [47]. SEAL adopts online subgraph sampling due to huge memory demands for offline subgraph extraction. Fig. 3 (a) compares the time cost for subgraph sampling across different SGRL methods. We adopt the official implementations of all baselines with tuned parameters that match their reported results.

Hyperparameters By default, SUREL+ uses the walk-based sampler, the structural encoder LP and the better set neural encoder tuned between mean pooling and attention. SUREL+ adopts a 2-layer MLP as $enc(\cdot)$ in Eq. (2) followed by a 2-layer classifier to map set-aggregated representations for final predictions. Default training hyperparameters are: learning rate lr=1e-3 with the early stopping of 5 epochs, dropout p=0.1, Adam [21] as the optimizer. Analysis of parameters M and m to control the walk-based sampler and K to control the metric-based sampler and selection of structure encoders and set neural encoders are studied in Sec. 4.4.

Evaluation Metrics The evaluation metrics include Hits@P, Mean Reciprocal Rank (MRR), and Area Under Curve (ROC-AUC).

Table 3: Prediction Performance for Links, Relation Types and Higher-Order Patterns: the best (bold) and the second best (underlined).

Models	citation2 MRR (%)	click MRR (%)	twitter MRR (%)	collab Hits@50 (%)	ppa Hits@100 (%)	vessel ROC-AUC	Models	MAG(P-A) MRR (%)	MAG(P-P) MRR (%)	tags-math MRR (%)	DBLP-coauthor MRR (%)
GCN	84.74±0.21	5.31±0.17	OOM	44.75±1.07	18.67±1.32	43.53±9.61	**H*GCN**	39.43±0.29	57.43±0.30	51.64±0.27	37.95±2.59
GraphSAINT	79.85±0.40	2.86±0.63	4.12±0.73	53.12±0.52	3.83±1.33	47.14±6.83	**H*SAGE**	25.35±1.49	60.54±1.60	54.68±2.03	22.91±0.94
GDGNN	86.96±0.28	13.30±0.45	49.86±0.39	54.74±0.48	45.92±2.14	75.84±0.08	**R-GCN**	37.10±1.05	56.82±4.71	-	-
SEAL	87.67±0.32	OOM	OOM	63.64±0.71	48.80±3.16	80.50±0.21	**SUREL**	45.33±2.94	**82.47±0.26**	71.86±2.15	97.66±2.89
SUREL	**89.74±0.18**	40.39±0.61	OOM	63.34±0.52	53.23±1.03	**86.16±0.39**	**SUREL+**	**58.81±0.42**	80.45±0.13	**77.73±0.16**	**99.83±0.02**
SUREL+	88.90±0.06	**60.87±0.15**	**55.67±0.67**	**64.10±1.06**	**54.32±0.44**	85.73±0.88	/	/	/	/	/

Table 4: Breakdown of Runtime, Memory Consumption for Different Models on criteo-click, twitter, citation2 and ppa. The column Train records the runtime per 10K queries.

Models	Runtime (s)			Memory (GB)		Runtime (s)			Memory (GB)		Runtime (s)			Memory (GB)		Runtime (s)			Memory (GB)	
	Prep.	Train	Inf.	RAM	SDRAM	Prep.	Train	Inf.	RAM	SDRAM	Prep.	Train	Inf.	RAM	SDRAM	Prep.	Train	Inf.	RAM	SDRAM
Dataset	criteo-click					twitter					citation2					ppa				
GCN	3	0.085	8	3.1	62.74	-	-	-	-	OOM	17	21.74	105	9.3	36.84	2	0.026	1.2	4.6	11.35
GraphSAINT	1	0.012	20	13.1	8.79	111	0.009	920	253	76.60	151	1.79	107	9.6	9.78	10	0.003	1.5	4.9	23.06
GDGNN	215	1.43	2,928	16.2	23.77	1204	1.84	9,744	188	79.34	338	2.26	5,460	40.6	16.96	127	1.77	902	21.1	10.27
SEAL	-	-	-	OOM	-	-	-	-	OOM	-	46	3.52	24,626	35.4	5.71	46	10.57	3,988	9.5	12.13
SUREL	2	1.59	2,307	11.7	16.25	-	-	-	OOM	-	151	4.14	6,081	25.1	9.68	31	2.68	1,429	13.6	31.01
SUREL+	22	0.23	502	10.4	11.93	327	0.26	3,779	210	49.44	130	0.35	1,389	16.7	4.75	69	0.72	201	9.8	19.02

Hit@P counts the ratio of positive samples ranked at the top-P place against negative ones. MRR first computes the inverse of the rank of the first correct prediction and then takes the average of obtained reciprocal ranks for a sample of queries. For all datasets adopting MRR, each positive query is paired with 1000 randomly sampled negative test queries, except tags-math using 100 and crieo-click using 650. ROC-AUC follows the standard definition to measure the model's performance in binary classification.

