HARP: Hierarchical Representation Learning for Networks

Haochen Chen
Stony Brook University
haocchen@cs.stonybrook.edu

Bryan Perozzi
Google Research
bperozzi@acm.org

Yifan Hu
Yahoo! Research
yifanhu@oath.com

Steven Skiena
Stony Brook University
skiena@cs.stonybrook.edu

Abstract

We present HARP, a novel method for learning low dimensional embeddings of a graph's nodes which preserves higher-order structural features. Our proposed method achieves this by compressing the input graph prior to embedding it, effectively avoiding troublesome embedding configurations (i.e. local minima) which can pose problems to non-convex optimization.

HARP works by finding a smaller graph which approximates the global structure of its input. This simplified graph is used to learn a set of initial representations, which serve as good initializations for learning representations in the original, detailed graph. We inductively extend this idea, by decomposing a graph in a series of levels, and then embed the hierarchy of graphs from the coarsest one to the original graph.

HARP is a general meta-strategy to improve *all* of the state-of-the-art neural algorithms for embedding graphs, including *DeepWalk*, *LINE*, and *Node2vec*. Indeed, we demonstrate that applying HARP's hierarchical paradigm yields improved implementations for all three of these methods, as evaluated on classification tasks on real-world graphs such as *DBLP*, *Blog-Catalog*, and *CiteSeer*, where we achieve a performance gain over the original implementations by up to 14% Macro F1.
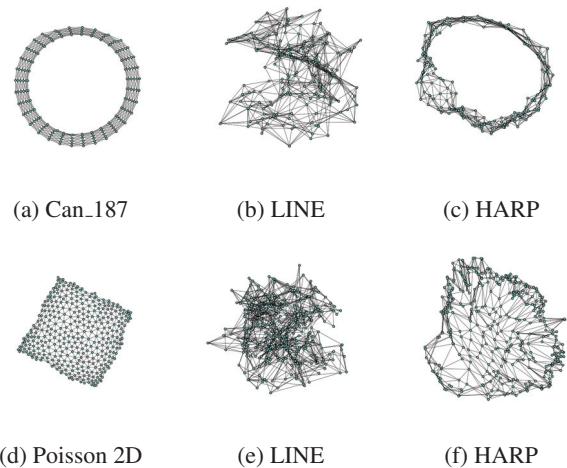
Figure 1: Comparison of two-dimensional embeddings from LINE and our proposed method, for two distinct graphs. Observe how HARP's embedding better preserves the higher order structure of a ring and a plane.

Introduction

From social networks to the World Wide Web, graphs are a ubiquitous way to organize a diverse set of real-world information. Given a network's structure, it is often desirable to predict missing information (frequently called *attributes* or *labels*) associated with each node in the graph. This missing information can represent a variety of aspects of the data – for example, on a social network they could represent the communities a person belongs to, or the categories of a document's content on the web.

Because many information networks can contain billions of nodes and edges, it can be intractable to perform complex inference procedures on the entire network. One technique which has been proposed to address this problem is *dimensionality reduction*. The central idea is to find a mapping function which converts each node in the graph to a low-dimensional latent representation. These representations can

then be used as features for common tasks on graphs such as multi-label classification, clustering, and link prediction.

Traditional methods for graph dimensionality reduction (Belkin and Niyogi 2001; Roweis and Saul 2000; Tenenbaum, De Silva, and Langford 2000) perform well on small graphs. However, the time complexity of these methods are at least quadratic in the number of graph nodes, makes them impossible to run on large-scale networks.

A recent advancement in graph representation learning, DeepWalk (Perozzi, Al-Rfou, and Skiena 2014) proposed online learning methods using neural networks to address this scalability limitation. Much work has since followed (Cao, Lu, and Xu 2015; Grover and Leskovec 2016; Perozzi et al. 2017; Tang et al. 2015). These neural network-based methods have proven both highly scalable and performant, achieving strong results on classification and link prediction tasks in large networks.

Despite their success, all these methods have several shared weaknesses. Firstly, they are all local approaches – limited to the structure immediately around a node. Deep-

Walk (Perozzi, Al-Rfou, and Skiena 2014) and Node2vec (Grover and Leskovec 2016) adopt short random walks to explore the local neighborhoods of nodes, while LINE (Tang et al. 2015) is concerned with even closer relationships (nodes at most two hops away). This focus on local structure implicitly ignores long-distance global relationships, and the learned representations can fail to uncover important global structural patterns. Secondly, they all rely on a non-convex optimization goal solved using stochastic gradient descent (Goldberg and Levy 2014; Mikolov et al. 2013) which can become stuck in a local minima (e.g. perhaps as a result of a poor initialization). In other words, all previously proposed techniques for graph representation learning can accidentally learn embedding configurations which disregard important structural features of their input graph.

In this work, we propose *HARP*, a meta strategy for embedding graph datasets which preserves higher-order structural features. *HARP* recursively coalesces the nodes and edges in the original graph to get a series of successively smaller graphs with similar structure. These coalesced graphs, each with a different granularity, provide us a view of the original graph's global structure. Starting from the most simplified form, each graph is used to learn a set of initial representations which serve as good initializations for embedding the next, more detailed graph. This process is repeated until we get an embedding for each node in the original graph.

