# Subgraph Contrastive Link Representation Learning

A Preprint

**Jiajun Zhou**
Zhejiang University of Technology
Hangzhou, China
jjzhou@zjut.edu.cn

**Qi Xuan**
Zhejiang University of Technology
Hangzhou, China
xuanqi@zjut.edu.cn

December 3, 2021

## Abstract

Graph representation learning (GRL) has emerged as a powerful technique for solving graph analytics tasks. It can effectively convert discrete graph data into a low-dimensional space where the graph structural information and graph properties are maximumly preserved. While there is rich literature on node and whole-graph representation learning, GRL for link is relatively less studied and less understood. One common practice in previous works is to generate link representations by directly aggregating the representations of their incident nodes, which is not capable of capturing effective link features. Moreover, common GRL methods usually rely on full-graph training, suffering from poor scalability and high resource consumption on large-scale graphs. In this paper, we design Subgraph Contrastive Link Representation Learning (SCLRL) — a self-supervised link embedding framework, which utilizes the strong correlation between central links and their neighborhood subgraphs to characterize links. We extract the "link-centric induced subgraphs" as input, with a subgraph-level contrastive discrimination as pretext task, to learn the intrinsic and structural link features via subgraph mini-batch training. Extensive experiments conducted on five datasets demonstrate that SCLRL has significant performance advantages in link representation learning on benchmark datasets and prominent efficiency advantages in terms of training speed and memory consumption on large-scale graphs, when compared with existing link representation learning methods.

## Introduction

Thanks to the widespread application of graphs in many real scenarios, Graph Representation Learning (GRL) has attracted considerable attention recently. The main goal of GRL is to convert discrete graph structure into a low-dimensional space, in which the vital structural information and properties are maximumly preserved as continuous vector representations, known as *embeddings*. Different types of embeddings could be task-driven and facilitate different downstream applications. For instance, node embedding can potenitally benefit a wide variety of node-level analytics tasks such as node classification [1] and node clustering [2], and whole-graph embedding serves the graph-level tasks such as graph classification [3]. So far, GRL has led to advances in multiple successful applications across different domains such as social networks [4], molecular graphs [5], knowledge graphs [6], etc.

While there is rich literature on node and whole-graph representation learning, GRL for link is relatively less studied and less understood. One common practice in previous works [7, 8] is two-stage, which obtains a link representation via aggregateing the representations of its incident nodes generated in the previous phase. For instance, Node2Vec [7] extends random walk to node pairs and uses a bootstrapping approach over the individual node embeddings to obtain link representation. We refer to such two-stage link embedding mechanism as "node2link" pattern which suffers from several drawbacks. First, the indirect generation of link representations ignores the fact that different types of objects (e.g., nodes, links) do not share the same embedding space. Actually, node embedding methods mainly focus on the nodes and their properties (e.g. the node proximity), while link embedding methods should pay more attention to links (or pairwise nodes), and explicitly capture the structural interactions between nodes. Thus, the "node2link" pattern cannot preserve the complete properties of links, resulting in a failure to achieve the best performance in the downstream edge analytics tasks such as link prediction. Second, commonly used GRL methods usually suffer from poor scalability and high resource consumption on large-scale graphs. For instance, random walks embedding techniques such as DeepWalk [4] and Node2Vec [7] rely on a large amount of CPU kernels and time to generate a sufficient number of walks. Deep GRL methods such as graph convolutional network (GCN) and graph attention network (GAT) generally accept the entire graphs as input and perform full-graph training to learn graph representations, which restricts their scalability.

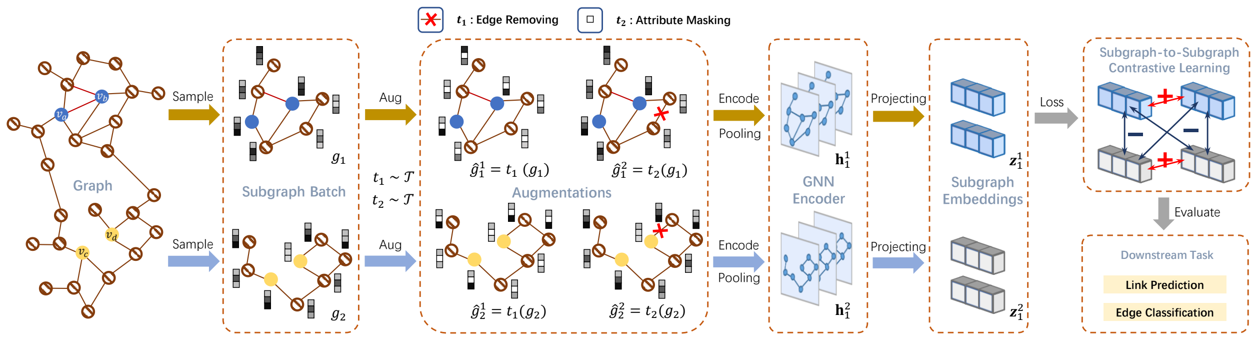

Figure 1: The architecture of SCLRL. The complete workflow proceeds as follows: 1) extracting and sampling subgraph centered on target links to form input batchs; 2) applying two augmentation operators on each subgraph to generate two correlated views; 3) feeding these augmented subgraphs into the GNN encoder and projection head to generate subgraph embeddings as link representations; 4) maximizing the consistency between two augmented views of subgraphs via a subgraph-level contrast.

In view of (1) the unsound expressiveness of link representations generated via "node2link" pattern and (2) the poor scalability of universal GRL techniques, an immediate question arises here: *how can we design a universal and scalable representation learning framework for highly-expressive link representations?*

Intuitively, links (interactions, relations) can reflect the correlation between nodes, and exploiting the key interaction patterns between node pairs can facilitate learning highly-expressive link representation. For two nodes of a link in graph, the message propagation between them can be achieved through both direct interaction (i.e., edge) or indirect ones (e.g., common neighbors, shortest paths), all of which can be contained in a subgraph surrounding this link. In this work, we study the "subgraph2link" pattern following the facts: (1) subgraph consisting of target link and their local neighborhood information (neighbors and their interactions) is informative and plays a critical role to provide structure contexts for link representation learning; (2) subgraph is actually the receptive field of center link, which is generally much smaller than the whole graph.