Environment We use a server with two Intel Xeon Gold 6248R CPUs, 512 GB DRAM, and NVIDIA A100 (80GB) GPU. SUREL+ is built on PyTorch 1.12 and PyG 2.2. Set samplers are implemented in C, OpenMP, NumPy, Numba, and uhash, integrated into Python scripts; SpG is customized based on the CSR format of Scipy.

4.2 Prediction Accuracy Comparison

Table 3 shows the prediction performance of different methods. SGRL models significantly outperform canonical GNNs on these six link prediction benchmarks, especially on two challenging biological datasets ppa and vessel. Link prediction in biological datasets relies on richer structural information that canonical GNNs have limited expressive power to capture. Within SGRL models, SUREL+ achieves comparable performance to SUREL and outperforms SEAL, which validates the effectiveness of our proposed set-based representations. For predictions of relation type and higher-order pattern, we observe additional performance gains (+2∼13%) from SUREL+ compared to SUREL on three of the four datasets. A large performance gap exists between canonical GNNs and SGRL models, particularly in the higher-order case. This demonstrates the inherent limitations of canonical GNNs to make predictions of complex relations involving multiple nodes.

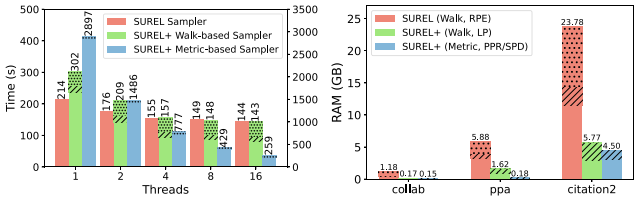
4.3 Efficiency and Scalability Analysis

Improved Efficiency in Training and Inference. Table 4 compares model runtime and memory usage on the four largest benchmarks. SUREL+ offers a reasonable training time compared with canonical GNNs. It shows clear improvement in inference compared to the current SOTA framework SUREL (3-11× speedups across all datasets) and its predecessor SEAL (∼20× speedups). SUREL+ achieves comparable and even lower RAM usage than canonical GNNs. Compared to other SGRL models, it can save up to half of RAM with lower usage of GPU DRAM, since set-based subgraphs eliminate node duplicates and their associated structural features. Tables 1 and 4 analytically and empirically show that the key factor of SUREL+ scaling up to billion-size graphs is its set-based representation with the sparse design, while GCN (full adjacency matrix), SEAL (complex subgraph extraction) and SUREL (dense walks with duplicate nodes) are all out of memory (OOM) on twitter.

Profiling Different Strategies for Offline Processing Fig. 6a reports the time cost of different samplers with multithreading on citation2. Fig. 6b shows memory consumption to store different types of sampled data (walks in SUREL [47] or sets in SUREL+) and their associated structural features (LPs, SPDs, PPR scores). Compared to SUREL sampler[47], the walk-based sampler in SUREL+ is more scalable and only adds an extra minute for encoding and converting data in SpG format (slash/dash marked in Fig. 6a), while achieving 6.94×, 3.63× and 4.12× memory savings on three OGB datasets respectively to store sampled sets and their structural features. Those memory savings are more crucial for overall scalability as they reduce the workload of data transfer from CPU to GPU and save many GPU operations to encode the data, which dominates the time cost of the online stage. This leads to the efficiency improvement of SUREL+ in Table 4. In addition, the metric-based sampler that adopts PPR scores has better scaling performance when using more threads. When adopting PPR scores or SPDs as structural features, SUREL+ further reduces the memory cost, though they often harm prediction performance slightly.