We illustrate the effectiveness of this multilevel paradigm in Figure 1, by visualizing the two-dimension embeddings from an existing method (*LINE* (Tang et al. 2015)) and our improvement to it, *HARP(LINE)*. Each of the small graphs we consider has an obvious global structure (that of a ring (1a) and a grid (1d)) which is easily exposed by a force direced layout (Hu 2005). The center figures represent the two-dimensional embedding obtained by LINE for the ring (1b) and grid (1e). In these embeddings, the global structure is lost (i.e. that is, the ring and plane are unidentifiable). However, the embeddings produced by using our meta-strategy to improve LINE (right) clearly capture both the local and global structure of the given graphs (1c, 1f).

Our contributions are the following:

- **New Representation Learning Paradigm.** We propose *HARP*, a novel multilevel paradigm for graph representation which seamlessly blends ideas from the graph drawing (Fruchterman and Reingold 1991) and graph representation learning (Perozzi, Al-Rfou, and Skiena 2014; Tang et al. 2015; Grover and Leskovec 2016) communities to build substantially better graph embeddings.

- **Improved Optimization Primitives.** We demonstrate that our approach leads to improved implementations of **all** state-of-the-art graph representation learning methods, namely *DeepWalk* (DW), *LINE* and *Node2vec* (N2V). Our improvements on these popular methods for learning latent representations illustrate the broad applicability of our hierarchical approach.

- **Better Embeddings for Downstream Tasks.** We demonstrate that *HARP(DW)*, *HARP(LINE)* and *HARP(N2V)* embeddings consistently outperform the originals on clas-

sification tasks on several real-world networks, with improvements as large as 14% Macro F_1.

Problem Formulation

We desire to learn latent representations of nodes in a graph. Formally, let $G = (V, E)$ be a graph, where V is the set of nodes and E is the set of edges. The goal of graph representation learning is to develop a mapping function $\Phi : V \mapsto \mathbb{R}^{|V| \times d}, d \ll |V|$. This mapping Φ defines the latent representation (or *embedding*) of each node $v \in V$. Popular methods for learning the parameters of Φ (Perozzi, Al-Rfou, and Skiena 2014; Tang et al. 2015; Grover and Leskovec 2016) suffer from two main disadvantages: (1) higher-order graph structural information is not modeled, and (2) their stochastic optimization can fall victim to poor initialization.

In light of these difficulties, we introduce the *hierarchical representation learning* problem for graphs. At its core, we seek to find a graph, $G_s = (V_s, E_s)$ which captures the essential structure of G, but is smaller than our original (i.e. $|V_s| << |V|, |E_s| << |E|$). It is likely that G_s will be easier to embed for two reasons. First, there are many less pairwise relationships ($|V_s|^2$ versus $|V|^2$) which can be expressed in the space. As the sample space shrinks, there is less variation in training examples – this can yield a smoother objective function which is easier to optimize. Second, the diameter of G_s may be smaller than G, so algorithms with a local focus can exploit the graph's global structure.

In summary, we define the hierarchical representation learning problem in graphs as follows:

Given a large graph $G(V, E)$ and a function f, which embeds G using initialization θ, $f : G \times \theta \mapsto \Phi_G$,

Simplify G to a series of successively smaller graphs $G_0 \ldots G_L$,

Learn a coarse embedding $\Phi_{G_L} = f(G_L, \emptyset)$,

Refine the coarse embedding into Φ_G by iteratively applying $\Phi_{G_i} = f(G_i, \Phi_{G_{i+1}}), 0 \le i < L$.

Method

Here we present our hierarchical paradigm for graph representation learning. After discussing the method in general, we present a structure-preserving algorithm for its most crucial step, graph coarsening.

Algorithm: `HARP`

Our method for multi-level graph representation learning, HARP, is presented in Algorithm 1. It consists of three parts - graph coarsening, graph embedding, and representation refinement - which we detail below:

1. *Graph Coarsening* (line 1): Given a graph G, graph coarsening algorithms create a hierarchy of successively smaller graphs $G_0, G_1, \cdots, G_L$, where $G_0 = G$. The coarser (smaller) graphs preserve the global structure of the original graph, yet have significantly fewer nodes and edges. Algorithms for generating this hierarchy of graphs will be discussed in detail below.

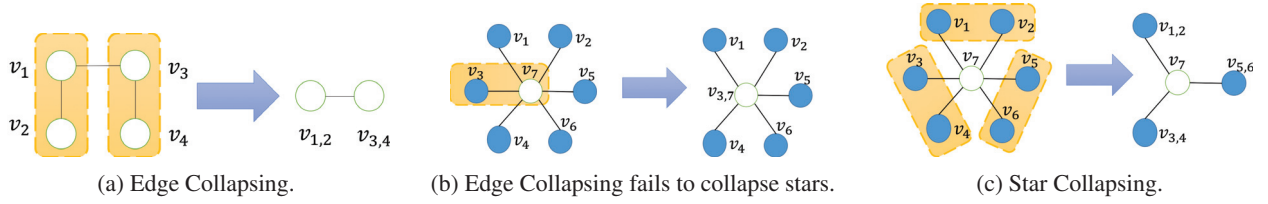| (a) Edge Collapsing. | (b) Edge Collapsing fails to collapse stars. | (c) Star Collapsing. |

Figure 2: Illustration of graph coarsening algorithms. 2a: Edge collapsing on a graph snippet. 2b: How edge collapsing fails to coalesce star-like structures. 2c: How star collapsing scheme coalesces the same graph snippet efficiently.