Inspired by the existing researches of structural link learning [9], unsupervised pre-training [10, 11] and scalable graph learning [12, 13], we propose Subgraph Contrastive Link Representation Learning (SCLRL) — a novel scalable self-supervised link representation learning framework via subgraph contrast, which utilizes the strong correlation between central links and their regional subgraphs. Conceptually, we leverage the idea of graph contrastive learning [10] to design the self-supervised pretext task as subgraph discrimination. More specifically, we first extract and sample the subgraphs centered on target links, and these informative subgraphs facilitate the learning of link representations. Then we introduce a series of graph augmentation strategies to generate multiple views for each subgraph, and multiple views generated from the same subgraph form positive pairs while those generated from different subgraphs form negative pairs. Subsequently, these augmented subgraphs are fed into a GNN-based encoder to obtain the subgraph representations after pooling. Finally, SCLRL uses a "subgraph-subgraph" contrast to train the encoder to distinguish positive and negative samples, so that links with similar subgraph patterns keep high consistency while those with different patterns can be well-differentiated, further achieving high-performance downstream link prediction. The major contributions of our work are summarized as follows:

- **SCLRL**: We propose a universal self-supervised link representation learning framework via subgraph contrast. It utilizes the informative local subgraphs surrounding links to learning highly-expressive link representations.
- **GraphAug**: We introduce two novel graph augmentation strategies, one of them is attribute-level augmentation named *Attribute Similarity* and the other is structure-level augmentation named *KNN Graph*, which aims to generate effective graph views for contrastive learning.

- **Scalability**: We take the receptive field subgraphs extracted from a batch of links as the input during each training step, so as to learn link representation efficiently and make our SCLRL scale well on large-scale graphs.
- **Effectiveness**: Extensive experiments demonstrate the superiority of our framework in terms of performance and efficiency on link representation learning.

## Preliminaries and Related Works

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an undirected and unweighted graph, where $\mathcal{V} = \{v_i \mid i = 1, 2, \cdots, N\}$ and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ represent the sets of nodes and edges respectively. We regard an edge in graph $(v_i, v_j) \in \mathcal{E}$ as a positive link while those nonexistent edges $(v_i, v_j) \notin \mathcal{E}$ are treated as negative links. Generally, graph structural data consist of two features: a node attribute matrix $\mathbf{X} \in \mathbb{R}^{N \times F}$ and an adjacency matrix of graph topology $\mathbf{A} \in \mathbb{R}^{N \times N}$, where $\mathbf{x}_i \in \mathbb{R}^F$ is the feature of dimension $F$ for node $v_i$, $\mathbf{A}_{ij} = 1$ if $(v_i, v_j) \in \mathcal{E}$ and 0 otherwise. We use a diagonal degree matrix $\mathbf{D} \in \mathbb{R}^{N \times N}$ to define the degree distribution of $\mathbf{A}$, and $\mathbf{D}_{ii} = \sum_{j=0}^{N} \mathbf{A}_{ij}$.

The problem we consider in this work is self-supervised link representation learning on graphs. Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$, our goal is to learn a GNN encoder $\mathbf{h} = f(\mathbf{X}, \mathbf{A})$ which maps a subgraph centered on target link to a vector representation without the presence of any labels related to link existence. We briefly summarize the related work below.

### Unsupervised Representation Learning on Graphs

**Random Walks** [4, 7] transform discrete graphs into node sequences through a stochastic process of sampling nodes along the connections in graphs, and use language models to learn node representations. Since random walks are naturally based on the connectivity structure between nodes in graphs, they can be extended to learning edge features using a bootstrapping approach over the feature representations of the individual nodes [7].

**Graph Auto-Encoders (GAE)** [8] train encoders that impose the topology proximity of nodes in the graph structure on the latent space by reconstructing adjacency information. They use a graph convolutional network (GCN) as encoder to generate latent representation of nodes and scores the existence of links between pairwise nodes by an inner product decoder. Both random walks and GAE over-emphasize the proximity information at the expense of the structural information [14].

**Graph Kernels** [15, 16] decompose graphs into sub-structures and use kernel functions to measure graph similarity between them, while **Whole-Graph Embeddings** [17, 18] embed the entire graph into a vector space where the embeddings of two graphs preserve their graph-graph proximity.

**Contrastive Methods** [10, 12, 14] are built on the idea of mutual information maximization, which learns by predicting the agreement between two augmented instances. Deep Graph Infomax (DGI) [14] and SUGCON [12] learn node representation by contrasting node and graph encodings, achieving state-of-the-art results in node classification benchmarks. GraphCL [10] learns graph-level representation by contrasting two different augmented views of original graphs, outperforming previous models on unsupervised graph classification tasks.

## Methodology

In this section, we give the details of the proposed framework SCLRL, as schematically depicted in Figure 1. Our framework is mainy composed of the following components: (1) a subgraph extractor which captures the subgraphs centered on target links from the graph topology; (2) a subgraph augmentor which generates a series of variant graph views using various transformations on attributes and topology of subgraphs; (3) a GNN-based encoder which learns graph-level representation for generated graph views; (4) a subgraph-level contrast maximizes the consistency between two augmented views of the same subgraph. Next, we describe the details of each component.

