

# Deep Node Ranking for Neuro-symbolic Structural Node Embedding and Classification

Blaž Škrlič

*Jožef Stefan Institute, Jamova 39, 1000 Ljubljana, Slovenia  
Jožef Stefan Int. Postgraduate School, Jamova 39, 1000 Ljubljana, Slovenia*

Jan Kralj

*Jožef Stefan Institute, Jamova 39, 1000 Ljubljana, Slovenia  
Cosylab d.o.o., Ljubljana, Slovenia*

Janez Konc

*National Institute of Chemistry, Ljubljana, Slovenia*

Marko Robnik-Šikonja

*Faculty of Computer and Information Science, Ljubljana, Slovenia*

Nada Lavrač

*Jožef Stefan Institute, Jamova 39, 1000 Ljubljana, Slovenia  
University of Nova Gorica, Vipavska 13, 5000 Nova Gorica, Slovenia*

---

## Abstract

Network node embedding is an active research subfield of complex network analysis. This paper contributes a novel approach to learning network node embeddings and direct node classification using a node ranking scheme coupled with an autoencoder-based neural network architecture. The main advantages of the proposed Deep Node Ranking (DNR) algorithm are competitive or better classification performance, significantly higher learning speed and lower space requirements when compared to state-of-the-art approaches on 15 real-life node classification benchmarks. Furthermore, it enables exploration of the relationship between symbolic and the derived sub-symbolic node representations, offering insights into the learned node space structure. To avoid the space complex-

---

<sup>1</sup>blaz.skrlic@ijs.si

ity bottleneck in a direct node classification setting, DNR computes stationary distributions of personalized random walks from given nodes in mini-batches, scaling seamlessly to larger networks. The scaling laws associated with DNR were also investigated on 1488 synthetic Erdős-Rényi networks, demonstrating its scalability to tens of millions of links.

*Keywords:* network node embedding, complex networks, deep learning

---

## 1. Introduction

Numerous real-world systems consisting of interconnected entities can be represented as complex networks. Analysis of such networks provides insights into the underlying patterns applicable in various practical scenarios, including the discovery of drug targets, modelling of disease outbreaks, author profiling, modelling of transportation and the study of social dynamics [1].

Modern machine learning approaches applied to complex networks offer intriguing opportunities for developing fast and accurate algorithms that can learn based on the structural topology of a given network. Recently, approaches based on network node embedding [2, 3, 4] became prevalent for many common tasks, such as node classification, edge prediction and unsupervised node clustering (community detection). Node embedding refers to the process of learning node representations in a numeric vector format that captures the topological properties of network nodes [5]. Embeddings are useful, as vector representations are suitable for conventional machine learning algorithms capable of addressing the tasks from classification and regression to clustering.

In this work, we propose a new network node embedding and classification algorithm named *Deep Node Ranking (DNR)*, which combines efficient node ranking with the non-linear approximation power of deep neural networks. The developed framework uses deep neural networks to obtain node embeddings directly from *stationary random walk distributions* produced by random walkers with a restart with respect to individual nodes of interest. Compared to existing methods, DNR is, to our knowledge, one of the first *neuro-symbolic* node repre-

sentation learning algorithms, as it offers joint construction of low-dimensional latent representations via symbolic (inspectable) node features.

Even though there already exist embedding approaches based on higher-order random walks [2, 4] (i.e. random walkers with memory), the stationary distribution of first-order random walkers has not yet been fully explored in a deep learning setting. Widely used methods such as node2vec and struc2vec perform well; however, they do not necessarily scale to larger networks and often require extensive hyperparameter tuning for good performance. Further, these methods mostly learn representations via rather shallow, single latent layer-like optimization schemes, potentially missing the abstraction learning power of deeper neural networks. Finally, in massive networks, not all nodes need to be accounted for during representation learning – information relevant to representing a given node can depend on its relation to a small number of key nodes. We demonstrate that the neuro-symbolic paradigm offers an elegant solution to this problem via ranking-based pivoting (selection of symbolic features before deep learning), scaling to networks comprised of tens of millions of links and tens of thousands of nodes on commodity hardware. This paper also offers ablation studies of DNR’s scalability on more than 1,400 synthetic networks of different sizes – this type of analysis is seldom considered in related work.

We showcase the developed algorithm’s capabilities on the challenging problems of node classification and network visualization, highlighting its ability to learn and accurately predict node labels at scale. Further, we compiled one of the largest collections of node classification data sets and used it for empirical evaluation of the methods. Key contributions of this paper are:

1. A fast network embedding algorithm named Deep Node Ranking (DNR) based on global personalized node ranks. It performs competitively and can be used for a multitude of downstream learning tasks, including node classification, network visualization and similar. The proposed neuro-symbolic algorithm is also faster than many state-of-the-art embedding algorithms, and scales better.

2. To our knowledge, the proposed node embedding algorithms are for the first time benchmarked against contemporary approaches on such scale (15 real data sets), as commonly the algorithms are tested only on a handful of data sets.
3. We conducted an extensive empirical evaluation on 1488 synthetic networks to study the effects of node pivoting, for which we hypothesized that it substantially improves scalability.
4. The proposed DNR performs better than the competition when the labelled data is scarce (small percentage of labelled nodes).

The remainder of this work is structured as follows. In Section 2, we shortly review the related work on neuro-symbolic representation learning, network node classification and network node ranking. Section 3 presents the proposed network node embedding algorithm that combines deep neural networks with network node ranking. In Section 4, we describe the experimental setting and different non-synthetic complex networks from different domains used in the evaluation, including the newly composed data sets. The experimental results are presented in Section 5. In Section 6 we conclude the work and present plans for further work.

## 2. Background and related work

This section presents deep and neuro-symbolic learning preliminaries that describe how algorithms learn from complex networks and what is learned, followed by an overview of node ranking algorithms relevant to this work.

### 2.1. Neuro-symbolic representation learning

We first discuss the branch of methods that exploits the insights from the fields of deep learning and symbolic learning, which can be referred to as *neuro-symbolic representation learning*. This paradigm of learning has been actively studied for the past twenty years (see [6]); however, it resurged recently with many works that demonstrated this paradigm’s utility when compared to

symbolic/sub-symbolic-only learning. The interest in neuro-symbolic learning, for example, spiked recently [7] by the development of a neuro-symbolic system that partially operates via symbolic and partially via a sub-symbolic space, used to distil human-understandable concepts from images. The recent work on closing the loop between recognition (neural) and reasoning (symbolic) [8] introduced a grammar model as a symbolic prior to bridge neural perception and symbolic reasoning, alongside a top-down, human-like induction procedure. This work demonstrated that such a combined approach significantly outperforms the conventional reinforcement learning-based baselines. The Microsoft research division (MSR) recently explored the interplay between visual recognition and reasoning [9]. They introduced a framework to isolate and evaluate the reasoning aspect of visual question answering separately from its perception, followed by a calibration procedure that offers an exploration of the relation between reasoning and perception. Further, a neuro-symbolic approach to logical deduction was proposed as *Neural Logic Machines* [10]. This architecture was shown to have inductive logic learning capabilities, which was demonstrated on simple tasks such as sorting. Finally, the two recent approaches from the field of inductive logic programming (ILP) explored the interplay between the logical input structures and how they perform when associated with neural network-based learning. The Deep Relational Machines [11] were one of the first approaches to showcase the utility of combining the two paradigms. Further, the recent work of Srinivasan et al. [12] explored how Deep Relational Machines can be *explained*, emphasizing that being able to explain what a given association system does is highly relevant in e.g., the field of biomedicine.

In the last years, links between sub-symbolic and logic programming were also established. For example, the DeepProbLog system [13] demonstrates how neural predicates could be useful for constructing expressive (and short) programs for complex tasks such as image-based enumeration. Furthermore, links between statistical learning and the neuro-symbolic paradigm were also studied [14]. Finally, recent endeavours in this direction also introduce the notion of stochasticity as a programming component [15].

Albeit being actively explored, the notion of neuro-symbolic representation learning was, to our knowledge, not yet considered in the context of node representation learning, which is the key focus of this work.

## 2.2. Network node classification

Complex networks, representing real-world phenomena such as financial markets, transportation, biological interactions or social dynamics [1, 16, 17] often possess interesting properties such as scale invariance, non-trivial partitioning, presence of hub nodes, weakly connected components, heavy-tailed node degree distributions, occurrence of communities, significant motif pattern counts, etc. [18, 19]. *Learning from complex networks* considers different aspects of complex networks, e.g., network structure and node labels, which are used as inputs to machine learning algorithms to address learning tasks such as link prediction, node classification, etc.

In this paper we focus on node classification, i.e. the problem of classifying nodes into two or more distinct classes. This task is considered as semi-supervised learning, given that the whole network is used to obtain representations of individual nodes, from which the network classification model is learned. Information propagation algorithms [20] propagate label information via nodes' neighbors until all nodes are labeled. These algorithms learn in an *end-to-end* manner, meaning that no intermediary representation of a network is first obtained and subsequently used for training e.g., a classifier.

Another class of node classification algorithms learns node labels from node embeddings, i.e. node representations in vector form [21]. Here, the whole network is first transformed into an information-rich, compact low-dimensional representation (a dense matrix). This representation serves as an input to plethora of more general machine learning approaches that can be used for node classification.

We distinguish between two main branches of embedding-based learning algorithms, discussed next: graph neural networks and random walk-based learners. Graph Neural Networks (GNNs), introduced in the recent years, attempt

to incorporate a given network’s adjacency structure as new neural network layers. Amongst first such approaches were the Graph Convolutional Networks (GCNs) [22], their generalization with the attention mechanism [23], and the more recent isomorphism-based variants with provable properties [24]. Treating the adjacency structure as a neural network has also shown promising results [25]. The key characteristic of this branch of methods is their capability to account for *node features* by multiplication of the normalized adjacency matrix as part of a special layer during learning from features. On the other hand, if node features are not available, which is the case with the majority of freely available public data sets, more optimized methods focused on *structure-based learning* are preferred. For example, the LINE algorithm [26] uses the network’s *eigendecomposition* in order to learn a low dimensional network representation, e.g., a representation of the network’s nodes in 128 dimensions instead of the dimension that matches the number of nodes. Approaches that use random walks to sample the network include DeepWalk [5] and its generalization node2vec [2]. It was recently proven that DeepWalk, node2vec, and LINE can be reformulated as implicit matrix factorization [27]. Furthermore, approaches such as struc2vec [4] demonstrated how more complex, multilayer structure can be compressed into node representations for better performance. Despite many promising approaches developed, a recent extensive evaluation of network embedding techniques [28] suggests that node2vec [2] remains one of the best embedding approaches for the task of **structural** node classification.

### 2.3. Network node ranking

Node ranking algorithms assess the relevance of a node in a network either globally (relative to the whole network) or locally (relative to a sub-network) by assigning a *score* (i.e. *rank*) to each node in the network. In this work we only consider node ranking algorithms that compute a local relevance score of a node based on its direct neighborhood. The key such node ranking algorithm is the Personalized PageRank (P-PR) algorithm [29], sometimes referred to as random walk with restart [30]. Personalized PageRank uses random walks to calculate

the relevance of nodes in a network. It obtains the stationary distribution of a random walk that starts at a given node. The P-PR-based approaches were used successfully to study cellular networks, social phenomena [31], and many other real-world networks [32]. Efficient implementation of P-PR algorithms remains an active research field, for example, the recent bidirectional variation of the P-PR was introduced to speed up the node ranking process [33]. The obtained stationary distribution of a random walk can be used directly for network-based learning tasks, as demonstrated in HINMINE methodology [34].

#### 2.4. Combining node ranking and node representation learning

Exploring the ideas of augmenting learning with ranking was in the recent years explored in the context of graph neural networks. For example, ranking was used to prioritize propagation [35] and to scale graph neural networks [36]. A similar idea was exploited by [37], where a more efficient propagation scheme was proposed by using node ranking. The proposed Deep Node Ranking algorithm is novel with respect to these works, as it exploits both the fast, parallel personalized node rank computation and the *representation learning* power of deep neural networks.