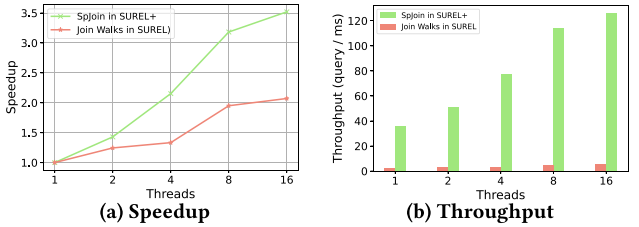
Note that in the above comparison of memory cost, compressed structural features are adopted both in SUREL (locally) and SUREL+ (globally), i.e., the secondary index based on SFptr D and D_{SF}, which achieve compression of 493×, 11318×, 19527× on three datasets listed in Fig. 6b, when one adopts LPs as structural features.

Scaling Analysis for SpJoin Fig. 7 shows the speedups and throughput of the sparse join operator SpJoin via multithreading, where the join operation in SUREL to construct query-level structural features for nodes on walks [47] is used as a comparison.



(a) Runtime (b) Memory

Figure 6: Comparison of Runtime, Memory Consumption across Different Offline Processing Strategies (the walk-based sampler: $m = 4, M = 200$, the metric-based sampler: $K = 150$). The areas highlighted break down the total consumption w.r.t. (a) sampling, structure encoding, sparse object construction; (b) structural features, node indices/pointers, and sampled walks (SUREL sampler only).



(a) Speedup (b) Throughput

Figure 7: Scaling Performance Comparison of `SpJoin` in SUREL+ (with average set size $|\tilde{S_u}| = 351$) and Join Walks in SUREL (with walk size $m = 4, M = 200$) against the Numbers of Threads.

SUREL uses a hash-based search for walk joins, which has unfavorable memory access patterns and suffers from workload imbalance due to inconsistent searching times in hash tables among different threads. SUREL+ earns more benefits from multithreading, thanks to the use of `SpJoin` and batch-wise load balancing.

4.4 Comparison between Different Set Samplers, Structural Features and Set Neural Encoders

SUREL+ is a modularized framework that supports different set samplers (walk- and metric-based), structural features (LP, SPD, PPR), and set neural encoders `AGGR` (mean pooling, LSTM, attention).

Table 5 shows the prediction performance and inference runtime by adopting different combinations of structure encoders and set neural encoders. Landing probabilities (LPs) as structural features perform the best on all three OGB datasets while being the slowest for inference. By recording the landing probabilities over different steps of walks, LPs provide structural information in finer granularity than both SPDs and PPR scores which are just scalars. Also, it might be due to the adopted link prediction task, which favors more local information held by LPs and SPDs than global information carried by PPR scores. The authors conjecture that other tasks that rely on more global information may favor PPR scores. In comparison, there is no set neural encoder as always a winner. Attention seems to perform the best on average while slower than mean pooling. LSTM is the slowest. On the two social networks (`citation2` and `collab`), mean pooling can provide comparable prediction results with much fewer parameters. However, prediction on the biological network (`ppa`) requires more expressive and complicated encoders, where LSTM and attention are favored as they can model more complex interactions between nodes in $\mathcal{S}_Q$.

Fig. 8 compares prediction results and inference time by using different hyperparameters m, M, and K of set samplers, which heavily

Table 5: Prediction Performance and Inference Runtime of SUREL+ with Different Combinations of Structure Features (LP, SPD, PPR) and Set Neural Encoders (Mean, LSTM, Attn.). The best and the second best are highlighted in bold and underlined accordingly.

Dataset	PPR+Mean	SPD+Mean	LP+Mean	LP+LSTM	LP+Attention
`citation2`	78.59±0.38	87.99±1.07	<u>88.55±0.15</u>	88.46±0.34	**88.90±0.06**
	834s	<u>1057s</u>	1389s	3678s	2171s
`collab`	47.15±0.21	62.11±0.13	**64.10±1.06**	61.31±1.37	<u>62.85±1.19</u>
	1.4s	<u>1.7s</u>	2.0s	3.5s	2.3s
`ppa`	13.28±1.20	41.06±1.70	46.41±1.65	**54.45±1.35**	<u>54.32±0.44</u>
	63s	<u>126s</u>	165s	322s	201s

