Big Data
Volume 12, Number 6, 2024
Mary Ann Liebert, Inc.
DOI: 10.1089/big.2021.0107

ORIGINAL ARTICLE

An Empirical Evaluation of Network Representation Learning Methods

Alexandru Cristian Mara,* Jefrey Lijffijt, and Tijl De Bie

Abstract

Network representation learning methods map network nodes to vectors in an embedding space that can preserve specific properties and enable traditional downstream prediction tasks. The quality of the representations learned is then generally showcased through results on these downstream tasks. Commonly used benchmark tasks such as link prediction or network reconstruction, however, present complex evaluation pipelines and an abundance of design choices. This, together with a lack of standardized evaluation setups, can obscure the real progress in the field. In this article, we aim at investigating the impact on the performance of a variety of such design choices and perform an extensive and consistent evaluation that can shed light on the state-of-the-art on network representation learning. Our evaluation reveals that only limited progress has been made in recent years, with embedding-based approaches struggling to outperform basic heuristics in many scenarios.

Keywords: representation learning; network embedding; benchmark; evaluation; network reconstruction; link prediction

Introduction

Networks are data representations that arise naturally in many fields from social sciences to medicine, biology, or computer science.[1,2–4] They model entities as graph nodes and their relations as edges. These particular structures, though highly expressive, do not directly support traditional machine-learning tasks such as classification, regression, or clustering. In this context, network representation learning or network embedding (NE) methods learn mappings from network nodes to low dimensional vectors of an embedding space.[5–9] The obtained vector representations generally preserve certain network properties and constitute convenient transformations that support downstream prediction and inference tasks.

To assess the quality of the learned representations, downstream tasks such as data visualization,[10,11] node multi-label classification,[12,13] link prediction,[14,15] and network reconstruction[8,16] are evaluated. Among these tasks, data visualization is an effective evaluation tool that is limited to small-sized networks. Multi-label classification is a relatively straightforward task that can be performed on larger networks and, thus, has been extensively studied in recent literature.[17–19]

Link prediction and network reconstruction, on the other hand, have received far less attention. The main reason for this is their need for more complex evaluation pipelines, with several preprocessing steps and design choices that can confound the results and are prone to errors. A recent empirical study[20] does, however, consider the link prediction task. In this study, the authors unify various representative NE models under the same framework and aim at explaining their differences by using a small set of experimental setups.

In this article, we focus on link prediction,* where we aim at estimating the likelihood of the existence

*Note that *link reconstruction* would be a more appropriate term for referring to the task of predicting missing edges of a static network. *Link prediction*, on the other hand, should specifically denote the prediction of new links in time evolving networks. This distinction, however, is rarely made in the representation learning literature, where the latter term is used to refer to predictions in both static and dynamic networks.

Department of Electronics and Information Systems, Ghent University, Ghent, Belgium.

*Address correspondence to: *Alexandru Cristian Mara, Department of Electronics and Information Systems, Ghent University, Technologiepark-Zwijnaarde, Ghent 9052, Belgium, E-mail:* alexandru.mara@ugent.be

of edges between given node-pairs not connected in the input graph, and on network reconstruction, which amounts to predicting the edges of a given network from its node embeddings. We also cover a broad range of experimental setups and implementations of unsupervised NE methods in a transductive scenario. The two evaluated tasks are strongly related and present similar challenges, as discussed next:

1. Data preprocessing: To evaluate the generalization performance of NE methods, specifically for link prediction, sets of train and test/validation edges (i.e., connected node-pairs) are required. These sets can be obtained by using different approaches that might impact the evaluation results. For instance, a typical assumption 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 is used for testing/validation.[2–4] Unfortunately, networks with such a temporal component are scarce, and therefore authors ordinarily resort to *sampling edges* from a network as test/validation examples and using the remaining edges for training.[9,15,21,22] The sampling process varies between scientific works in different aspects. The sizes of the train and test/validation sets vary from 50-50,[15,21] to 60-40[9] or 80-20.[22] The algorithms used to generate these splits also differ and while some aim at constructing training networks with similar, scaled-down, properties such as, for example, node degrees,[23] others generate training graphs that preserve the general topology of the original network.[24]
2. Prediction pipeline: Different NE methods require different pipelines to obtain link predictions. Although some embedding methods directly compute link probabilities,[15,16] for others, these need to be learned on top of the node embeddings. Two common approaches are: (i) interpreting some notion of similarity between two node embeddings (e.g., dot product) as the link probability and (ii) casting the problem as a binary classification task. The latter 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 that are, in turn, fed into a binary classifier to perform link prediction or network reconstruction. The choice of operator not only varies between scientific works[13,21] but is also, in some cases, completely overlooked in the experimental setup discussion.[10,22] Moreover, many valid choices exist for the binary classifier, and classifier training requires—in addition to "positive" examples or edges—"negative" samples or non-edges. These non-edges can also be selected by using different strategies and be of varying sizes.[25]
3. Hyperparameter tuning: When comparing new methods with the state-of-the-art, for the baseline methods the default parameter settings are often used,[13,16] 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 unfavorable picture of the baseline methods.
4. Evaluation metrics: Finally, no consensus exists on which evaluation criteria should be used for comparing different methods. Although some papers advocate for the use of precision@*Np* for a range of *Np* values,[8,10,16] others use area under the ROC curve (AUC)[15,21] or precision and recall at fixed thresholds.[26]

These challenges have led to an inconsistency in evaluation procedures throughout papers. Hence, the practical performance of NE methods is poorly understood. Moreover, as several recent studies have found,[27,28] the inconsistency of evaluation procedures is a central issue that has stalled progress in many sub-fields within Artificial Intelligence. As such, in this article, we aim at clarifying the impact of these design choices on the evaluation pipelines and ultimately on method performance. At the same time, we aim at shedding light on the real state-of-the-art by means of a standard evaluation process.

Contributions

We provide an in depth empirical analysis of the state-of-the-art on network representation learning. We focus on link prediction and network reconstruction and evaluate a total of 29 implementations of 23 different NE methods on 7 popular real-world networks. We study the impact on method performance of different evaluation pipelines and specific components such as hyperparameter tuning, embedding dimensionality, train set size, edge sampling strategy, node-pair embedding operator, and binary classifier.

Finally, there are inevitable limitations to the scope of our evaluation, such as an exclusive focus on undirected and unweighted networks without attributes, moderate-sized networks, and a representative but finite set of evaluated methods. To alleviate these limitations, we based our evaluation on Eva1NE,[24] a framework that ensures the reproducibility of results and allows for direct extensions of our work in all the aforementioned aspects.

This article is an extension of Ref.[29] The main difference is the extended experimental evaluation: We have included one additional downstream task (network reconstruction; Network Reconstruction section, Network Reconstruction Performance section; Table 4 and Fig. 10), one new heuristic baseline, and six new NE methods (changes throughout); included a comparison of different evaluation pipelines (Prediction Pipeline section; Fig. 8); and deepened the analysis of the link prediction performance and the use of embedding dimensions by different approaches (Prediction Pipeline section and Method Performance and Network Structure section; Fig. 9).

The remainder of this article is organized as follows. In the Methods section, we present the methods evaluated. The Datasets section introduces the real-world datasets used. A detailed description of the evaluation setup is given in the Experimental Setup section. In the Experimental Results section, we summarize our findings and finally, the Conclusions section concludes this article.

Methods

In this section, we present the NE methods (NE Methods section) and baseline heuristics (Baseline Heuristics section) evaluated, and we discuss the specific implementations used and method hyperparameters tuned (Implementations and Hyperparameters section).

Before continuing with method descriptions, we first lay down some notation. We represent an undirected network or graph 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-pair sets as $\mathbf{E}_{test}$ and $\mathbf{D}_{test}$. The adjacency matrix of a graph $\mathbf{G}$ is represented as $\mathbf{A}\in\{0,1\}^{N\times N}$, with element at row i and column j equal to 1 if $\{i,j\}\in\mathbf{E}$ and to 0 otherwise. We use $\mathbf{X}=(\mathbf{x}_1,\mathbf{x}_2,\ldots,\mathbf{x}_N)$ with $\mathbf{X}\in\mathrm{IR}^{\mathrm{N}\times\mathrm{d}}$ to denote a d-dimensional node embedding and $\Gamma(i)$ to refer to the neighborhood of node i.

NE methods

Embedding methods can be broadly categorized into four classes according to the strategy used for learning node similarities. In this taxonomy, we can distinguish methods based on the Skip-Gram model,[30] matrix factorization, neural architectures, and probabilistic approaches. Next, we describe the methods from each family included in our evaluation.

Methods based on Skip-Gram.

These methods leverage the Skip-Gram model to compute node representations such that the posterior probability of observing a neighboring node in a random walk is maximized.

DeepWalk[12] is the first method to use language modeling inspired techniques for NE. It uses random walks with fixed transition probabilities to measure node similarities, and embeddings are derived by using Skip-Gram approximated via hierarchical softmax.

Node2vec[21] is a generalization of DeepWalk that uses truncated random walks for node neighborhood 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[31] extracts structural information from graphs through node-pair similarities for a range of neighborhood 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[11] is designed to learn embeddings from heterogeneous networks. The authors extend the concept of random walks to account for nodes of different types and use a heterogeneous Skip-Gram model to learn the embeddings.

LINE[6] uses joint and conditional probability distributions to model the first- and second-order adjacencies between linked nodes in $\mathbf{G}$. Embeddings are then obtained by maximizing the postulated objective function.

VERSE[13] learns embeddings by training a single-layer neural network that minimizes the Kullback-Leibler (KL) divergence between node similarities in $\mathbf{G}$ and vector similarities in $\mathbf{X}$. Noise Contrastive Estimation is used for scalability.

Watch Your Step (WYS)[32] is an attention model learned on the power series of the transition matrix of **G**. The importance of different terms in the power series is parametrized by a probability distribution learned from data via backpropagation.

Methods based on matrix factorization. These approaches use representations of node similarities such as high-order proximities expressed as polynomials of **A**, the incidence matrix, Katz similarity, or the graph Laplacian. Node embeddings are then obtained by factorizing the selected matrix.

Graph factorization (GF)[33] uses regularized Gaussian matrix factorization to recover a matrix **Z** such that $\mathbf{ZZ}^T$ is close to **A** in terms of observed non-zeros.

GraRep[7] is based on factorization of different polynomials of the adjacency matrix **A** that identify relations between nodes in **G** at different resolutions.

HOPE[8] preserves high-order proximities in graphs and accounts for the asymmetric transitivity property of directed networks.

Laplacian Eigenmaps (LE)[5] constructs a weighted representation $\hat{\mathbf{A}}$ of **A** by leveraging first-order proximities on **G**. The Laplacian **L** computed from $\hat{\mathbf{A}}$ is then factorized to obtain node embeddings.

Locally linear embeddings (LLE)[34] operates under the assumption that nodes and their neighbors lie on locally linear patches of a high-dimensional manifold. The embedding of a node can, thus, be derived from the linear coefficients that better reconstruct the node from the embeddings of its neighbors.