### Subgraph Extraction and Sampling

Generic GNNs like GCN and GAT generally accept the whole graph as input and rely on full-graph training, which restricts their scalability on large-scale graphs for node representation learning. Existing improved studies use layer sampling [13, 19] or subgraph sampling [20, 21] to alleviate the aforementioned limitations. Inspired by the mechanism of "neighborhood aggregation" in GNN, we know that to characterize and classify a target node, GNN actually

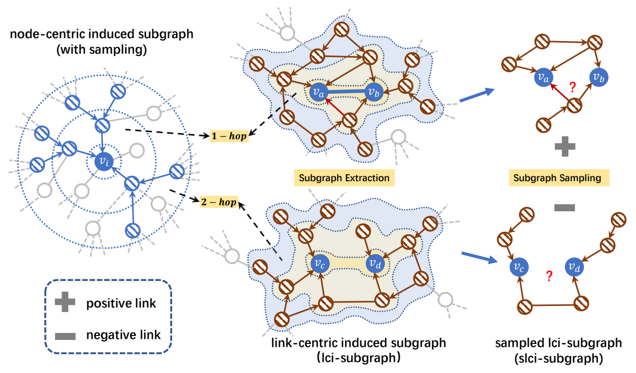

Figure 2: Schematic depiction of subgraphs. *Left*: a sampled 2-hop node-centric induced subgraph of node $v_i$. *Middle*: 2-hop link-centric induced subgraphs (lci-subgraph) of $(v_a, v_b)$ and $(v_c, v_d)$. *Right*: sampled 2-hop link-centric induced subgraphs (slci-subgraph) of which the labels reflect the existence of links between target node pairs.

aggregates the node features in the subgraph centered on it. The node-centric induced subgraph is actually the receptive field of center node, as shown in Figure 2, which is generally much smaller than the whole graph. Thus, extracting the subgraphs to form a batch as the inputs of GNNs can significantly reduce the resource consumption during model training. In this work, we consider the subgraph extraction and sampling in the scenario of link representation learning, which are similarly adopted in [9].

## Subgraph Extraction

We name the subgraph centered on link as *link-centric induced subgraph* (abbreviated "lci-subgraph" in this paper) and give the definition as follows.

**Definition 0.1. Link-centric Induced Subgraph.** For a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, given target link $l = (v_i, v_j)$ where $v_i, v_j \in \mathcal{V}$, the $h$-hop link-centric induced subgraph for link $l$ is the subgraph $g_l^h$ induced from $\mathcal{G}$ by the set of nodes $\cup_{v\in(v_i,v_j)}\{v_k \mid d(v_k, v) \leq h\}$.

The lci-subgraph $g_l^h$ of link $l = (v_i, v_j)$ contains all the $h$-hop neighbors of $v_i$ and $v_j$. Figure 2 illustrates the 2-hop lci-subgraphs for links $(v_a, v_b)$ and $(v_c, v_d)$. The informative subgraph patterns can effectively reflect the existence of links between center node pairs, which helps to characterize the link structure.

## Subgraph Sampling

Furthermore, we know that a $h$-layer GNN expands the receptive field by one-hop during each iteration and after $h$ iterations the features of nodes within $h$-hops will be aggregated. When it comes to a deeper model, the size (i.e., the number of nodes) of receptive field subgraph grows exponentially with layers, which results in neighborhood explosion problem. In this work, we use subgraph sampling to control the size of lci-subgraphs, even though it can only alleviate this problem to some extent. Specifically, for a $h$-hop lci-subgraph of link $l = (v_i, v_j)$, we select the top-$K_1$ important 1-hop neighbors (with node degree value) for $v_i$ and $v_j$ respectively, and again select the top-$K_2$ important 1-hop neighbors for each selected node at hop 1, and recursive ones in the downstream hops. The recursive sampling can be formulated as follows:

$$\mathcal{V}_t = \bigcup_{v\in\mathcal{V}_{t-1}} \mathsf{topK}(\mathcal{N}_v, K_t, \mathbf{D}[\mathcal{N}_v, \mathcal{N}_v]), \tag{1}$$

where $\mathcal{V}_t$ is the set of nodes sampled at hop $t$, $\mathcal{N}_v$ is the 1-hop neighborhs set of node $v$, $K_t$ is the sampling number at hop $t$, $\mathbf{D}[\mathcal{N}_v, \mathcal{N}_v]$ is the degree sequence of nodes in $\mathcal{N}_v$, and topK is the function that returns the nodes of top-$K$ largest degree values. $\mathcal{V}_0$ is initialized as $\{v_i, v_j\}$. After $h$ iterations, the set of nodes sampled from the $h$-hop lci-subgraph is $\mathcal{V}_l = \cup_{t=0}^{h}\mathcal{V}_t$. The sampled lci-subgraph (abbreviated "slci-subgraph" in this paper) $g_l = (\mathbf{A}_l, \mathbf{X}_l)$ is induced by $\mathcal{V}_l$, and its adjacency matrix $\mathbf{A}_l$ and feature matrix $\mathbf{X}_l$ are denoted respectively as

$$\mathbf{X}_l = \mathbf{X}[\mathcal{V}_l, :]\,, \quad \mathbf{A}_l = \mathbf{A}[\mathcal{V}_l, \mathcal{V}_l]\,. \tag{2}$$

Note that we remove the edge between $v_i$ and $v_j$ if $l = (v_i, v_j)$ is a positive link. Finally, we anonymize the slci-subgraph $g_l$ by relabeling its nodes to be $\{1, 2, \cdots, |\mathcal{V}_l|\}$, in arbitrary order. For a set of links $L_{emb}$ to be embedded, we get the slci-subgraph for each link in $L_{emb}$ and all the slci-subgraphs form a set: $\mathcal{D} = \{g_i \mid i = 1, 2, \cdots, |L_{emb}|\}$.