### 3. Deep Node Ranking

This section presents the Deep Node Ranking (DNR) algorithm for neuro-symbolic structural network node embedding and end-to-end node classification (overview shown in Figure 1). The name of the algorithm, Deep Node Ranking, reflects the two main ideas considered: network node ranking step (*symbolic*) and the subsequent deep neural network learning step (*neural/sub-symbolic*). In the first step of DNR, personalized node ranks are computed for each node, resulting in Personalized PageRank with shrinking (P-PRS) vectors. These vectors are symbolic, as each dimension corresponds to a given node. In the second step, the P-PRS vectors are considered by a deep neural network consisting of at least a single dense embedding layer of size equal to the predefined

embedding dimension. This embedding is sub-symbolic, as one can no longer interpret the meaning of individual (latent) dimensions. The third, output step, consists either of an output layer with the number of its neurons equal to the number of target classes (top) enabling direct classification of nodes or embeddings (bottom), which correspond to the embedding layer from Step 2. The obtained embeddings can be used for downstream machine learning tasks, such as classification, network visualization, and comparison.

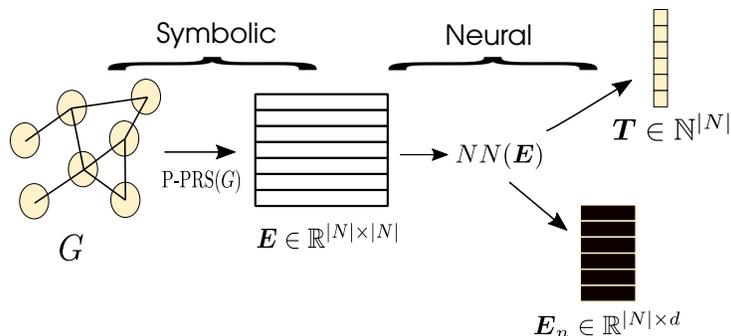


Figure 1: Deep Node Ranking algorithm. The symbolic part of the algorithm computes personalized PageRank vectors ( $\mathbf{E}$ ), which are subsequently compressed with a neural network (NN) into either a lower dimensional representation ( $\mathbf{E}_n$ ), or used for end-to-end classification ( $\mathbf{T}$ ). Note that the intermediary symbolic P-PRS based representation remains interpretable (features are nodes).

The DNR algorithm, which takes as input a partially labeled complex network, consists of three steps outlined below.

1. Network node ranking results in learned node representations, obtained by using the Personalized PageRank with Shrinking (P-PRS) algorithm. This step results in a matrix of P-PRS vectors of dimension  $|\mathcal{N}|$ .
2. Representation distillation. A neural network architecture compresses the prepared personalized PageRank vectors into compact representations (of dimension  $d$ ).
3. Output phase. The output of the network can be either node classification, that is, direct learning of node labels or a collection of low-dimensional

node representations.

### *3.1. Node ranking with the Personalized PageRank with Shrinking algorithm*

We first build and upgrade the representation learning idea, introduced in previous work [34], where node representations are obtained via personalized node ranking. The following description represents a substantial theoretical extension of the original idea, which was further parallelized for the first time in this work. Furthermore, this work introduces node pivoting, which substantially speeds up the personalized ranking time. We consider a version of the Personalized PageRank [38] algorithm to which we refer to as Personalized PageRank with Shrinking (P-PRS) (algorithm 1). This variant of the widely known PPR algorithm produces node representations (or P-PRS vectors) by simulating random walks for each node of the input network. Compared to the network adjacency matrix, P-PRS vectors contain traversal information for each node, reflecting its ranking based on a node’s position with respect to a given network’s topology.

---

**Algorithm 1: P-PRS: Personalized PageRank with Shrinking**

---

**Data:** A complex network's adjacency matrix  $A$ , with nodes  $N$  and edges  $E$ , starting node  $u \in N$

**Parameters** : damping factor  $\delta$ , spread step  $\sigma$ , spread percent  $\tau$  (default 50%), stopping criterion  $\epsilon$

**Result:** P-PRS $_u$  vector describing stationary distribution of random walker visits with respect to  $u \in N$

```
1  $A \leftarrow \text{toRightStochasticMatrix}(A);$  ▷ Transpose and normalize rows of  $A$ 
2  $\text{core\_vector} \leftarrow [0, \dots, 0];$  ▷ Initialize zero vector of size  $|N|$ 
3  $\text{core\_vector}[u] \leftarrow 1; \text{rank\_vector} \leftarrow \text{core\_vector}; v \leftarrow \text{core\_vector};$ 
4  $\text{steps} \leftarrow 0;$  ▷ Shrinking part
5  $\text{nz} \leftarrow 1;$  ▷ Number of non-zero P-PRS values
6 while  $\text{nz} < |N| \cdot \tau \wedge \text{steps} < \sigma$  do
7    $\text{steps} \leftarrow \text{steps} + 1;$ 
8    $v = v + A \cdot v;$  ▷ Update transition vector
9    $\text{nzn} \leftarrow \text{nonZero}(v);$  ▷ Identify non-zero values
10  if  $\text{nzn} = \text{nz}$  then
11     $\text{shrink} \leftarrow \text{True};$ 
12    end while;
13  end
14   $\text{nz} \leftarrow \text{nzn};$ 
15 end
16 if  $\text{shrink}$  then
17    $\text{toReduce} \leftarrow \{i; v[i] \neq 0\};$  ▷ Indices of non-zero entries in vector  $v$ 
18    $\text{core\_rank} \leftarrow \text{core\_rank}[\text{toReduce}]; \text{rank\_vector} \leftarrow \text{rank\_vector}[\text{toReduce}];$ 
19    $A \leftarrow A[\text{toReduce}, \text{toReduce}];$  ▷ Shrink a sparse adjacency matrix
20 end
21  $\text{diff} \leftarrow \infty;$ 
22  $\text{steps} \leftarrow 0;$  ▷ Node ranking - power iteration
23 while  $\text{diff} > \epsilon \wedge \text{steps} < \text{max\_steps}$  do
24    $\text{steps} \leftarrow \text{steps} + 1;$ 
25    $\text{new\_rank} \leftarrow A \cdot \text{rank\_vector}; \text{rank\_sum} \leftarrow \sum_i \text{rank\_vector}[i];$ 
26   if  $\text{rank\_sum} < 1$  then
27      $\text{new\_rank} \leftarrow \text{new\_rank} + \text{start\_rank} \cdot (1 - \text{rank\_sum});$ 
28   end
29    $\text{new\_rank} \leftarrow \delta \cdot \text{new\_rank} + (1 - \delta) \cdot \text{start\_rank};$ 
30    $\text{diff} \leftarrow \|\text{rank\_vec} - \text{new\_rank}\|;$  ▷ Norm computation
31    $\text{rank\_vec} \leftarrow \text{new\_rank};$ 
32 end
33 if  $\text{shrink}$  then
34    $\text{P-PRS}_u \leftarrow [0, \dots, 0];$  ▷ Zero vector of dimension  $|N|$ 
35    $\text{P-PRS}_u[\text{toReduce}] \leftarrow \text{rank\_vec};$ 
36 else
37    $\text{P-PRS}_u \leftarrow \text{rank\_vec}$ 
38 return P-PRS $_u$ ;
```

---

The P-PRS algorithm consists of two main parts:

1. In the first part named the *shrinking step* (lines 5–20 of Algorithm 1), in each iteration, the walker spreads from nodes with non-zero PageRank values to their neighbors.
2. In the second part of the algorithm, named the *P-PRS computation step* (lines 23–38 of Algorithm 1), P-PRS vectors corresponding to individual network nodes are computed using the power iteration method (Eq. 1).

**Shrinking step.** In the shrinking step we take into account the following:

- If no path exists between node  $u$  (the starting node) and node  $i$ , the P-PRS value assigned to node  $i$  will be zero.
- The P-PRS values for nodes reachable from  $u$  will be equal to P-PRS values calculated for a reduced network  $G_u$ , obtained from the original network by only accounting for the subset of nodes reachable from  $u$  and connections between them (lines 6–15 in Algorithm 1).

If the network is strongly connected,  $G_u$  will be equal to the original network, yielding no change in performance compared to the original P-PRS algorithm. However, if the resulting network  $G_u$  is smaller, the calculation of P-PRS values will be faster as they are calculated on  $G_u$  instead of on the whole network. In our implementation, we first estimate if network  $G_u$  contains less than 50% (i.e. spread percentage) of nodes of the whole network (lines 6–14 in Algorithm 1). This is achieved by expanding all possible paths from node  $i$  and checking the number of visited nodes in each step. If the number of visited nodes stops increasing after a maximum of 15 steps, we know we have found a network  $G_u$ , and we count its nodes. If the number of nodes is still increasing, we abort the calculation of  $G_u$ . We limit the maximum number of steps because each step of computing  $G_u$  is computationally comparable to one step of the power iteration used in the PageRank algorithm [38] which converges in about 50 steps. Therefore we can considerably reduce the computational load if we

limit the number of steps in the search for  $G_u$ . Next, in lines 16–20, the P-PRS algorithm shrinks the personalized rank vectors based on non-zero values obtained as the result of the shrinking step.

**P-PRS computation step.** In the second part of the algorithm (lines 23–38), node ranks are computed using the power iteration (Eq. 1), whose output consists of P-PRS vectors. An example stationary distribution is shown in Figure 2 for the *Cora* network.

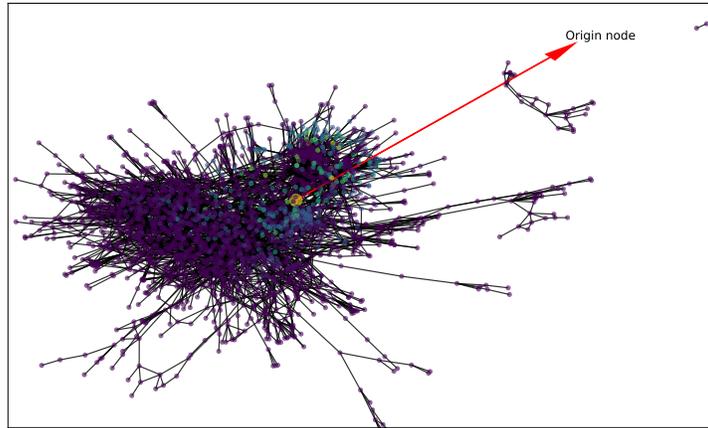


Figure 2: Stationary distribution, visualized (*Cora*). The origin node is the large yellow node pointed from. The upper right part of the network contains nodes that a simulated walker (from the origin node) likely ends up at (green-colored nodes).

For each node  $u \in V$ , a feature vector  $\gamma_u$  (with components  $\gamma_u(i)$ ,  $1 \leq i \leq |N|$ ) is computed by calculating the stationary distribution of a random walk, starting at node  $u$ . The stationary distribution is approximated using power iteration, where the  $i$ -th component  $\gamma_u(i)^{(k)}$  of approximation  $\gamma_u^{(k)}$  is computed in the  $k + 1$ -st iteration as follows:

$$\gamma_u(i)^{(k+1)} = \alpha \cdot \sum_{j \rightarrow i} \frac{\gamma_u(j)^{(k)}}{d_j^{out}} + (1 - \alpha) \cdot v_u(i); k = 1, 2, \dots \quad (1)$$

The number of iterations  $k$  is increased until the visit distribution converges to

the final stationary distribution vector (P-PRS value for node  $i$ ). In the above equation,  $\alpha$  is the damping factor that corresponds to the probability that a random walk follows a randomly chosen outgoing edge from the current node rather than restarting its walk. The summation index  $j$  runs over all nodes of the network that have an outgoing connection towards  $i$ , (denoted as  $j \rightarrow i$  in the sum), and  $d_j^{out}$  is the out-degree of node  $d_j$ . Term  $v_u(i)$  is the restart distribution that corresponds to a vector of probabilities for a walker’s return to the starting node  $u$ , i.e.  $v_u(u) = 1$  and  $v_u(i) = 0$  for  $i \neq u$ . This vector guarantees that the walker will jump back to the starting node  $u$  in case of restart.<sup>2</sup>

In a single iteration ( $k \rightarrow k + 1$ ), all stationary distribution vector components  $\gamma_u(i)$ ,  $1 \leq i \leq |N|$ , are updated which result in the P-PRS vector  $\gamma_u^{(k+1)}$ . Increasing  $k$  thus leads to the  $\gamma_u^{(k)}$  eventually converging to the PageRank  $\gamma_u$  of a random walk starting from node  $u$  (see Algorithm 1). Eq. 1 is optimized by using the *power iteration*, which is especially suitable for large sparse matrices, since it does not rely on spatially expensive matrix factorization in order to obtain the eigenvalue estimates.<sup>3</sup>

The P-PRS algorithm simulates a first-order random walk in which no past information is incorporated in the final stationary distribution. The time complexity of the described P-PRS algorithm with shrinking for  $k$  iterations is  $\mathcal{O}(|N|(|E| + |N|) \cdot k)$  for the whole network, and  $\mathcal{O}(|E| + |N|) \cdot k$  for a single node.

