Network Representation Learning for Link Prediction: Are we improving upon simple heuristics?

Alexandru Mara [1] **Jefrey Lijffijt** [1] **Tijl De Bie** [1]

Abstract

Network representation learning has become an active research area in recent years with many new methods showcasing their performance on downstream prediction tasks such as Link Prediction. Despite the efforts of the community to ensure reproducibility of research by providing method implementations, important issues remain. The complexity of the evaluation pipelines and abundance of design choices have led to difficulties in quantifying the progress in the field and identifying the state-of-the-art. In this work, we analyse 17 network embedding methods on 7 real-world datasets and find, using a consistent evaluation pipeline, only thin progress over the recent years. Also, many embedding methods are outperformed by simple heuristics. Finally, we discuss how standardized evaluation tools can repair this situation and boost progress in this field.

1. Introduction

Network representation learning or network embedding (NE) methods have attracted much interest in recent years (Belkin & Niyogi, 2002; Perozzi et al., 2014; Tang et al., 2015; Cao et al., 2015; Ou et al., 2016; Gao et al., 2018). NE methods aim to bridge the gap between network data and traditional machine learning by constructing low-dimensional embeddings of network entities as vectors in the metric space. Using these embeddings, traditional machine learning methods can be used on network data. A prominent application of NEs is in link prediction (LP) (Grover & Leskovec, 2016; Lai et al., 2017; Kang et al., 2019), which amounts to estimating the probability of the existence of edges between nodes not connected in the input graph.

In tasks such as multi-label classification, a general consensus on the evaluation setup and criteria exists (Hamilton et al., 2017; Zhang et al., 2018a; Goyal & Ferrara, 2018), but for LP this is not the case. The evaluation of LP performance requires a pipeline with several preprocessing steps and design choices that can confound the results and are prone to errors. The main challenges for LP evaluation are:

Data preprocessing: To perform LP, sets of train and test edges (i.e. connected node-pairs) are required that can be generated using different approaches. For instance, a typical assumption in LP is that the train edges span a (train) sub-network of a more complete (test) network. Thus, a principled approach is to use snapshots of the network at two different points in time. The first snapshot is used for training and the difference between the two snapshots for testing (Lichtenwalter & Chawla, 2012; Yang et al., 2015; Garcia-Gasulla et al., 2015). Unfortunately, networks with such a temporal component are scarce and therefore, authors ordinarily resort to *sampling edges* from a network as test examples and using the remaining edges for training Grover & Leskovec (2016); Lai et al. (2017); Gao et al. (2018); Kang et al. (2019). The sampling process varies between scientific works in different aspects. The sizes of the train and test sets vary—between 50-50 in Grover & Leskovec (2016); Kang et al. (2019), to 60-40 in Gao et al. (2018) and 80-20 in Lai et al. (2017). The algorithms used to generate these splits also differ and while some aim to construct train networks with similar, scaled-down, properties such as, e.g., node degrees (Gurukar et al., 2019), others generate train graphs that preserve the general topology of the original network (Mara et al., 2019).

Prediction pipeline: Different NE methods require different pipelines to obtain link predictions. While some embedding methods directly compute link probabilities (Zhang et al., 2018b; Kang et al., 2019), for others, these need to be learned on top of the node embeddings. Two common approaches are, interpreting some notion of similarity between two node embeddings (e.g. dot product) as the link probability or casting the problem as a binary classification task. The latter, which has been shown to be more effective Gurukar et al. (2019), requires as a pre-step the computation of node-pair embeddings. Thus, an operator must be applied on the node embeddings to obtain node-pair representations

[1]Department of Electronics and Information Systems, Ghent University, Ghent, Belgium. Correspondence to: Alexandru Mara <alexandru.mara(at)ugent.be>, Jefrey Lijffijt <jefrey.lijffijt(at)ugent.be>, Tijl De Bie <tijl.debie(at)ugent.be>.

Preliminary work under review.

that are in turn fed into a binary classifier to perform LP. The choice of operator varies between scientific works (Grover & Leskovec, 2016; Tsitsulin et al., 2018) and is sometimes not discussed (Wang et al., 2016; Lai et al., 2017). Moreover, many valid choices exist for the binary classifier, and classifier training requires—in addition to 'positive' examples of edges—'negative' samples or non-edges that can be selected using different strategies and be of varying sizes (Kotnis & Nastase, 2017).

Hyperparameter tuning: When comparing new methods with the state-of-the-art, for the baseline methods the default parameter settings are often used (Zhang et al., 2018b; Tsitsulin et al., 2018), yet care is taken in tuning the parameters of the introduced method. When the recommended default settings were informed by experiments on other graphs than those used in the study at hand, this can paint an unduly unfavourable picture of the baseline methods.

Evaluation metrics: Finally, no consensus exists on which evaluation criteria should be used for comparing different methods. While some papers advocate for the use of precision@Np for a range of Np values (Ou et al., 2016; Wang et al., 2016; Zhang et al., 2018b), others use AUROC (Grover & Leskovec, 2016; Kang et al., 2019) or precision and recall at fixed thresholds (Wei et al., 2017).

These challenges have led to an inconsistency in evaluation procedures throughout papers. Hence, the practical performance of NE methods for LP is poorly understood. Here, we study these design choices to clarify their effect on the evaluation pipelines and ultimately on method performance.

Contributions. We provide an in-depth empirical analysis of the state-of-the-art on network representation learning for LP. In our experiments, we show that embedding-based LP methods are generally matched in performance and sometimes even outperformed by simple heuristics. We also study the effect on performance of hyperparameter tuning, embedding dimensionality, train set size, edge sampling strategy, node-pair embedding operator, and classification methods. Inevitably there are also limitations on the scope of the evaluation, such as an exclusive focus on undirected and unweighted networks without attributes, an analysis of moderate-sized networks, and a representative but finite set of evaluated methods and evaluation metrics. However, our evaluation is based upon a well-developed open source evaluation framework, `EvalNE` (Mara et al., 2019), ensuring reproducibility of results and allowing for direct extensions of this work in all the aforementioned aspects.

The paper is organized as follows. Section 2 describes the NE methods evaluated, Section 3 presents the datasets used, Section 4 discusses the evaluation setup, Section 5 summarizes the results, and finally, Section 6 concludes this paper.

2. Methods

In this section we present the NE methods and baseline LP heuristics evaluated. When available, we compare implementations of NE methods from distinct sources. Specifically, in addition to original implementations we consider those in two actively maintained libraries, i.e. `OpenNE` and `GEM` (Goyal & Ferrara, 2018). Table 1 summarizes the hyperparameters tuned and implementations used. Detailed hyperparameter descriptions are provided in the supplementary material. All LP heuristics are presented in Table 1 aggregated in a single *Heuristics* field. For all methods where this is relevant, the node-pair embedding operator is tuned as an additional hyperparameter (see Section 4).

As for the notation, we represent an undirected network as $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ with vertex set $\mathbf{V} = \{1, \ldots, N\}$ and edge set $\mathbf{E} \subseteq \binom{\mathbf{V}}{2}$. Edges or connected node-pairs are represented as unordered pairs $\{i, j\} \in \mathbf{E}$. Non-edges or disconnected pairs are represented as $\{i, j\} \in \mathbf{D}$. We refer to the sets of train node-pairs as $\mathbf{E}_{train}$ and $\mathbf{D}_{train}$ and to the test node-pairs as $\mathbf{E}_{test}$ and $\mathbf{D}_{test}$. The adjacency matrix of a graph $\mathbf{G}$ is represented as $\mathbf{A}$. We use $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N)$ with $\mathbf{X} \in \mathbb{R}^{\mathrm{N} \times \mathrm{d}}$ to denote a d-dimensional node embedding and $\Gamma(i)$ to refer to the neighbourhood of node i.