FREDE[35] uses Personalized PageRank as a node similarity matrix and adapts the Frequent Directions sketching algorithm[36] to produce embeddings.

NetMF[37] factorizes the DeepWalk transition probability matrix via singular value decomposition to reach the global optimum. To address the method's limited scalability, the authors have proposed an approximation method in Ref..[38]

M-NMF[39] incorporates community structure information in the embedding learning process via modularized non-negative matrix factorization.

AROPE,[16] similarly to GraRep, proposes embeddings as found by the truncated singular value decomposition of polynomials of **A**. The authors describe a fast eigen-decomposition for these polynomials based on shifting or reweighing the decomposition of **A**.

Methods based on neural networks. Neural approaches return as embedding the latent representations learned by deep layers of the networks. In this category, we include models with a wide range or architectures, from shallow to deep models and from *classical* convolutional networks to graph convolutional approaches.

Structural deep network embedding (SDNE)[10] uses a deep neural network for learning embeddings that capture first- and second-order graph proximities.

PRUNE[22] relies on a deep Siamese neural network for learning node embeddings and can incorporate node ranking as additional information.

GAE/VGAE[14] is based on the variational autoencoder model and computes embeddings from latent variables. The inference model is parametrized by a two-layer graph convolutional network, and the generative model computes inner products of latent representations. VGAE is a non-probabilistic variant of GAE.

Graph isomorphism network (GIN)[40] is a neural model based on the Weisfeiler-Lehman test[41] that achieves maximum discriminative power among graph neural networks.

GatedGCN[42] belongs to the subclass of message passing graph convolutional approaches, where node representations are learned by performing local updates only. The model incorporates edge gating mechanisms into vanilla graph convolutional networks.

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

CNE[15] uses a Bayesian approach to generate embeddings that 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.

Baseline heuristics

In addition to the NE approaches, we evaluate a set of heuristics as baseline. Among these, we include heuristics that derive similarity scores—which can be interpreted as link probabilities—from the neighborhoods $\Gamma(i)$ and $\Gamma(j)$ of node-pairs $\{i,j\}$. Specifically, we evaluate Common Neighbors 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, $RA_{\{i,j\}} = \sum_{k \in \Gamma(i) \cap \Gamma(j)} 1/|\Gamma(k)|$; and Preferential Attachment, $PA_{\{i,j\}} = |\Gamma(i)| \cdot |\Gamma(j)|$.

We also include a method based on paths, that is, Katz similarity. This method is defined as

Table 1. Implementation used and hyperparameters tuned for each method

Methods	*Implementations*	*Hyperparameters tuned*
Heuristics	EvalNE	—
DeepWalk	Original, OpenNE	*num_walks* = *walk_len* = [5, 10, 20, 40, 80], *window_size* = [5, 10, 20]
Node2vec	Original, OpenNE	*num_walks* = *walk_len* = [5, 10, 20, 40, 80], *window_size* = [5, 10, 20], $p = q = [0.5, 1, 2]$
Struc2vec	Original	*num_walks* = *walk_len* = [5, 10, 20, 40, 80], *window_size* = [5, 10, 20]
Metapath2vec	Original	$\alpha = [0.01, 0.025]$, $negative = [5, 10]$
LINE	Original, OpenNE	$\rho = [0.01, 0.025]$, $negative\_ratio = [5, 10]$
VERSE	Original	*nsamples* = [3, 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	—
FREDE	Original	*log_transform* = [*log*, *add*, *max*]
NetMF	Original	*window_size* = [1, 10]
M-NMF	Original	*clusters* = [10, 20, 50]
AROPE	Original	*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	Original	$\lambda = [0.01, 0.05]$
GAE	Original	—
VGAE	Original	—
GIN	Other	*lr* = [0.01, 0.001], *step_size* = [50, 100]
GatedGCN	Original	*lr* = [0.01, 0.001], *step_size* = [50, 100]
CNE	Original	*lr* = [0.01, 0.05]

Except for AROPE, CNE, GAE, VGAE, GIN, GatedGCN, and the heuristics, the edge embedding operator is also tuned as a hyperparameter.

$Katz_{\{i,j\}} = \sum_{l=1}^{\infty} \beta^l \cdot |\rho(i,j|l)|$, where $\beta \in [0,1]$ is a damping factor and $|\rho(i,j|l)|$ represents the number of paths of length l from node i to node j.

Lastly, we generate a "heuristics embedding" by concatenating CN, JC, AA, RA, and PA as a five-dimensional node-pair embedding. Logistic regression (LR) is then used to obtain link predictions. We will refer to this method as *NE_heuristics*.

Implementations and hyperparameters

Regarding the code used in our evaluation, we primarily rely on reference implementations by the original authors of different manuscripts. In addition to these, we also use code from other sources such as the extensively used OpenNE[43] and Graph Embedding Methods (GEM)[19] libraries. Table 1 summarizes the implementations used and hyperparameters tuned.

In the table, all heuristics are summarized in a single *Heuristics* field and have no tunable hyperparameters (for Katz we fix $\beta = 0.001$ throughout our evaluation). Detailed hyperparameter descriptions are provided in the Supplementary Material. For all methods where this is relevant, the node-pair embedding operator is tuned as an additional hyperparameter (see the Experimental Setup section). The evaluation metrics used and hyperparameter tuning strategy are task dependent and, thus, are discussed later in the Link Prediction section, Network Reconstruction section, and Evaluation Setup section.

Lastly, some notable differences between method implementations and important method specifics are as follows: For LINE, the original implementation uses second-order proximity by default (LINE-2) whereas the OpenNE implementation uses a concatenation of first and second orders (LINE-1 + 2). For HOPE, the OpenNE implementation uses Common Neighbors as a similarity matrix, which is then factorized, whereas the GEM implementation uses Katz similarity.

For VERSE, the reference implementation provides an unsupervised version of the method with Personalized PageRank as a similarity metric. In NetMF, context window sizes of 1 and 10 correspond to evaluating NetMF-small and NetMF-large, respectively. Finally, for GIN and GatedGCN, end-to-end link

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

Dataset	*Category*	$\|\mathbf{V}\|$	$\|\mathbf{E}\|$	$2\|\mathbf{E}\|/\|\mathbf{V}\|$
StudentDB	Relational	395	3423	17.33
Facebook	Social	4039	88,234	43.69
BlogCatalog	Social	10,312	333,983	64.77
GR-QC	Collaboration	4158	26,844	6.45
AstroPH	Collaboration	18,772	396,160	22.00
PPI	Biological	3852	37,841	19.64
Wikipedia	Language	4777	92,295	38.64

prediction is performed by adding a new layer to each network trained to minimize the cross-entropy between the logits of this layer and the ground truth labels (existence or not of an edge).

Datasets

We conduct our experimental evaluation on seven 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[44] 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, that is, student-in-track, student-in-program, student-take-course, professor-teach-course, and course-in-room.

Facebook[45] and *BlogCatalog*[46] are online social networks, where nodes represent different users and edges indicate friendships.

GR-QC[45] and *AstroPh*[45] describe collaboration networks in the fields of General Relativity and Astrophysics. Nodes represent papers, and edges denote citations between them.

PPI[47] is a biological protein–protein interaction network and constitutes a subset of the *Homo Sapiens* PPI network.

Finally, *Wikipedia*[48] contains nodes representing words in Wikipedia pages and links denoting co-occurrences.

Experimental Setup

In this section, we give details on the link prediction and network reconstruction tasks, present the experimental setups used, and discuss the limitations to the scope of the evaluation and reproducibility of results.

Link prediction

As pointed out in the Introduction section, the objective in link prediction is to identify missing links in an incomplete graph $\mathbf{G}$. Thus, the first step for evaluation is to preprocess $\mathbf{G}$ and obtain sets of train and test node-pairs. For our main experiments, we use the standard approach of generating an incomplete training 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 $\{i, j\} \in \mathbf{E} \backslash \mathbf{E}_{train}$ 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 three approaches, namely *random*,[23] *spanning tree (ST)*[24], and *depth first tree (DFT)*. The first starts by randomly sampling a set of "preliminary" test edges ($\mathbf{E}_{test}$). The main connected component of the remaining graph is then computed and all edges in it are considered train edges ($\mathbf{E}_{train}$). Finally, the test set is refined such that $\{i, j\} \in \mathbf{E}_{test}$ only if $i \in \mathbf{V}_{train}$ and $j \in \mathbf{V}_{train}$. The ST and DFT approaches are very similar.

These construct a spanning tree of $\mathbf{G}$ and add randomly selected edges until the desired size of $\mathbf{E}_{train}$ is reached. All remaining edges are used for testing ($\mathbf{E}_{test}$). The main difference between these methods lies in the approach taken for computing the spanning tree. In the case of ST, this is done by selecting a tree uniformly at random from all possible ones, whereas in DFT a faster depth first search is used.

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 by 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 set $|\mathbf{D}_{train}| = |\mathbf{E}_{train}|$ and $|\mathbf{D}_{test}| = |\mathbf{E}_{test}|$.

For hyperparameter tuning, the train sets need to be further split into train and validation. We fix their proportions to 90% train and 10% validation in all our experiments, and the validation split is always performed by 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, GAE, VGAE, GIN, GatedGCN, and the 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, that is., $\mathbf{x}_{\{i,j\}} = \mathbf{x}_i \circ \mathbf{x}_j$. In our evaluation, we use the operators introduced in Ref.[21], 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.

Network reconstruction

Network reconstruction evaluates how accurately an embedding method captures the link structure of a network. In other words, it evaluates how well the adjacency matrix $\mathbf{A}$ of the input graph $\mathbf{G}$ can be recovered from the embeddings generated by an NE method. Similarly to the link prediction setting, we formulate network reconstruction as a binary classification problem where we expect connected node-pairs in $\mathbf{G}$ to be ranked higher than non-connected pairs.

First, for each method and network we perform hyperparameter tuning via a grid search. The best set of parameters is then used to compute node embeddings, and node-pair embeddings are derived by using the same operators $\circ$ as for link prediction. Finally, binary classification is performed and results are reported.

We note that for this task there are two fundamental differences with respect to the link prediction pipeline. First, since the objective is to determine how well a method captures the structure of a graph, training of the NE methods, training of the binary edge classifier, hyperparameter tuning, and model assessment are all performed on the complete input graph $\mathbf{G}$ rather than on train and validation graphs, that is, $\mathbf{G} = \mathbf{G}_{train}$ and $\mathbf{E} = \mathbf{E}_{train}$.

Second, this task requires us to provide a prediction for all $(N \cdot (N-1))$ entries in $\mathbf{A}$ to determine whether the probabilities of edges $\{i, j\} \in \mathbf{E}$ are higher than those of non-edges $\{i, j\} \in \mathbf{D}$. As this is unfeasible even for moderate-sized networks, we randomly sample 1% of node-pairs (0.1% for AstroPH and BlogCatalog) and approximate the method performance by these samples. Note that for the sparse graphs considered in our evaluation, this random sampling results in many more non-edges than edges to predict and, thus, AUC would not be an adequate performance metric. In this case, we report precision@*Np* for a range of *Np* values.