### 3.2. Additional shrinking by rank-based pivoting

We next present an additional step of shrinking explored as part of this work that offers scaling to very large networks. Recall (Algorithm 1) that the PageRank iteration, if the network is reduced, operates on the smaller adjacency

---

<sup>2</sup>If the binary vector was composed exclusively of ones, the iteration would compute the global PageRank vector, and Eq. 1 would reduce to the standard PageRank iteration.

<sup>3</sup>The power iteration (Eq. 1) converges exponentially, that is, the error is proportional to  $\alpha^k$ , where  $\alpha$  is the damping factor and  $k$  is the iteration number.

matrix indexed via the set of nodes toReduce. This step, as offered in the Algorithm 1, prunes out the nodes unreachable via traversal from the current node. This step preserves the computed vectors’ properties, however, it does not guarantee asymptotically faster computation and is largely dependent on a given network’s structure. For DNR to scale to very large networks, a more lossy selection scheme can be adopted. Recall the toReduce, the set of nodes that define the final set of iterations that yield a given node’s P-PRS-based representation. The idea discussed next defines the set toReduce upfront; the size of this set is parametrized with an integer value  $p$  (number of *pivot nodes*). Members of this set are obtained as follows. We hypothesize that two main types of *pivot nodes* need to be preserved in toReduce; namely, the nodes local to the node of interest, but also global nodes (via their relation to the node of interest). To address both concerns, we first define a given target node  $u$ ’s neighbors as  $\text{Ne}(u)$ . Next, we define with  $\text{argSortDes}(\text{PR}(\text{G}))$  the set of initial nodes, sorted by their PageRank values in descending order. Note that this step takes only  $\mathcal{O}(|N| \log |N| + |E|)$  steps, and as such, scales to very large networks. The final set of toReduce is constructed by first including all nodes from the neighborhood, followed by the global nodes which are not already in the neighborhood until the set is of cardinality  $p$ . We can formally define the set of  $\text{toReduce}_u$  pivot nodes as the first  $p$  nodes of the ordered union of the neighborhood and the top-ranked nodes, i.e.

$$T^u = \{a \in \text{Ne}(u), b \in \text{argSortDes}(\text{PR}(\text{G})) \setminus \text{Ne}(u)\}$$

$$\text{toReduce}_u = \{T_1^u, T_2^u, \dots, T_p^u\}.$$

Here,  $T^u$  is the ordered union set (combined ordered sets), and  $T_i^u$  the  $i$ -th element of that set (obtained with respect to node  $u$ ). This heuristic selection of pivot nodes implies local neighborhood for low values of  $p$ , and mixed neighborhood for larger  $p$  values. The computed (symbolic) node representations are used as inputs to the subsequent step of (neural) representation compression.

The advantage of the deep neural network architecture discussed in the following section is that it can learn incrementally, from small batches of calculated

P-PRS vectors. In contrast, the previously developed HINMINE approach [34] requires that all P-PRS vectors are calculated prior to learning, which is due to HINMINE using the  $k$ -nearest neighbors and support vector machine classifiers. This incurs substantial space requirements as the P-PRS vectors for the entire network require  $\mathcal{O}(|N|^2)$  space. The DNR algorithm presented here uses a deep neural network instead, which can take as small input batches of P-PRS vectors. Therefore, only a small percentage of vectors need to be computed before the second step of the algorithm (*neural network training*) can begin. This results in improved space and time complexities of the learning process.

### 3.3. Node representation learning

In this section we address the second step of Deep Node Ranking algorithm (outlined in Figure 1) – the incremental compression of batches of personalized PageRank vectors via neural network learning. We next discuss the key formalisms used to describe the two types of learning implemented as part of DNR. We can formalize the key idea underlying DNR as the following mapping

$$\text{DNR} : \underbrace{\mathbb{R}^{|N| \times |N|}}_{\text{Adjacency matrix}} \xrightarrow{\text{P-PRS}} \underbrace{[0, 1]^{|N| \times |N|}}_{\text{P-PRS vectors}} \xrightarrow{\text{NN}_f} \underbrace{\mathbb{R}^{|N| \times d}}_{\text{Node embeddings}},$$

where  $d$  represents a *latent dimension*,  $N$  the set of nodes and DNR the mapping approximated by the proposed approach. Note how the second space consists of visit *probabilities* with respect to individual nodes. We will next focus on the two mapping methods displayed in the scheme above; P-PRS and  $\text{NN}_f$ .

The first mapping (P-PRS) takes as input the network adjacency matrix and, if executed for each node, outputs the same dimensional matrix which contains richer, walk convergence-based information describing individual nodes (instead of only their neighbors). The initial adjacency can be stored as a sparse data structure, requiring only  $\mathcal{O}(|E|)$  space. However, the probability matrix is commonly dense (with the exception of nodes in different components, for example),  $\mathcal{O}(|N|^2)$  space can already pose a problem to the method’s utility. To address this concern, we can consider *batches* of nodes ( $b$ ) from the first mapping

onwards, potentially at no point requiring the full dense  $\mathcal{O}(|N|^2)$  matrix. The first mapping adheres to this implementation due to the fact that with respect to individual nodes, P-PRS vectors can be obtained *independently*. We denote with  $\text{P-PRS}_b$  a produced *batch* of such vectors. The union of all such batches (forming the set  $B$ ) can be used to construct the whole probability matrix, i.e.

$$[0, 1]^{|N| \times |N|} = \left|_{b \in B} \text{P-PRS}_b(A), \quad (2)$$

where  $A$  represents the adjacency matrix and  $|_b$  the concatenation alongside the first axis. Here, we assume the batches preserve the order of input nodes. The same property holds for transforming the  $b \times |N|$ -dimensional vectors into  $b \times d$  dimensional ones with a *trained* neural network  $\text{NN}_f^b$ . Similarly to the E.q. 2, the final matrix can be written as

$$\mathbb{R}^{|N| \times d} = \left|_{b \in B} \text{NN}_f^b(\text{P-PRS}_b(A)).$$

The two equations above assume projection-ready mapping methods ( $\text{NN}_f$  and P-PRS). The probability vector computation P-PRS indeed requires no additional training. However, this is not the case for the neural network NN. Note that we denoted with  $\text{NN}_f$  only the forward pass up to the hidden layer with  $d$  outputs – the embedding. To describe the whole process, the missing point remains the *neural network training*. Denoted with NN, we represent a single epoch (forward and backward pass) of training the neural network (for all nodes). If we denote with  $\omega$  the number of epochs required to train either an autoencoder-like, or an end-to-end architecture ( $d$  is in this case the number of classes), DNR requires  $\mathcal{O}(\omega \cdot (\text{NN} + |N|(|E| + |N|)))$  operations. Furthermore, if the whole probability matrix fits into memory, the second product is decoupled, making the full probability matrix computed only once, resulting in complexity  $\mathcal{O}(\omega \cdot \text{NN} + |N|(|E| + |N|))$ . The initial complexity which re-computes the rank vectors for each batch ( $b$ ) has, compared to the pre-computed rank matrix version lower space complexity, i.e.  $\mathcal{O}(b \cdot |N|) \ll \mathcal{O}(|N|^2)$ . This analysis demonstrates that for larger networks, additional computation needs to be performed in order to maintain the DNR’s space complexity. Finally, the node pivoting

scheme similarly reduces the space complexity of the rank matrix computation from  $\mathcal{O}(|N|^2)$  to  $\mathcal{O}(|N| \cdot p)$  ( $p$  is the number of pivot nodes). Similarly, the time complexity reduces linearly ( $|N| \rightarrow p$ ) for the ranking step. Hence, the pivoting scheme was hypothesized to improve both space and time-related performances substantially. Having discussed the coupled and the de-coupled (memoization) variants of DNR, it is apparent that the low space version will take much longer to compute compared to the memory-intensive version. To fine-tune this to a given hardware setting, DNR is able to estimate the approximate RAM utilization by assuming 32-bit floating point precision and takes as hyperparameter an integer number denoting the upper RAM bound. Should this bound be exceeded, the memory-efficient version is considered, and the faster one otherwise. In this work, we set this bound to 16GB.

#### 3.4. *DNRNet: A neural network architecture*

In the previous section, a description of the core feature construction process based on personalized node ranking was described alongside its time and space complexities. We next discuss in more detail the considered neural network architecture and the training regime, which is also a contribution of this work.

We are interested in compressing the P-PRS-based representation (Equation 2) we henceforth refer to as  $\mathbf{P}$ . The goal of the designed neural network is to compress this representation from dimension  $|N|$  to  $d$ , in *unsupervised* manner. To achieve this compression, we implemented an autoencoder-like architecture

with a forward pass defined as follows (note the indexing).

$$\begin{aligned}
 l_i &= \text{Dropout}(\text{ELU}(\mathbf{W}_{\mathbf{d}}^T \cdot \mathbf{P} + b_i)) \\
 h_1 &= \text{ELU}(\mathbf{W}_{\mathbf{d1}}^T \cdot l_i + b_{h1}) \\
 &\dots && \text{Initial embedding order} \\
 h_k &= \text{ELU}(\mathbf{W}_{\mathbf{dk}}^T \cdot h_{k-1} + b_{hk}) \\
 r_1 &= \text{ELU}(\mathbf{W}_{\mathbf{rk}}^T \cdot l_i + b_{r1}) \\
 &\dots && \text{Reversed embedding order} \\
 r_k &= \text{ELU}(\mathbf{W}_{\mathbf{r1}}^T \cdot r_{k-1} + b_{rk}) \\
 l_o &= \mathbf{W}_{\mathbf{o}}^T \left( \frac{1}{2} \cdot r_k \oplus h_k \right).
 \end{aligned}$$

The first part of the architecture projects and activates the input probabilities ( $\mathbf{P}$ ) to a lower dimension ( $d$ ). The ELU activation is defined as

$$\text{ELU}(x) = \begin{cases} x; & x > 0 \\ \alpha \cdot (e^x - 1); & x \leq 0 \end{cases} .$$

The parameter  $\alpha$  was set to 1 throughout this work. The inner part of the architecture consists of multiple same-dimensional ( $d$ ) layers, which refine the representation. The final layer projects the refined representation back to the initial dimension ( $|N|$ ). The key component is the regularization (dropout) prior to the embedding layers, as it notably improved the architecture’s stability during design. The loss function used is the Smooth L1 Loss defined as:

$$\mathcal{L}_n = \begin{cases} \frac{1}{2} \cdot (x_n - y_n)^2 / \beta; & |x_n - y_n| < \beta \\ |x_n - y_n| - \frac{1}{2} \cdot \beta; & \text{otherwise} \end{cases}$$

Here,  $x_n$  represents the prediction,  $y_n$  the actual value and  $\beta$  a parameter (set to 1.0 in this work). The loss is averaged on the batch level. The key novelty of the proposed neural network-based compression is not the architecture but the way *forward passes are conducted*. We impose an additional constraint on

the intermediary representations by implementing the forward pass so that it includes multiplication *in both ways* across the hidden layers (note the shared parameters – there is no weight duplication). This means that each forward pass incorporates a scenario where the first hidden layer is first, but also last in the forward pass (inverted latent space); with this, we enforce reverse consistency, as *all* intermediary representations are used to obtain the final embedding. This is possible due to the symmetric nature of the activation-dense layers – inverting the order during the forward pass amplifies the effect of different hidden layers with the same type of output. The  $\oplus$  denotes the Hadamard summation (elementwise). Note that such inverse projections during the same forward pass are possible because the dimensionalities of the intermediary representations are all the same (*d*). A schematic overview of this idea is shown in Figure 3.

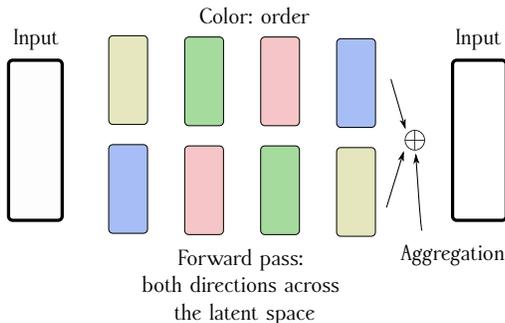


Figure 3: The forward pass with the inclusion of reversed latent spaces.