2.1. Network Embedding Methods

NE methods can be broadly categorized into four classes, by the strategy used for learning node similarities. In this taxonomy we can distinguish methods based on random walks, matrix factorization, neural networks, and probabilistic approaches. We describe the methods included in the evaluation below, per class.

2.1.1. Methods based on random walks

These methods determine node similarities using random walks on the input graph. The Skip-Gram model, described in Mikolov et al. (2013), is then generally used to generate node embeddings from the random walks.

DeepWalk (Perozzi et al., 2014) is the first method to use techniques inspired by the deep learning community for NE. It uses random walks with fixed transition probabilities to measure node similarity and embeddings are derived using the Skip-Gram model.

Node2vec (Grover & Leskovec, 2016) is a generalization of DeepWalk which uses truncated random walks for node neighbourhood exploration and the Skip-Gram model, approximated via negative sampling, for embedding generation. The random walk properties are controlled by a return parameter p and an in-out parameter q.

Struc2vec (Ribeiro et al., 2017) extracts structural information from graphs through node pair similarities for a range of

Table 1. Hyperparameters tuned and implementations evaluated for each method. Except for AROPE, CNE and the LP heuristics, we also tune the edge embedding operator as model hyperparameter.

Methods	Implementations	Parameters
Heuristics	`EvalNE`	-
DeepWalk	Orig., `OpenNE`	$num\_walks = walk\_len = [5, 10, 20, 40, 80]$, $window\_size = [5, 10, 20]$
Node2vec	Orig., `OpenNE`	$num\_walks = walk\_len = [5, 10, 20, 40, 80]$, $window\_size = [5, 10, 20]$, $p = q = [0.5, 1, 2]$
Struc2vec	Orig.	$num\_walks = walk\_len = [5, 10, 20, 40, 80]$, $window\_size = [5, 10, 20]$
Metapath2vec	Orig.	$\alpha = [0.01, 0.025]$, $negative = [5, 10]$
WYS	Other	$lr = [0.01, 0.05]$, $num\_walks = [20, 40, 80]$, $window\_size = [5, 10, 20]$
GF	`OpenNE`	-
GraRep	`OpenNE`	$kstep = [2, 4, 8]$
HOPE	`OpenNE`, `GEM`	$\beta = [0.1, 0.01, 0.001, 0.0001]$
LE	`OpenNE`, `GEM`	-
LLE	`GEM`	-
M-NMF	Orig.	$clusters = [10, 20, 50]$
AROPE	Orig.	$weights = [[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0], [0, 0, 0, 1], [1, 0.1, 0.01, 0.001], [1, 0.5, 0.05, 0.005]]$
SDNE	`OpenNE`, `GEM`	$\beta = [2, 5, 10]$, $encoder\_list = [[128], [512, 128], [1024, 512, 128]]$
PRUNE	Orig.	$\lambda = [0.01, 0.05]$
VERSE	Orig.	$nsamples = [3, 5, 10]$
LINE	Orig. `OpenNE`	$\rho = [0.01, 0.025]$, $negative\_ratio = [5, 10]$
CNE	Orig.	$lr = [0.01, 0.05]$

neighbourhood sizes. This information is then summarized as a multi-layer weighted graph $\hat{\mathbf{G}}$. Subsequently, random walks on $\hat{\mathbf{G}}$ are used to generate the embeddings.

Metapath2vec (Dong et al., 2017) is a method capable of learning embeddings from heterogeneous networks. The authors extend the concept of random walks to account for nodes of different types and further use a heterogeneous Skip-Gram model to learn the embeddings.

Watch Your Step (WYS) (Abu-El-Haija et al., 2018) is an attention model learned on the power series of the transition matrix of $\mathbf{G}$. Node context importance is learned with minimal manually-tunable hyperparameters.

2.1.2. Methods based on matrix Factorization

Factorization based methods use representations of node similarities such as high-order proximities expressed as polynomials of $\mathbf{A}$, the incidence matrix, Katz similarity or the graph Laplacian $\mathbf{L}$. Node embeddings are then obtained by factorizing the selected matrix.

Graph Factorization (GF) (Ahmed et al., 2013) uses regularized Gaussian matrix factorization to recover a matrix $\mathbf{Z}$ such that $\mathbf{Z}\mathbf{Z}^T$ is close to $\mathbf{A}$ in terms of observed non-zeros.

GraRep (Cao et al., 2015) is based on factorization of different polynomials of the adjacency matrix $\mathbf{A}$ which identify relations between nodes in $\mathbf{G}$ at different resolutions.

HOPE (Ou et al., 2016) is also based on matrix factorization and preserving high-order proximities of graphs. The method additionally accounts for the asymmetric transitivity property of directed networks.

Laplacian Eigenmaps (LE) (Belkin & Niyogi, 2002) first constructs a weighted representation $\hat{\mathbf{A}}$ of $\mathbf{A}$ by leveraging first order proximities on $\mathbf{G}$. The Laplacian matrix $\mathbf{L}$ is computed using $\hat{\mathbf{A}}$ and embeddings are obtained from the d eigenvectors corresponding to the lowest eigenvalues of $\mathbf{L}$.

Locally Linear Embeddings (LLE) (Roweis & Saul, 2000) operates under the assumption that nodes and their neighbours lie on locally linear patches of a high-dimensional manifold. The embedding of a node, therefore, can be derived from the linear coefficients that better reconstruct the node from the embeddings of its neighbours.

M-NMF (Wang et al., 2017) incorporates community structure information in the embedding learning process via modularized non-negative matrix factorization.

AROPE (Zhang et al., 2018b) similarly to GraRep, proposes embeddings as found by the truncated singular value decomposition of polynomials of $\mathbf{A}$. The authors describe a fast eigen-decomposition method for these polynomials based on shifting or reweighing the decomposition of $\mathbf{A}$.

2.1.3. Methods based on neural Networks

Due to their ability to capture highly non-linear relations, deep models and particularly auto-encoders have also been used for network representation learning.

Structural Deep Network Embedding (SDNE) (Wang et al., 2016) uses a deep neural network for learning embeddings that capture first and second order proximities on the graph.

PRUNE (Lai et al., 2017) relies on a deep siamese neural network for learning node embeddings and can incorporate node ranking as additional information.

VERSE (Tsitsulin et al., 2018) learns embeddings by training a single layer neural network which minimizes the Kullback-Leibler (KL) divergence between node similarity in $\mathbf{G}$ and the vector similarity in $\mathbf{X}$. Noise Contrastive Estimation is used for scalability.

2.1.4. Probabilistic Methods

These methods use probabilistic approaches to model node similarities and learn embeddings.

LINE (Tang et al., 2015) uses joint and conditional probability distributions to model the first and second order adjacencies between linked nodes in $\mathbf{G}$. The skip-gram model is used to obtain node embeddings.

CNE (Kang et al., 2019) uses a Bayesian approach to generate embeddings which model the observed network while taking prior information into account. The prior can incorporate structural graph properties such as node degrees or block densities for clustered or multi-partite networks.