Evaluation setup

For link prediction, we use two experimental setups, called LP1 and LP2, with model hyperparameter tuning as the only difference. 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. We quantify method performance in terms of AUC and also report execution times. For visualization purposes, we use $-\log(1 - AUC)$, as it better reflects differences in performance for AUC values $\approxeq 1$.

Unless otherwise specified, we use embedding dimensionality $d = 128$, train fraction $f = 0.8$, ST as the edge sampling strategy, and LR (with fivefold cross-validation for tuning the regularization parameter) as a binary classifier. Finally, all methods are run for their predefined number of iterations.

Setup LP1 is used to study 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 remaining pipeline components.

For network reconstruction we use a single experimental setup, where method hyperparameters and node-pair embedding operators are tuned. We present method results for the best embedding dimension $d \in \{8, 32, 128\}$ and use LR with fivefold cross-validation (LRCV) as a binary classifier.

Reproducibility notes

To ensure the reproducibility of our results and foster further research in this area, we have based our experimental evaluation on the EvalNE framework. This open-source Python toolbox aims at simplifying the evaluation of NE methods for link prediction, network reconstruction, node classification, and visualization. The toolbox automates tasks such as hyperparameter tuning, selection of train and test edges, or non-edge sampling.

EvalNE also implements widely used node-pair embedding operators and can incorporate any classifier for prediction. Moreover, its design ensures that common errors, 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, for maximum reproducibility, configuration files describing our complete evaluation setups, and that can be used directly in EvalNE to replicate our results, are provided as Supplementary Material.

Experimental Results

In this section, we present the results of our empirical study. First, we focus on link prediction and discuss performance in the Link Prediction Performance section, the effect of hyperparameter tuning in the Hyperparameter Tuning section, embedding dimensionality in the Embedding Dimensionality section, train-test splits in the Train-Test Split Size section, edge sampling in the Edge Sampling section, and finally prediction pipelines in the Prediction Pipeline section.

Then, in the Network Reconstruction Performance section, we present the results for the network reconstruction task and in the Method Performance and Network Structure section, the relationship between method performance and network structure. All experiments were conducted on a machine equipped with two 12 Core Intel(R) Xeon(R) Gold processors and 256 GB of RAM.

Link prediction performance

We start in Table 3 by presenting the best AUC scores for each method on the link prediction task. These results were obtained by using setup LP1, where for each method we selected the embedding dimension $d \in \{8,\ 32,\ 128\}$ that resulted in the highest scores. GIN, GatedGCN, and CNE performed best for $d = 8$ whereas all the remaining methods performed best for $d = 128$ (see the Embedding Dimensionality section for a detailed discussion).

In the table, we group methods according to the taxonomy in the Methods section with the best performing method per group highlighted in bold and the overall best for each network on a gray background. The last two columns show averages over the seven evaluated networks of: (i) the AUC performance and (ii) the ranking of each method among all other approaches.

The results in Table 3 show the excellent performance, on most datasets, of the baseline heuristics and in particular of RA, Katz, and the NE_heuristics. The latter, together with GraRep, achieves the highest average AUC among all methods of 0.963. In addition, although NE_heuristics is not a top performer on any particular network, its AUC scores are consistently high across all networks and never more than 2% lower than the best scores in each case. This effect is also reflected by the method's excellent average AUC rank (second overall).

Among Skip-Gram based approaches, VERSE is a clear top performer. GraRep is the matrix factorization-based method with the highest scores and it presents a consistent performance across networks. Nonetheless, AROPE and HOPE also exhibit good performance. The neural architecture-based approaches present similar performances, with SDNE and GAE obtaining marginally higher scores.

Overall, according to their average AUCs and AUC ranks, the best performing methods are: VERSE, NE_heuristics, GraRep, and CNE. It is worth noting that for these top performers we tuned at most one hyperparameter (see Table 2) and, thus, their excellent performance cannot be ascribed to an exhaustive model selection process. Also noteworthy is the fact that CNE achieves state-of-the-art results, with an 8-dimensional embedding as compared with the 128 dimensions required by other methods.

In the absence of additional data, such as node or edge attributes, our experiments suggest that performance generally improves as the order of proximity between nodes captured by a model increases (i.e., as a node's neighborhood includes vertices that are further and further away). GraRep, the best performing NE method, captures relations up to order 8. HOPE and AROPE, also top performers, capture relations up to order 4 whereas the remaining first- and second-order methods exhibit lower performance.

Two interesting exceptions to this pattern are VERSE and CNE. In the case of VERSE, the second-order method employs a nonlinear transformation that is able to preserve more information from the original network. CNE, on the other hand, finds an embedding based on first-order information. Its excellent performance can be explained by the fact that it additionally models structural information (node degrees) in a prior, leaving more flexibility to the embedding.

Another important observation from the results in Table 3 is that method performance varies significantly between different implementations of the same NE methods (up to 11.7% for LINE on StundentDB). These differences are especially large for the implementations of LINE (3.3% difference, on average, over all networks), HOPE (2.4%) and Node2vec (2%). The main factors causing these differences are numerical instability, approximation of computations, and dependencies on different software versions with varying default parameters.

In our Supplementary Material, we also show fluctuations in method performance of up to 44.5% for GEM implementations due to package dependencies and up to 5% for metapath2vec due to multi-core execution.

Table 3. Area under the ROC curve scores and standard deviations over three experiment repetitions for setup LP1 where hyperparameters are tuned and $d = 128$ for all methods except GIN, GatedGCN, and CNE where $d = 8$

Methods	*StudentDB*	*Facebook*	*BlogCat.*	*GR-QC*	*AstroPH*	*PPI*	*Wikipedia*	*Average AUC*	*Average rank*
CN	0.630 ± 0.011	0.992 ± 0.001	0.948 ± 0.003	0.959 ± 0.000	0.990 ± 0.000	0.863 ± 0.002	0.900 ± 0.001	0.897 ± 0.210	20.79
JC	0.630 ± 0.011	0.990 ± 0.001	0.770 ± 0.001	0.959 ± 0.000	0.990 ± 0.002	0.839 ± 0.005	0.624 ± 0.001	0.829 ± 0.260	26.93
AA	0.630 ± 0.011	0.993 ± 0.001	0.952 ± 0.003	0.959 ± 0.000	0.991 ± 0.000	0.867 ± 0.002	0.919 ± 0.001	0.902 ± 0.211	16.14
PA	0.922 ± 0.008	0.842 ± 0.003	0.955 ± 0.002	0.839 ± 0.001	0.878 ± 0.001	0.905 ± 0.001	0.920 ± 0.006	0.894 ± 0.041	18.93
RA	0.630 ± 0.011	**0.994 ± 0.001**	**0.958 ± 0.003**	0.959 ± 0.000	0.991 ± 0.000	0.867 ± 0.002	**0.931 ± 0.001**	0.904 ± 0.212	12.93
Katz	0.737 ± 0.008	0.994 ± 0.000	0.954 ± 0.001	**0.988 ± 0.003**	**0.995 ± 0.000**	0.920 ± 0.000	0.914 ± 0.002	0.929 ± 0.086	9.79
NE_heuristics	**0.966 ± 0.004**	0.993 ± 0.000	0.956 ± 0.001	0.976 ± 0.000	0.993 ± 0.001	**0.927 ± 0.004**	0.929 ± 0.003	**0.963 ± 0.026**	**5.71**
DeepWalk	0.906 ± 0.005	0.990 ± 0.000	0.943 ± 0.001	0.986 ± 0.000	0.984 ± 0.000	0.905 ± 0.002	0.903 ± 0.001	0.945 ± 0.040	15.57
DeepWalk_opne	0.906 ± 0.010	0.991 ± 0.000	0.943 ± 0.001	0.985 ± 0.000	0.983 ± 0.000	0.905 ± 0.001	0.904 ± 0.002	0.945 ± 0.039	15.43
Node2vec	0.948 ± 0.009	0.994 ± 0.000	0.938 ± 0.006	0.985 ± 0.001	0.989 ± 0.001	0.840 ± 0.002	0.893 ± 0.003	0.941 ± 0.054	16.29
Node2vec_opne	0.897 ± 0.004	0.991 ± 0.001	0.929 ± 0.001	0.986 ± 0.000	0.992 ± 0.001	0.900 ± 0.001	0.901 ± 0.002	0.942 ± 0.043	16.00
Struc2vec	0.933 ± 0.010	0.833 ± 0.004	0.953 ± 0.002	0.842 ± 0.001	0.874 ± 0.001	0.904 ± 0.001	0.918 ± 0.005	0.894 ± 0.042	20.57
Metapath2vec	**0.981 ± 0.005**	0.942 ± 0.003	0.948 ± 0.003	0.803 ± 0.002	0.858 ± 0.000	0.880 ± 0.001	0.903 ± 0.006	0.902 ± 0.058	22.29
LINE	0.963 ± 0.004	0.993 ± 0.001	0.931 ± 0.002	0.984 ± 0.000	0.991 ± 0.000	0.877 ± 0.002	0.882 ± 0.002	0.946 ± 0.048	15.14
LINE_opne	0.850 ± 0.010	0.991 ± 0.000	0.931 ± 0.002	0.933 ± 0.001	0.963 ± 0.000	0.895 ± 0.003	0.894 ± 0.001	0.923 ± 0.045	22.86
VERSE	0.935 ± 0.010	**0.994 ± 0.001**	**0.956 ± 0.002**	**0.990 ± 0.000**	**0.996 ± 0.002**	**0.919 ± 0.002**	**0.918 ± 0.002**	**0.959 ± 0.033**	**5.14**
WYS	0.819 ± 0.016	0.940 ± 0.003	0.915 ± 0.005	0.833 ± 0.004	0.855 ± 0.002	0.853 ± 0.010	0.864 ± 0.008	0.868 ± 0.042	30.00
GF_opne	0.868 ± 0.007	0.983 ± 0.000	0.897 ± 0.004	0.933 ± 0.001	0.947 ± 0.001	0.837 ± 0.003	0.834 ± 0.005	0.900 ± 0.054	28.14
GraRep_opne	0.969 ± 0.003	**0.993 ± 0.000**	**0.962 ± 0.001**	**0.984 ± 0.000**	**0.990 ± 0.001**	**0.921 ± 0.001**	**0.922 ± 0.002**	**0.963 ± 0.029**	**6.14**
HOPE_gem	**0.989 ± 0.001**	0.990 ± 0.000	0.955 ± 0.002	0.952 ± 0.001	0.950 ± 0.000	0.909 ± 0.001	0.919 ± 0.002	0.952 ± 0.029	13.00
HOPE_opne	0.914 ± 0.002	0.989 ± 0.000	0.944 ± 0.005	0.920 ± 0.000	0.947 ± 0.000	0.872 ± 0.001	0.915. ± 0.005	0.929 ± 0.034	21.43
LE_gem	0.906 ± 0.010	0.992 ± 0.000	0.800 ± 0.005	0.975 ± 0.000	0.934 ± 0.003	0.760 ± 0.005	0.767 ± 0.003	0.876 ± 0.097	24.64
LE_opne	0.906 ± 0.011	0.992 ± 0.000	0.802 ± 0.003	0.977 ± 0.002	0.932 ± 0.005	0.764 ± 0.006	0.771 ± 0.001	0.878 ± 0.092	24.14
LLE_gem	0.889 ± 0.008	0.990 ± 0.000	0.704 ± 0.008	0.970 ± 0.006	0.894 ± 0.002	0.726 ± 0.005	0.741 ± 0.004	0.845 ± 0.114	28.43
FREDE	0.815 ± 0.005	0.747 ± 0.000	0.627 ± 0.001	0.604 ± 0.001	0.548 ± 0.002	0.621 ± 0.002	0.910 ± 0.002	0.696 ± 0.125	32.71
NetMF	0.942 ± 0.016	0.990 ± 0.000	0.952 ± 0.001	0.970 ± 0.003	0.968 ± 0.000	0.890 ± 0.001	0.914 ± 0.001	0.947 ± 0.033	17.00
M-NMF	0.944 ± 0.009	0.992 ± 0.000	0.936 ± 0.008	0.983 ± 0.000	0.983 ± 0.001	0.878 ± 0.001	0.913 ± 0.002	0.947 ± 0.040	15.79
AROPE	0.982 ± 0.002	0.991 ± 0.001	0.955 ± 0.001	0.968 ± 0.000	0.967 ± 0.001	0.910 ± 0.002	0.918 ± 0.001	0.956 ± 0.029	12.50
SDNE_gem	**0.987 ± 0.004**	0.979 ± 0.002	0.952 ± 0.002	0.945 ± 0.001	0.971 ± 0.000	**0.909 ± 0.001**	0.918 ± 0.002	**0.952 ± 0.028**	15.14
SDNE_opne	0.985 ± 0.002	0.987 ± 0.000	0.953 ± 0.005	0.957 ± 0.002	0.969 ± 0.000	0.898 ± 0.001	0.917 ± 0.007	**0.952 ± 0.032**	15.71
PRUNE	0.901 ± 0.010	0.838 ± 0.002	**0.956 ± 0.003**	0.836 ± 0.001	0.874 ± 0.000	0.903 ± 0.001	**0.920 ± 0.003**	0.890 ± 0.042	20.14
GAE	0.955 ± 0.012	**0.993 ± 0.001**	0.943 ± 0.005	0.977 ± 0.003	**0.991 ± 0.001**	0.856 ± 0.002	0.913 ± 0.002	0.947 ± 0.047	**14.36**
VGAE	0.952 ± 0.007	0.993 ± 0.000	0.935 ± 0.003	**0.978 ± 0.002**	0.989 ± 0.000	0.881 ± 0.003	0.898 ± 0.007	0.946 ± 0.042	15.86
GIN	0.820 ± 0.055	0.855 ± 0.011	0.725 ± 0.012	0.748 ± 0.013	0.826 ± 0.005	0.843 ± 0.005	0.897 ± 0.002	0.816 ± 0.059	31.00
GatedGCN	0.713 ± 0.045	0.823 ± 0.040	0.756 ± 0.014	0.776 ± 0.005	0.812 ± 0.008	0.852 ± 0.008	0.871 ± 0.036	0.800 ± 0.054	32.43
CNE	**0.946 ± 0.009**	**0.994 ± 0.000**	**0.967 ± 0.001**	**0.980 ± 0.000**	**0.976 ± 0.001**	**0.928 ± 0.001**	**0.922 ± 0.000**	**0.959 ± 0.026**	**6.93**