## Subgraph-level Contrastive Learning for Link Representation

To learn high-expressive link representations, SCLRL utilizes subgraph-level contrastive learning to train a GNN encoder $f$ which is capable of characterizing the link information in the input slci-subgraphs via a self-supervised manner. In our contrastive learning framework, for each slci-subgraph $g_i$, its two correlated views $\hat{g}_i^1$ and $\hat{g}_i^2$ are generated by undergoing two augmentation operators $t_1$ and $t_2$ sampled from the family of all operators $\mathcal{T}$, where $\hat{g}_i^1 = t_1(g_i)$ and $\hat{g}_i^2 = t_2(g_i)$. The correlated augmented views are fed into a GNN encoder $f_\theta$ with pooling layer, producing the whole subgraph representations $\mathbf{h}_i^1$ and $\mathbf{h}_i^2$, which are then mapped into an embedding space via a projection head $f_\phi$, yielding $\mathbf{z}_i^1$ and $\mathbf{z}_i^2$. Note that $\theta$ and $\phi$ are the parameters of graph encoder and projection head respectively. The representation of a slci-subgraph, $\mathbf{z}$, is treated as the representation of its central link, following a "subgraph2link" pattern. Finally, the goal of subgraph-level contrast is to maximize the consistency between two correlated augmented views of subgraphs in the embedding space via Eq. (3):

$$\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n}\mathcal{L}_i, \tag{3}$$

where $n$ is the number of subgraphs in a batch (i.e., batch size). The loss for each subgraph $\mathcal{L}_i$ can be computed as:

$$\mathcal{L}_i = -\log\frac{e^{\mathrm{s}\left(\mathbf{z}_i^1,\mathbf{z}_i^2\right)/\tau}}{\sum_{j=1,j\neq i}^{n} e^{\mathrm{s}\left(\mathbf{z}_i^1,\mathbf{z}_j^2\right)/\tau}}, \tag{4}$$

where $\mathrm{s}(\cdot,\cdot)$ is the cosine similarity function having $\mathrm{s}(\mathbf{z}_i^1,\mathbf{z}_i^2) = \mathbf{z}_i^{1\top}\cdot\mathbf{z}_i^2/\|\mathbf{z}_i^1\|\|\mathbf{z}_i^2\|$, and $\tau$ is the temperature parameter. The two correlated views $\mathbf{z}_i^1$ and $\mathbf{z}_i^2$ of slci-subgraph $g_i$ are treated as positive pair while the rest view pairs in the batch are treated as negative pairs. The objective aims to maximize the consistency of positive pairs as opposed to negative ones. Note that here we use an asymmetrtic and simplified loss compared to the SimCLR loss [22], i.e., we generate negative pairs by only treating view 1 ($\mathbf{z}_i^1$) as the anchor and contrasting with view 2 ($\mathbf{z}_j^2$) of all other subgraphs, as shown in Eq. (4).

## Graph Augmentation

Contrastive learning relies heavily on well-designed data augmentation strategies for view generation. So far, commonly used graph augmentation techniques concentrate on structure-level and attribute-level augmentation. In this paper, we use two existing augmentation methods, *Attribute Masking* and *Edge Removing*, and design two novel augmentation techniques, *Attribute Similarity* and *KNN Graph*, as discussed below.

### Attribute Masking

This augmentor was proposed in [23]. Given the attribute matrix $\mathbf{X}$, we randomly set a fraction of dimensions to zero in node attributes. Formally, we first sample a mask vector $\mathbf{m}^X \in \{0,1\}^F$, where each dimension of it is independently drawn from a probability distribution $\mathbf{m}_i^X \sim \mathsf{Bernoulli}(1-p_X)$, where $p_X$ is the probability of masking arbitrary dimension of node attributes. The resulting attribute matrix $\hat{\mathbf{X}}$ can be computed as

$$\hat{\mathbf{X}} = t_{AM}(\mathbf{X}) = [\mathbf{x}_1\circ\mathbf{m}^X;\mathbf{x}_2\circ\mathbf{m}^X;\cdots;\mathbf{x}_N\circ\mathbf{m}^X]^\top, \tag{5}$$

where $\circ$ is the Hadamard product and $[\cdot;\cdot]$ is the concatenation operation.

### Edge Removing

This augmentor was used in [10, 24] and belongs to structure-level augmentation. Given the adjacency matrix $\mathbf{A}$, we randomly remove a portion of edges from graph. Formally, we first sample a mask matrix $\mathbf{M}^A \in \{0,1\}^{N\times N}$, whose entry is independently drawn from a probability distribution $\mathbf{M}_{ij}^A \sim \mathsf{Bernoulli}(1-p_A)$, where $p_A$ is the probability of removing an arbitrary edge from graph. The resulting adjacency matrix $\hat{\mathbf{A}}$ can be computed as

$$\hat{\mathbf{A}} = t_{ER}(\mathbf{A}) = \mathbf{A}\circ\mathbf{M}^A. \tag{6}$$

### Attribute Similarity

This augmentor builds new node features based on node similarity. Specifically, the new node feature matrix is actually the node similarity matrix $\mathbf{S}\in\mathbb{R}^{N\times N}$, in which each entry $\mathbf{S}_{ij}$ represents the similarity between node $v_i$ and $v_j$, and can be calculated by $S_{ij} = \mathbf{x}_i^\top\mathbf{x}_j$. Note that the similarity matrix generated by dot product between $\mathbf{X}$ and its transpose is unique, so we further apply *Attribute Masking* on the similarity matrix for diverse view generation. The resulting attribute matrix $\hat{\mathbf{X}}$ can be computed as

$$\hat{\mathbf{X}} = t_{AM}(\mathbf{S}) = t_{AM}(\mathbf{X}\mathbf{X}^\top). \tag{7}$$

### KNN Graph

This augmentor builds new adjacency matrix based on feature similarity. Regarding arbitrary node $v_i$ in graph as an independent sample with features $\mathbf{x}_i$, we first calculate the feature similarity matrix by dot product, $\mathbf{S} = \mathbf{X}\mathbf{X}^\top$. Then for each sample, we find its top $K_T$ similar samples as neighbors and set edges to connect it and its neighbors, formulated as $\hat{\mathbf{a}}_i = \mathsf{bT}(\mathbf{S}_i)$ where $\mathsf{bT}$ is the function that binarizes elements in a vector by setting the largest $K_T$ elements as 1 and other elements as 0. The resulting adjacency matrix $\hat{\mathbf{A}}$ can be computed as

$$\hat{\mathbf{A}} = t_{KG}(\mathbf{S}) = [\mathsf{bT}(\mathbf{S}_1); \cdots; \mathsf{bT}(\mathbf{S}_N)]^\top. \tag{8}$$

## Experiments

### Datasets

To assess how well our approach can learn highly-expressive link representation while keeping high scalability on large-scale graphs, we evaluate SCLRL on edge-level graph analytics task — link prediction, using publicly available real-world datasets as follows: three popular benchmark citation network datasets (Cora, Citeseer and Pubmed [25]) together with two large-scale datasets[1] of web and social networks collected from Facebook and Github. The specifications of datasets are given in Table 1 and additional details can be found in Appendix A.