2.2. Baseline Heuristics

In addition to the NE methods, we evaluate a set of LP heuristics as baselines. These heuristics use the neighbourhoods $\Gamma(i)$ and $\Gamma(j)$ of nodes pairs $\{i, j\}$ to derive similarity scores which can be interpreted as link probabilities. In our experiments we consider Common Neighbours defined as: $CN_{\{i,j\}} = |\Gamma(i) \cap \Gamma(j)|$; Jaccard Coefficient, $JC_{\{i,j\}} = |\Gamma(i) \cap \Gamma(j)|/|\Gamma(i) \cup \Gamma(j)|$; Adamic Adar Index, $AA_{\{i,j\}} = \sum_{k \in \Gamma(i) \cap \Gamma(j)} 1/\log|\Gamma(k)|$; Resource Allocation Index, $RAI_{\{i,j\}} = \sum_{k \in \Gamma(i) \cap \Gamma(j)} 1/|\Gamma(k)|$ and Preferential Attachment $PA_{\{i,j\}} = |\Gamma(i)| \cdot |\Gamma(j)|$.

Additionally, we generate a new heuristic by joining in a 5-dimensional node-pair embedding CN, JC, AA, RAI and PA, followed by logistic regression to obtain link predictions. We will refer to this new heuristic as *NE_heuristics*.

3. Datasets

We conduct our experimental evaluation on 7 undirected real-world networks from different domains. These networks are medium-sized to ensure successful execution of all methods and constrain the computational resources needed. Next, we present a short description of each network and in Table 2, we summarize their main statistics.

StudentDB (Goethals et al., 2010) represents a snapshot of Antwerp University's relational student database. Nodes in the network represent entities such as students, professors, tracks, etc. and edges constitute binary relations, i.e. student-in-track, student-in-program, student-in-contract, student-take-course, professor-teach-course, course-in-room. **Facebook** (Leskovec & Krevl, 2015) and **BlogCatalog** (Zafarani & Liu, 2009) are online social networks where nodes represent different users and edges indicate friendships. **GR-QC** (Leskovec & Krevl, 2015) and **AstroPh** (Leskovec & Krevl, 2015) describe collaboration networks in the fields of General Relativity and Astrophysics. Nodes represent papers and edges denote citations between them. **PPI** (Breitkreutz et al., 2007) is a biological protein-protein interaction network constituting a subset of the Homo Sapiens PPI network. **Wikipedia** (Mahoney, 2011) contains nodes representing words in Wikipedia pages and links denote co-occurrences.

Table 2. Summary of dataset statistics where $2|\mathbf{E}|/|\mathbf{V}|$ denotes the average node degrees.

Dataset	Category	$\lvert\mathbf{V}\rvert$	$\lvert\mathbf{E}\rvert$	$2\lvert\mathbf{E}\rvert/\lvert\mathbf{V}\rvert$
StudentDB	Relational	395	3,423	17.33
Facebook	Social	4,039	88,234	43.69
BlogCatalog	Social	10,312	333,983	64.77
GR-QC	Collaboration	4,158	26,844	6.45
AstroPH	Collaboration	18,772	396,160	22.00
PPI	Biological	3,852	37,841	19.64
Wikipedia	Language	4,777	92,295	38.64

4. Experimental Setup

In this section we give details on the LP task, present the experimental setup used and discuss the limitations to the scope of the evaluation and reproducibility of results.

4.1. Link Prediction

As pointed out in Section 1 the objective in LP is to identify missing links in an incomplete graph $\mathbf{G}$. Thus, the first step to evaluate LP is to preprocess $\mathbf{G}$ and obtain sets of train and test node-pairs. We use the standard approach of generating an incomplete train graph $\mathbf{G}_{train} = (\mathbf{V}, \mathbf{E}_{train})$ from a more complete graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ where the connected node-pairs $\{i, j\} \in \mathbf{E} \setminus \mathbf{E}_{train}$ are used for testing. The proportion $f = |\mathbf{E}_{train}|/|\mathbf{E}|$ (or train fraction) is a user-defined parameter. For obtaining $\mathbf{E}_{train}$ and $\mathbf{E}_{test}$ we evaluate 3 approaches, namely *random* (Gurukar et al., 2019), *spanning tree (ST)* (Mara et al., 2019) and *depth first tree (DFT)*. The first is a random sampling strategy followed by a main connected components computation, while ST and DFT constructively ensure that $\mathbf{G}_{train}$ contains a spanning tree of $\mathbf{G}$. More details on these strategies are provided in the supplementary material. In addition to splitting connected pairs, we also generate sets of train and test non-edges, $\mathbf{D}_{train}$ and $\mathbf{D}_{test}$. The node-pairs in these sets are randomly selected using an open world assumption where any pair $\{i, j\} \notin \mathbf{E}_{train}$ is considered a valid train non-edge. Test non-edges are selected as pairs $\{i, j\} \notin (\mathbf{E} \cup \mathbf{D}_{train})$. In our experiments, we use the same number of connected and non-connected node-pairs.

For hyperparameter tuning the train sets need to be further split into train and validation. We fix these values to 90% train and 10% validation in all our experiments and the validation split is always performed using the same algorithm as the initial train-test split. Grid search is adopted as the strategy for learning the best model hyperparameters.

For most methods, with the exception of CNE, AROPE and the LP heuristics, for which node-pair similarities are directly computed, the link predictions are learned through binary classification. First, node-pair embeddings for $\mathbf{E}_{train}$, $\mathbf{D}_{train}$, $\mathbf{E}_{test}$ and $\mathbf{D}_{test}$ need to be obtained from the node embeddings $\mathbf{X}$ learned by a method on $\mathbf{G}_{train}$. The embedding of a pair $\{i, j\}$ can be computed by applying different operators $\circ$ to the embeddings of the incident nodes i and j i.e. $\mathbf{x}_{\{i,j\}} = \mathbf{x}_i \circ \mathbf{x}_j$. In our evaluation, we select the operators introduced in Grover & Leskovec (2016), namely *Average* $((\mathbf{x}_i + \mathbf{x}_j)/2)$, *Hadamard* $(\mathbf{x}_i \cdot \mathbf{x}_j)$, *Weighted* L_1 $(|\mathbf{x}_i - \mathbf{x}_j|)$ and *Weighted* L_2 $(|\mathbf{x}_i - \mathbf{x}_j|^2)$. A binary classifier is then fitted with the node-pair embeddings and labels $\{0, 1\}$ representing non-edges and edges, respectively.

4.2. Evaluation Setup

We used two experimental setups, called LP1 and LP2, with model hyperparameter tuning as main difference. While in LP1 we tune all hyperparameters presented in Table 1 as well as the node-pair operator, in LP2 we use default recommended parameters for all methods while still tuning the node-pair embedding strategy. In both cases, we provide results averaged over 3 experiment repeats and report test AUROC values and execution times. We quantify method accuracy in terms of AUROC as it is easily interpretable and the most commonly used LP evaluation metric. Sometimes we report $-\log(1 - AUROC)$, as it better reflects differences in performance for AUROC values $\approxeq 1$. Unless otherwise specified we use embedding dimensionality $d = 128$, train fraction $f = 0.8$, ST as edge sampling strategy and logistic regression (with 5-fold CV to tune the regularization parameter) as binary classifier. Finally, all methods are run for their pre-defined number of iterations.

Setup LP1 is used to analyse the performance of NE methods with respect to parameters $d \in \{8, 32, 128\}$ and $f \in \{0.2, 0.5, 0.8\}$. Setup LP2, on the other hand, is used to investigate the effect of the edge sampling strategies introduced in Section 4.1, and the choice of binary classifier.

4.3. Limits to the scope of the evaluation

Constraints on overall computation time (the results in Section 5 took 50.2 days to compute) inevitably imply boundaries to the scope of this evaluation: It is limited to popular mid-sized undirected, unweighted networks. It evaluates a representative but undoubtedly non-exhaustive set of state-of-the-art NE methods. It only evaluates methods purely exploiting the network structure (and not node/edge metadata), such that some methods are not shown in their full potential (but note that the same holds for the heuristic baselines). Furthermore, the selected heuristics are very simple, other more powerful similarity metrics between network nodes have been proposed. Including these could only strengthen the main conclusion of this paper though, namely that heuristic LP approaches, to date, remain competitive and often outperform LP based on complex NE methods.