Note that 0.000 in the table means <0.0005. The best method within each type of approach is highlighted in bold, and the overall best for each column is depicted in a gray background.

Also interesting is the difference in performance between Node2vec and DeepWalk. In this case, one would expect the former (which is a generalization of the latter) to perform consistently better. This would, indeed, be the case if both methods would approximate Skip-Gram via negative sampling.[21] In our experiments, however, the Skip-Gram is approximated via hierarchical softmax for DeepWalk (as proposed by the original authors) and via negative sampling for Node2vec (also as proposed by the authors), which explains the observed differences.

Further experiments comparing the two Skip-Gram approximations for both methods reveal significant fluctuations in AUC of up to 0.02, with a standard deviation of 0.02. Moreover, we find that no approximation provides consistently better results than the other across all datasets. Ultimately, this experiment reveals the importance of accounting for the approximation used when comparing methods based on Skip-Gram.

Hyperparameter tuning

Figure 1 presents a heatmap of the increment in link prediction accuracy obtained through hyperparameter tuning, that is, the difference in AUC between setups LP1 and LP2. Only methods with tuned hyperparameters are shown in this figure. The results reveal limited improvements in most cases. For $\sim$7% of the network-method combinations, however, we observe increments in AUC greater than 5%. WYS, SDNE and CNE benefit the most from hyperparameter tuning. Popular random walk methods such as DeepWalk and Node2vec, for which parameter tuning is tedious, show minimal gains in AUC.

In Figure 2, we present the node-pair embedding operator selected by using hyperparameter tuning for each method and network on experimental setup LP1. The results indicate that Hadamard is the most frequently selected (in 71/161 cases) and that some methods prefer specific operators. This is: Hadamard in the case of VERSE, HOPE and SDNE, 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 AUC between the best and the worst performing operator of 0.141, with a standard deviation of 0.092. 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 seven evaluated networks for experimental setups LP1 and LP2 is presented in Figure 3. As expected, runtimes for LP1 are larger than those of LP2 and the differences are especially significant for the methods with more tuned hyperparameters, for example, Node2vec, Struc2vec, and SDNE (as this implies computing an embedding of the validation graph for each combination of hyperparameters).

Nevertheless, the highest increase in runtime can be found for GraRep, for which a single hyperparameter was tuned (the *k-step*). This indicates that computing high-order proximities in GraRep is a slow process and, as shown in Figure 1, does not result in a large improvement in performance for *k-step* >4.

We also observe from Figure 3 that naive sequential implementations of the heuristics are still faster than heavily optimized and parallelized NE methods, with only AROPE presenting comparable runtimes. Finally,

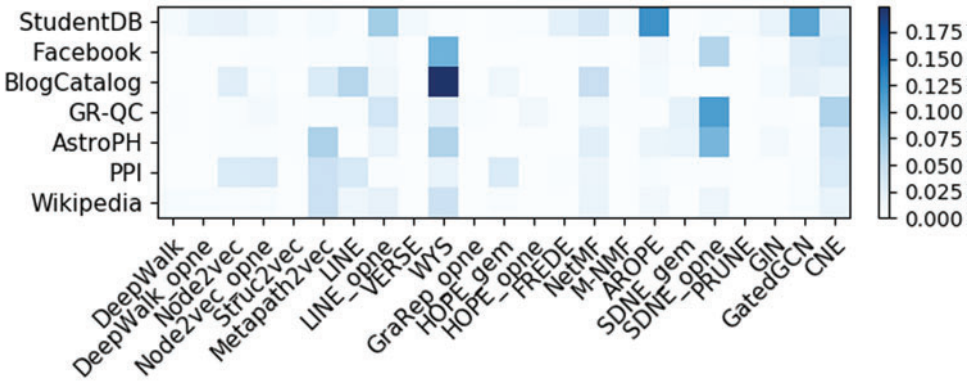

FIG. 1. Improvement in AUC of tuning model hyperparameters. Only methods with tuned parameters are shown. AUC, area under the ROC curve. Color images are available online.

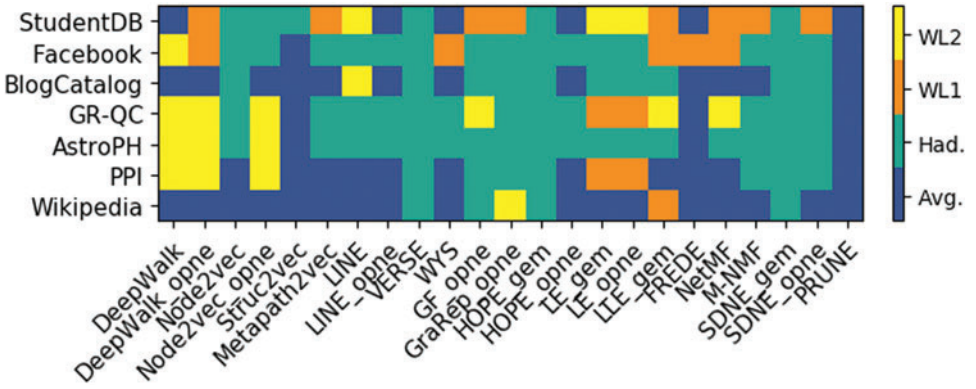

FIG. 2. Best performing node-pair embedding operators on setup LP1. Only methods for which the node-pair operator was tuned are shown. Color images are available online.

with no hyperparameter tuning, similar patterns can be observed within each method family with factorization methods being the fastest.

Embedding dimensionality

We have evaluated the effect of embedding dimensionality on method performance by modifying setup LP1 and computing the AUCs of all methods for $d \in \{8, 32, 128\}$. In Figure 4, we summarize the average AUC and standard deviation for each method and embedding dimension over all evaluated networks. The heuristics do not depend on d but are shown for reference. These results show that most methods significantly improve in performance as d increases. The opposite effect can be seen for GIN, GatedGCN, and CNE, which may indicate that these methods are underfitting.

Train-Test split size

We further used experimental setup LP1 to analyze the effect on method performance of the train fraction f. These results are summarized as three heatmaps in Figure 5 showing performance as $-\log(1-AUC))$ for $f \in \{0.2, 0.5, 0.8\}$. An interesting observation here is that for $f > 0.5$ most methods capture well the network structures whereas this is not the case

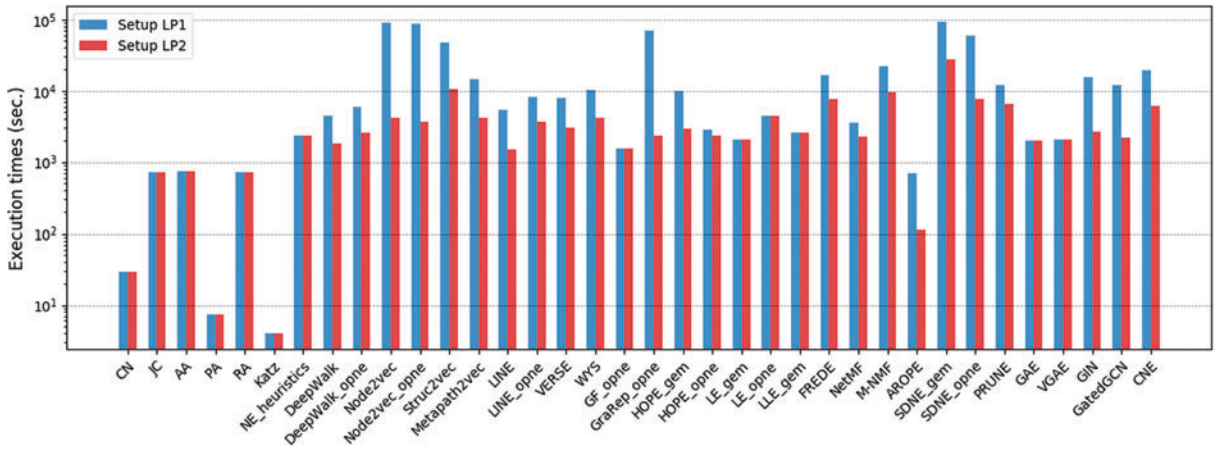

FIG. 3. Execution times in seconds of setups LP1 and LP2. Color images are available online.