### Comparison Methods

Since our approach belongs to unsupervised link representation learning and is evaluated on downstream link prediction task, we use 4 broad categories of methods for comparison: link prediction heuristics, graph kernel, unsupervised GRL and supervised task learning. For heuristics, we compare with 4 well-known node-similarity measures which are commonly used in link prediction: Common Neighbors (CN), Salton Index, Adamic-Adar Index (AA) and Resource Allocation Index (RA) [26]. For graph kernels and whole-graph embedding methods, we compare with them since that SCLRL follows the "subgraph2link" pattern and learns link representation by extracting local subgraph features. We use 2 graph kernels and 2 whole-graph embedding methods for comparison: Weisfeiler-Lehman Kernel (WL) [15], Shortest Path Kernel (SP) [16], Spectral Feature embedding (SF) [17] and Graph2Vec [18]. Besides, we also compare with 2 random walks and 2 graph auto-encoders which provide node representations and further calculate existence scores for links following [8]: DeepWalk [4], Node2Vec [7], Graph Auto-Encoder (GAE) and Variational Graph Auto-Encoder (VGAE) [8]. Furthermore, we also compare SCLRL with a supervised GNN-based framework, SEAL [9], which also characterizes links using local subgraphs, but in an end-to-end manner.

Table 1: Dataset properties.

| | Dataset | Type | Nodes | Edges | Features |
|---|---|---|---|---|---|
| Small-scale | Cora | Citation network | 2708 | 5429 | 1433 |
| | Citeseer | Citation network | 3327 | 4732 | 3703 |
| | Pubmed | Citation network | 19717 | 44338 | 500 |
| Large-scale | Facebook | Web network | 22470 | 171002 | 128* |
| | Github | Social network | 37700 | 289,003 | 128* |

* The original node features are processed into 128-dimensional initial features by Doc2Vec [27].

### Experimental Settings

### Data Preparation

For graph kernels, whole-graph embedding methods, SEAL and SCLRL, we sample the same number of positive and negative links from each dataset, with the proportion of $\{40\%, 40\%, 10\%, 10\%, 20\%\}$ for $\{$Cora, Citeseer, Pubmed, Facebook, Github$\}$. All links are split into training, validation and testing sets with a proportation of 8:1:1, and we further extract "slci-subgraphs" for each link. For heuristics, random walks methods and graph auto-encoders which rely on full-graph training, we randomly sample a certain number of positive links as well as the same number of additionally negative links as testing data, and the remaining partially observed graph is used for training. We repeat 10-fold cross validation for 5 times and report the average Area Under Curve (AUC) and Average Precision (AP)

[1]http://snap.stanford.edu/data/

Table 2: Performance on link prediction task reported in Area Under Curve (AUC) and Average Precision (AP) measures. The best result per category is highlighted in blod.