Lastly, we discuss how the final representations (node embeddings) are obtained. Recall that  $h_1, \dots, h_k$  represent the outputs of the intermediary embedding layers. The final node representations ( $\mathbf{E}$ ) are obtained by performing Hadamard summation across these intermediary representations and dividing with the number of hidden outputs, i.e.,

$$\mathbf{E} = k^{-1} \cdot \bigoplus_i \mathbf{h}_i.$$

Schematic overview of this process is shown in Figure 4. The main reason

such multi-space aggregation is conducted is that the information used for reconstructing the origin rank space is likely distributed across all hidden layers, implying that by considering, e.g., only the last layer, valuable parts of the final representation could be lost. This idea was inspired by how representations are obtained from contextual language models [39].

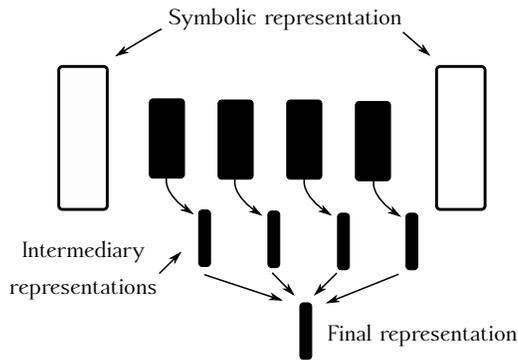


Figure 4: DNRNet’s representation construction process. The final representation ( $\mathbf{E}$ ) is obtained as an aggregate of *all* relevant intermediary layers.

#### 4. Data sets and experimental setting

This section first describes the data sets used, the experimental setting and the DNR implementations tested together with their hyperparameters, followed by a description of the compared baseline approaches.

##### 4.1. Data sets

We evaluated the proposed approach on 15 real-world complex networks, three of them introduced in this work, which is one of the largest collections of complex networks for the task of node classification. The *Homo Sapiens* (proteome) [40], POS tags [41] and Blogspot data sets [42] are used in the same form as in [2].

The *Homo sapiens* data set represents a subset of the human proteome, i.e. a set of interacting proteins. The sub-network consists of all proteins for

which biological states are known [43]. The goal is to predict protein function annotations. The POS data set represents part-of-speech tags obtained from Wikipedia—a co-occurrence network of words appearing in the first million bytes of the Wikipedia dump [41]. Thus, different POS tags are predicted. The Blogspot data set represents a social network of bloggers (Blogspot website) [42]. The labels represent bloggers’ interests inferred through the metadata provided by the bloggers. The CiteSeer citation network consists of scientific publications classified into one of the six classes (categories) [44]. The Cora citation network consists of scientific publications classified into one of seven classes (categories) [44]. The E-commerce network is a heterogeneous network connecting buyers with different products. As DNR and the compared baseline algorithms operate on homogeneous networks, the E-commerce network was transformed to a homogeneous network prior to learning using a term frequency weighting scheme [34]. The created edges represent mutual purchases of two persons, i.e. two customers are connected if they purchased an item from the same item category. We refer the interested reader to [34] for a detailed description of the data set and its transformation to a homogeneous network. The two-class values being predicted correspond to the buyers’ gender. The *film*, *squirrel*, *chameleon*, *wisconsin*, *texas* and *cornell* data sets are based on a recent study about geometric deep learning [45]. Given that some of the data sets have features, and the purpose of this paper is structure-only learning, instead of neglecting the feature spaces, we converted them into weights between nodes as follows. If the cardinality of the feature spaces of a given node was the same for all nodes, we computed the weights as inverse Euclidean distances with one added to the denominator (similarities). If the features were sets, we computed the weights as cardinalities of the intersection sets between pairs of nodes.

One of the contributions of this work is also three novel node classification data sets, which we constructed as follows. Two data sets are related to Bitcoin trades [46]. The two networks correspond to transactions within two different platforms, namely Bitcoin OTC and Bitcoin Alpha. Each edge in this network

represents a transaction along with an integer score denoting trust in the range  $[-10, 10]$  (zero-valued entries are not possible). We reformulate this as a classification problem by collecting the trust values associated with individual nodes and considering them as target classes. The resulting integer values can thus belong to one of the 20 possible classes. Note that more than a single class is possible for an individual node, as we did not attempt to aggregate trust scores for individual nodes.

The *ions* data set is based on the recently introduced protein-ion binding site similarity network [47]. The network was constructed by structural alignment using the ProBiS family of algorithms [48, 49, 50] where all known protein-ion binding sites were considered. The obtained network was pruned for structural redundancy as described in [47]. Each node corresponds to one of 12 possible ions, and each weighted connection corresponds to the ion-binding site similarity between the two considered binding sites. Overall, this is to date one of the largest collections of structure-only node classification benchmark data sets.

The considered data sets are summarized in Table 1. In the table, CC denotes the number of connected components. The clustering coefficient measures how nodes in a graph tend to cluster together and is computed as the ratio between the number of closed triplets and the number of all triplets. The network density is computed as the number of actual connections divided by all possible connections. The mean degree corresponds to the average number of connections of a node. Links to data sets, along with other material presented in this paper, are discussed in Section 7.

Furthermore, to test the DNR’s scalability, we created 1488 Erdős-Rényi networks in node range from 2,500 to 35,000 in the increments of 1,000 with different seeds and the probability parameter set to 0.05 (sparser networks).

#### 4.2. Experimental setting

In this section, we describe the experimental setting used to evaluate the proposed method against the existing baselines.

There are two main evaluation aspects relevant to this paper; investigating

Table 1: Networks used in this study and their basic statistics.

Name	#Classes	#Nodes	#Edges	Mean deg	CC	CCoef	Density
<i>cornell</i>	4	183	280	3.06	1	0.17	0.0168
<i>texas</i>	4	183	295	3.22	1	0.20	0.0177
<i>wisconsin</i>	4	251	466	3.71	1	0.21	0.0149
<i>ions</i>	12	1969	16092	16.35	326	0.53	0.0083
<i>chameleon</i>	4	2277	31421	27.60	1	0.48	0.0121
<i>cora</i>	7	2708	5278	3.90	78	0.24	0.0014
<i>citeseer</i>	6	3327	4676	2.81	438	0.14	0.0008
<i>Bitcoin_alpha</i>	20	3783	14124	7.47	5	0.18	0.0020
<i>Homo_sapiens</i>	50	3890	38739	19.92	35	0.15	0.0051
<i>POS</i>	40	4777	92517	38.73	1	0.54	0.0081
<i>squirrel</i>	4	5201	198493	76.33	1	0.42	0.0147
<i>Bitcoin</i>	20	5881	21492	7.31	4	0.18	0.0012
<i>film</i>	4	7600	14056	3.70	1975	0.04	0.0005
<i>Blogspot</i>	39	10312	333983	64.78	1	0.46	0.0063
<i>ecommerce_tf</i>	2	29999	178608	11.91	8304	0.48	0.0004

the quantitative performance of embeddings on a given downstream task and computation time. To assess the classification performance, we use the same evaluation scheme as in related work on node classification [26, 2, 5, 51]. Here, as all methods for node embedding construction are unsupervised, an embedding is first constructed and used as input to a logistic regression-based classification scheme suitable for multiclass and multilabel classification tasks.

We repeated the classification experiments five times and averaged the results to obtain stable performance estimates with corresponding variabilities. The performance of trained classifiers was evaluated by using micro and macro  $F_1$  scores, as these two measures are used in the majority of related node classification studies [26, 2, 5, 51].

Due to many classifier comparisons, we utilize the Friedman test with Nemenyi *post hoc* correction to compute the statistical significance of the differences. The results are visualized as critical difference diagrams, where average

ranks of individual algorithms according to scores across all data set splits are presented [52]. The selected algorithms are also compared via Bayesian hierarchical t-test [53] with a prior value of  $\rho = 0.8$  and rope region value set to 2%. All experiment repetitions were used for posterior sampling.

All experiments were conducted on a machine with 64GB RAM, 6 core Intel(R) Core(TM) i7-6800K CPU @ 3.40GH with a Nvidia 1080 GTX GPU. As the maximum amount of RAM available for all approaches was 64GB, the run is marked as unsuccessful, should this amount be exceeded. Further, we gave each algorithm at most five hours for learning the embeddings and subsequent classification. We selected these constraints as the networks used are of medium size, and if a given method cannot work on these networks, it will not scale to larger networks; e.g., social networks with millions of nodes and tens of millions, or even billions of edges without substantial adaptation of the method. The unsuccessful runs were replaced with a random embedding. Node ranking was implemented by using sparse matrices from the SciPy module [54] and the PyTorch library [55].

### 4.3. DNR implementations

We implemented the following variants of DNR, each emphasizing a different aspect of the algorithm.

The **DNR** represents a default DNR implementation with no node pivoting, two hidden layers, trained for at most 100 epochs with the stopping criterion of five epochs. The learning rate was set to 0.01 and adaptively decreased throughout the training. The upper memory bound was set to 16GB, meaning that networks that would require more space would be computed incrementally, on the fly, reducing the space but increasing the computation time. The latent dimension was for this and all other embedding-based methods set to 128 (as also seen in related work). The **DNR4** architecture includes four hidden layers instead of two, and **DNR8** eight hidden layers. The **DNRPH** is a DNR variant with the pivoting node number set to  $|N|/2$ . The **DNRPQ** to  $|N| \cdot 0.75$  and **DNRPM** to  $\sqrt{|N|}$ . All pivot number estimates were rounded to the nearest

integer. Finally, we implemented the symbolic-only learner we refer to as **DNR-symbolic**, which outputs the  $\mathbf{E}_s \in |N|^2$  matrix of personalized rank vectors (symbolic part of full DNR).

The P-PRS algorithm parameters (constant throughout all experiments) were set as follows.  $\epsilon$ , the error bound, which specifies the end of an iteration, was set to  $10^{-6}$ . Max steps, the number of maximum steps allowed during one iteration was set to 100,000 steps. Damping factor; the probability that a random walker continues at a given step was set to 0.5. Spread step, the number of iteration steps allowed for the shrinking part was set to 10. Spread percent, the maximum percentage of the network to be explored during shrinking was set to 50%.

#### 4.4. The baseline approaches

We tested the proposed approach against different baselines outlined below. The baselines were selected as they are currently considered as either very weak (random) or strong (node2vec, struc2vec). All approaches apart from label propagation are node embedding algorithms. For label propagation, the same data splits were used for classification evaluation as for the logistic regression when considering embedding-based learning.

- **node2vec** [2]. This algorithm maximizes the likelihood of preserving network neighborhoods of nodes. This is achieved via biased random walk sampling. This algorithm is considered a strong baseline for structure-only learning.
- **struc2vec** [4]. This algorithm uses a hierarchy-like structure to measure node similarity at different scales and constructs a multilayer graph to encode structural similarities and generate structural context for nodes. It remains one of the key approaches capable of including information on structural similarity.
- **Label Propagation (LP)** [56]. Label propagation is a well-known algorithm for node classification. It operates by incrementally sending in-

formation from the neighboring nodes to the unlabeled nodes, eventually reaching an equilibrium and yielding the final set of predictions for the masked part of the network.