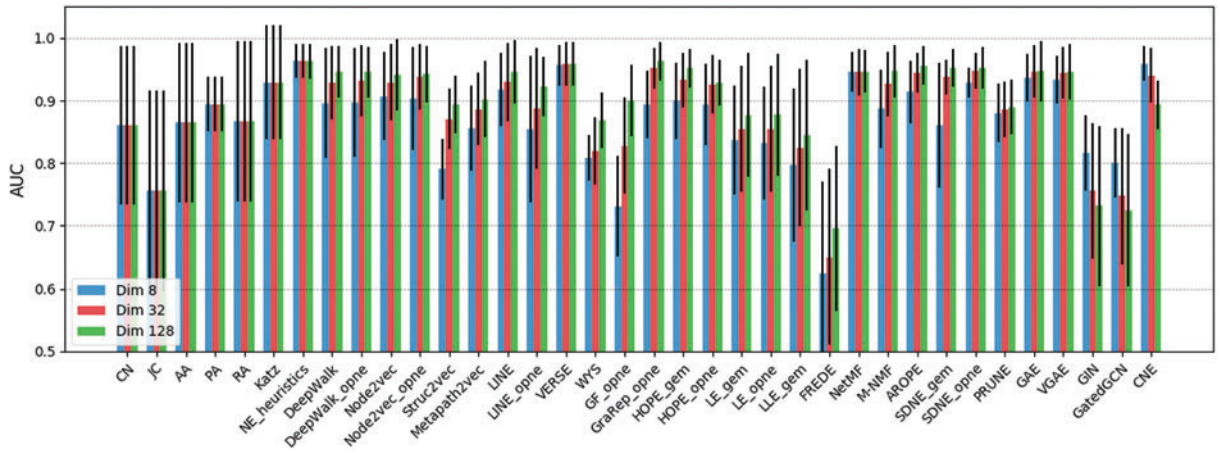

FIG. 4. Method performance (AUC) on setup LP1 for $d \in \{8, 32, 128\}$. Color images are available online.

for $f < 0.5$. This is reflected by an average increase in AUC over all methods of 6.7% between $f = 0.2$ and $f = 0.5$ whereas only a 1.9% difference can be observed between $f = 0.5$ and $f = 0.8$.

From this experiment, we also observe that probabilistic approaches present the highest robustness to varying train set sizes followed by Skip-Gram, factorization, neural approaches, and, finally, the heuristics. VERSE, GraRep, CNE, and NetMF are the methods that best capture the network structure when only 20% of the edges are used for training. Lastly, regarding method runtimes, these increase by $\sim$60% from evaluations with $f = 0.2$ to $f = 0.8$. Some notable exceptions are Metapath2vec, Struc2vec, and SDNE_gem, for which the runtimes increase by 84.9%, 83.7%, and 81.1%, respectively.

Edge sampling

We also conducted an experiment to compare the three strategies introduced in the Link Prediction section, namely random, ST, and DFT, for splitting $\mathbf{E}$ into $\mathbf{E}_{train}$ and $\mathbf{E}_{test}$ and determine their impact on method performance. We used experimental setup LP2, where no method hyperparameters, apart from the node-pair embedding operator, were tuned. Our results show minimal differences in AUC between the three strategies. More precisely, the average AUC and standard deviation over all methods and datasets for random edge sampling is 0.894 ± 0.114; for the ST strategy, it is 0.897 ± 0.114; and for DFT, it is 0.894 ± 0.109.

For large networks, edge sampling can become a bottleneck and, thus, sampling runtimes were also worth investigating. Our results, depicted in Figure 6, show that the random strategy is the slowest followed by ST and finally, DFT, which is up to one order of magnitude faster on specific networks. The slower runtimes of the random strategy are mainly due to set intersections to obtain the correct $\mathbf{E}_{train}$ and $\mathbf{E}_{test}$.

Our experiments also reveal that the random edge split strategy does not preserve all nodes from the input graph $\mathbf{G}$ in the training 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 networks with lower average degrees, such as 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 also compared the ST strategy with a split based on edge timestamps to determine whether the excellent performance of the heuristics is a result of the edge sampling strategy used. For this, we selected two temporal networks CollegeMsg and MathOverflow from the SNAP repository.[45] The first network encodes messages sent between students at UC Irvine, whereas the second captures interactions between users of the Math Overflow social platform.

As both networks present temporal information on the edges, we can compare a timestamp-based edge sampling with the random ST strategy. In this experiment, we used setup LP2 with a fixed train fraction of 0.8. In the timestamp sampling, we selected the 20% most recent edges such that, when these were removed, the remaining training graph was still connected.

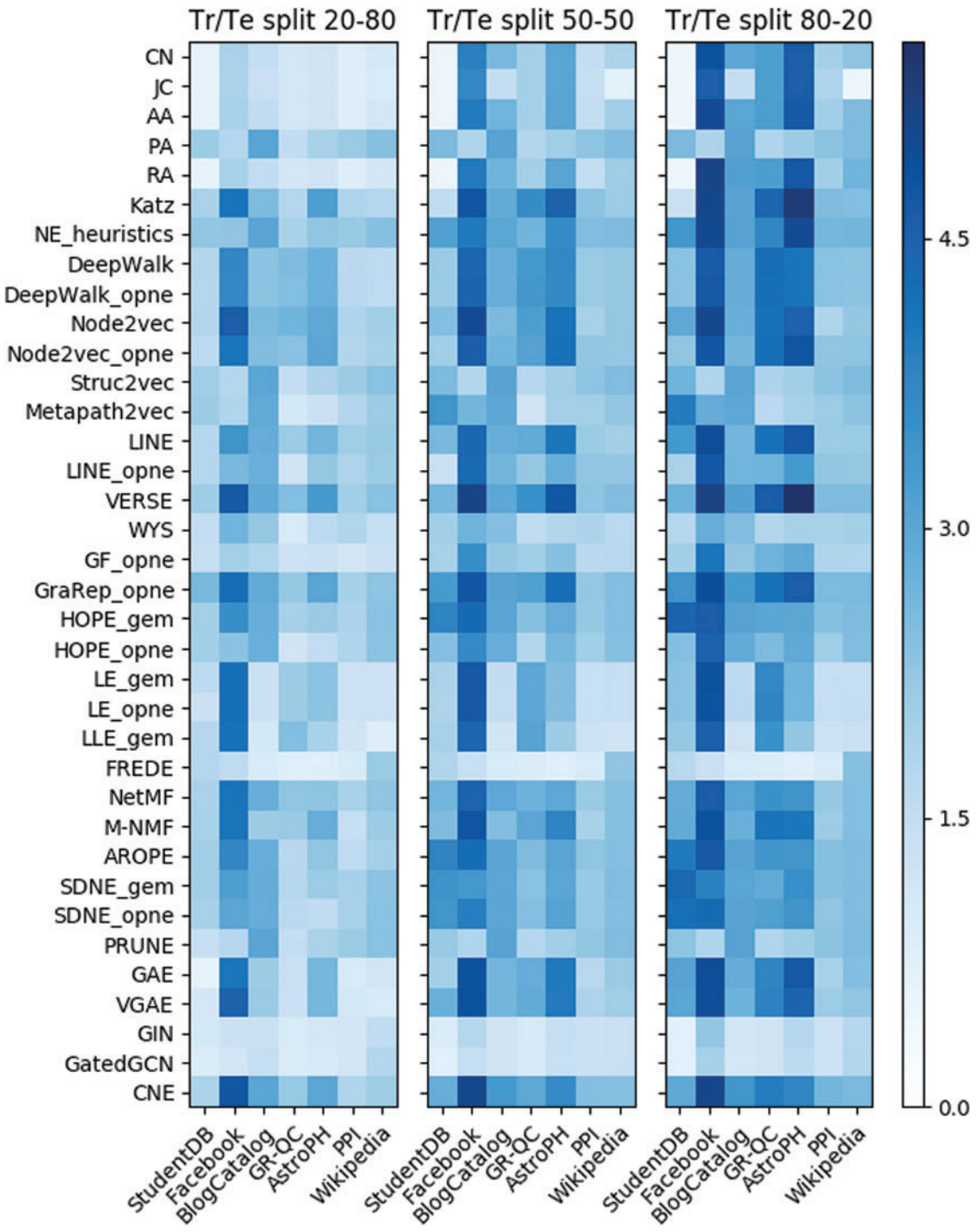

FIG. 5. Method performance as $-\log(1 - AUC)$ on setup LP1 for train-test edge splits 20-80, 50-50, and 80-20. A darker color indicates better performance. Color images are available online.

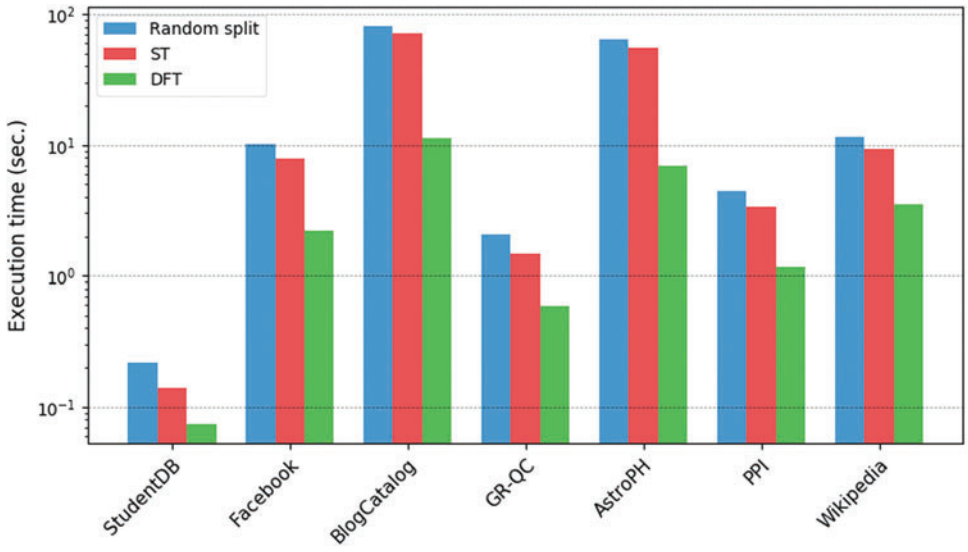

FIG. 6. Execution times of different train-test split algorithms. Color images are available online.