| | Method | Cora | | Citeseer | | Pubmed | | Facebook | | Github | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | AUC (%) | AP (%) | AUC (%) | AP (%) | AUC (%) | AP (%) | AUC (%) | AP (%) | AUC (%) | AP (%) |
| Heuristics | CN | 56.19±0.099 | 63.08±0.059 | 58.76±0.095 | 65.31±0.060 | **65.53±0.050** | 67.54±0.034 | **88.70±0.076** | **91.72±0.045** | **67.87±0.105** | 73.26±0.069 |
| | Salton | 56.85±0.094 | 61.73±0.065 | **59.32±0.090** | 63.79±0.065 | 64.78±0.057 | 66.12±0.047 | 88.61±0.077 | 91.07±0.054 | 62.84±0.085 | 60.69±0.054 |
| | AA | 57.33±0.087 | 59.36±0.081 | 58.97±0.089 | 60.90±0.083 | 65.26±0.049 | 67.49±0.035 | 87.24±0.093 | 91.18±0.060 | 67.69±0.108 | 75.23±0.085 |
| | RA | **57.77±0.090** | **64.75±0.055** | 59.11±0.091 | **65.88±0.058** | 65.27±0.047 | **67.78±0.032** | 85.08±0.101 | 90.43±0.057 | 67.56±0.078 | **77.14±0.050** |
| Kernel | $^\star$ WL | **96.31±0.008** | **96.19**±0.012 | **97.39**±0.011 | **97.66**±0.010 | **93.77**±0.012 | **93.67**±0.012 | **93.78**±0.005 | **93.77**±0.005 | **78.77**±0.009 | **77.06**±0.011 |
| | $^\star$ SP | 90.50±0.087 | 91.04±0.020 | 92.12±0.022 | 93.56±0.018 | 90.86±0.016 | 89.25±0.022 | 74.34±0.009 | 72.17±0.008 | 61.31±0.008 | 58.32±0.007 |
| Supervised | $^\star$ SEAL($h = 1$, $K_1 =$ ours) | 63.36±0.043 | 65.13±0.046 | 67.64±0.053 | 70.23±0.053 | 85.30±0.023 | 84.17±0.031 | 95.05±0.004 | 95.38±0.004 | **89.39±0.006** | **89.05±0.007** |
| | $^\star$ SEAL($h = 2$, $K_1, K_2 =$ ours) | 62.23±0.046 | 64.74±0.043 | 66.55±0.048 | 68.96±0.052 | 83.69±0.017 | 83.32±0.019 | 94.38±0.006 | 94.94±0.006 | 87.90±0.006 | 87.74±0.007 |
| | $^\star$ SEAL($h = 1$, $K_1 =$ all) | 88.70±0.019 | 88.72±0.023 | 88.84±0.029 | 89.71±0.031 | 93.49±0.011 | 94.16±0.010 | 98.50±0.023 | 98.69±0.020 | $>$ 5 days | $>$ 5 days |
| | $^\star$ SEAL($h = 2$, $K_1, K_2 =$ all) | **91.04±0.017** | **90.79±0.021** | **92.72±0.023** | **93.02±0.022** | **95.15±0.009** | **95.13±0.011** | **98.62±0.001** | **98.79±0.001** | $>$ 5 days | $>$ 5 days |
| Unsupervised | $^\star$ SF | 96.38±0.011 | 94.29±0.020 | 98.24±0.008 | 97.22±0.017 | 93.46±0.013 | 92.13±0.018 | 89.16±0.006 | 90.65±0.005 | 76.61±0.008 | 76.33±0.009 |
| | $^\star$ Graph2Vec | 96.23±0.010 | 94.12±0.020 | 98.15±0.009 | 97.08±0.018 | 92.90±0.013 | 91.54±0.017 | 91.24±0.005 | 92.39±0.005 | 77.74±0.008 | 76.18±0.011 |
| | $^\diamond$ DeepWalk | 88.14±0.055 | 87.87±0.045 | 85.67±0.052 | 86.11±0.044 | 90.88±0.021 | 87.48±0.026 | 87.65±0.012 | 85.41±0.011 | **81.25±0.013** | **80.25±0.012** |
| | $^\diamond$ Node2Vec | 88.65±0.058 | 89.62±0.049 | 87.36±0.065 | 88.15±0.059 | 90.60±0.021 | 89.29±0.027 | 85.64±0.022 | 85.25±0.028 | 80.49±0.019 | 79.62±0.016 |
| | $^\diamond$ GAE | 94.17±0.018 | 93.98±0.021 | 93.14±0.023 | 91.90±0.028 | 94.20±0.019 | 93.35±0.023 | OOM | OOM | OOM | OOM |
| | $^\diamond$ VGAE | 94.30±0.015 | 94.30±0.016 | 95.61±0.017 | 95.47±0.019 | 91.58±0.007 | 90.49±0.010 | OOM | OOM | OOM | OOM |
| | $^\star$ SCLRL($h = 1$, $K_1 =$ ours) | **99.49±0.003** | **99.56**±0.002 | **99.32±0.005** | **99.17±0.007** | **97.05±0.006** | **97.41**±0.006 | **94.89±0.001** | **95.20±0.001** | 79.77±0.005 | 78.63±0.005 |

$^\diamond$ This method generates link representation with "node2link" pattern.
$^\star$ This method generates link representation with "subgraph2link" pattern.

measures as well as their standard deviations. Note that we use the same testing links during each evaluation for each method to make a fair comparison. We include more details of data preparation in Appendix B.

## Parameter Configuration

For all graph kernels, whole-graph embedding methods and SCLRL, we implement link prediction by feeding the generated link representations into a downstream classifier, which is Support Vector Machine for {Cora, Citeseer, Pubmed} and Random Forest for {Facebook, Github}. The dimension of generated link representation is set to 128. In SCLRL, the hop number $h$ is set to 1 and the both probabilities ($p_A$, $p_x$) associated with data augmentation are set to 20%. For the parameter $K_1$ that refers to the number of sampled neighborhs per node in 1-hop, we set it for all datasets to 3. We use 2-layer graph isomorphism network (GIN) with the default setting in [28] as the graph encoder in SCLRL. The batch size is set to 128. More details about parameter configuration of SCLRL and baselines are included in Appendix C.

## Evaluation on Link Prediction

The results of evaluating unsupervised link representation learning using downstream link prediction task are presented in Table 2, from which we can see that our SCLRL achieves state-of-the-art results with respect to previous unsupervised methods. For instance, on three citation networks, we achieve 99.49%, 99.32% and 97.05% AUC, respectively, which are 3.2%, 1.1% and 2.7% relative improvement over previous state-of-the-art. When compared to graph kernel methods, our SCLRL surpasses strong baseline: on three citation benchmarks we observe 3.3%, 2.0% and 3.5% relative improvement over WL kernel in terms of AUC, respectively. Our SCLRL also significantly outperforms heuristic baselines across all datasets.

We also compare with the state-of-the-art supervised model, SEAL, in a variety of hyper-parameter settings. The results shown in Table 2 suggest that our model significantly outperforms SEAL (with same hyper-parameter settings) on three citation benchmarks, and achieves highly competitive results in Facebook datasets. For instance, on three citation benchmarks we observe 57.0%, 46.8% and 13.8% relative improvement over SEAL in terms of AUC. When compared to SEAL that accepts larger subgraphs, our SCLRL still beats SEAL on three citation benchmarks while performs 3.73 $\sim$ 8.13% worse than SEAL in large-scale datasets. These results show that for link prediction, SEAL relies on enclosing subgraphs of larger size to learn high-order link features while SCLRL prefers to find link patterns from small subgraphs.

**Subgraph2link vs Node2link:** Furthermore, when comparing the two categories of methods with different patterns, the results suggest that "subgraph2link" models generally perform on par with or better than strong baselines with “node2link” pattern. Specifically, kernel method (WL kernel) and whole-graph embedding methods (SF, Graph2Vec) achieve comparable or sometimes even better performance than graph auto-encoders, and surpass random walks by substantial performance margins in 4 out of 5 datasets. More importantly, “subgraph2link” models are trained on slci-subgraphs while “node2link” models are trained on partially observed graph where only the testing links are masked. In summary, compared with “node2link” methods, “subgraph2link” models use relatively less training data and achieve comparable or superior performance, which validates the effectiveness of our proposal, i.e., *local subgraph can provide informative structure contexts for link representation learning*.