4.4. Reproducibility Notes

In order to ensure the reproducibility of our results and foster further research in this area, we have based our experimental evaluation upon the `EvalNE` framework. This open source Python toolbox aims to simplify the evaluation of NE methods for LP, network reconstruction, node classification, and visualization. The toolbox automates tasks such as hyperparameter tuning, selection of train and test edges or non-edge sampling. It implements widely used node-pair embedding operators and can incorporate any classifier for prediction. Moreover, its design ensures that common erros, such as the computation of features on $\mathbf{G}$ rather than just on $\mathbf{G}_{train}$, or other forms of label leakage, are ruled out. Finally, `EvalNE` can assess the scalability of methods through wall clock time and performance through a wide range of accuracy measures e.g. AUROC, PR, F-score, and threshold curves. Configuration files describing our complete evaluation setup and which can be used directly in `EvalNE` to replicate our results, are made available in the supplementary material.

5. Experimental Results

In this section we present the results of our empirical study. More specifically, in Section 5.1 we discuss LP accuracy, in Section 5.2 hyperparameter tuning, embedding dimensionality in Section 5.3, train-test splits in Section 5.4, edge sampling in Section 5.5 and finally binary classification in Section 5.6. All experiments were conducted on a machine equiped with two 12 Core Intel(R) Xeon(R) Gold processors and 256GB of RAM.

5.1. Link Prediction Accuracy

We start in Table 3 by presenting the AUROC scores for each method on the LP task. These results were obtained using setup LP1 where we additionally tuned the embedding dimensionality, $d \in \{8, 32, 128\}$. All methods perform best for $d = 128$, except CNE, which performs best for $d = 8$ (see Section 5.3 for detailed discussion). The methods are grouped in the table according to the taxonomy in Section 2 with the best performing method per group highlighted in bold and the overall best for each network on grey background. The last two columns show the average performance of each method across all networks and their average rank among all methods.

The results in Table 3 show top performance of the LP heuristics on most datasets and in particular of RAI and our proposed NE_heuristics which, together with GraRep, achieves the best overall AUROC of 0.9629. Although the individual heuristics occasionally underperform, NE_heuristics is a robust top performer. Among random walk approaches DeepWalk and Node2vec perform best. GraRep is the matrix factorization-based method with highest scores and consistently outperforms its competitors. VERSE and CNE are respectively the best neural architecture-based and probabilistic method. Overall, NE_heuristics, GraRep, CNE, VERSE, AROPE and SDNE present competitive performance. It is worth noting that CNE achieves these state-of-the-art results with only 8 dimensions. Also noteworthy is that the results vary significantly (by up to 11,7%) between different implementations of the same NE method. Additional results in Appendix C show fluctuations of up to 44.5% in AUROC for `GEM`

Table 3. AUROC scores for setup LP1 where hyperparameters are tuned and $d = 128$ for all methods except CNE where $d = 8$.

Methods	StudentDB	Facebook	BlogCat.	GR-QC	AstroPH	PPI	Wikipedia	Avg. AUROC	Avg. AUROC Rank
CN	0.3701	0.9921	0.948	0.9588	0.9900	0.8629	0.8998	0.8602	17.21
JC	0.3701	0.9897	0.7704	0.9586	0.990	0.8387	0.3765	0.7563	22.07
AA	0.3701	0.9931	0.9522	0.9592	0.9906	0.8671	0.9191	0.8645	13.57
PA	0.9218	0.8420	0.9550	0.8392	0.8785	0.9047	0.9198	0.8944	16.50
RAI	0.3701	**0.9942**	**0.9575**	0.9592	0.9908	0.8670	**0.9313**	0.8672	10.64
NE_heuristics	**0.9663**	0.9933	0.9557	**0.9761**	**0.9930**	**0.9273**	0.9288	**0.9629**	**5.00**
DeepWalk	0.9060	0.9904	0.9428	**0.9860**	0.9837	0.9048	0.9031	0.9453	13.14
DeepWalk_opne	0.9059	0.9908	0.9430	0.9852	0.9834	**0.9055**	0.9044	**0.9455**	**13.00**
Node2vec	0.9479	**0.9937**	0.9377	0.9847	0.9890	0.8399	0.8932	0.9409	13.29
Node2vec_opne	0.8971	0.9912	0.9292	0.9856	**0.9916**	0.9004	0.9009	0.9423	13.57
Struc2vec	0.9333	0.8334	**0.9532**	0.8419	0.8735	0.9036	**0.9182**	0.8939	18.00
Metapath2vec	**0.9805**	0.9421	0.9482	0.8035	0.8580	0.8796	0.9030	0.9021	19.29
WYS	0.8188	0.9401	0.9151	0.8325	0.8549	0.8533	0.8638	0.8684	25.57
GF_opne	0.8678	0.9831	0.8975	0.9328	0.9469	0.8374	0.8337	0.8999	23.57
GraRep_opne	0.9686	**0.9926**	**0.9620**	**0.9841**	**0.9900**	**0.9210**	**0.9218**	**0.9629**	**5.29**
HOPE_gem	**0.9885**	0.9903	0.9554	0.9518	0.9501	0.9088	0.9193	0.9520	11.29
HOPE_opne	0.9139	0.9891	0.9437	0.9204	0.9465	0.8724	0.9155	0.9288	18.57
LE_gem	0.9061	0.9923	0.8001	0.9753	0.9337	0.7604	0.7673	0.8765	20.36
LE_opne	0.9062	0.9918	0.8025	0.9770	0.9324	0.7637	0.7712	0.8778	20.00
LLE_gem	0.8895	0.9895	0.7041	0.9697	0.8945	0.7255	0.7409	0.8448	23.57
M-NMF	0.9435	0.9923	0.9360	0.9834	0.9825	0.8783	0.9133	0.9470	13.36
AROPE	0.9817	0.9907	0.955	0.9678	0.9671	0.9097	0.9178	0.9557	10.79
SDNE_gem	**0.9872**	0.9787	0.9519	0.9450	0.9714	0.9095	0.9177	0.9516	13.29
SDNE_opne	0.9850	0.9866	0.9534	0.9573	0.9688	0.8979	0.9171	0.9523	13.86
PRUNE	0.9009	0.8377	0.9561	0.8357	0.8739	0.9035	**0.9197**	0.8896	17.71
VERSE	0.9349	**0.9943**	**0.9564**	**0.9902**	**0.9960**	**0.9194**	0.9185	**0.9585**	**4.57**
LINE	**0.9633**	0.9928	0.9308	**0.9842**	**0.9908**	0.8766	0.8823	0.9458	12.64
LINE_opne	0.8501	0.9910	**0.9315**	0.9333	0.9633	0.8947	0.8937	0.9225	19.29
CNE (d=8)	0.9459	**0.9938**	**0.9671**	0.9799	0.9763	**0.9281**	**0.9216**	**0.9589**	**6.00**

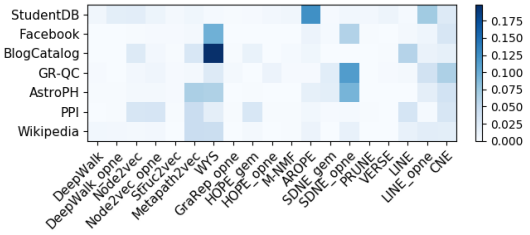

Figure 1. Improvement in AUROC of tuning model hyperparameters. Only methods with tuned parameters are shown.