Algorithm 1 HARP(G, $Embed()$)

Input:
    graph $G(V, E)$
    arbitrary graph embedding algorithm EMBED()
Output: matrix of vertex representations $\Phi \in \mathbb{R}^{|V| \times d}$
1: $G_0, G_1, \cdots, G_L \leftarrow$ GRAPHCOARSENING(G)
2: Initialize Φ'_{G_L} by assigning zeros
3: $\Phi_{G_L} \leftarrow$ EMBED(G_L, Φ'_{G_L})
4: **for** $i = L - 1$ to 0 **do**
5:     $\Phi'_{G_i} \leftarrow$ PROLONGATE($\Phi_{G_{i+1}}, G_{i+1}, G_i$)
6:     $\Phi_{G_i} \leftarrow$ EMBED(G_i, Φ'_{G_i})
7: **end for**
8: **return** Φ_{G_0}

---

2. *Graph Embedding on the Coarsest Graph* (line 2-3): The graph embedding is obtained on the coarsest graph G_L with the provided graph embedding algorithm. As the size of G_L is usually very small, it is much easier to get a high-quality graph representation.

3. *Graph Representation Prolongation and Refinement* (line 4-7): We prolong and refine the graph representation from the coarsest to the finest graph. For each graph G_i, we prolong the graph representation of G_{i+1} as its initial embedding Φ'_{G_i}. Then, the embedding algorithm $Embed()$ is applied to (G_i, Φ'_{G_i}) to further refine Φ'_{G_i}, resulting in the refined embedding Φ_{G_i}. We discuss this step in the embedding prolongation section below.

4. *Graph Embedding of the Original Graph* (line 8): We return Φ_{G_0}, which is the graph embedding of the original graph.

We can easily see that this paradigm is algorithm independent, relying only on the provided functions $Embed()$. Thus, with minimum effort, this paradigm can be incorporated into any existing graph representation learning methods, yielding a multilevel version of that method.

Graph Coarsening

In Algorithm 2, we develop a hybrid graph coarsening scheme which preserves global graph structural information at different scales. Its two key parts, namely edge collapsing and star collapsing, preserve *first-order proximity* and *second-order proximity* (Tang et al. 2015) respectively. First-order proximity is concerned with preserving the observed edges in the input graph, while second-order proximity is

---

Algorithm 2 GraphCoarsening(G)

Input: graph $G(V, E)$
Output: Series of Coarsened Graphs $G_0, G_1, \cdots, G_L$
1: $L \leftarrow 0$
2: $G_0 \leftarrow G$
3: **while** $|V_L| \geq threshold$ **do**
4:     $L \leftarrow L + 1$
5:     $G_L \leftarrow$ EDGECOLLAPSE(STARCOLLAPSE(G))
6: **end while**
7: **return** $G_0, G_1, \cdots, G_L$

---

based on the shared neighborhood structure of the nodes.

Edge Collapsing. Edge collapsing (Hu 2005) is an efficient algorithm for preserving first-order proximity. It selects $E' \subseteq E$, such that no two edges in E' are incident to the same vertex. Then, for each $(u_i, v_i) \in E'$, it merges (u_i, v_i) into a single node w_i, and merge the edges incident to u_i and v_i. The number of nodes in the coarser graph is therefore at least half of that in the original graph. As illustrated in Figure 2a, the edge collapsing algorithm merges node pairs (v_1, v_2) and (v_3, v_4) into supernodes $v_{1,2}$ and $v_{3,4}$ respectively, resulting in a coarser graph with 2 nodes and 1 edge. The order of merging is arbitrary; we find different merging orders result in very similar node embeddings in practice.

Star Collapsing. Real world graphs are often scale-free, which means they contain a large number of star-like structures. A star consists of a popular central node (sometimes referred to as *hubs*) connected to many peripheral nodes. Although the edge collapsing algorithm is simple and efficient, it cannot sufficiently compress the star-like structures in a graph. Consider the graph snippet in Figure 2b, where the only central node v_7 connects to all the other nodes. Assume the degree of the central node is k, it is clear that the edge collapsing scheme can only compress this graph into a coarsened graph with $k - 1$ nodes. Therefore when k is large, the coarsening process could be arbitrarily slow, takes $O(k)$ steps instead of $O(\log k)$ steps.

One observation on the star structure is that there are strong second-order similarities between the peripheral nodes since they share the same neighborhood. This leads to our star collapsing scheme, which merges nodes with the same neighbors into supernodes since they are similar to each other. As shown in Figure 2c, (v_1, v_2), (v_3, v_4) and (v_5, v_6) are merged into supernodes as they share the same neighbors (v_7), generating a coarsened graph with only $k/2$

nodes.