- **GraphWave** [57] is a method that represents each node’s local network neighborhood via a low-dimensional embedding by leveraging spectral graph wavelet diffusion patterns. This is one of the more scalable methods considered in this work.<sup>4</sup>
- **Graph Neural Networks** (GAT and GCN) [23]. We trained the models with the stopping criterion of 100 epochs for up to 1000 epochs. Due to unstable performance, we report the best performance (epoch scoring best). Further, as GATs were not initially implemented for multilabel classification, we extended them, so they minimize binary cross-entropy and output a sigmoid-activated space with the same dimension as the number of targets (the multiclass version does not work for such problems). As this branch of models operates with additional features assigned to nodes, and the considered benchmark data sets do not possess such features, we used the identity matrix of the adjacency matrix as the feature space, thus expecting sub-optimal performance. This Algorithm was shown to outperform other variants of graph neural networks such as the GCNs [22] which were also considered under the same training regime.
- **Random baseline** which is a random float matrix  $\in [0, 1]^{|N| \times d}$ . The PyTorch-Geometric library was used for the two baselines [58]

For all baselines, suggested default hyperparameter settings were used (either taken from papers or from the codebases). Similarly, default configurations of DNR variants were used to ensure fair comparisons (no additional hyperparameter optimization was conducted across data sets). Thus, we evaluated

---

<sup>4</sup>Python 3 implementation used: <https://github.com/benedekrozemberczki/GraphWaveMachine>

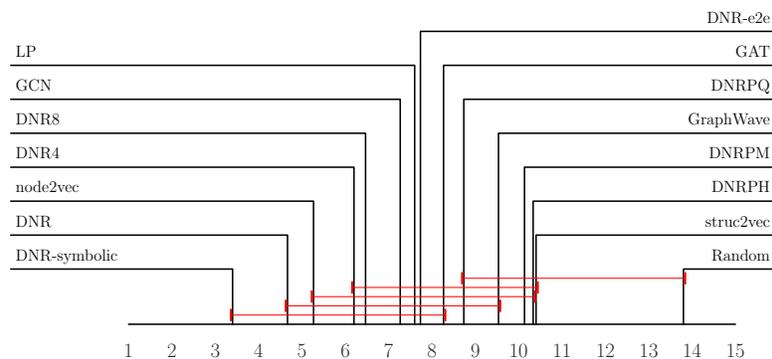
out-of-the-box performance – additional hyperparameter tuning could significantly increase the training time and render some of the methods inapplicable even at the mid-scale networks considered in this work.

## 5. Results

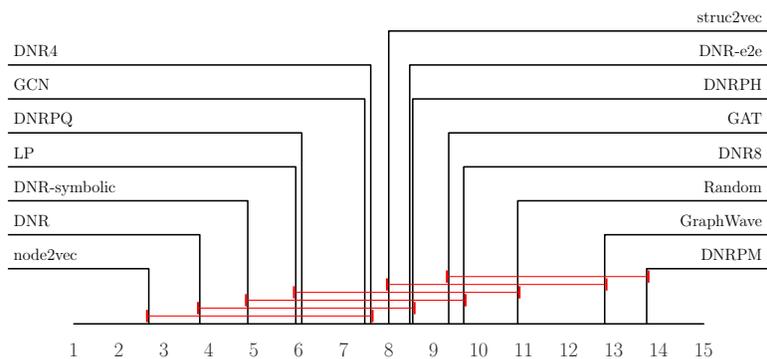
In this section, we present the empirical results and discuss their qualitative as well as quantitative aspects. We first present the results for the node classification task, followed by a qualitative evaluation of the proposed DNR algorithm for the task of node visualization.

### 5.1. Classification performance

We first present the results of classification experiments. In Figure 5 the reader can observe the critical difference plots of micro and macro F1 scores aggregated across all data sets. It can be observed that similar algorithms dominate with respect to both scores; node2vec, DNR, DNR-symbolic and label propagation are amongst the best-performing ones. The differences between the best performers are insignificant, as demonstrated via statistical analysis (CD diagrams) [52]. Next, GraphWave and DNRPM underperform w.r.t. macro F1 (Figure 6). This observation could be due to multiple factors, ranging from GraphWave’s hyperparameter sensitivity, poor performance on small networks (too much information is lost) or similar. Amongst the best performing algorithms are either the default DNR variant with two hidden layers, DNR-symbolic or node2vec. The DNR variant implementing a deeper neural network (DNR8) performed worse than the more shallow versions, indicating overfitting (highly likely especially for smaller networks). The DNRPM variant, which uses a substantially reduced version of the adjacency matrix for rank computation, performed better than random when considering micro F1. However, it was overall amongst the worst performing variants of DNR. The DNRPQ variant performed better, indicating that node pruning can have a substantial impact on the final representation – too low values of  $p$  indicate detrimental effects on



(a) Critical differences – micro F1.



(b) Critical differences – macro F1.

Figure 5: Overview of classification performance – critical difference diagrams.

the final performance. The end-to-end variant of DNR performed competitively w.r.t. micro F1; however, it performed worse when considering macro F1. This result indicates over-fitting, but also the method’s potential sensitivity to the classification of nodes in smaller networks (see the appendix materials for detailed scores on smaller networks). Note that the proposed end-to-end DNR out-performed the two GNN baselines. Current results indicate that structure-only learning is harder for GCN and GAT-based models – either due to higher possibility of overfitting or due to space complexity which arises if consider-

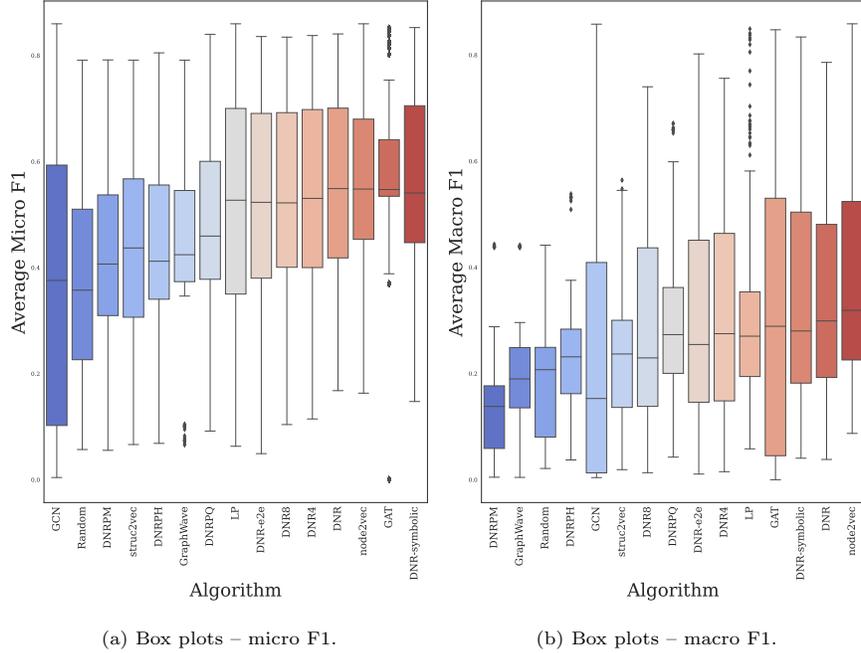


Figure 6: Micro and macro F1 performance distributions for considered algorithms.

ing the the attention-based architecture. The results indicate that the neural network in neuro-symbolic DNR variants, as expected, acts as a compression layer, losing some of the expressive power of the origin rank space at the cost of being more efficient space-wise. Bayesian comparison of default DNR with node2vec and struc2vec confirms the results obtained via frequentist analysis and is shown in Figure 7. The numbers denote the posterior probability estimates (higher is better). Note the insignificant difference between DNR and node2vec (most of the density is in the rope region), but the significant difference (as also confirmed via frequentist analysis) between DNR and struc2vec. Overall, the Bayesian analysis confirms the findings supported by the classical analysis.

## 5.2. Execution time analysis

Overview of the execution times is shown in Figure 8. We present overall execution times followed by the per-dataset execution times.

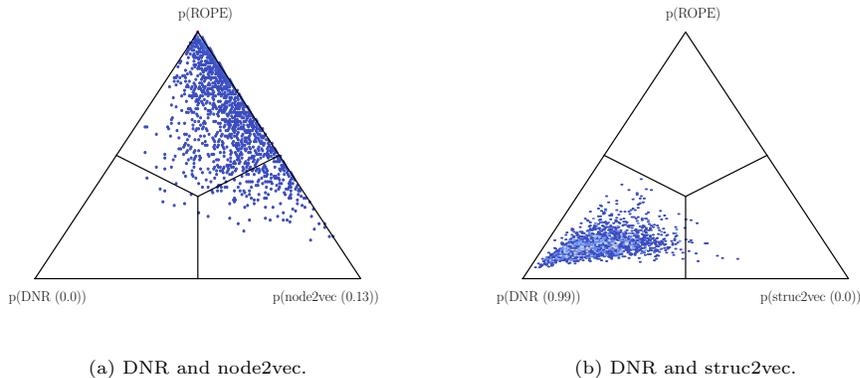


Figure 7: Bayesian comparison of selected algorithm pairs.

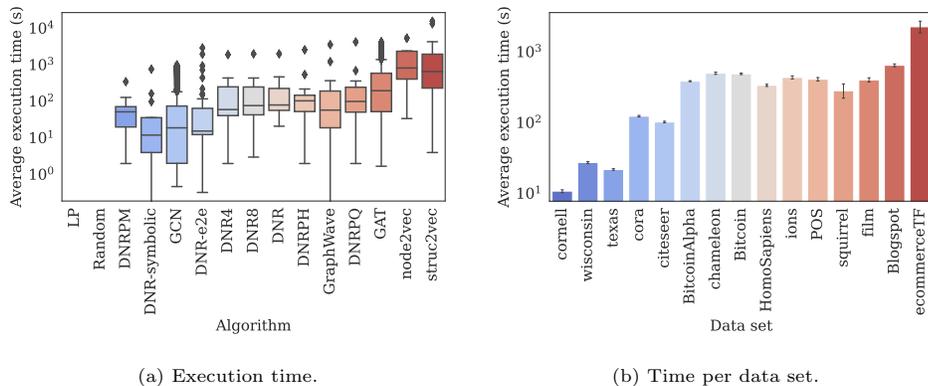


Figure 8: Execution time analysis. The proposed DNR algorithm performs substantially faster than struc2vec and node2vec.

It can be observed that the fastest DNR variants perform up to two orders of magnitude faster than, e.g., struc2vec and node2vec. Given that, e.g., DNR offers very similar performance, this result serves as a strong case for using DNR-based embeddings, especially on larger networks. Further, note that the execution time includes both embedding construction and classification, rendering the random baseline slower than label propagation (logistic regression is the bottleneck in this case).

### 5.3. Number of pivot nodes and scalability

We finally present the results on synthetic Erdős-Rényi networks, where the effect of the number of pivot nodes on the execution time was studied. The main result is shown in Figure 9. The result indicates that the number of pivot nodes can reduce the execution time by more than an order of magnitude – with no pivoting nodes, DNR’s execution time increases observably faster. More detailed results displaying the dependence with the node and link numbers are shown in subfigures 9a and 9b. The complexity, if considering pivoting, remains linear with respect to the number of nodes (constant  $d = p$  instead of  $d = |N|$  in the symbolic step of DNR). Without pivoting, the complexity increases substantially, which is problematic for larger networks. Consistent

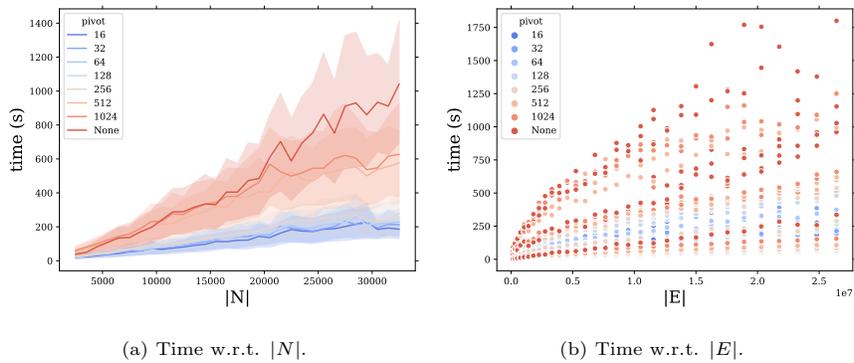


Figure 9: Execution time with respect to the number of pivot nodes. Smaller number of pivot nodes induces substantially faster execution times.

improvement with respect to the number of pivot nodes was observed – the lower the pivot number, the faster the overall process. This observation confirms our theoretical analysis which indicated that substantial improvements could be observed, especially for larger networks. The results indicate that for larger networks comprised of tens of millions of edges, the pivoting-based solutions could offer more than an *order of magnitude faster* embedding construction. For completion, tabular results summarised in this section are available as Appendix A.