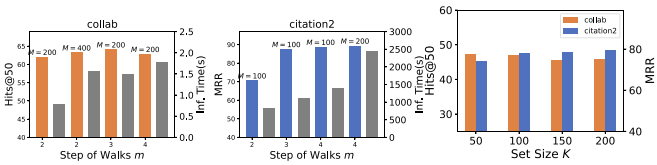


Figure 8: Hyperparameter Analysis of Set Samplers (Performance v.s. Time Cost). Walk-based: the number M and the step m of walks, LPs as structural features; Metric-based: the set size K, PPR scores as structural features.

affects the coverage of sampled neighborhoods and the computation workload. The performance consistently increases if the walk-based sampler uses a larger M (high granularity of local structures) but is not guaranteed for a larger m (border exploration). Better coverage with a larger K is generally beneficial for the metric-based sampler over `citation2` but not for `collab`, which is due to different characteristics of these two datasets and is also observed by [47]. In general, small sampling parameters m ($2 \sim 4$), M ($100 \sim 400$) and K ($50 \sim 200$) with fast inference can yield satisfactory performance that achieves the trade-off between accuracy and scalability.

5 CONCLUSION

In this work, we propose a novel framework SUREL+ for scalable subgraph-based graph representation learning. SUREL+ avoids costly query-specific subgraph extraction by decoupling it into sampled node sets whose join can be used as a proxy of query-induced subgraphs for prediction. SUREL+ benefits from the reusability of presampled node sets across different queries. Compared to the current SOTA framework SUREL, the set-based representation of SUREL+ substantially reduces memory and time consumption by avoiding heavy node duplicates in sampled walks. To handle irregularly sized node sets, SUREL+ designs a dedicated sparse storage `SpG` and a sparse join operator `SpJoin`, providing memory-efficient storage and rapid access. In addition, SUREL+ adopts a modular design, which enables users to flexibly choose different set samplers, structure encoders, and set neural encoders based on the nature of their SGRL tasks. Extensive experiments on three types of prediction tasks over nine real-world graph benchmarks show that SUREL+ significantly improves scalability, memory efficiency, and prediction accuracy compared to current SGRL methods and canonical GNNs.

ACKNOWLEDGMENTS

The authors would like to thank Rongzhe Wei and Yanbang Wang for their helpful discussions and valuable feedback. Haoteng Yin and Pan Li are supported by the 2021 JPMorgan Faculty Award, NSF awards OAC-2117997, IIS-2239565.

REFERENCES

[1] Emily Alsentzer, Samuel Finlayson, Michelle Li, and Marinka Zitnik. 2020. Subgraph neural networks. *Advances in Neural Information Processing Systems* 33 (2020), 8017–8029.

[2] Reid Andersen, Fan Chung, and Kevin Lang. 2006. Local graph partitioning using pagerank vectors. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*. IEEE, 475–486.

[3] Austin R Benson, Rediet Abebe, Michael T Schaub, Ali Jadbabaie, and Jon Kleinberg. 2018. Simplicial closure and higher-order link prediction. *Proceedings of the National Academy of Sciences* 115, 48 (2018), E11221–E11230.

[4] Aleksandar Bojchevski, Johannes Klicpera, Bryan Perozzi, Amol Kapoor, Martin Blais, Benedek Rózemberczki, Michal Lukasik, and Stephan Günnemann. 2020. Scaling graph neural networks with approximate pagerank. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2464–2473.

[5] Giorgos Bouritsas, Fabrizio Frasca, Stefanos P Zafeiriou, and Michael Bronstein. 2022. Improving graph neural network expressivity via subgraph isomorphism counting. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2022).

[6] Lei Cai, Zhengzhang Chen, Chen Luo, Jiaping Gui, Jingchao Ni, Ding Li, and Haifeng Chen. 2021. Structural temporal graph neural networks for anomaly detection in dynamic graphs. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 3747–3756.

[7] Benjamin Paul Chamberlain, Sergey Shirobokov, Emanuele Rossi, Fabrizio Frasca, Thomas Markovich, Nils Hammerla, Michael M Bronstein, and Max Hansmire. 2022. Graph Neural Networks for Link Prediction with Subgraph Sketching. *arXiv preprint arXiv:2209.15486* (2022).

[8] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. Fastgcn: fast learning with graph convolutional networks via importance sampling. *arXiv preprint arXiv:1801.10247* (2018).

[9] Zhengdao Chen, Lei Chen, Soledad Villar, and Joan Bruna. 2020. Can graph neural networks count substructures? *Advances in Neural Information Processing Systems* 33 (2020), 10383–10395.