Hybrid Coarsening Scheme. By combining edge collapsing and star collapsing, we present a hybrid scheme for graph coarsening in Algorithm 2, which is adopted on all test graphs. In each coarsening step, the hybrid coarsening scheme first decomposes the input graph with star collapsing, then adopts the edge collapsing scheme to generate the coalesced graph. We repeat this process until a small enough graph (with less than 100 vertices) is obtained.

Embedding Prolongation

After the graph representation for G_{i+1} is learned, we prolong it into the initial representation for G_i. We observe that each node $v \in G_{i+1}$ is either a member of the finer representation ($v \in G_i$), or the result of a merger, $(v_1, v_2, \cdots, v_k) \in G_i$. In both cases, we can simply reuse the representation of the parent node $v \in G_i$ - the children are quickly separated by gradient updates.

Complexity Analysis

In this section, we discuss the time complexity of *HARP(DW)* and *HARP(LINE)* and compare with the time complexity of *DeepWalk* and *LINE* respectively. *HARP(N2V)* has the same time complexity as *HARP(DW)*, thus it is not included in the discussion below.

HARP(DW): Given the number of random walks γ, walk length t, window size w and representation size d, the time complexity of *DeepWalk* is dominated by the training time of the Skip-gram model, which is $\mathcal{O}(\gamma|V|tw(d + d\log|V|))$. For *HARP(DW)*, coarsening a graph with $|V|$ nodes produces a coarser graph with about $|V|/2$ nodes. The total number of nodes in all levels is approximately $|V| \sum_{i=0}^{\log_2|V|} (\frac{1}{2})^i = 2|V|$. Therefore, the time complexity of *HARP(DW)* is $O(|V|)$ for copying binary tree and $\mathcal{O}(\gamma|V|tw(d+d\log|V|))$ for model training. Thus, the overall time complexity of *HARP(DW)* is also $\mathcal{O}(\gamma|V|tw(d + d\log|V|))$.

HARP(LINE): The time complexity of *LINE* is linear to the number of edges in the graph and the number of iterations r over edges, which is $\mathcal{O}(r|E|)$. For *HARP(LINE)*, coarsening a graph with $|E|$ nodes produces a coarsened graph with about $|E|/2$ edges. The total number edges in all levels is approximately $|E| \sum_{i=0}^{\log_2|E|} (\frac{1}{2})^i = 2|E|$. Thus, the time complexity of *HARP(LINE)* is also $\mathcal{O}(r|E|)$.

Experiment

In this section, we provide an overview of the datasets and methods used for experiments and evaluate the effectiveness of our method on challenging multi-label classification tasks in several real-life networks. We further illustrate the scalability of our method and discuss its performance with regard to several important parameters.

Datasets

Table 1 gives an overview of the datasets used in our experiments.

Name	DBLP	Blogcatalog	CiteSeer
# Vertices	29,199	10,312	3,312
# Edges	133,664	333,983	4,732
# Classes	4	39	6
Task	Classification	Classification	Classification

Table 1: Statistics of the graphs used in our experiments.

- *DBLP* (Perozzi et al. 2017) – DBLP is a co-author graph of researchers in computer science. The labels indicate the research areas a researcher publishes his work in. The 4 research areas included in this dataset are DB, DM, IR, and ML.

- *BlogCatalog* (Tang and Liu 2009) – BlogCatalog is a network of social relationships between users on the BlogCatalog website. The labels represent the categories a blogger publishes in.

- *CiteSeer* (Sen et al. 2008) – CiteSeer is a citation network between publications in computer science. The labels indicate the research areas a paper belongs to. The papers are classified into 6 categories: Agents, AI, DB, IR, ML, and HCI.

Baseline Methods

We compare our model with the following graph embedding methods:

- *DeepWalk* — *DeepWalk* is a two-phase method for embedding graphs. Firstly, *DeepWalk* generates random walks of fixed length from all the vertices of a graph. Then, the walks are treated as sentences in a language model and the Skip-Gram model for learning word embeddings is utilized to obtain graph embeddings. *DeepWalk* uses hierarchical softmax for Skip-gram model optimization.

- *LINE* — *LINE* is a method for embedding large-scale networks. The objective function of *LINE* is designed for preserving both first-order and second-order proximities, and we use first-order *LINE* for comparison. Skip-gram with negative sampling is used to solve the objective function.

- *Node2vec* — *Node2vec* proposes an improvement to the random walk phase of *DeepWalk*. By introducing the return parameter p and the in-out parameter q, *Node2vec* combines DFS-like and BFS-like neighborhood exploration. *Node2vec* also uses negative sampling for optimizing the Skip-gram model.

For each baseline method, we combine it with *HARP* and compare their performance.