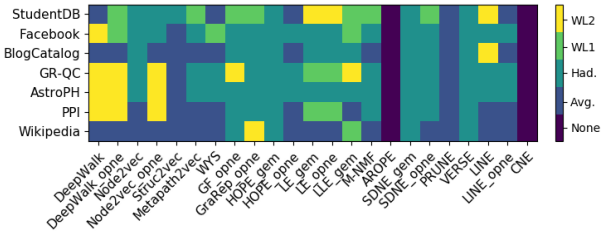

Figure 2. Best performing node-pair embedding operators for each method and datasets on setup LP1.

implementations due to package dependencies and up to 5% for metapath2vec due to parallelization.

5.2. Hyperparameter Tuning

Figure 1 presents a heatmap of the increment in LP accuracy obtained through hyperparameter tuning, i.e. the difference in AUROC between setups LP1 and LP2. Only methods for which hyperparameters were tuned are shown in this figure. The results reveal only limited improvements in most cases, however, in 8% of the network-method combinations significant improvements of up to 5% can be observed. Methods including WYS, SDNE and CNE benefit most from parameter tuning. Popular random walk methods such as DeepWalk and Node2vec for which parameter tuning is tedious, show minimal improvements in AUROC.

In Figure 2 we present the node-pair embedding operator selected using hyperparameter tuning for each method and network on experimental setup LP1. The results indicate that Hadamard is the most frequently selected and that the neural network-based models prefer specific operators, Hadamard in the case of SDNE and VERSE and Average for PRUNE. The remaining methods present a mix of operators. On average, over all methods and datasets we observe a difference in validation AUROC between the best and the worst performing operator of 0.144 with a standard deviation of 0.091. This clearly highlights the need to tune the node-pair embedding operator as a method hyperparameter.

For each method, the sum of execution times on the 7 evaluated networks for experimental setups LP1 and LP2 are compared in Figure 3. As expected, runtimes for LP1 are larger than those of LP2 with significant increments of more than one order of magnitude for Node2vec and GraRep. Naive sequential implementations of the LP heuristics are still faster than heavily optimized and parallelized NE meth-

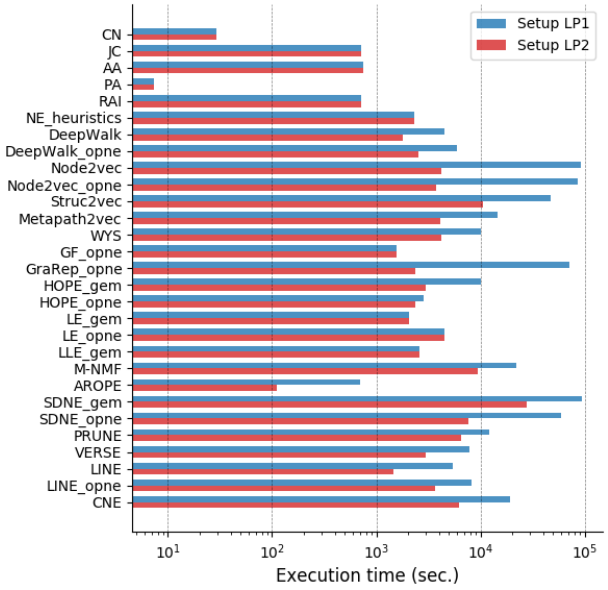

Figure 3. Execution times in seconds of setups LP1 and LP2.

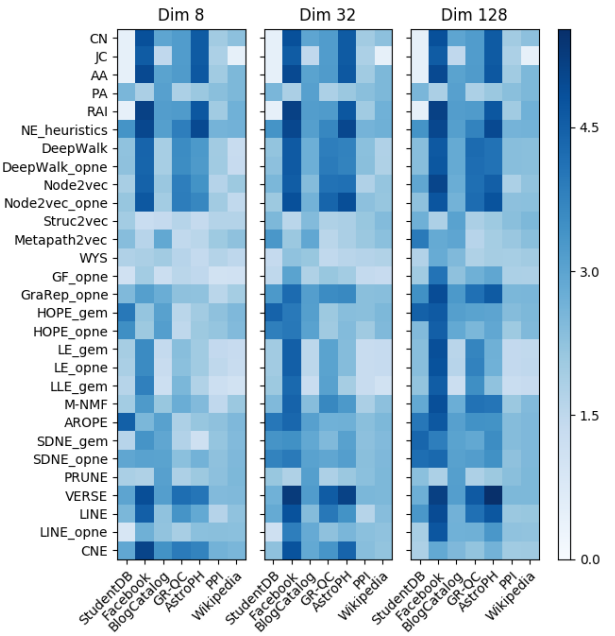

Figure 4. Method performance as $-\log(1 - AUROC)$ on setup LP1 for $d \in \{8, 32, 128\}$. Dark colours indicate better results.

ods. AROPE is the only embedding method with a runtime comparable to those of the heuristics. Finally, with no parameter tuning, similar patterns can be observed within each method family with factorization methods being the fastest.

5.3. Embedding Dimensionality

We evaluate the effect of embedding dimensionality on method performance by modifying setup LP1 and computing the AUROC of all methods for $d \in \{8, 32, 128\}$. For visualization purposes we compute $-\log(1 - AUROC)$ and present the results in three heatmaps in Figure 4 where a darker colour indicates better performance. The LP heuristics do not depend on d but are shown for reference. These results support the conclusions extracted from Table 3 regarding the best performing methods. They also show improved performance as d grows for all methods except CNE for which the opposite is true. For AROPE we can observe overfitting on the small StudentDB network. In this case, the method computes embeddings by eigen-decomposition of $\mathbf{A}^3$, which performs well for low values of d but significantly worse as d increases.

5.4. Train-Test Split Size

We further use experimental setup LP1 to analyse the effect on method performance of the train fraction f. Due to space constraints heatmaps showing method performance for $f \in \{0.2, 0.5, 0.8\}$ are included in the supplementary material (Appendix D). For GR-QC performing a 20-80 split while keeping the train graph connected was not possible due to a low edge density, thus we resort to using a 35-65 split. An interesting observation from these results is that for $f > 0.5$ most methods capture well the network structures while this is not the case for $f < 0.5$. This is reflected by an average increase in AUROC over all methods of 7% between $f = 0.2$ and $f = 0.5$ while only a 2% difference can be observed between $f = 0.5$ and $f = 0.8$. Additionally, we find that neural network-based NE methods are most robust to varying train sizes followed by probabilistic, random walk-based, matrix factorization, and finally the LP heuristics. The proposed NE_heuristics method, GraRep, VERSE, and CNE best capture the network structure when only 20% of train edges are available. Lastly, regarding method runtimes, these increase by approximatelly 50% from evaluations with $f = 0.2$ to $f = 0.8$. Some notable exceptions are Metapath2vec, SDNE and Node2vec_opne, for which the runtimes increase 82.6%, 78.3% and 73.2% respectively.

5.5. Edge Sampling

We also conduct an experiment to compare the three strategies introduced in Subsection 4.1, i.e. random, ST and DFT for splitting $\mathbf{E}$ into $\mathbf{E}_{train}$ and $\mathbf{E}_{test}$. Our aim is to study the impact of these strategies on LP performance and sampling execution time. We isolate the effect of this pipeline component by using experimental setup LP2 where hyperparameter tuning is not performed. Our results show minimal differences in method accuracy between the three strategies. More precisely, the average AUROC over all methods and datasets using random edge split is 0.897 with a standard

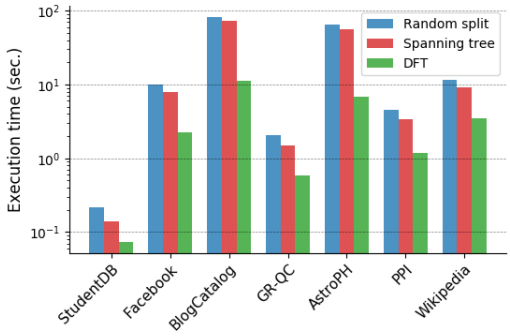

Figure 5. Execution times of different train-test split algorithms.

deviation of 0.107 while for the ST strategy the average AUROC is 0.902 with the same standard deviation of 0.107. For DFT the average AUROC is also 0.902 with a standard deviation of 0.123.