[10] DGL. 2022. 6.7 Using GPU for Neighborhood Sampling — DGL 0.9.1post1 documentation. https://docs.dgl.ai/guide/minibatch-gpu-sampling.html

[11] Diemert Eustache, Meynet Julien, Pierre Galland, and Damien Lefortier. 2017. Attribution Modeling Increases Efficiency of Bidding in Display Advertising. In *Proceedings of the AdKDD and TargetAd Workshop, KDD, Halifax, NS, Canada, August, 14, 2017*. ACM, To appear.

[12] Fabrizio Frasca, Beatrice Bevilacqua, Michael M Bronstein, and Haggai Maron. 2022. Understanding and Extending Subgraph GNNs by Rethinking Their Symmetries. *Advances in Neural Information Processing Systems* 35 (2022).

[13] Vikas Garg, Stefanie Jegelka, and Tommi Jaakkola. 2020. Generalization and representational limits of graph neural networks. In *International Conference on Machine Learning*. PMLR, 3419–3430.

[14] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. *Advances in Neural Information Processing Systems* 30 (2017), 1025–1035.

[15] William L Hamilton. 2020. Graph representation learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning* 14, 3 (2020), 1–159.

[16] William L. Hamilton, Rex Ying, and Jure Leskovec. 2017. Representation Learning on Graphs: Methods and Applications. *IEEE Data Eng. Bull.* 40, 3 (2017), 52–74.

[17] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2020. Open graph benchmark: Datasets for machine learning on graphs. *Advances in Neural Information Processing Systems* 33 (2020), 22118–22133.

[18] Kexin Huang and Marinka Zitnik. 2020. Graph meta learning via local subgraphs. *Advances in Neural Information Processing Systems* 33 (2020), 5862–5874.

[19] Glen Jeh and Jennifer Widom. 2003. Scaling personalized web search. In *Proceedings of the 12th International Conference on World Wide Web*. 271–279.

[20] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. 2021. Highly accurate protein structure prediction with AlphaFold. *Nature* 596, 7873 (2021), 583–589.

[21] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*.

[22] Thomas N Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*.

[23] Daphne Koller, Nir Friedman, Sašo Džeroski, Charles Sutton, Andrew McCallum, Avi Pfeffer, Pieter Abbeel, Ming-Fai Wong, Chris Meek, Jennifer Neville, et al. 2007. *Introduction to statistical relational learning*. MIT press.

[24] Lecheng Kong, Yixin Chen, and Muhan Zhang. 2022. Geodesic Graph Neural Network for Efficient Graph Representation Learning. *Advances in Neural Information Processing Systems* 35 (2022).

[25] Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon. 2010. What is Twitter, a social network or a news media?. In *Proceedings of the 19th international conference on World wide web*. 591–600.

[26] Pan Li, I Chien, and Olgica Milenkovic. 2019. Optimizing generalized pagerank methods for seed-expansion community detection. *Advances in Neural Information Processing Systems* 32 (2019), 11710–11721.

[27] Pan Li, Yanbang Wang, Hongwei Wang, and Jure Leskovec. 2020. Distance Encoding: Design Provably More Powerful Neural Networks for Graph Representation Learning. *Advances in Neural Information Processing Systems* 33 (2020), 4465–4478.

[28] Xin Liu, Haojie Pan, Mutian He, Yangqiu Song, Xin Jiang, and Lifeng Shang. 2020. Neural subgraph isomorphism counting. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1959–1969.

[29] Yunyu Liu, Jianzhu Ma, and Pan Li. 2022. Neural Predicting Higher-Order Patterns in Temporal Networks. In *Proceedings of the Web Conference 2022*. ACM, 1340–1351.

[30] Zhaoyu Lou, Jiaxuan You, Chengtao Wen, Arquimedes Canedo, Jure Leskovec, et al. 2020. Neural Subgraph Matching. *arXiv preprint arXiv:2007.03092* (2020).

[31] Yuhong Luo and Pan Li. 2022. Neighborhood-aware Scalable Temporal Network Representation Learning. *Learning on Graphs Conference* (2022).

[32] Changping Meng, S Chandra Mouli, Bruno Ribeiro, and Jennifer Neville. 2018. Subgraph pattern neural networks for high-order graph evolution prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32.