Parameter Settings

Here we discuss the parameter settings for our models and baseline models. Since *DeepWalk*, *LINE* and *Node2vec* are all sampling based algorithms, we always ensure that the total number of samples seen by the baseline algorithm is the **same** as that of the corresponding *HARP* enhanced algorithm.
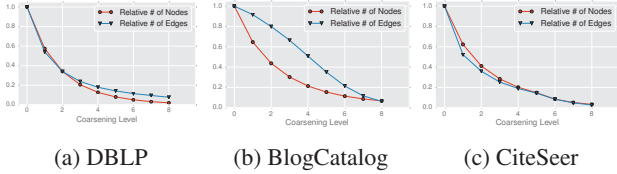
(a) DBLP (b) BlogCatalog (c) CiteSeer

Figure 3: The ratio of nodes/edges of the coarsened graphs to that of the original test graphs. For disconnected graphs, the graph coarsening result on the largest connected component is shown.

DeepWalk. For *DeepWalk* and *HARP(DW)*, we need to set the following parameters: the number of random walks γ, walk length t, window size w for the Skip-gram model and representation size d. In *HARP(DW)*, the parameter setting is $\gamma = 40, t = 10, w = 10, d = 128$. For *DeepWalk*, all the parameters except γ are the same as in *HARP(DW)*. Specifically, to ensure a fair comparison, we increase the value of γ for *DeepWalk*. This gives *DeepWalk* a larger training dataset (as large as all of the levels of *HARP(DW)* combined). We note that failure to increase γ in this way resulted in substantially worse *DeepWalk* (and *Node2vec*) models.

LINE. For *HARP(LINE)*, we run 50 iterations on all graph edges on all coarsening levels. For *LINE*, we increase the number of iterations over graph edges accordingly, so that the amount of training data for both models remain the same. The representation size d is set to 64 for both *LINE* and *HARP(LINE)*.

Node2vec. For *HARP(N2V)*, the parameter setting is $\gamma = 40, t = 10, w = 10, d = 128$. Similar to *DeepWalk*, we increase the value of γ in *Node2vec* to ensure a fair comparison. Both in-out and return hyperparameters are set to 1.0. For all models, the initial learning rate and final learning rate are set to 0.025 and 0.001 respectively.

Graph Coarsening

Figure 3 demonstrates the effect of our hybrid coarsening method on all test graphs. The first step of graph coarsening for each graph eliminates about half the nodes, but the number of edges only reduce by about 10% for *BlogCatalog*. This illustrates the difficulty of coarsening real-world graphs. However, as the graph coarsening process continues, the scale of all graphs drastically decrease. At level 8, all graphs have less than 10% nodes and edges left.

Visualization

To show the intuition of the *HARP* paradigm, we set $d = 2$, and visualize the graph representation generated by *HARP(LINE)* at each level.

Figure 4 shows the level-wise 2D graph embeddings obtained with *HARP(LINE)* on *Poisson 2D*. The graph layout of level 5 (which has only 21 nodes) already highly resembles the layout of the original graph. The graph layout on each subsequent level is initialized with the prolongation of the previous graph layout, thus the global structure is kept.

(a) Level 7 (b) Level 6 (c) Level 5

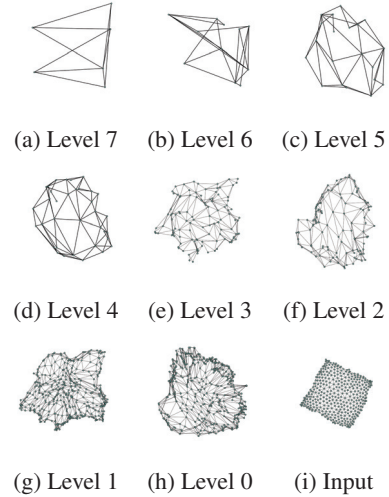(d) Level 4 (e) Level 3 (f) Level 2

(g) Level 1 (h) Level 0 (i) Input

Figure 4: Two-dimensional embeddings generated with *HARP(LINE)* on different coarsening levels on *Poisson 2D*. Level 7 denotes the smallest graph, while level 0 denotes the original graph. The last subfigure is the graph layout generated by a force-direct graph drawing algorithm.

Multi-label Classification

We evaluate our method using the same experimental procedure in (Perozzi, Al-Rfou, and Skiena 2014). Firstly, we obtain the graph embeddings of the input graph. Then, a portion (T_R) of nodes along with their labels are randomly sampled from the graph as training data, and the task is to predict the labels for the remaining nodes. We train a one-vs-rest logistic regression model with L2 regularization on the graph embeddings for prediction. The logistic regression model is implemented by LibLinear (Fan et al. 2008). To ensure the reliability of our experiment, the above process is repeated for 10 times, and the average Macro F_1 score is reported. The other evaluation metrics such as Micro F_1 score and accuracy follow the same trend as Macro F_1 score, thus are not shown.