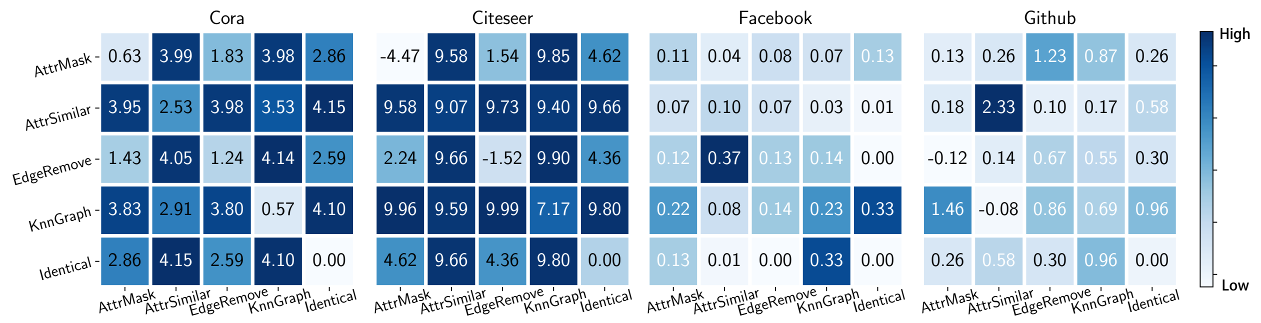

Figure 3: Link prediction AUC gain (%) when contrasting different augmentation pairs, compared to “Identical-Identical” which stands for a no-augmentation baseline for contrastive learning, under three categories of datasets. The baseline “Identical-Identical” AUCs are 95.53%, 90.30%, 94.54% and 77.95% for the four datasets, respectively.

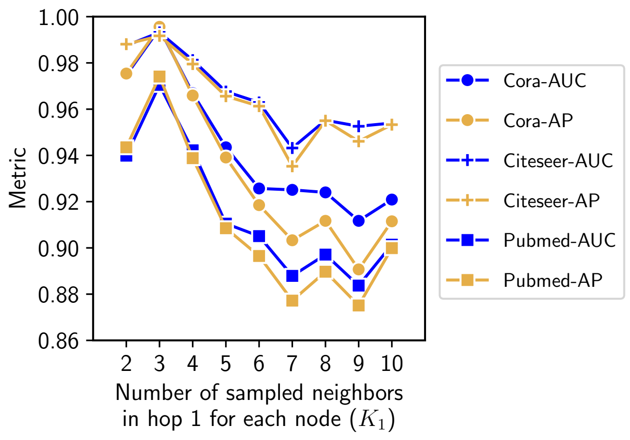

Figure 4: Performance versus subgraph size.

## Ablation Studies

### Effect of Augmented View

We investigate four augmentation techniques by applying various pairs of augmentation types to three categories of graph datasets, as illustrated in Figure 3. As we can see, for citation networks, using either “AttrSimilar” or “KnnGraph” as one of augmented views can significantly benefit the downstream performance, when compared to other augmentation pairs. And for the web and social networks, best performance is still achieved by using “AttrSimilar” as an augmented view: 0.37% for Facebook, and 2.33% for Github. The observation meets our intuition. In link prediction scenario, a larger similarity between nodes leads to a higher likelihood of links between them. Based on this, the new node features and adjacency information generated by “AttrSimilar” and “KnnGraph” respectively, can effectively characterize the correlation between nodes. Therefore, contrasting the new graph features with original (“Identical”) or perturbed (generated by “AttrMask” or “EdgeRemove”) ones, can avoids exploit shortcuts that can greatly decrease representation quality and learn more generalizable features, as conjectured in [22].

Table 3: Performance versus number of hops in subgraph.

| Metrix | Hop | Dataset | | | | | Remark |
|---|---|---|---|---|---|---|---|
| | | Cora | Citeseer | Pubmed | Facebook | Github | |
| AUC | $h = 1$ | 99.49% | 99.32% | 97.05% | 94.56% | 79.77% | $K_1 = 3$ $K_2 = 1$ |
| | $h = 2$ | 97.36% | 98.32% | 94.40% | 90.74% | 75.80% | |
| | gain | 2.2% | 1.0% | 2.8% | 4.2% | 5.2% | |
| AP | $h = 1$ | 99.56% | 99.17% | 97.41% | 94.94% | 78.63% | |
| | $h = 2$ | 97.55% | 98.67% | 94.56% | 91.46% | 75.87% | |
| | gain | 2.1% | 0.5% | 3.0% | 3.8% | 3.6% | |

Table 4: Efficiency analysis on five datasets.

| Item | Method | Dataset | | | | |
|---|---|---|---|---|---|---|
| | | Cora | Citeseer | Pubmed | Facebook | Github |
| Training Time (s) | DeepWalk | 609.2 | 580.7 | 6023.0 | 13531.5 | 21948.7 |
| | GAE | 0.0 | 0.0 | 0.0 | – | – |
| | SCLRL | 1.6 | 1.2 | 5.1 | 8.7 | 13.3 |
| Memory (MB) | DeepWalk | 346 | 414 | 1679 | 3082 | 3099 |
| | GAE | 4935 | 4932 | 4994 | OOM | OOM |
| | SCLRL | 5698 | 6000 | 5589 | 6044 | 8740 |

### Effect of Subgraph Size

We further investigate the impact of subgraph size in our framework on five datasets (refer to Appendix E for more details on large datasets.). In this paper, the subgraph size is determined by two hyper-parameters: $h$ and $K$.

**Number of sampled neighborhs per node ($K$):** we first fix $h = 1$ and adjust $K_1$ from 2 to 10, and evaluate the results shown in Fig 4. We observe that our model obtains best results across all citation networks consistently when $K_1$ is set to 3. More specifically, as the number of sampled neighborhs increases, the performance on all datasets increases first, reaches the peak when $K_1$ is 3, and then goes down.