For very large networks, the edge sampling strategy can become a bottleneck and, thus, sampling runtimes are also worth investigating. Our results, depicted in Figure 5, show that the random strategy is the slowest followed by ST and finally, DFT which is up to one order of magnitude faster on most networks. The slower runtimes of the random strategy are due to set intersections to obtain $\mathbf{E}_{train}$ and $\mathbf{E}_{test}$. In addition, we have found that the random edge split strategy does not preserve all nodes from the input graph $\mathbf{G}$ in the train graph $\mathbf{G}_{train}$. On average, over all datasets 2.5% of the nodes in $\mathbf{G}$ are lost. This effect is especially severe for the networks with the lowest average degrees, i.e. StudentDB and GR-QC, which lose up to 8% of their nodes. The ST and DFT strategies, on the other hand, preserve all nodes.

We have also monitored the variation in performance when embeddings were learned from different train graphs $\mathbf{G}_{train}$. For each edge split strategy we have run 3 different executions with varying initial random seeds. We did not find any significant differences with a maximum standard deviation observed across all datasets, methods and strategies of only 0.004. The largest variations were observed for the smallest network, StudentDB. Ultimately, this experiment reveals that averaging LP results over different random edge splits to obtain unbiased estimates of method performance is unnecessary for networks of similar sizes.

5.6. Binary Classification

In this experiment we evaluate the changes in AUROC due to the binary classification method used. We adapt experimental setup LP2 and consider the following classifiers: logistic regression (LR), logistic regression with 5-fold cross validation (LRCV) and decision trees (DT). A heatmap of standard deviations of the AUROCs obtained with each classifier are included in Appendix E. In contrast to recent results suggesting large variations in performance between LR and LRCV (Gurukar et al., 2019), we did not find significant differences in our evaluation. The large values in the figure are fundamentally due to the DT classifier. When predictions were given by a DT classifier, the performance of two matrix factorization approaches, LE and LLE, improved. For LE we observed an increment in average AUROC over all networks of 6.5%, from 0.87 using LRCV to 0.93 with DT. For LLE the increment was of 7.8%, from an AUROC of 0.84 with LRCV to 0.91 with DT. All other methods, and particularly the `OpenNE` implementations of Node2vec and LINE, showed lower AUROC scores for DT as compared to LRCV.

5.7. External validation of the results

Due to differences in the evaluations, we can only validate our results, to some extent, against the empirical results in the Node2vec and CNE papers. In the first case, although our AUROC scores are consistently higher, the same main conclusion holds: node2vec outperforms its competitors on the three networks originally selected. However, a properly tuned LINE exhibits similar performance or even outperforms node2vec on other networks. In the second case, our results are very similar to those reported by the authors of CNE. Finally, our evaluation also corroborates the conclusions reported by the authors of AROPE and VERSE as well as the choice of node-pair operator for the latter.

6. Conclusions and Discussion

In this paper we have conducted an extensive empirical study on NE methods for LP. Our results show that, despite the surge of interest in the field in recent years, little progress has been made. Complex state-of-the-art NE methods are still consistently matched or even outperformed by simple baselines with no hyperparameters, in terms of both accuracy and runtime. On average, the proposed heuristic-based baseline outperforms all others. We have also shown that for all methods, with the exception of CNE, performance improves as embedding dimensionality increases. In contrast with recently published results, our experiments indicate no significant difference between LR and LRCV as binary classifiers in this setting. We also highlight the need to tune the node-pair embedding operator as model hyperparameter and that a single train-test split provides a good estimation of method accuracy resulting in higher evaluation efficiency. Finally, we hope this study and evaluation toolboxes such as `EvalNE` serve as initial steps towards the creation of standard evaluation pipelines for NE methods, and shed light on the current state-of-the-art. Specifically, we plan to use these results as a basis for an online resource that is continuously augmented with results for newly developed methods and/or other networks.

Acknowledgements

The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC Grant Agreement no. 615517, from the Flemish Government under the "Onderzoeksprogramma Artificile Intelligentie (AI) Vlaanderen" programme, from the FWO (project no. G091017N, G0F9816N, 3G042220).

References

Abu-El-Haija, S., Perozzi, B., Al-Rfou, R., and Alemi, A. Watch your step: Learning node embeddings via graph attention. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS18, pp. 91989208, Red Hook, NY, USA, 2018. Curran Associates Inc.

Ahmed, A., Shervashidze, N., Narayanamurthy, S., Josifovski, V., and Smola, A. J. Distributed large-scale natural graph factorization. In *Proc. of WWW*, pp. 37–48, 2013.

Belkin, M. and Niyogi, P. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Proc. of NIPS*, pp. 585–591, 2002.

Breitkreutz, B.-J., Stark, C., Reguly, T., Boucher, L., Breitkreutz, A., Livstone, M., Oughtred, R., Lackner, D. H., Bähler, J., Wood, V., et al. The biogrid interaction database: 2008 update. *Nucleic acids research*, 36:D637–D640, 2007.

Cao, S., Lu, W., and Xu, Q. GraRep: Learning graph representations with global structural information. In *Proc. of CIKM*, pp. 891–900, 2015.

Dong, Y., Chawla, N. V., and Swami, A. Metapath2vec: Scalable representation learning for heterogeneous networks. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, pp. 135–144, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-4887-4. doi: 10.1145/3097983.3098036.

Gao, M., Chen, L., He, X., and Zhou, A. Bine: Bipartite network embedding. In *Proc. of SIGIR*, pp. 715–724, 2018.

Garcia-Gasulla, D., Cortes, C. U., Ayguade, E., and Labarta, J. J. Evaluating link prediction on large graphs. In *Proc. of CAAI*, pp. 90–99, 2015.

Goethals, B., Le Page, W., and Mampaey, M. Mining interesting sets and rules in relational databases. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, pp. 997–1001, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-639-7. doi: 10.1145/1774088.1774299.

Goyal, P. and Ferrara, E. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems*, 151:78–94, 2018. doi: 10.1016/j.knosys.2018.03.022.

Grover, A. and Leskovec, J. node2vec: Scalable feature learning for networks. In *Proc. of KDD*, pp. 855–864, 2016.

Gurukar, S., Vijayan, P., Srinivasan, A., Bajaj, G., Cai, C., Keymanesh, M., Kumar, S., Maneriker, P., Mitra, A., Patel, V., Ravindran, B., and Parthasarathy, S. Network representation learning: Consolidation and renewed bearing. *ArXiv*, abs/1905.00987, 2019.

Hamilton, W. L., Ying, R., and Leskovec, J. Representation learning on graphs: Methods and applications, 2017.

Kang, B., Lijffijt, J., and Bie, T. D. Conditional network embeddings. In *International Conference on Learning Representations*, 2019.

Kotnis, B. and Nastase, V. Analysis of the impact of negative sampling on link prediction in knowledge graphs. *ArXiv*, abs/1708.06816, 2017.

Lai, Y.-A., Hsu, C.-C., Chen, W. H., Yeh, M.-Y., and Lin, S.-D. Prune: Preserving proximity and global ranking for network embedding. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 30*, pp. 5257–5266. Curran Associates, Inc., 2017.