#### 5.4. Performance in a low-data regime

One of the main limitations of many existing node classification algorithms is their performance when only a small portion of a given network is labelled. We next present the DNR’s behaviour when considering only 10% of the labelled data in Figure 10. The experiment indicates that two of the DNR variants

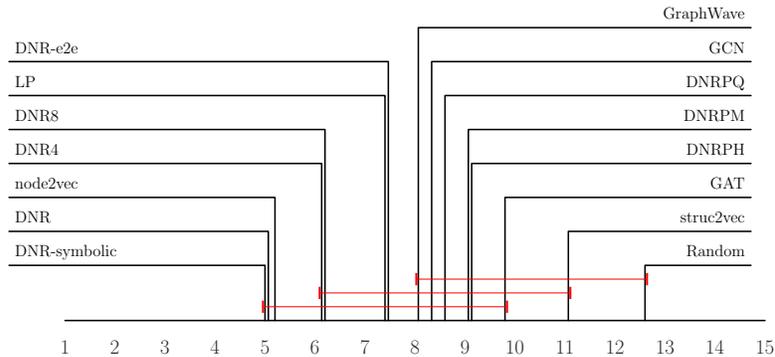


Figure 10: Micro F1 performance when the labelled data is scarce (10%)

perform well when only a relatively few labelled data are available. This result potentially indicates the link between using a symbolic (global) rank space instead of using the more local, sampled walks, indicating that neuro-symbolic node representation learning has exciting potential for low-resource learning. Note that for larger data sets, this amount of labelled nodes can already be at the limit of what can be learned on a given commodity hardware setup, rendering DNR relevant for larger data sets.

#### 5.5. Network visualization

We next demonstrate how DNR’s results can be compressed to two dimensions with UMAP [59] to visualize a given network. By considering ten nearest neighbors with the minimum distance parameter set to 0.5, we obtained the visualization (colored by node labels) as shown in Figure 11.

The visualization shows distinct clustering patterns which to some extent also correspond to the label space (colors). Note that this representation was

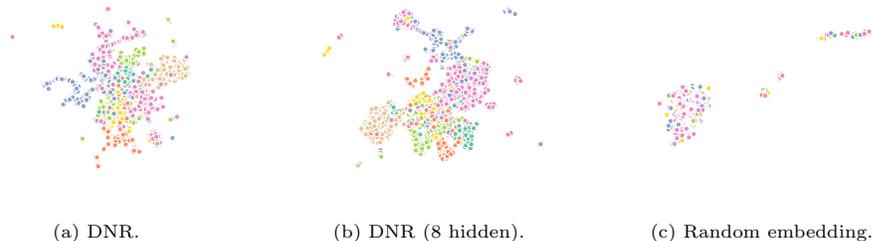
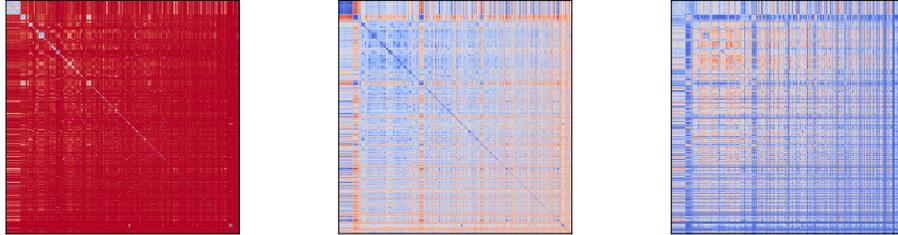


Figure 11: Embedding visualization with overlaid labels (Cora). The random embedding is added as a reference of a non-structure-preserving projection visualization.

obtained in unsupervised manner, hence some variability with respect to label-position assignment is expected. A prominent use for this type of visualizations is when inspection of potentially interesting, structurally similar units is investigated via overlay of additional information. This visualization was, alongside the embedding to 2D, computed in a matter of seconds.

### 5.6. Comparing symbolic and subsymbolic representations

The proposed DNR’s neuro-symbolic capabilities render it open for exploration of all intermediary representations and the relations between them. For the Cora network, we first computed node representations with DNR-symbolic ( $d = |N|$ ) and DNR ( $d = 128$ ) and investigated the distances between representations of individual nodes. For each node representation, we computed the cosine distance and normalized all values in the matrix by subtracting the minimum and dividing with the difference between the maximum and minimum to ensure a more fair comparison with respect to the distance bias for individual representations. The result is shown in Figure 12. We observe the following. The symbolic representation comparison matrix (a) mostly consists of entries indicating very high distances between a given embedding pair (intense red color  $\implies$  higher distance). This observation indicates that representations in high dimensional spaces (in this case  $d = |N|$ ) are far apart. The exceptions (similar nodes) are in the upper left part (blue). On the contrary, many more nodes are closer if we consider the more compact node representations obtained via



(a) DNR-symbolic ( $d = |N|$ ). (b) DNR ( $d = 128$ ). (c)  $\text{abs}(\text{DNR-symbolic} - \text{DNR})$ .

Figure 12: Representation space comparison. Each cell in (a) and (b) represents cosine distance between a given embedding pair. Cells in (c) represent the absolute difference values. The rows and columns correspond to the same nodes for all three sub-figures. Red represents high values and blue low ones.

the DNR algorithm (b). This result indicates that the neural network compresses the space (as expected), yielding fewer node representations that are distant from the others (red strips in the matrix). The final representation (c) represents the difference between the representations – blue color in this case represents similar representations. The reader can observe that distant node representations obtained by DNR are relatively close to the ones obtained by DNR-symbolic (blue horizontal and vertical strips). The red squares in the upper left part, however, indicate node similarities that were not amplified by DNR-symbolic, but with DNR. This type of ablation is possible only for neuro-symbolic representation learners, and is to our knowledge one of the first of its kind for the considered task.

## 6. Discussion and conclusions

In this work, we presented Deep Node Ranking, a methodology for scalable neuro-symbolic node embedding and direct end-to-end classification based on a given network’s structure. In extensive empirical evaluation, we demonstrated DNR’s competitive performance and superior scalability on multiple real-life and synthetic benchmark problems.

The proposed methodology offers one of the first neuro-symbolic node representation learners – the initial node features that are interpretable are compressed with a novel neural network architecture (DNRNet). Albeit the resulting representations are latent and non-interpretable to a human, the input to obtaining such a representation can be manipulated in a symbolic manner (e.g., effects of node removal), offering a simple-to-use testbed for investigating the effects of different structural interventions on a given network. Extensive empirical evaluation indicates that symbolic features are highly competitive. However, they could be impractical to compute, rendering the proposed neuro-symbolic variant of DNR highly useful for many contemporary network-based learning tasks. Furthermore, DNRNet performs well in low-data regimes, which was an interesting finding – we expected that the symbolic-only variant would dominate in such settings.

We demonstrated that neuro-symbolic approaches could scale better than purely subsymbolic ones (e.g., node2vec or struc2vec), indicating that not all interpretability is necessarily sacrificed for good performance. We demonstrated that out-of-the-box DNR implementation performs competitively and in terms of micro F1 better than state-of-the-art, and further, it offers at least an order of magnitude speedup. By introducing the concept of *node pivoting*, we demonstrate that DNR can scale to very large networks with tens of millions of links – a scale where other considered methods do not operate well without specialized hardware. We confirmed the findings related to algorithms’ performance with frequentist and Bayesian analysis. As Bayesian analysis was previously not conducted in such evaluation settings, we believe further work which will investigate the suitability (and scalability) of this branch of tests for network-related tasks is an interesting research direction.

In terms of further work related to the proposed algorithm, we see the following main directions. First, the effects of studying different pivoting schemes could offer better trade-offs between efficiency and performance. Next, by considering GPU-based implementations of the power iteration considered for computing the stationary random walk distributions, we believe additional speedups

could be observed. Neuro-symbolic node ranking offers the direct study of the effects of perturbing specific, e.g., nodes and observing the properties of the resulting low-dimensional representations. Such *structural interventions* potentially offer a more native explanation mechanism compared to *post-hoc* approximation schemes considered in contemporary machine learning. Finally, we plan to explore the scalability of DNR across multiple machines by sharing the input network and performing the rankings only locally. Such implementation could scale to much larger networks than considered by current state-of-the-art approaches.

Finally, this work demonstrates that deeper neural networks are suitable models for structure-only learning, albeit, as shown, in a neuro-symbolic setting. Current results indicate that deeper neural networks are possible and potentially offer superior performance (unless overfitting takes place). A promising direction that would offer additional improvements is also the automatic development of neural network architectures via neuroevolution.

## 7. Availability

The DNR and the datasets allowed to be shared w.r.t. their licenses will be freely accessible at <https://github.com/SkBlaz/DNR>.

## *Acknowledgments*

The work of the first author was funded by the Slovenian Research Agency through a young researcher grant (BŠ). The work of other authors was supported by the Slovenian Research Agency (ARRS) core research programs P2-0103 and P6-0411, and research projects J7-7303, L7-8269, and N2-0078 (financed under the ERC Complementary Scheme). The work was also supported by European Union’s Horizon 2020 research and innovation programme under grant agreement No 825153, project EMBEDDIA (Cross-Lingual Embeddings for Less-Represented Languages in European News Media).

## References

- [1] A. R. Benson, D. F. Gleich, J. Leskovec, Higher-order organization of complex networks, *Science* 353 (6295) (2016) 163–166.
- [2] A. Grover, J. Leskovec, node2vec: Scalable feature learning for networks, in: B. Krishnapuram, M. Shah, A. J. Smola, C. C. Aggarwal, D. Shen, R. Rastogi (Eds.), *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Francisco, CA, USA, August 13-17, 2016, ACM, 2016, pp. 855–864. doi:10.1145/2939672.2939754.  
URL <https://doi.org/10.1145/2939672.2939754>
- [3] M. Žitnik, J. Leskovec, Predicting multicellular function through multi-layer tissue networks, *Bioinformatics* 33 (14) (2017) i190–i198.
- [4] L. F. Ribeiro, P. H. Saverese, D. R. Figueiredo, struc2vec: Learning node representations from structural identity, in: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '17*, Association for Computing Machinery, New York, NY, USA, 2017, p. 385–394. doi:10.1145/3097983.3098061.  
URL <https://doi.org/10.1145/3097983.3098061>
- [5] B. Perozzi, R. Al-Rfou, S. Skiena, Deepwalk: online learning of social representations, in: S. A. Macskassy, C. Perlich, J. Leskovec, W. Wang, R. Ghani (Eds.), *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14*, New York, NY, USA - August 24 - 27, 2014, ACM, 2014, pp. 701–710. doi:10.1145/2623330.2623732.  
URL <https://doi.org/10.1145/2623330.2623732>
- [6] A. d’Avila Garcez, L. C. Lamb, Neurosymbolic ai: The 3rd wave (2020).  
arXiv:2012.05876.
- [7] J. Mao, C. Gan, P. Kohli, J. B. Tenenbaum, J. Wu, The neuro-symbolic concept learner: Interpreting scenes, words, and sentences from natural

- supervision, in: 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019, OpenReview.net, 2019.  
URL <https://openreview.net/forum?id=rJgMlhRctm>
- [8] Q. Li, S. Huang, Y. Hong, Y. Chen, Y. N. Wu, S. Zhu, Closed loop neural-symbolic learning via integrating neural perception, grammar parsing, and symbolic reasoning, in: Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event, Vol. 119 of Proceedings of Machine Learning Research, PMLR, 2020, pp. 5884–5894.  
URL <http://proceedings.mlr.press/v119/li20f.html>
- [9] S. Amizadeh, H. Palangi, A. Polozov, Y. Huang, K. Koishida, Neuro-symbolic visual reasoning: Disentangling "visual" from "reasoning", in: Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event, Vol. 119 of Proceedings of Machine Learning Research, PMLR, 2020, pp. 279–290.  
URL <http://proceedings.mlr.press/v119/amizadeh20a.html>
- [10] H. Dong, J. Mao, T. Lin, C. Wang, L. Li, D. Zhou, Neural logic machines, in: 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019, OpenReview.net, 2019.  
URL <https://openreview.net/forum?id=B1xY-hRctX>
- [11] H. Lodhi, Deep relational machines, in: Proceedings, Part II, of the 20th International Conference on Neural Information Processing - Volume 8227, ICONIP 2013, Springer-Verlag, Berlin, Heidelberg, 2013, p. 212–219.
- [12] A. Srinivasan, L. Vig, M. Bain, Logical explanations for deep relational machines using relevance information, Journal of Machine Learning Research 20 (130) (2019) 1–47.
- [13] R. Manhaeve, S. Dumancic, A. Kimmig, T. Demeester, L. D. Raedt, Deepproblog: Neural probabilistic logic programming, in: S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, R. Garnett

(Eds.), *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018*, December 3-8, 2018, Montréal, Canada, 2018, pp. 3753–3763.