Our results are summarized in Figure 7. We observe that the ST strategy significantly boosts the performance of both heuristics and NE methods when compared with the timestamp sampling. We also observe that for ST sampling the performances of heuristics and NE methods are very similar, whereas for the timestamp sampling, the latter perform slightly better. Ultimately, this experiment indicates that the sampling strategy induces properties on the training graphs that some approaches might exploit better than others.

Finally, we also monitored the variation in performance when embeddings are learned from different initial training graphs $\mathbf{G}_{train}$. For each of the three edge split strategies, we performed three different

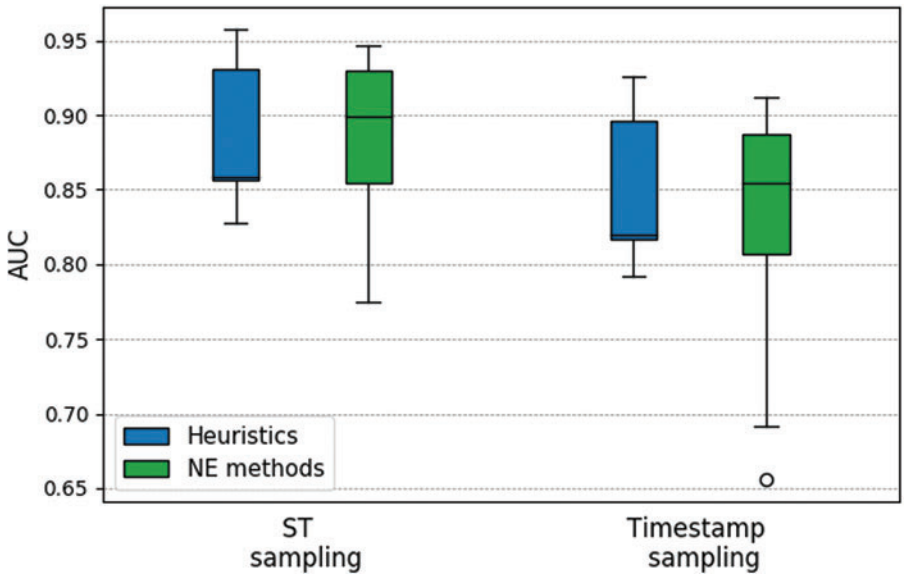

FIG. 7. Performance of the heuristics and NE methods compared for the spanning tree and timestamp-based sampling strategies. NE, network embedding. Color images are available online.

evaluations with varying initial random seeds and fixed $f = 0.8$. Our results show no significant differences, with a maximum standard deviation observed across all datasets, methods, and strategies of only 0.003. The largest variations are observed for the smallest network, StudentDB.

This experiment, thus, indicates that averaging link prediction results over several sets $\mathbf{E}_{train}$ and $\mathbf{E}_{test}$ generated by using different random seeds—to obtain unbiased estimates of method performance—becomes less necessary as the train network sizes ($\mathbf{G}_{train}$) exceed a few thousand nodes.

Prediction pipeline

For all end-to-end methods considered in our experiments (i.e., GAE, VGAE, GIN, GatedGCN, AROPE, and CNE), we compared their original pipelines with the same pipeline used for other methods, that is, computation of node-pair embeddings followed by binary classification with LR. The results are summarized in Figure 8 and show that for most methods and networks, computing node-pair embeddings and performing LR can result in a higher overall performance.

We have conducted additional experiments using setup LP2 comparing all methods with different binary classifiers: LR, LRCV, and decision trees (DT). Our experiments do not reveal any significant differences in performance for most methods, with the exception of LE and LLE, which are significantly boosted by the DT classifier. For LE we observe an increment in average AUC over all networks of 6.5%, from 0.87 using LRCV to 0.93 with DT. For LLE, the increment is of 7.8%, from an AUC of 0.84 with LRCV to 0.91 with DT.

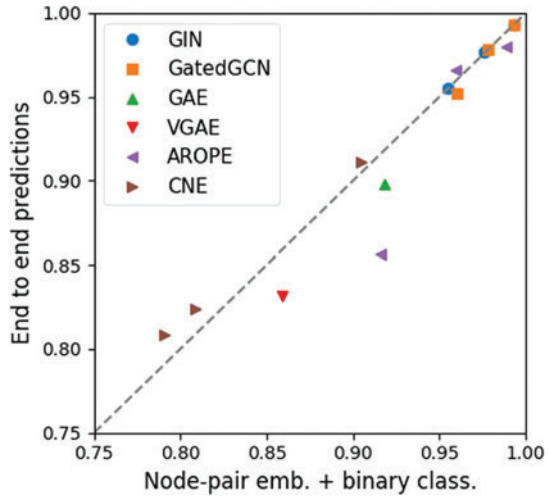

FIG. 8. End-to-end prediction pipelines compared with node-pair embeddings and binary classification. Color images are available online.

The LR binary classifier can also be used to gain insights on how NE methods distribute information over the available embedding dimensions. In Figure 9, we show, for each method, the sorted log odds of the LR coefficients. These are presented for a specific network (GR-QC) and train-test split on setup LP2 with $d = 128$, but they remain consistent across different settings. We observe that most methods present similar patterns, with a few dimensions encoding most of the information and the remaining ones capturing progressively less.

Notable exceptions are some of the top performers, that is, CNE, VERSE, and AROPE, and also other methods such as GF, M-NMF, GAE, or PRUNE, which spread the information more evenly. A second interesting finding is that the OpenNE implementation of LINE evaluated appears to encode no information beyond dimension 64. This could indicate an implementation error when calculating the second-order proximities (recall that this implementation uses concatenated first and second orders). We do not observe this pattern in the original implementation of LINE (LINE-2), which takes full advantage of all 128 dimensions.

Network reconstruction performance

For the network reconstruction task, we first summarize method performance in terms of precision@Np, for $Np = 100$, in Table 4. We chose this threshold, as it best highlights the differences between evaluated methods. We observe from the table that Katz is the top performer, closely followed by VGAE, AROPE, HOPE_opne, and VERSE.

In Figure 10, we plot the precision@Np for a range of Np values for a subset of methods and networks. The methods shown are the top performers from Table 4, that is, Katz, VERSE, HOPE_opne, AROPE, and VGAE. We observe that Katz is the only method that maintains a very high precision on all networks as the threshold increases. AROPE and HOPE perform very similarly on all networks and maintain overall good precision scores. For VERSE and VGAE, we observe large variations between different networks and overall lower performances.

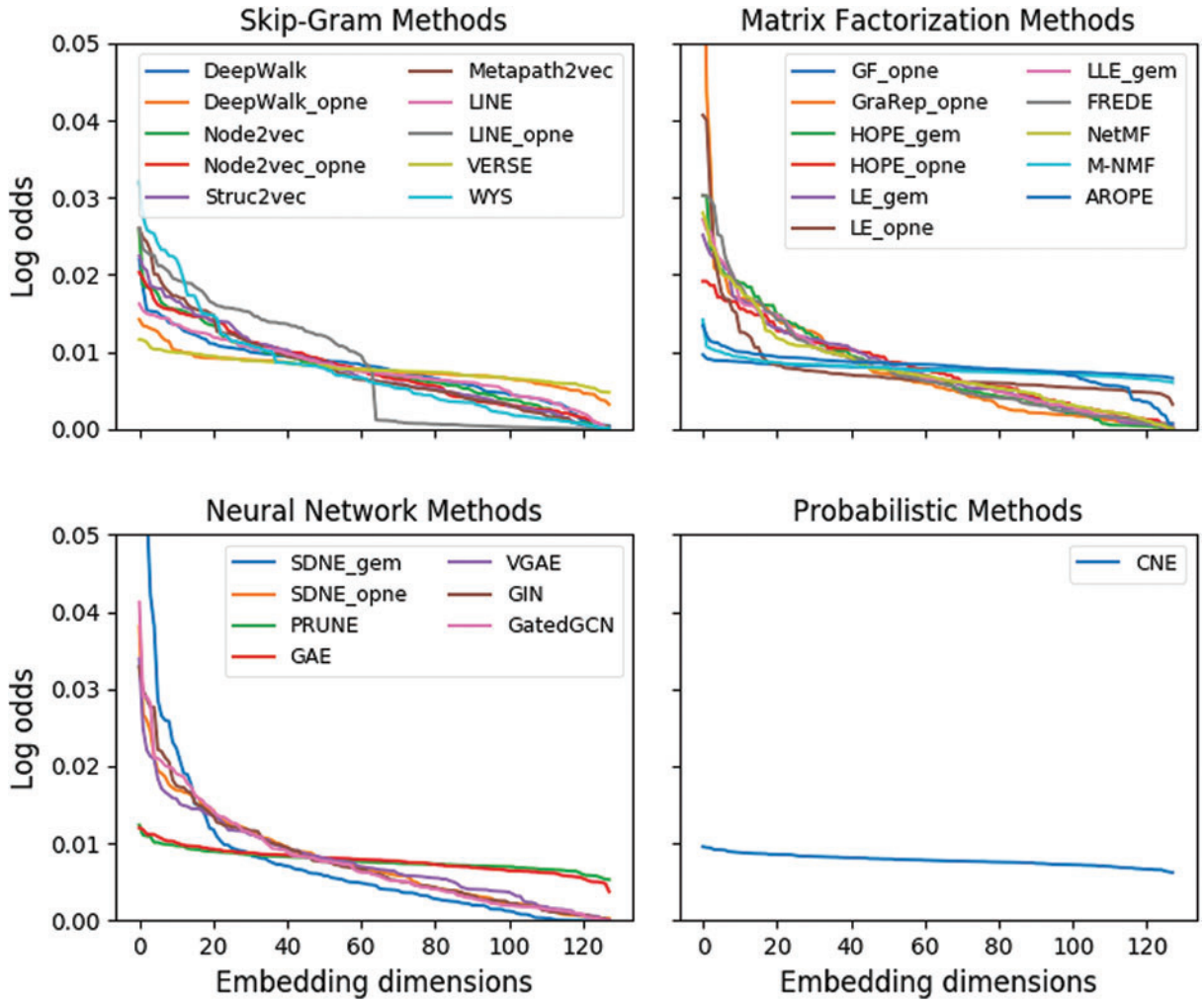

FIG. 9. Log odds for each coefficient of the LR binary classifier. Results are presented for the GR-QC dataset, but identical patterns are observed for other networks and experimental setups. LR, logistic regression. Color images are available online.

Method performance and network structure

The results in Tables 3 and 4 also show that most NE methods and heuristics perform well on Facebook, AstroPH, and GR-QC (a graphical representation of the results in Table 3 is provided in the rightmost heatmap in Fig. 5[‡]). These networks share similar topologies with large diameters and high clustering coefficients, which indicate the presence of community structures within the networks.

These observations are consistent with the data represented by the networks, that is, groups of friends in Facebook and citations between papers in different research sub-fields for AstroPH and GR-QC. The community structures with high intra-community edge densities and low inter-community densities could explain the higher link prediction performance of most methods.