Table 2 reports the Macro F_1 scores achieved on *DBLP*, *BlogCatalog*, and *CiteSeer* with 5%, 50%, and 5% labeled nodes respectively. The number of class labels of *BlogCatalog* is about 10 times that of the other two graphs, thus we use a larger portion of labeled nodes. We can see that our method improves all existing neural embedding techniques on all test graphs. In *DBLP*, the improvements introduced by *HARP(DW)*, *HARP(LINE)* and *HARP(N2V)* are 7.8%, 3.0% and 0.3% respectively. Given the scale-free nature of *BlogCatalog*, graph coarsening is much harder due to a large amount of star-like structures in it. Still, *HARP(DW)*, *HARP(LINE)* and *HARP(N2V)* achieve gains of 4.0%, 4.6% and 4.7% over the corresponding baseline methods respectively. For *CiteSeer*, the performance improvement is also striking: *HARP(DW)*, *HARP(LINE)* and *HARP(N2V)* outperforms the baseline methods by 4.8%, 13.6%, and 2.8%.

To have a detailed comparison between HARP and the baseline methods, we vary the portion of labeled nodes for classification, and present the macro F_1 scores in Fig-
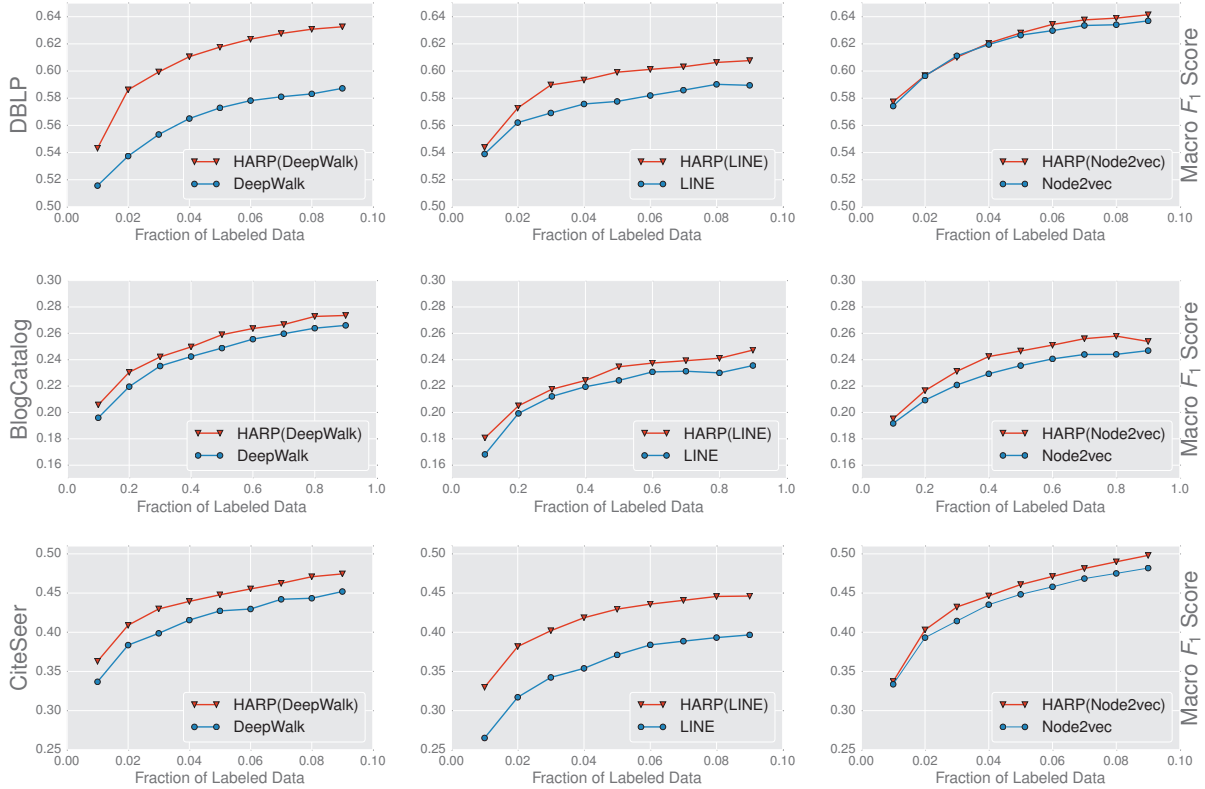
Figure 5: Detailed multi-label classification result on *DBLP*, *BlogCatalog*, and *CiteSeer*.

Algorithm	Dataset		
	DBLP	BlogCatalog	CiteSeer
DeepWalk	57.29	24.88	42.72
HARP(DW)	**61.76***	**25.90***	**44.78***
Gain of HARP[%]	**7.8**	**4.0**	**4.8**
LINE	57.76	22.43	37.11
HARP(LINE)	**59.51***	**23.47***	**42.95***
Gain of HARP[%]	**3.0**	**4.6**	**13.6**
Node2vec	62.64	23.55	44.84
HARP(N2V)	**62.80**	**24.66***	**46.08***
Gain of HARP[%]	**0.3**	**4.7**	**2.8**

Table 2: Macro F_1 scores and performance gain of *HARP* on *DBLP*, *BlogCatalog*, and *CiteSeer* in percentage. * indicates statistically superior performance to the corresponding baseline method at level of 0.001 using a standard paired t-test. Our method improves **all** existing neural embedding techniques.

ure 5. We can observe that *HARP(DW)*, *HARP(LINE)* and *HARP(N2V)* consistently perform better than the corresponding baseline methods.