URL <https://proceedings.neurips.cc/paper/2018/hash/dc5d637ed5e62c36ecb73b654b05ba2a-Abstract.html>

- [14] L. D. Raedt, S. Dumancic, R. Manhaeve, G. Marra, From statistical relational to neuro-symbolic artificial intelligence, in: C. Bessiere (Ed.), *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, ijcai.org, 2020, pp. 4943–4950. doi: 10.24963/ijcai.2020/688.  
URL <https://doi.org/10.24963/ijcai.2020/688>
- [15] T. Winters, G. Marra, R. Manhaeve, L. D. Raedt, *Deepstochlog: Neural stochastic logic programming (2021)*. arXiv:2106.12574.
- [16] C. Nowzari, V. M. Preciado, G. J. Pappas, Analysis and control of epidemics: A survey of spreading processes on complex networks, *IEEE Control Systems* 36 (1) (2016) 26–46.
- [17] D.-H. Le, A novel method for identifying disease associated protein complexes based on functional similarity protein complex networks, *Algorithms for Molecular Biology* 10 (1) (2015) 14.
- [18] L. d. F. Costa, F. A. Rodrigues, G. Travieso, P. R. Villas Boas, Characterization of complex networks: A survey of measurements, *Advances in Physics* 56 (1) (2007) 167–242.
- [19] R. Van Der Hofstad, *Random graphs and complex networks (2016)*.
- [20] X. Zhu, Z. Ghahramani, Learning from labeled and unlabeled data with label propagation, Tech. rep. (2002).
- [21] P. Cui, X. Wang, J. Pei, W. Zhu, A survey on network embedding, *IEEE Transactions on Knowledge and Data Engineering*.

- [22] T. N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, in: 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings, OpenReview.net, 2017.  
URL <https://openreview.net/forum?id=SJU4ayYg1>
- [23] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, Y. Bengio, Graph attention networks, in: 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings, OpenReview.net, 2018.  
URL <https://openreview.net/forum?id=rJXmpikCZ>
- [24] K. Xu, W. Hu, J. Leskovec, S. Jegelka, How powerful are graph neural networks?, in: 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019, OpenReview.net, 2019.  
URL <https://openreview.net/forum?id=ryGs6iA5Km>
- [25] W. L. Hamilton, Z. Ying, J. Leskovec, Inductive representation learning on large graphs, in: I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, R. Garnett (Eds.), Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA, 2017, pp. 1024–1034.  
URL <https://proceedings.neurips.cc/paper/2017/hash/5dd9db5e033da9c6fb5ba83c7a7e9bea9-Abstract.html>
- [26] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, Q. Mei, LINE: large-scale information network embedding, in: A. Gangemi, S. Leonardi, A. Panconesi (Eds.), Proceedings of the 24th International Conference on World Wide Web, WWW 2015, Florence, Italy, May 18-22, 2015, ACM, 2015, pp. 1067–1077. doi:10.1145/2736277.2741093.  
URL <https://doi.org/10.1145/2736277.2741093>
- [27] J. Qiu, Y. Dong, H. Ma, J. Li, K. Wang, J. Tang, Network embedding

as matrix factorization: Unifying deepwalk, line, pte, and node2vec, in: Y. Chang, C. Zhai, Y. Liu, Y. Maarek (Eds.), Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining, WSDM 2018, Marina Del Rey, CA, USA, February 5-9, 2018, ACM, 2018, pp. 459–467. doi:10.1145/3159652.3159706.

URL <https://doi.org/10.1145/3159652.3159706>

- [28] P. Goyal, E. Ferrara, Graph embedding techniques, applications, and performance: A survey, arXiv preprint arXiv:1705.02801.
- [29] L. Page, S. Brin, R. Motwani, T. Winograd, The PageRank citation ranking: Bringing order to the web., Tech. rep., Stanford InfoLab (1999).
- [30] H. Tong, C. Faloutsos, J.-Y. Pan, Fast random walk with restart and its applications, in: Proceedings of the Sixth International Conference on Data Mining, Washington, DC, USA, 2006, pp. 613–622.
- [31] A. Halu, R. J. Mondragón, P. Panzarasa, G. Bianconi, Multiplex pagerank, PloS one 8 (10) (2013) e78293.
- [32] X. Yu, T. G. Lilburn, H. Cai, J. Gu, T. Korkmaz, Y. Wang, Pagerank influence analysis of protein-protein association networks in the malaria parasite plasmodium falciparum, International Journal of Computational Biology and Drug Design 10 (2) (2017) 137–156.
- [33] P. Lofgren, S. Banerjee, A. Goel, Personalized pagerank estimation and search: A bidirectional approach, in: P. N. Bennett, V. Josifovski, J. Neville, F. Radlinski (Eds.), Proceedings of the Ninth ACM International Conference on Web Search and Data Mining, San Francisco, CA, USA, February 22-25, 2016, ACM, 2016, pp. 163–172. doi:10.1145/2835776.2835823.  
URL <https://doi.org/10.1145/2835776.2835823>
- [34] J. Kralj, M. Robnik-Šikonja, N. Lavrač, HINMINE: heterogeneous infor-

- mation network mining with information retrieval heuristics, *Journal of Intelligent Information Systems* (2017) 1–33.
- [35] J. Klicpera, A. Bojchevski, S. Günnemann, Predict then propagate: Graph neural networks meet personalized pagerank, in: 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019, OpenReview.net, 2019.  
URL <https://openreview.net/forum?id=H1gL-2A9Ym>
- [36] A. Bojchevski, J. Klicpera, B. Perozzi, M. Blais, A. Kapoor, M. Lukasik, S. Günnemann, Is pagerank all you need for scalable graph neural networks?, in: ACM KDD, MLG Workshop, 2019.
- [37] K. Xu, C. Li, Y. Tian, T. Sonobe, K. Kawarabayashi, S. Jegelka, Representation learning on graphs with jumping knowledge networks, in: J. G. Dy, A. Krause (Eds.), Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholm, Sweden, July 10-15, 2018, Vol. 80 of Proceedings of Machine Learning Research, PMLR, 2018, pp. 5449–5458.  
URL <http://proceedings.mlr.press/v80/xu18c.html>
- [38] L. Page, S. Brin, R. Motwani, T. Winograd, The PageRank citation ranking: Bringing order to the web., Tech. rep., Stanford InfoLab (1999).
- [39] N. Reimers, I. Gurevych, Sentence-bert: Sentence embeddings using siamese bert-networks, in: Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, 2019.  
URL <https://arxiv.org/abs/1908.10084>
- [40] C. Stark, B.-J. Breitkreutz, A. Chatr-Aryamontri, L. Boucher, R. Oughtred, M. S. Livstone, J. Nixon, K. Van Auken, X. Wang, X. Shi, et al., The biogrid interaction database: 2011 update, *Nucleic acids research* 39 (suppl\_1) (2010) D698–D704.

- [41] M. Mahoney, Large text compression benchmark, URL: <http://www.mattmahoney.net/text/text.html>.
- [42] R. Zafarani, H. Liu, Social computing data repository at ASU, <http://socialcomputing.asu.edu> (2009).
- [43] C. Stark, B.-J. Breitkreutz, T. Reguly, L. Boucher, A. Breitkreutz, M. Tyers, BioGRID: A general repository for interaction datasets, *Nucleic Acids Research* 34 (suppl\_1) (2006) D535–D539.
- [44] Q. Lu, L. Getoor, Link-based classification, in: T. Fawcett, N. Mishra (Eds.), *Machine Learning, Proceedings of the Twentieth International Conference (ICML 2003)*, August 21-24, 2003, Washington, DC, USA, AAAI Press, 2003, pp. 496–503.  
URL <http://www.aaai.org/Library/ICML/2003/icml03-066.php>
- [45] H. Pei, B. Wei, K. C. Chang, Y. Lei, B. Yang, Geom-gcn: Geometric graph convolutional networks, in: *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, OpenReview.net, 2020.  
URL <https://openreview.net/forum?id=S1e2agrFvS>
- [46] S. Kumar, F. Spezzano, V. Subrahmanian, C. Faloutsos, Edge weight prediction in weighted signed networks, in: *Data Mining (ICDM), 2016 IEEE 16th International Conference on*, IEEE, 2016, pp. 221–230.
- [47] B. Škrlić, T. Kunej, J. Konc, Insights from ion binding site network analysis into evolution and functions of proteins, *Molecular Informatics*.
- [48] J. Konc, D. Janežič, Probis-ligands: a web server for prediction of ligands by examination of protein binding sites, *Nucleic Acids Research* 42 (W1) (2014) W215–W220.
- [49] J. Konc, M. Depolli, R. Trobec, K. Rozman, D. Janežič, Parallel-ProBiS: Fast parallel algorithm for local structural comparison of protein struc-

- tures and binding sites, *Journal of Computational Chemistry* 33 (27) (2012) 2199–2203.
- [50] J. Konc, B. Skrlj, N. Erzen, T. Kunej, D. Janezic, Genprobis: web server for mapping of sequence variants to protein binding sites, *Nucleic Acids Research* 45 (W1) (2017) W253–W259.
- [51] J. Qiu, Y. Dong, H. Ma, J. Li, K. Wang, J. Tang, Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec, in: Y. Chang, C. Zhai, Y. Liu, Y. Maarek (Eds.), *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining, WSDM 2018, Marina Del Rey, CA, USA, February 5-9, 2018*, ACM, 2018, pp. 459–467. doi:10.1145/3159652.3159706.  
URL <https://doi.org/10.1145/3159652.3159706>
- [52] J. Demšar, Statistical comparisons of classifiers over multiple data sets, *Journal of Machine learning research* 7 (Jan) (2006) 1–30.
- [53] A. Benavoli, G. Corani, J. Demšar, M. Zaffalon, Time for a change: a tutorial for comparing multiple classifiers through bayesian analysis, *The Journal of Machine Learning Research* 18 (1) (2017) 2653–2688.
- [54] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, Í. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, A. Vijaykumar, A. P. Bardelli, A. Rothberg, A. Hilboll, A. Kloeckner, A. Scopatz, A. Lee, A. Rokem, C. N. Woods, C. Fulton, C. Masson, C. Häggström, C. Fitzgerald, D. A. Nicholson, D. R. Hagen, D. V. Pasechnik, E. Olivetti, E. Martin, E. Wieser, F. Silva, F. Lenders, F. Wilhelm, G. Young, G. A. Price, G.-L. Ingold, G. E. Allen, G. R. Lee, H. Audren, I. Probst, J. P. Dietrich, J. Silterra, J. T. Webber,

J. Slavič, J. Nothman, J. Buchner, J. Kulick, J. L. Schönberger, J. V. de Miranda Cardoso, J. Reimer, J. Harrington, J. L. C. Rodríguez, J. Nunez-Iglesias, J. Kuczynski, K. Tritz, M. Thoma, M. Newville, M. Kümmerer, M. Bolingbroke, M. Tartre, M. Pak, N. J. Smith, N. Nowaczyk, N. Shebanov, O. Pavlyk, P. A. Brodtkorb, P. Lee, R. T. McGibbon, R. Feldbauer, S. Lewis, S. Tygier, S. Sievert, S. Vigna, S. Peterson, S. More, T. Pudlik, T. Oshima, T. J. Pingel, T. P. Robitaille, T. Spura, T. R. Jones, T. Cera, T. Leslie, T. Zito, T. Krauss, U. Upadhyay, Y. O. Halchenko, Y. Vázquez-Baeza, S. 1.0 Contributors, Scipy 1.0: fundamental algorithms for scientific computing in python, *Nature Methods* 17 (3) (2020) 261–272. doi:10.1038/s41592-019-0686-2. URL <https://doi.org/10.1038/s41592-019-0686-2>

- [55] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala, Pytorch: An imperative style, high-performance deep learning library, in: H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. B. Fox, R. Garnett (Eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada, 2019*, pp. 8024–8035.

URL <https://proceedings.neurips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html>

- [56] X. Zhu, Z. Ghahramani, Learning from labeled and unlabeled data with label propagation.

- [57] C. Donnat, M. Zitnik, D. Hallac, J. Leskovec, Learning structural node embeddings via diffusion wavelets, in: Y. Guo, F. Farooq (Eds.), *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2018, London, UK, August 19-23, 2018*,

ACM, 2018, pp. 1320–1329. doi:10.1145/3219819.3220025.