On the other hand, the lowest method performance can be observed for StudentDB, BlogCatalog, Wikipedia, and PPI.

For StudentDB, the k-partite structure of the network poses a challenge to both heuristics and NE methods; first, due to the fact that, in this case, similar node neighborhoods do not necessarily imply that two nodes should be connected. For instance, two students following the same courses will have identical node

[‡]With the exception of GIN, GatedGCN and CNE are presented in Figure 4 for $d = 128$ and in Table 3 with $d = 8$.

Table 4. Network reconstruction performance shown as precision@100 where method hyperparameters are tuned and $d = 128$ for all methods except CNE, GIN, and GatedGCN where $d = 8$

Methods	*StudentDB*	*Facebook*	*BlogCatalog*	*GR-QC*	*AstroPH*	*PPI*	*Wikipedia*	*Average prec@Np*	*Average rank*
CN	0.00	0.95	0.52	0.81	0.91	0.41	0.73	0.61	18.50
JC	0.00	0.88	0.03	0.69	0.80	0.04	0.03	0.35	27.29
AA	0.00	0.95	0.51	0.93	0.93	0.44	0.77	0.64	17.00
PA	0.20	0.41	0.43	0.22	0.07	0.30	0.87	0.35	28.36
RA	0.00	0.97	0.59	0.95	0.98	0.40	0.90	0.68	13.64
Katz	**0.30**	**1**	**1**	**1**	**1**	**1**	**1**	**0.90**	**3.43**
NE_heuristics	0.22	0.95	0.50	0.94	0.97	0.41	0.83	0.68	15.71
DeepWalk	0.30	0.89	0.47	0.86	0.86	0.68	0.90	0.70	14.21
DeepWalk_opne	0.30	0.89	0.45	0.89	0.84	0.59	0.87	0.69	16.00
Node2vec	0.30	0.94	0.62	0.98	0.86	0.58	0.69	0.71	13.71
Node2vec_opne	0.30	0.86	0.56	0.89	0.87	0.65	0.72	0.69	15.43
Struc2vec	0.23	0.29	0.50	0.38	0.19	0.30	**0.92**	0.40	25.43
Metapath2vec	0.30	0.31	0.39	0.04	0.11	0.10	0.68	0.27	29.36
LINE	0.30	0.87	0.24	0.96	0.94	0.43	0.76	0.64	16.36
LINE_opne	0.29	0.87	0.53	0.67	0.79	0.63	0.81	0.65	18.00
VERSE	0.30	**0.97**	**0.71**	**0.99**	**0.97**	**0.87**	0.80	**0.80**	**9.00**
WYS	0.30	0.40	0.26	0.36	0.29	0.17	0.79	0.36	27.07
GF_opne	0.29	0.87	0.33	**0.95**	0.88	0.72	0.61	0.66	18.43
GraRep_opne	0.23	0.84	0.57	0.72	0.73	0.51	0.70	0.61	20.21
HOPE_gem	0.30	0.98	0.48	0.65	0.70	0.64	**1**	0.67	13.21
HOPE_opne	0.30	0.99	0.88	0.78	0.89	0.96	**1**	**0.82**	8.14
LE_gem	0.00	0.93	0.45	0.55	0.20	0.30	0.45	0.48	27.00
LE_opne	0.30	0.93	0.45	0.55	0.20	0.31	0.45	0.45	13.79
LLE_gem	0.30	0.73	0.36	0.62	0.44	0.23	0.88	0.50	23.64
FREDE	0.26	0.12	0.06	0.00	0.02	0.05	0.92	0.20	30.50
NetMF	0.30	0.98	0.92	0.66	0.58	0.99	**1**	0.77	10.00
M-NMF	0.30	0.83	0.32	0.61	0.44	0.44	0.75	0.52	23.00
AROPE	0.30	**1**	**1**	0.50	**0.94**	**1**	**1**	**0.82**	**7.71**
SDNE_gem	0.29	0.97	0.53	0.62	**0.68**	0.36	0.91	0.62	17.29
SDNE_opne	0.29	0.96	0.46	0.84	0.58	0.33	0.91	0.62	18.14
PRUNE	0.29	0.56	0.46	0.23	0.06	0.28	0.77	0.37	27.79
GAE	**0.60**	0.80	**0.90**	**1**	0.60	0.70	0.90	0.78	11.07
VGAE	**0.60**	**1**	**0.90**	**1**	0.60	**0.90**	**1**	**0.85**	**5.64**
GIN	**0.60**	0.80	**0.90**	0.10	0.20	0.50	0.30	0.48	21.29
GatedGCN	0.40	0.90	**0.90**	0.10	0.20	0.60	0.20	0.47	19.50
CNE	**0.29**	**0.83**	**0.27**	**0.91**	**0.33**	**0.24**	**0.63**	**0.50**	**25.64**

The best method/methods within each type of approach are highlighted in bold, and the overall best for each column are depicted in a gray background.

The average precision@100 over all networks and average rank for each method are also presented.

AUC, area under the ROC curve.

neighborhoods; however, they should never be connected, as links between nodes of the same types do not exist in the data. Second, NE methods must at the same time represent similarity between nodes of different types and dissimilarity for those of the same type. For example, students following the same course must be embedded close to the said course yet far from each other. Methods that are able to capture high-order proximities between nodes (e.g., GraRep, HOPE, AROPE) or those that learn node roles (e.g., Metapath2vec) maintain high AUC scores in this case.

The BlogCatalog and Wikipedia networks present very similar structures with small diameters, high clustering coefficients, and large average degrees. This can result in cluttered representations, where all nodes are close by, making downstream classification tasks harder. Lastly, the PPI network presents the opposite scenario with a large diameter and a small clustering coefficient. This can result in embeddings, where most nodes lie equally far from each other and, thus, overall lower AUCs.

We further analyzed the reasons behind the large variations in method performance on different networks by computing the average true positive rates (TPR) and false positive rates (FPR) over all methods for each network. As a threshold, we used the value in the AUC that maximized the accuracy of predictions. We omit showing the full statistics for brevity.

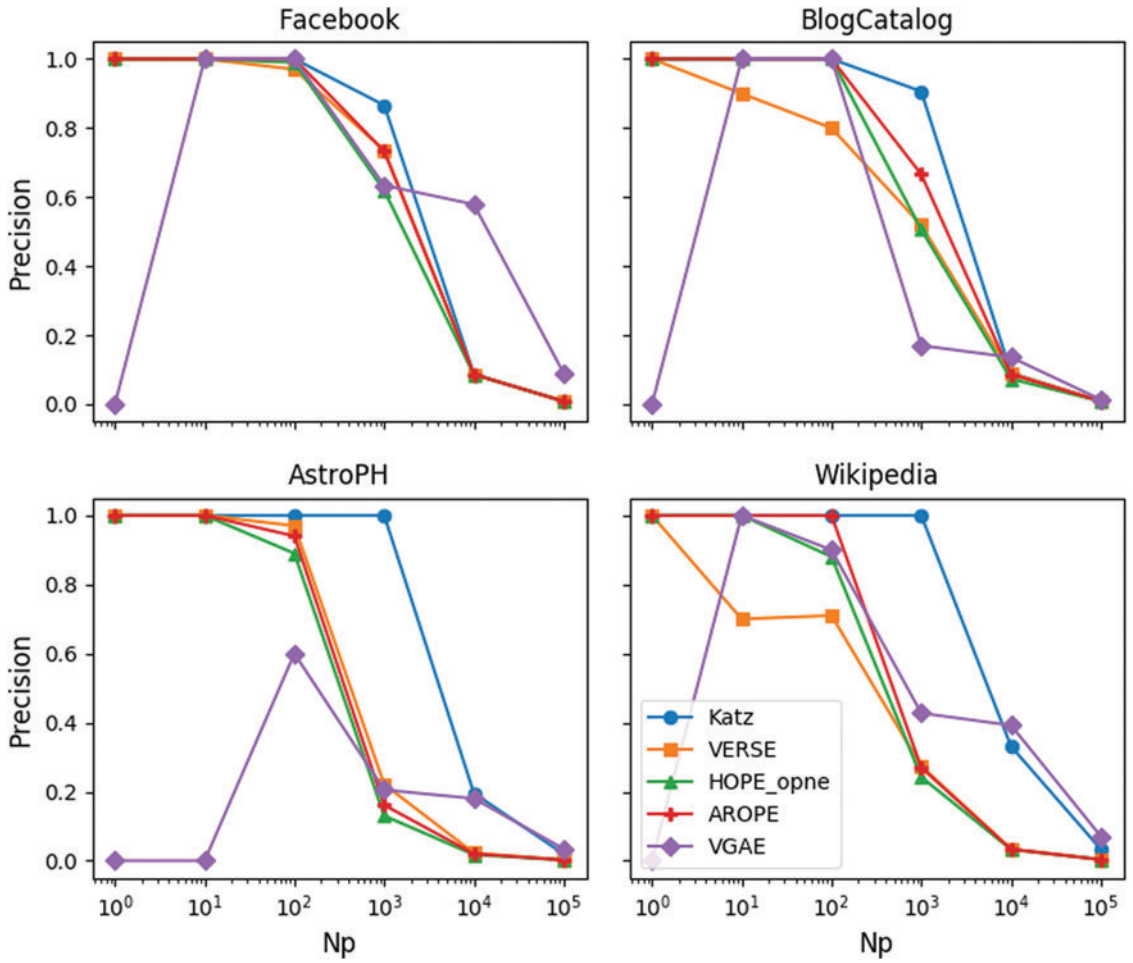

FIG. 10. Network reconstruction performance in terms of precision@*Np* shown only for the top-performing approaches. Color images are available online.

We observe that although most TPRs at this threshold vary around 0.8, the FPRs for StudentDB, BlogCatalog, PPI, and Wikipedia are three orders of magnitude higher than those of other networks. This indicates that, indeed, a clear community structure simplifies the link prediction task whereas cluttered graphs negatively impact the prediction of non-edges.

Limits to the scope of this evaluation

Constraints on overall computation time (the results in this section required ∼65 days to compute) inevitably imply boundaries to the scope of this evaluation: It is limited to popular mid-sized undirected and unweighted networks. It evaluates a representative but non-exhaustive set of NE methods. It only evaluates methods purely exploiting the network structure (and not node/edge meta-data), such that some methods are not used to their full potential (but note that the same holds for the heuristic baselines).

Further, the selected heuristics are very simple; other more powerful similarity metrics between network nodes have been proposed, for example, in Refs.[49,50] These are not included, as a thorough analysis of link prediction heuristic performance is not the main goal of our work. For a detailed overview and experimental evaluation of different heuristics, we refer the reader to Ref.[51]

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 AUC scores are consistently higher,

the same main conclusion holds: node2vec outperforms its competitors on the three networks originally evaluated. 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 regarding network reconstruction performance and the choice of the node-pair operator for VERSE.