DBLP. For *DBLP*, the relative gain of *HARP(DW)* is over 9% with 4% labeled data. With only 2% labeled data, *HARP(DW)* achieves higher macro F_1 score than *Deep-Walk* with 8% label data. *HARP(LINE)* also consistently outperforms *LINE* given any amount of training data, with macro F_1 score gain between 1% and 3%. *HARP(N2V)* and *Node2vec* have comparable performance with less than 5% labeled data, but as the ratio of labeled data increases, *HARP(N2V)* eventually distances itself to a 0.7% improvement over *Node2vec*. We can also see that *Node2vec* generally has better performance when compared to *DeepWalk*, and the same holds for *HARP(N2V)* and *HARP(DW)*. The difference in optimization method for Skip-gram (negative sampling for *Node2vec* and hierarchical softmax for *Deep-Walk*) may account for this difference.

BlogCatalog. As a scale-free network with complex structure, *BlogCatalog* is challenging for graph coarsening. Still, by considering both first-order proximity and second-order proximity, our hybrid coarsening algorithm generates an appropriate hierarchy of coarsened graphs. With the same amount of training data, *HARP(DW)* always leads by at least 3.0%. For *HARP(LINE)*, it achieves a relative gain of 4.8% with 80% labeled data. For *HARP(N2V)*, its gain over *Node2vec* reaches 4.7% given 50% labeled nodes.

Citeseer. For *CiteSeer*, the lead of *HARP(DW)* on Macro F_1 score varies between 5.7% and 7.8%. For *HARP(LINE)*, its improvement over *LINE* with 4% labeled data is an impressive 24.4%. *HARP(N2V)* also performs better than *Node2vec* on any ratio of labeled nodes.
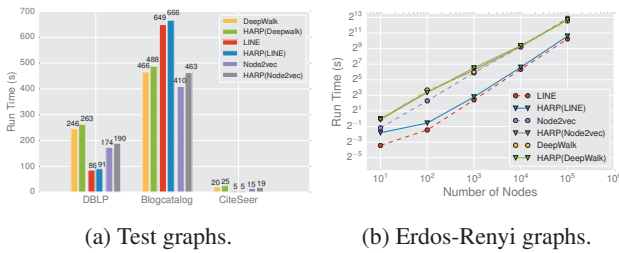
(a) Test graphs.   (b) Erdos-Renyi graphs.

Figure 6: Runtime analysis.

Scalability

We already shown that introducing *HARP* does not affect the time complexity of the underlying graph embedding algorithms. Here, we compare the actual run time of *HARP* enhanced embedding algorithms with the corresponding baseline methods on all test graphs. All models run on a single machine with 128GB memory, 24 CPU cores at 2.0GHZ with 20 threads. As shown in Figure 6a, applying *HARP* typically only introduces an overhead of less than 10% total running time. The time spent on sampling and training the Skip-gram model dominates the overall running time.

Additionally, we learn graph embeddings on Erdos-Renyi graphs with node count ranging from 100 to 100,000 and constant average degree of 10. In Figure 6b, we can observe that the running time of *HARP* increases linearly with the number of nodes in the graph. Also, when compared to the corresponding baseline method, the overhead introduces by the graph coarsening and prolongation process in HARP is negligible, especially on large-scale graphs.

Related Work

The related work is in the areas of graph representation learning and graph drawing, which we briefly describe here.

Graph Representation Learning. Most early methods treated representation learning as performing dimension reduction on the Laplacian and adjacency matrices (Belkin and Niyogi 2001; Cox and Cox 2000; Tenenbaum, De Silva, and Langford 2000). These methods work well on small graphs, but the time complexity of these algorithms is too high for the large-scale graphs commonly encountered today.

Recently, neural network-based methods have been proposed for constructing node representation in large-scale graphs. Deepwalk (Perozzi, Al-Rfou, and Skiena 2014) presents a two-phase algorithm for graph representation learning. In the first phase, Deepwalk samples sequences of neighboring nodes of each node by random walking on the graph. Then, the node representation is learned by training a Skip-gram model (Mikolov et al. 2013) on the random walks. A number of methods have been proposed which extend this idea. First, several methods use different strategies for sampling neighboring nodes. LINE (Tang et al. 2015) learns graph embeddings which preserve both the first-order and second-order proximities in a graph. Walklets (Perozzi et al. 2017) captures multiscale node representation on graphs by sampling edges from higher powers of the graph

adjacency matrix. Node2vec (Grover and Leskovec 2016) combines DFS-like and BFS-like exploration within the random walk framework. Second, matrix factorization methods and deep neural networks have also been proposed (Cao, Lu, and Xu 2015; Ou et al. 2016; Wang, Cui, and Zhu 2016; Abu-El-Haija, Perozzi, and Al-Rfou 2017) as alternatives to the Skip-gram model for learning the latent representations.

Although these methods are highly scalable, they all rely on optimizing a non-convex objective function. With no prior knowledge of the graph, the latent representations are usually initialized with random numbers or zero. With such an initialization scheme, these methods are at risk of converging to a poor local minima. *HARP* overcomes this problem by introducing a multilevel paradigm for graph representation learning.