Leskovec, J. and Krevl, A. SNAP Datasets: Stanford large network dataset collection, 2015.

Lichtenwalter, R. N. and Chawla, N. V. Link prediction: Fair and effective evaluation. In *Proc. of ASONAM*, pp. 376–383, 2012.

Mahoney, M. Large text compression benchmark. *URL: http://www.mattmahoney.net/text/text.html*, 2011.

Mara, A., Lijffijt, J., and Bie, T. D. Evalne: A framework for evaluating network embeddings on link prediction, 2019.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. Efficient estimation of word representations in vector space, 2013.

Ou, M., Cui, P., Pei, J., Zhang, Z., and Zhu, W. Asymmetric transitivity preserving graph embedding. In *Proc. of KDD*, pp. 1105–1114, 2016.

Perozzi, B., Al-Rfou, R., and Skiena, S. Deepwalk: Online learning of social representations. In *Proc. of KDD*, pp. 701–710, 2014.

Ribeiro, L. F., Saverese, P. H., and Figueiredo, D. R. struc2vec: Learning node representations from structural identity. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 385–394. ACM, 2017.

Roweis, S. T. and Saul, L. K. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290 (5500):2323–2326, 2000. ISSN 0036-8075. doi: 10.1126/science.290.5500.2323.

Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., and Mei, Q. LINE: Large-scale information network embedding. In *Proc. of WWW*, pp. 1067–1077, 2015.

Tsitsulin, A., Mottin, D., Karras, P., and Müller, E. Verse: Versatile graph embeddings from similarity measures. In *Proceedings of the 2018 World Wide Web Conference*, WWW 18, pp. 539548, Republic and Canton of Geneva, CHE, 2018. International World Wide Web Conferences Steering Committee. ISBN 9781450356398. doi: 10.1145/3178876.3186120.

Wang, D., Cui, P., and Zhu, W. Structural deep network embedding. In *Proc. of KDD*, pp. 1225–1234, 2016.

Wang, X., Cui, P., Wang, J., Pei, J., Zhu, W., and Yang, S. Community preserving network embedding. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI'17, pp. 203–209. AAAI Press, 2017.

Wei, X., Xu, L., Cao, B., and Yu, P. S. Cross view link prediction by learning noise-resilient representation consensus. In *Proc. of WWW*, pp. 1611–1619, 2017.

Wilson, D. B. Generating random spanning trees more quickly than the cover time. In *Proc. of STOC*, pp. 296–303, 1996.

Yang, Y., Lichtenwalter, R. N., and Chawla, N. V. Evaluating link prediction methods. *KAIS*, 45(3):751–782, 2015.

Zafarani, R. and Liu, H. Social computing data repository at asu, 2009.

Zhang, D., Yin, J., Zhu, X., and Zhang, C. Network representation learning: A survey, 2018a.

Zhang, Z., Cui, P., Wang, X., Pei, J., Yao, X., and Zhu, W. Arbitrary-order proximity preserved network embedding. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, pp. 2778–2786, New York, NY, USA, 2018b. ACM. ISBN 978-1-4503-5552-0. doi: 10.1145/3219819.3219969.

Supplementary material for the paper: Network Representation Learning for Link Prediction: Are we improving upon simple heuristics?

A. Hyperparameters and Implementations

In this section we descibe the hyperparameters tuned for each embedding method as well as the implementations conssidered.

DeepWalk: For DeepWalk we evaluate the author's original implementation and the one available in the `OpenNE` library [1]. The main difference between these implementations is the approximation of the Skip-Gram model via hierarchical softmax and negative sampling, respectively. Regarding parameters, we tune the window size, number of walks and walk length.

Node2vec: For Node2vec we also evaluate the original and `OpenNE` implementations. We tune the same parameters as for DeepWalk and additionally the p and q values governing the random walks.

Struc2vec: For struc2vec we only evaluate the original implementation by the authors and tune the same window size, number of walks and walk length as for DeepWalk and Node2vec.

Metapath2vec: In our experiments we consider the original implementation of metapath2vec++ and tune the Skip-Gram learning rate (α) and number of negative samples.

Watch Your Step (WYS): For WYS the original implementation is not directly usable so we resort to evaluating the one provided at `https://github.com/benedekrozemberczki/AttentionWalk`. We tune the number of random walks, Skip-Gram window size and learning rate.

Graph Factorization (GF): The implementation of GF evaluated is the one available in the `OpenNE` library and no model parameters were tuned.

GraRep: For GraRep we study the `OpenNE` implementation and tune the k-step parameter which encodes the order of the transition probability matrix.

HOPE: The `OpenNE` and `GEM` implementations of this method are evaluated and we tune the decay factor β.

[1] `https://github.com/thunlp/OpenNE`

Laplacian Eigenmaps (LE): In our empirical evaluation we consider the `OpenNE` and `GEM` implementations of LE. No method-specific hyperparameters are tuned.

Locally Linear Embeddings (LLE): For LLE we evaluate the implementation available in the `GEM` library and no parameter are tuned.

M-NMF: For M-NMF we use the original code by the authors and we tune the number of clusters as hyperparameter.

AROPE: We evaluate the original AROPE code and tune the weight parameter which indicates the order of the proximity. We restrict our analysis to polynomials of order 4.

Structural Deep Network Embedding (SDNE): For SDNE we evaluate the `OpenNE` and `GEM` implementations. We tune the edge reconstruction weight β and the number of neurons and layers of the deep network.

PRUNE: We consider the original implementation of PRUNE and tune the λ parameter controlling the relative importance of node ranking w.r.t. the node proximity in the graph.

VERSE: In our experiments we evaluate the Python implementation released by the authors and tune the number of negative samples ($nsamples$).

LINE: For LINE we evaluate the original and `OpenNE` implementations and tune ρ, the initial learning rate and *negative_ratio*, the number of non-edges per true edge used in the optimization procedure.

CNE: For CNE only the original implementation is available and we tune the learning rate parameter.

B. Edge selection algorithms

In this section we present the three train and test set selection procedures compared in our experimental setup.

Random: This approach proposed in Gurukar et al. (2019) starts by randomly selecting a set of edges from $\mathbf{E}$ as 'initial' test edges $\hat{\mathbf{E}}_{test}$. The main connected component spanned by the remaining edges $\{i, j\} \in \mathbf{E} \setminus \hat{\mathbf{E}}_{test}$ is then computed. These edges are considered as the training edges

$\mathbf{E}_{train}$ spanning a train graph $\mathbf{G}_{train} = (\mathbf{V}_{train}, \mathbf{E}_{train})$. Finally, only the test edges $\{i, j\} \in \hat{\mathbf{E}}_{test}$ such that $i \in \mathbf{V}_{train}$ and $j \in \mathbf{V}_{train}$ form the set of final 'refined' test edges $\mathbf{E}_{test}$. The sizes of the train and test edge sets are expected to vary between different executions of the algorithm.

Spanning tree: This approach was proposed in Mara et al. (2019). It amounts to computing the main connected component of the input graph. A spanning tree of the graph is selected uniformly at random using the algorithm proposed in Wilson (1996). These edges are part of the train edge set $\mathbf{E}_{train}$. Additional edges are selected uniformly at random from the graph and added to $\mathbf{E}_{train}$ until a set fraction $f = \mathbf{E}_{train}/\mathbf{E}$ is reached. The remaining edges are used for testing as $\mathbf{E}_{test}$. The sizes of the edge sets returned by this approach in different execution on the same graph are expected to be constant.