URL <https://doi.org/10.1145/3219819.3220025>

[58] M. Fey, J. E. Lenssen, Fast graph representation learning with PyTorch Geometric, in: ICLR Workshop on Representation Learning on Graphs and Manifolds, 2019.

[59] L. McInnes, J. Healy, J. Melville, Umap: Uniform manifold approximation and projection for dimension reduction, arXiv preprint arXiv:1802.03426.

### **Appendix A. Tabular results with deviations**

In this section, we present the performance results for micro and macro F1 scores. The first table represents the micro F1 scores, followed by a table showing macro F1 scores. The runs marked with NaN reached the time-out point (did not finish).

Table A.2: Micro F1.

dataset setting	Bitcoin	BitcoinAlpha	Blogspot	HomeSapiens	POS	chameleon	citeseer	cora	cornell	ecommerceTF	film	ions	squirrel	texas	wisconsin
DNR	0.71 (0.01)	0.71 (0.01)	0.2 (0.01)	0.2 (0.01)	0.45 (0.02)	0.56 (0.04)	0.55 (0.02)	0.78 (0.02)	0.52 (0.06)	0.83 (0.0)	0.34 (0.01)	0.64 (0.03)	0.41 (0.03)	0.59 (0.07)	0.53 (0.03)
DNR-e2e	0.69 (0.01)	0.7 (0.01)	0.09 (0.01)	0.07 (0.01)	0.41 (0.01)	0.45 (0.04)	0.57 (0.03)	0.77 (0.04)	0.49 (0.12)	0.83 (0.01)	0.35 (0.02)	0.66 (0.03)	0.39 (0.02)	0.61 (0.06)	0.49 (0.08)
DNR-symbolic	0.72 (0.01)	0.71 (0.01)	0.28 (0.04)	0.22 (0.03)	0.45 (0.02)	0.59 (0.04)	0.62 (0.04)	0.8 (0.04)	0.54 (0.04)	0.83 (0.01)	0.36 (0.01)	0.67 (0.04)	0.5 (0.05)	0.58 (0.07)	0.49 (0.04)
DNR4	0.71 (0.01)	0.71 (0.01)	0.15 (0.02)	0.15 (0.02)	0.4 (0.0)	0.52 (0.03)	0.55 (0.01)	0.74 (0.02)	0.48 (0.05)	0.83 (0.01)	0.37 (0.01)	0.63 (0.04)	0.42 (0.02)	0.58 (0.05)	0.51 (0.03)
DNR8	0.7 (0.01)	0.71 (0.0)	0.14 (0.02)	0.12 (0.01)	0.4 (0.0)	0.5 (0.02)	0.54 (0.02)	0.71 (0.03)	0.53 (0.02)	0.82 (0.01)	0.37 (0.01)	0.58 (0.05)	0.41 (0.01)	0.55 (0.04)	0.52 (0.05)
DNRPH	0.68 (0.01)	0.68 (0.01)	0.15 (0.01)	0.08 (0.01)	0.4 (0.01)	0.34 (0.01)	0.36 (0.01)	0.41 (0.01)	0.54 (0.03)	0.8 (0.0)	0.34 (0.01)	0.48 (0.01)	0.39 (0.01)	0.53 (0.03)	0.5 (0.03)
DNRPM	0.69 (0.0)	0.7 (0.0)	0.1 (0.0)	0.06 (0.01)	0.4 (0.01)	0.37 (0.01)	0.2 (0.01)	0.31 (0.0)	0.52 (0.02)	0.78 (0.0)	0.37 (0.01)	0.43 (0.01)	0.41 (0.01)	0.53 (0.02)	0.51 (0.07)
DNRPQ	0.69 (0.01)	0.69 (0.01)	0.2 (0.01)	0.11 (0.01)	0.41 (0.01)	0.42 (0.02)	0.4 (0.02)	0.6 (0.02)	0.51 (0.03)	0.83 (0.01)	0.34 (0.02)	0.58 (0.03)	0.37 (0.01)	0.54 (0.03)	0.44 (0.06)
GAT	0.54 (0.0)	0.55 (0.02)	NaN	0.0 (0.0)	0.35 (0.12)	0.6 (0.04)	0.66 (0.07)	0.82 (0.03)	0.58 (0.05)	NaN	0.37 (0.0)	0.51 (0.01)	NaN	0.65 (0.05)	0.53 (0.03)
GCN	0.58 (0.02)	0.57 (0.01)	0.11 (0.01)	0.03 (0.01)	0.05 (0.05)	0.61 (0.04)	0.69 (0.05)	0.83 (0.03)	0.56 (0.04)	0.82 (0.0)	0.37 (0.01)	0.19 (0.12)	0.44 (0.01)	0.63 (0.06)	0.51 (0.02)
GraphWave	0.69 (0.0)	0.7 (0.0)	0.1 (0.0)	0.07 (0.01)	0.4 (0.01)	0.37 (0.02)	0.37 (0.01)	0.42 (0.02)	0.5 (0.03)	0.78 (0.0)	0.37 (0.01)	0.53 (0.03)	0.41 (0.01)	0.54 (0.04)	0.47 (0.04)
LP	0.7 (0.0)	0.71 (0.01)	0.22 (0.02)	0.06 (0.0)	0.07 (0.0)	0.4 (0.02)	0.64 (0.06)	0.82 (0.04)	0.54 (0.05)	0.72 (0.02)	0.35 (0.01)	0.69 (0.06)	0.41 (0.01)	0.56 (0.02)	0.42 (0.06)
Random	0.65 (0.02)	0.66 (0.03)	0.07 (0.01)	0.06 (0.0)	0.37 (0.03)	0.3 (0.03)	0.18 (0.0)	0.21 (0.02)	0.47 (0.06)	0.78 (0.01)	0.33 (0.02)	0.36 (0.02)	0.34 (0.04)	0.5 (0.05)	0.42 (0.03)
node2vec	0.69 (0.01)	0.69 (0.02)	0.36 (0.03)	0.21 (0.02)	0.52 (0.02)	0.58 (0.02)	0.58 (0.02)	0.83 (0.03)	0.51 (0.04)	0.83 (0.01)	0.34 (0.02)	0.65 (0.03)	0.43 (0.03)	0.57 (0.05)	0.51 (0.02)
struc2vec	0.67 (0.02)	0.68 (0.01)	0.08 (0.01)	0.08 (0.0)	0.38 (0.02)	0.54 (0.03)	0.27 (0.01)	0.3 (0.02)	0.45 (0.04)	0.78 (0.01)	0.34 (0.01)	0.46 (0.02)	0.41 (0.02)	0.56 (0.04)	0.53 (0.1)

Table A.3: Macro F1.

dataset setting	Bitcoin	BitcoinAlpha	Blogspot	HomeSapiens	POS	chameleon	citeseer	cora	cornell	ecommerceTF	film	ions	squirrel	texas	wisconsin
DNR	0.32 (0.02)	0.3 (0.01)	0.06 (0.01)	0.16 (0.02)	0.06 (0.01)	0.54 (0.04)	0.5 (0.01)	0.77 (0.02)	0.25 (0.04)	0.67 (0.01)	0.2 (0.01)	0.25 (0.05)	0.3 (0.02)	0.28 (0.06)	0.39 (0.05)
DNR-e2e	0.27 (0.01)	0.27 (0.01)	0.01 (0.0)	0.03 (0.01)	0.04 (0.0)	0.39 (0.07)	0.5 (0.03)	0.74 (0.06)	0.22 (0.04)	0.66 (0.02)	0.17 (0.02)	0.2 (0.02)	0.18 (0.03)	0.31 (0.04)	0.32 (0.04)
DNR-symbolic	0.31 (0.02)	0.29 (0.01)	0.11 (0.03)	0.16 (0.04)	0.06 (0.01)	0.57 (0.04)	0.56 (0.04)	0.79 (0.05)	0.21 (0.05)	0.67 (0.04)	0.19 (0.02)	0.22 (0.05)	0.39 (0.08)	0.23 (0.06)	0.27 (0.04)
DNR4	0.31 (0.02)	0.28 (0.01)	0.03 (0.01)	0.09 (0.02)	0.04 (0.0)	0.48 (0.03)	0.48 (0.01)	0.72 (0.03)	0.19 (0.02)	0.65 (0.01)	0.15 (0.01)	0.2 (0.03)	0.25 (0.03)	0.26 (0.05)	0.34 (0.04)
DNR8	0.3 (0.02)	0.28 (0.01)	0.03 (0.01)	0.06 (0.01)	0.04 (0.0)	0.46 (0.02)	0.47 (0.02)	0.68 (0.05)	0.19 (0.03)	0.63 (0.01)	0.14 (0.0)	0.17 (0.02)	0.2 (0.03)	0.22 (0.04)	0.26 (0.03)
DNRPH	0.29 (0.01)	0.27 (0.01)	0.06 (0.01)	0.05 (0.01)	0.04 (0.0)	0.22 (0.01)	0.21 (0.01)	0.33 (0.02)	0.26 (0.03)	0.53 (0.01)	0.2 (0.01)	0.14 (0.02)	0.24 (0.01)	0.26 (0.04)	0.33 (0.03)
DNRPM	0.27 (0.01)	0.26 (0.01)	0.01 (0.0)	0.02 (0.0)	0.03 (0.0)	0.14 (0.0)	0.06 (0.0)	0.07 (0.0)	0.17 (0.0)	0.44 (0.0)	0.14 (0.0)	0.06 (0.0)	0.15 (0.0)	0.18 (0.02)	0.17 (0.01)
DNRPQ	0.3 (0.01)	0.28 (0.01)	0.08 (0.01)	0.09 (0.01)	0.05 (0.0)	0.37 (0.02)	0.36 (0.02)	0.58 (0.02)	0.24 (0.03)	0.66 (0.01)	0.21 (0.01)	0.22 (0.05)	0.25 (0.0)	0.28 (0.05)	0.35 (0.08)
GAT	0.04 (0.0)	0.05 (0.01)	NaN	0.0 (0.0)	0.01 (0.0)	0.59 (0.04)	0.62 (0.07)	0.81 (0.04)	0.35 (0.16)	NaN	0.15 (0.01)	0.08 (0.02)	NaN	0.36 (0.08)	0.36 (0.05)
GCN	0.08 (0.01)	0.08 (0.01)	0.01 (0.0)	0.01 (0.0)	0.02 (0.01)	0.6 (0.04)	0.64 (0.05)	0.82 (0.03)	0.32 (0.13)	0.64 (0.0)	0.15 (0.01)	0.08 (0.03)	0.34 (0.01)	0.35 (0.07)	0.38 (0.05)
GraphWave	0.27 (0.01)	0.26 (0.01)	0.0 (0.0)	0.03 (0.0)	0.04 (0.0)	0.21 (0.03)	0.25 (0.01)	0.21 (0.01)	0.21 (0.02)	0.44 (0.0)	0.14 (0.0)	0.14 (0.01)	0.14 (0.0)	0.18 (0.02)	0.22 (0.03)
LP	0.28 (0.01)	0.29 (0.01)	0.1 (0.02)	0.06 (0.0)	0.06 (0.0)	0.33 (0.03)	0.61 (0.06)	0.81 (0.04)	0.24 (0.05)	0.67 (0.02)	0.2 (0.01)	0.32 (0.04)	0.24 (0.02)	0.26 (0.02)	0.28 (0.09)
Random	0.27 (0.01)	0.26 (0.01)	0.02 (0.0)	0.05 (0.01)	0.04 (0.0)	0.25 (0.02)	0.15 (0.0)	0.13 (0.01)	0.21 (0.03)	0.44 (0.0)	0.22 (0.01)	0.08 (0.0)	0.22 (0.01)	0.23 (0.05)	0.23 (0.02)
node2vec	0.33 (0.01)	0.3 (0.01)	0.23 (0.03)	0.18 (0.02)	0.11 (0.01)	0.57 (0.02)	0.54 (0.02)	0.82 (0.03)	0.23 (0.03)	0.67 (0.0)	0.21 (0.01)	0.32 (0.05)	0.36 (0.03)	0.27 (0.04)	0.37 (0.06)
struc2vec	0.29 (0.01)	0.28 (0.01)	0.03 (0.0)	0.06 (0.01)	0.06 (0.0)	0.52 (0.03)	0.24 (0.01)	0.17 (0.01)	0.21 (0.04)	0.45 (0.0)	0.21 (0.02)	0.13 (0.01)	0.34 (0.01)	0.28 (0.07)	0.3 (0.03)