Conclusions

In this article, we have conducted an extensive empirical study on NE methods. Our results show that, despite the surge of interest in the field in recent years, thin progress has been made. Most of the NE methods evaluated perform very similarly, and in most cases not significantly better than simple heuristics for both tasks studied. On average, the proposed NE_heuristics baseline outperforms all other methods for the link prediction task. Nevertheless, some embedding-based methods such as GraRep, VERSE, and CNE do show promising results.

We have also shown that clear community structures in the graphs, high embedding dimensionalities, and capturing high-order proximities between nodes all impact method performance positively. 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 a model hyperparameter and that a single train-test split provides a good estimation of method accuracy, resulting in higher evaluation efficiency.

Methods such as Katz, VERSE, HOPE, AROPE, and VGAE present exceptionally good results for network reconstruction. Finally, we hope this study and evaluation toolboxes such as EvalNE serve as initial steps toward 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.

Authors' Contributions

A.M., J.L., and T.D.B. conceived and planned the experiments. Alexandru Mara designed and implemented the computational framework used in the experiments and carried out the evaluations. J.L. and T.D.B. verified the evaluations and contributed toward the interpretation of results. All authors provided critical feedback that helped shape the research and the article.

Author Disclosure Statement

No competing financial interests exist.

Funding Information

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, and from the FWO (project no. G091017N, G0F9816N, 3G042220).

Supplementary Material

The supplementary material includes: (i) the EvalNE configuration files to replicate our experiments, (ii) a PDF file detailing the hyperparameters and implementations used for each method, (iii) an installation guide for each method and (iv) main files to simplify the evaluation of some approaches.

References

1. Lu L, Zhou T. Link prediction in complex networks: A survey. Phys A: Stat Mech Appl. 2011;390:1150–1170.
2. Lichtenwalter RN, Chawla NV. Link prediction: Fair and effective evaluation. In: Proceedings of ASONAM, New York: Association for Computing Machinery, 2012. pp. 376–383.
3. Garcia-Gasulla D, Cortes CU, Ayguade E, et al. Evaluating link prediction on large graphs. In: Proceedings of CAAI, Amsterdam, The Netherlands: IOSPress, 2015. pp. 90–99.
4. Yang Y, Lichtenwalter RN, Chawla NV. Evaluating link prediction methods. KAIS. 2015;45:751–782.
5. Belkin M, Niyogi P. Laplacian eigenmaps and spectral techniques for embedding and clustering. In: Proceedings of NIPS, Cambridge: MIT Press, 2002. pp. 585–591.
6. Tang J, Qu M, Wang M, et al. LINE: Large-scale information network embedding. In: Proceedings of WWW, New York: Association for Computing Machinery, 2015. pp. 1067–1077.
7. Cao S, Lu W, Xu Q. GraRep: Learning graph representations with global structural information. In: Proceedings of CIKM, New York: Association for Computing Machinery, 2015. pp. 891–900.
8. Ou M, Cui P, Pei J, et al. Asymmetric transitivity preserving graph embedding. In: Proceedings of KDD, New York: Association for Computing Machinery, 2016. pp. 1105–1114.
9. Gao M, Chen L, He X, et al. Bine: Bipartite network embedding. In: Proceedings of SIGIR, New York: Association for Computing Machinery, 2018. pp. 715–724.
10. Wang D, Cui P, Zhu W. Structural deep network embedding. In: Proceedings of KDD, New York: Association for Computing Machinery, 2016. pp. 1225–1234.
11. Dong Y, Chawla NV, Swami A. Metapath2vec: Scalable representation learning for heterogeneous networks. In: Proceedings of KDD, New York: Association for Computing Machinery, 2017. pp. 135–144.
12. Perozzi B, Al-Rfou R, Skiena S. Deepwalk: Online learning of social representations. In: Proceedings of KDD, New York: Association for Computing Machinery, 2014. pp. 701–710.

13. Tsitsulin A, Mottin D, Karras P, et al. Verse: Versatile graph embeddings from similarity measures. In: Proceedings of WWW, Republic and Canton of Geneva, Switzerland: International World Wide Web Conferences Steering Committee, 2018. pp. 539–548.
14. Kipf TN, Welling M. Variational graph auto-encoders. NIPS Workshop on Bayesian Deep Learning, Barcelona, Spain, 2016.
15. Kang B, Lijffijt J, De Bie T. Conditional network embeddings. In: Proceedings of ICLR, New Orleans: OpenReview.net, 2019.
16. Zhang Z, Cui P, Wang X, et al. Arbitrary-order proximity preserved network embedding. In: Proceedings of KDD, New York: Association for Computing Machinery, 2018. pp. 2778–2786.
17. Hamilton WL, Ying R, Leskovec J. Representation learning on graphs: Methods and applications. Washington, DC: IEEE, 2017.
18. Zhang D, Yin J, Zhu X, et al. Network representation learning: A survey. Washington, DC: IEEE, 2018.
19. Goyal P, Ferrara E. Graph embedding techniques, applications, and performance: A survey. Knowl Based Syst. 2018;151:78–94.
20. Khosla M, Setty V, Anand A. A comparative study for unsupervised network representation learning. IEEE Trans Knowl Data Eng. 2021;330: 1807–1818.
21. Grover A, Leskovec J. node2vec: Scalable feature learning for networks. In: Proceedings of KDD, New York: Association for Computing Machinery, 2016. pp. 855–864.
22. Lai Y-A, Hsu C-C, Chen WH, et al. Prune: Preserving proximity and global ranking for network embedding. In: Proceedings of NIPS, Red Hook, NY: Curran Associates Inc., 2017. pp. 5257–5266.
23. Gurukar S, Vijayan P, Srinivasan A, et al. Network representation learning: Consolidation and renewed bearing. Arxiv. 2019.
24. Mara AC, Lijffijt J, De Bie T. Evalne: A framework for evaluating network embeddings on link prediction. In: Proceedings of EDML (SDM). Calgary, Canada: CEUR, 2019;2436. pp. 5–13.
25. Kotnis B, Nastase V. Analysis of the impact of negative sampling on link prediction in knowledge graphs. Arxiv. New York: Association for Computing Machinery, 2017.
26. Wei X, Xu L, Cao B, et al. Cross view link prediction by learning noise-resilient representation consensus. In: Proceedings of WWW, Republic and Canton of Geneva, Switzerland: International World Wide Web Conferences Steering Committee, 2017. pp. 1611–1619.
27. Hutson M. Core progress in AI has stalled in some fields. Science. 2020; 368:927–927.
28. Dacrema MF, Cremonesi P, Jannach D. Are we really making much progress? a worrying analysis of recent neural recommendation approaches. In: Proceedings of RecSys, New York: Association for Computing Machinery, 2019. pp.101–109.
29. Mara AC, Lijffijt J, De Bie T. Network embedding models for link prediction: Are we making progress? In: Proceedings of DSAA, Washington, DC: IEEE, 2020. pp. 138–147.
30. Mikolov T, Chen K, Corrado G, et al. Efficient estimation of word representations in vector space. In: Bengio Y, LeCun Y (Eds.): Proceedings of ICLR, Scottsdale, AZ: OpenReview.net, 2013.
31. Ribeiro L, Saverese P, Figueiredo D. struc2vec: Learning node representations from structural identity. In: Proceedings of KDD, New York: Association for Computing Machinery, 2017. pp. 385–394.
32. Abu-El-Haija S, Perozzi B, Al-Rfou R, et al. Watch your step: Learning node embeddings via graph attention. In: Proceedings of NIPS, Red Hook, NY: Curran Associates Inc., 2018. pp. 9198–9208.
33. Ahmed A, Shervashidze N, Narayanamurthy S, et al. Distributed large-scale natural graph factorization. In: Proceedings of WWW, New York: Association for Computing Machinery, 2013. pp. 37–48.
34. Roweis ST, Saul LK. Nonlinear dimensionality reduction by locally linear embedding. Science. 2000;290:2323–2326.
35. Tsitsulin A, Munkhoeva M, Mottin D, et al. FREDE: Anytime graph embeddings. Proc VLDB Endow. 2021;14:1102–1110.
36. Ghashami M, Liberty E, Phillips JM, et al. Frequent directions: Simple and deterministic matrix sketching. SIAM J Comput. 2016;45:1762–1792.
37. Qiu J, Dong Y, Ma H, et al. Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec. In: Proceedings of WSDM, New York: Association for Computing Machinery, 2018. pp. 459–467.
38. Qiu J, Dong Y, Ma H, et al. Netsmf: Large-scale network embedding as sparse matrix factorization. In: Proceedings of WWW, New York: Association for Computing Machinery, 2019.
39. Wang X, Cui P, Wang J, et al. Community preserving network embedding. In Proceedings of AAAI, San Francisco, California: AAAI Press, 2017. pp. 203–209.
40. Xu K, Hu W, Leskovec J, et al. How powerful are graph neural networks? In Proceedings of ICLR, New Orleans: OpenReview.net, 2019.
41. Weisfeiler B, Leman A. The reduction of a graph to canonical form and the algebra which appears therein. NTI Series. 1968;2:12–16.
42. Bresson X, Laurent T. Residual gated graph convnets. Press, 2018.
43. Tu C, Yang C, Liu Z, et al. Network representation learning: An overview. Sci Sin Inf. 2017;47:980–996.
44. Goethals B, Le Page W, Mampaey M. Mining interesting sets and rules in relational databases. In: Proceedings of SAC, New York: Association for Computing Machinery, 2010. pp. 997–1001.
45. Leskovec J, Krevl A. SNAP Datasets: Stanford large network dataset collection. Stanford, CA: Stanford University, 2015.
46. Zafarani R, Liu H. Social computing data repository at asu. Arizona: Arizona State University, 2009.
47. Breitkreutz B-J, Stark C, Reguly T, et al. The biogrid interaction database: 2008 update. Nucleic Acids Res. 2007;36:D637–D640.
48. Mahoney M. Large text compression benchmark. Florida: Florida Tech, 2011.
49. Cukierski W, Hamner B, Yang B. Graph-based features for supervised link prediction. In: Proceedings of IJCNN, Washington, DC: IEEE, 2011. pp. 1237–1244.
50. Hagberg A, Swart P, Chult DS. Exploring network structure, dynamics, and function using networkx. In: Proceedings of SciPy, Pasadena, CA, 2008. pp. 11–15.
51. Ghasemian A, Hosseinmardi H, Galstyan A, et al. Stacking models for nearly optimal link prediction in complex networks. Washington, DC: National Academy of Sciences, 2019.

Cite this article as: Mara AC, Lijffijt J, De Bie T (2024) An empirical evaluation of network representation learning methods. *Big Data* 12:6, 518–537, DOI: 10.1089/big.2021.0107.

Abbreviations Used

AUC = area under the ROC curve
DFT = depth first tree
DT = decision trees
FPR = false positive rates
GF = graph factorization
GIN = graph isomorphism network
LE = Laplacian Eigenmaps
LLE = locally linear embeddings
LR = logistic regression
LRCV = LR with fivefold cross validation
NE = network embedding
SDNE = structural deep network embedding
ST = spanning tree
TPR = true positive rates
WYS = Watch Your Step