[33] Johannes C Paetzold, Julian McGinnis, Suprosanna Shit, Ivan Ezhov, Paul Büschl, Chinmay Prabhakar, Anjany Sekuboyina, Mihail Todorov, Georgios Kaissis, Ali Ertürk, et al. 2021. Whole Brain Vessel Graphs: A Dataset and Benchmark for Graph Learning and Neuroscience. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

[34] PyG. 2022. Accelerating PyG on NVIDIA GPUs. https://www.pyg.org//ns-newsarticle-accelerating-pyg-on-nvidia-gpus

[35] 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 *European semantic web conference*. Springer, 593–607.

[36] Yili Shen, Jiaxu Yan, Cheng-Wei Ju, Jun Yi, Zhou Lin, and Hui Guan. 2022. Improving Subgraph Representation Learning via Multi-View Augmentation. *arXiv preprint arXiv:2205.13038* (2022).

[37] Balasubramaniam Srinivasan and Bruno Ribeiro. 2020. On the equivalence between positional node embeddings and structural graph representations. In *International Conference on Learning Representations*.

[38] Balasubramaniam Srinivasan, Da Zheng, and George Karypis. 2021. Learning over Families of Sets-Hypergraph Representation Learning for Higher Order Tasks. In *Proceedings of the 2021 SIAM International Conference on Data Mining (SDM)*. SIAM, 756–764.

[39] Komal Teru, Etienne Denis, and Will Hamilton. 2020. Inductive relation prediction by subgraph reasoning. In *International Conference on Machine Learning*. PMLR, 9448–9457.

[40] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2018. Graph attention networks. In *International Conference on Learning Representations*.

[41] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. 2020. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods* 17 (2020), 261–272.

[42] Kuansan Wang, Zhihong Shen, Chiyuan Huang, Chieh-Han Wu, Yuxiao Dong, and Anshul Kanakia. 2020. Microsoft academic graph: When experts are not enough. *Quantitative Science Studies* 1, 1 (2020), 396–413.

[43] Xiyuan Wang and Muhan Zhang. 2021. GLASS: GNN with Labeling Tricks for Subgraph Representation Learning. In *International Conference on Learning Representations*.

[44] Yanbang Wang, Yen-Yu Chang, Yunyu Liu, Jure Leskovec, and Pan Li. 2021. Inductive Representation Learning in Temporal Networks via Causal Anonymous Walks. In *International Conference on Learning Representations*.

[45] Lingfei Wu, Peng Cui, Jian Pei, Liang Zhao, and Xiaojie Guo. 2022. Graph neural networks: foundation, frontiers and applications. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 4840–4841.

[46] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How Powerful are Graph Neural Networks?. In *International Conference on Learning Representations*.

[47] Haoteng Yin, Muhan Zhang, Yanbang Wang, Jianguo Wang, and Pan Li. 2022. Algorithm and System Co-design for Efficient Subgraph-based Graph Representation Learning. *Proceedings of the VLDB Endowment* 15, 11 (2022), 2788–2796.

[48] Hanqing Zeng, Muhan Zhang, Yinglong Xia, Ajitesh Srivastava, Andrey Malevich, Rajgopal Kannan, Viktor Prasanna, Long Jin, and Ren Chen. 2021. Decoupling the depth and scope of graph neural networks. *Advances in Neural Information*

Processing Systems 34 (2021), 19665–19679.

[49] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. 2020. Graphsaint: Graph sampling based inductive learning method. In *International Conference on Learning Representations*.

[50] Muhan Zhang and Yixin Chen. 2018. Link prediction based on graph neural networks. *Advances in Neural Information Processing Systems* 31 (2018), 5165–5175.

[51] Muhan Zhang and Yixin Chen. 2020. Inductive Matrix Completion Based on Graph Neural Networks. In *International Conference on Learning Representations*.

[52] Muhan Zhang, Pan Li, Yinglong Xia, Kai Wang, and Long Jin. 2021. Labeling Trick: A Theory of Using Graph Neural Networks for Multi-Node Representation Learning. *Advances in Neural Information Processing Systems* 34 (2021), 9061–9073.

[53] Hongkuan Zhou, Da Zheng, Israt Nisa, Vasileios Ioannidis, Xiang Song, and George Karypis. 2022. TGL: A General Framework for Temporal GNN Training on Billion-Scale Graphs. *Proceedings of the VLDB Endowment* 15, 8 (2022), 1572–1580.