DFT: This approach is a faster version of the spanning tree method but with a fundamental difference i.e. there is no guarantee that $\mathbf{G}_{train}$ will be an unbiased representation of the original graph $\mathbf{G}$. In this case, a spanning tree of $\mathbf{G}$ is computed using a depth first tree (DFT) algorithm. Edges are added to this initial $\mathbf{E}_{train}$ until the quota is filed. The remaining edges are used for testing.

C. Performance variations

In this section we present a number of software and method implementation-related issues encountered during the benchmark process. Specifically, for Metapath2vec we have found an important degradation in performance when parallelism is increased beyond 4 threads. Table 4 summarizes the average AUROC, standard deviation and p-values over 5 independent runs of the method using 4 and 8 thread parallelism. The results are presented for all edge embedding heuristics considered in our evaluation using experimental setup LP2. The original implementation of LINE is shown for reference. The results using 1 and 16 threads, omitted in the table, are very similar to those obtained with 4 and 8 threads respectively. Our results indicate statistically significant differences at the 0.01 level for Metapath2vec on the Facebook dataset for different levels of parallelism.

Throughout our experiments we have also found strong variations in performance for all methods in the `GEM` library. These fluctuations in AUROC of up to 44.5% appear to be related to the versions of `GEM` dependencies installed. The authors of `GEM` do provide a list of minimal required versions for these libraries, however, there are many possible combinations which satisfy these premisses. In Figure 6 we present the difference un AUROC for two such sets of `GEM` dependencies. The average AUROC over all methods and datasets in one case is 0.673 while in the second is 0.902. In the figure, we observe that most of this variation is concentrated around the AstroPH, GR-QC and BlogCatalog datasets. The results of SDNE and HOPE also fluctuate more than those of LE and LLE.

Table 4. Mean and std deviation of AUC-ROC for LINE and Metapath2vec with 4 and 8 threads. Results are only shown for the Facebook, PPI and AstroPH networks. The last column reports p-values computed from 5 independent runs with 4 and 8 threads.

Dataset	Method	4 threads	8 threads	p
Facebook	LINE(avg.)	0.731 ± 0.005	0.002	0.283
	LINE(had.)	0.968 ± 0.001	0.964 ± 0.002	0.007
	LINE(W L_1)	0.682 ± 0.016	0.679 ± 0.013	0.707
	LINE(W L_2)	0.69 ± 0.017	0.686 ± 0.014	0.703
	Metapath2vec(avg.)	0.75 ± 0.003	0.73 ± 0.003	$4.30 \cdot 10^{-6}$
	Metapath2vec(had.)	0.89 ± 0.007	0.857 ± 0.006	$5.45 \cdot 10^{-5}$
	Metapath2vec(W L_1)	0.912 ± 0.005	0.912 ± 0.008	0.920
	Metapath2vec(W L_2)	0.896 ± 0.005	0.889 ± 0.009	0.205
PPI	LINE(avg.)	0.824 ± 0.001	0.821 ± 0.003	0.074
	LINE(had.)	0.763 ± 0.007	0.756 ± 0.008	0.189
	LINE(W L_1)	0.763 ± 0.003	0.761 ± 0.002	0.319
	LINE(W L_2)	0.766 ± 0.003	0.764 ± 0.002	0.364
	Metapath2vec(avg.)	0.839 ± 0.002	0.837 ± 0.006	0.661
	Metapath2vec(had.)	0.824 ± 0.003	0.822 ± 0.005	0.416
	Metapath2vec(W L_1)	0.706 ± 0.009	0.702 ± 0.005	0.422
	Metapath2vec(W L_2)	0.692 ± 0.006	0.69 ± 0.008	0.686
arXiv	LINE(avg.)	0.869 ± 0.001	0.869 ± 0.001	0.984
	LINE(had.)	0.982 ± 0.001	0.982 ± 0.001	0.878
	LINE(W L_1)	0.628 ± 0.012	0.624 ± 0.01	0.512
	LINE(W L_2)	0.636 ± 0.012	0.631 ± 0.009	0.463
	Metapath2vec(avg.)	0.768 ± 0.002	0.77 ± 0.005	0.360
	Metapath2vec(had.)	0.828 ± 0.006	0.837 ± 0.007	0.054
	Metapath2vec(W L_1)	0.646 ± 0.006	0.641 ± 0.01	0.336
	Metapath2vec(W L_2)	0.642 ± 0.006	0.638 ± 0.01	0.419

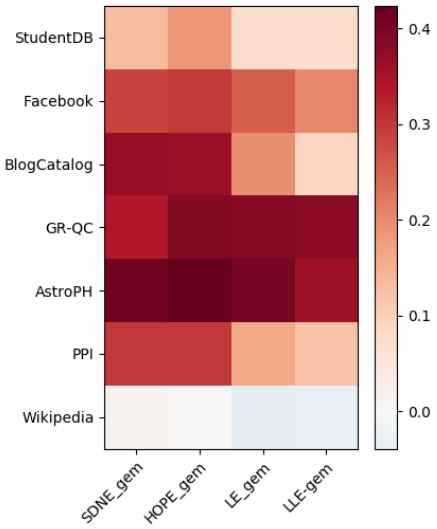

Figure 6. Difference in performance for two sets of `GEM` dependencies. A darker colour indicates a larger discrepancy between the results.

D. Train edge fraction

The results for different train fractions $f \in \{0.2, 0.5, 0.8\}$ are presented in Figure 7. A discussion of these results is provided in Section 5.4.

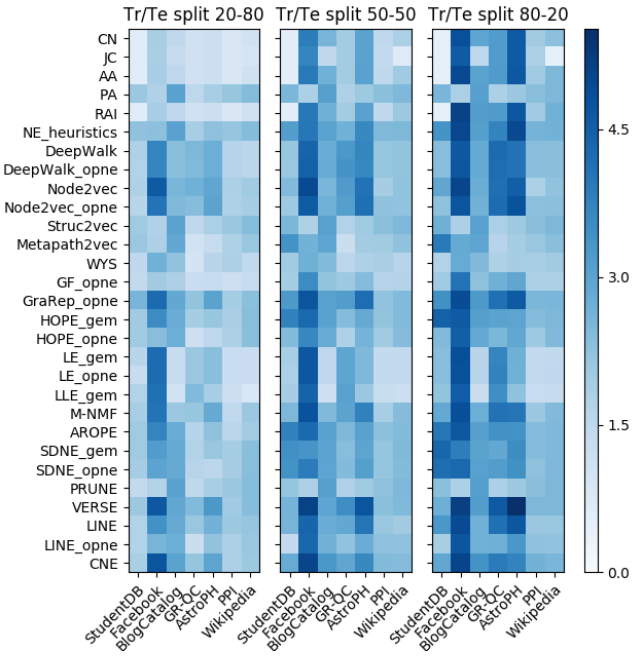

Figure 7. Method performance presented as $-\log(1 - AUROC)$ on setup LP1 for train-test edge splits 20-80, 50-50 and 80-20. Darker colours represent better performance.

E. Binary classifiers

The standard deviations between the AUROC results obtained with the three evaluated binary classifiers are presented in Figure 8. A detailed discussion of these results is provided in Section 5.6.

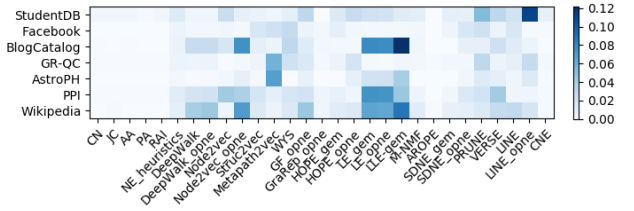

Figure 8. Difference in LP AUROC with different binary classifiers.