Graph Drawing. Multilevel layout algorithms are popular methods in the graph drawing community, where a hierarchy of approximations is used to solve the original layout problem (Fruchterman and Reingold 1991; Hu 2005; Walshaw 2003). Using an approximation of the original graph has two advantages - not only is the approximation usually simpler to solve, it can also be extended as a good initialization for solving the original problem. In addition to force-directed graph drawing, the multilevel framework (Walshaw 2004) has been proved successful in various graph theory problems, including the traveling salesman problem (Walshaw 2001), and graph partitioning (Karypis and Kumar 1998).

HARP extends the idea of the multilevel layout to neural representation learning methods. We illustrate the utility of this paradigm by combining *HARP* with three state-of-the-art representation learning methods.

Conclusion

Recent literature on graph representation learning aims at optimizing a non-convex function. With no prior knowledge of the graph, these methods could easily get stuck at a bad local minima as the result of poor initialization. Moreover, these methods mostly aim to preserve local proximities in a graph but neglect its global structure. In this paper, we propose a multilevel graph representation learning paradigm to address these issues. By recursively coalescing the input graph into smaller but structurally similar graphs, *HARP* captures the global structure of the input graph. By learning graph representation on these smaller graphs, a good initialization scheme for the input graph is derived. This multilevel paradigm is further combined with the state-of-the-art graph embedding methods, namely *DeepWalk*, *LINE*, and *Node2vec*. Experimental results on various real-world graphs show that introducing *HARP* yields graph embeddings of higher quality for all these three methods.

In the future, we would like to combine *HARP* with other graph representation learning methods. Specifically, as Skip-gram is a shallow method for representation learning, it would be interesting to see if *HARP* also works well with deep representation learning methods. On the other hand, our method could also be applied to language networks, possibly yielding better word embeddings.

Acknowledgements

References

Abu-El-Haija, S.; Perozzi, B.; and Al-Rfou, R. 2017. Learning edge representations via low-rank asymmetric projections. *arXiv preprint arXiv:1705.05615*.

Belkin, M., and Niyogi, P. 2001. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *NIPS*, volume 14, 585–591.

Cao, S.; Lu, W.; and Xu, Q. 2015. Grarep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, 891–900. ACM.

Cox, T. F., and Cox, M. A. 2000. *Multidimensional scaling*. CRC press.

Fan, R.-E.; Chang, K.-W.; Hsieh, C.-J.; Wang, X.-R.; and Lin, C.-J. 2008. Liblinear: A library for large linear classification. *The Journal of Machine Learning Research* 9:1871–1874.

Fruchterman, T. M., and Reingold, E. M. 1991. Graph drawing by force-directed placement. *Software: Practice and experience* 21(11):1129–1164.

Goldberg, Y., and Levy, O. 2014. word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method. *arXiv preprint arXiv:1402.3722*.

Grover, A., and Leskovec, J. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Hu, Y. 2005. Efficient, high-quality force-directed graph drawing. *Mathematica Journal* 10(1):37–71.

Karypis, G., and Kumar, V. 1998. A parallel algorithm for multilevel graph partitioning and sparse matrix ordering. *Journal of Parallel and Distributed Computing* 48(1):71–95.

Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G. S.; and Dean, J. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, 3111–3119.

Ou, M.; Cui, P.; Pei, J.; and Zhu, W. 2016. Asymmetric transitivity preserving graph embedding. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Perozzi, B.; Al-Rfou, R.; and Skiena, S. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 701–710. ACM.

Perozzi, B.; Kulkarni, V.; Chen, H.; and Skiena, S. 2017. Don't walk, skip!: Online learning of multi-scale network embeddings. In *Proceedings of the 2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2017*, ASONAM '17, 258–265. New York, NY, USA: ACM.

Roweis, S. T., and Saul, L. K. 2000. Nonlinear dimensionality reduction by locally linear embedding. *Science* 290(5500):2323–2326.

Sen, P.; Namata, G. M.; Bilgic, M.; Getoor, L.; Gallagher, B.; and Eliassi-Rad, T. 2008. Collective classification in network data. *AI Magazine* 29(3):93–106.

Tang, L., and Liu, H. 2009. Relational learning via latent social dimensions. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, 817–826. ACM.

Tang, J.; Qu, M.; Wang, M.; Zhang, M.; Yan, J.; and Mei, Q. 2015. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*, 1067–1077. International World Wide Web Conferences Steering Committee.

Tenenbaum, J. B.; De Silva, V.; and Langford, J. C. 2000. A global geometric framework for nonlinear dimensionality reduction. *Science* 290(5500):2319–2323.

Walshaw, C. 2001. *A multilevel Lin-Kernighan-Helsgaun algorithm for the travelling salesman problem*. Citeseer.

Walshaw, C. 2003. A multilevel algorithm for force-directed graph-drawing. *Journal of Graph Algorithms Applications* 7(3):253–285.

Walshaw, C. 2004. Multilevel refinement for combinatorial optimisation problems. *Annals of Operations Research* 131(1-4):325–372.

Wang, D.; Cui, P.; and Zhu, W. 2016. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.