**Number of hops ($h$):** we fix $K_1 = 3$ and $K_2 = 1$, and vary $h$ in $\{1,2\}$. We present the evaluation results on Table 3, from which we can see that 1-hop SCLRL yields up to 1.0 $\sim$ 5.2% and 0.5 $\sim$ 3.8% improvement in AUC and AP over 2-hop SCLRL, respectively.

Combining with the above results, we know that for citation networks, key structural information reflecting link existence and benefitting downstream prediction task is generally contained in subgraphs with smaller size.

### Computational Efficiency

Finally, we investigate the computational efficiency of SCLRL, as illustrated in Table 4, by summarizing the training time and memory consumption on all the five datasets. The training time refers to the time for training the encoder (exclude validation) per epoch, and the memory refers to total memory costs of model parameters and all hidden representations. We observe that for the two comparison methods, they rely on full-graph training and become less efficient as the scale of graph grows. When compared to DeepWalk belonging to random walks, our SCLRL can be trained much faster on datasets of various scales. When compared to GAE belonging to graph auto-encoder, our SCLRL achieves comparable efficiency on small-scale graphs, and beats GAE on large-scale graphs via subgraph mini-batch training. In summary, our SCLRL take both training speed and memory consumption into account, scaling well on large-scale graphs.

## Conclusion

Graph representation learning for link is relatively less studied and less understood. In this paper, we study the "subgraph2link" mechanism and propose a self-supervised framework for learning link representations by contrasting encodings from different view of subgraphs centered on links. Meanwhile, we introduce two novel augmentation techniques which are capable of characterizing node correlation. Experiments conducted on five datasets demonstrate that our framework can achieve new state-of-the-art in link prediction on benchmark datasets, and have satisfactory prediction performance, prominent advantages in training speed and memory consumption on large-scale datasets.

## References

[1] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *Proceedings of the 5th International Conference on Learning Representations*, 2017.

[2] Deyu Bo, Xiao Wang, Chuan Shi, Meiqi Zhu, Emiao Lu, and Peng Cui. Structural deep clustering network. In *Proceedings of The Web Conference 2020*, pages 1400–1410, 2020.

[3] Hanjun Dai, Bo Dai, and Le Song. Discriminative embeddings of latent variable models for structured data. In *International Conference on Machine Learning*, pages 2702–2711. PMLR, 2016.

[4] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 701–710, 2014.

[5] Arjun Subramonian. Motif-driven contrastive learning of graph representations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 15980–15981, 2021.

[6] Junheng Hao, Muhao Chen, Wenchao Yu, Yizhou Sun, and Wei Wang. Universal representation learning of knowledge bases by jointly embedding instances and ontological concepts. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1709–1719, 2019.

[7] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 855–864, 2016.

[8] Thomas N Kipf and Max Welling. Variational graph auto-encoders. *NIPS Workshop on Bayesian Deep Learning*, 2016.

[9] Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, volume 31, pages 5165–5175, 2018.

[10] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. Graph contrastive learning with augmentations. volume 33, pages 5812–5823, 2020.

[11] Jiezhong Qiu, Qibin Chen, Yuxiao Dong, Jing Zhang, Hongxia Yang, Ming Ding, Kuansan Wang, and Jie Tang. Gcc: Graph contrastive coding for graph neural network pre-training. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1150–1160, 2020.

[12] Yizhu Jiao, Yun Xiong, Jiawei Zhang, Yao Zhang, Tianqi Zhang, and Yangyong Zhu. Sub-graph contrast for scalable self-supervised graph representation learning. In *2020 IEEE International Conference on Data Mining (ICDM)*, pages 222–231. IEEE, 2020.

[13] William L Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 1025–1035, 2017.

[14] Petar Veličković, William Fedus, William L. Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. Deep graph infomax. In *Proceedings of the 7th International Conference on Learning Representations*, 2019.

[15] Nino Shervashidze, Pascal Schweitzer, Erik Jan Van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(9), 2011.

[16] Karsten M Borgwardt and Hans-Peter Kriegel. Shortest-path kernels on graphs. In *Fifth IEEE International Conference on Data Mining (ICDM'05)*, pages 8–pp. IEEE, 2005.

[17] Nathan de Lara and Edouard Pineau. A simple baseline algorithm for graph classification. *arXiv preprint arXiv:1810.09155*, 2018.

[18] Annamalai Narayanan, Mahinthan Chandramohan, Rajasekar Venkatesan, Lihui Chen, Yang Liu, and Shantanu Jaiswal. graph2vec: Learning distributed representations of graphs. *arXiv preprint arXiv:1707.05005*, 2017.

[19] Jie Chen, Tengfei Ma, and Cao Xiao. FastGCN: Fast learning with graph convolutional networks via importance sampling. In *Proceedings of the 6th International Conference on Learning Representations*, 2018.

[20] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 257–266, 2019.

[21] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. Graphsaint: Graph sampling based inductive learning method. In *In Proceedings of the 8th International Conference on Learning Representations*, 2020.

[22] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International Conference on Machine Learning*, pages 1597–1607. PMLR, 2020.

[23] Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. Deep graph contrastive representation learning. *arXiv preprint arXiv:2006.04131*, 2020.

[24] Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. Graph contrastive learning with adaptive augmentation. In *Proceedings of the Web Conference 2021*, pages 2069–2080, 2021.

[25] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI Magazine*, 29(3):93–93, 2008.

[26] Linyuan Lü and Tao Zhou. Link prediction in complex networks: A survey. *Physica A: Statistical Mechanics and its Applications*, 390(6):1150–1170, 2011.

[27] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *International Conference on Machine Learning*, pages 1188–1196. PMLR, 2014.

[28] Fan-Yun Sun, Jordon Hoffman, Vikas Verma, and Jian Tang. Infograph: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization. In *In Proceedings of the 8th International Conference on Learning Representations*, 2020.