Network Embedding Exploration Tool (NEExT)

Ashkan Dehghan[1], Paweł Prałat[1], and François Théberge[2]

[1] Department of Mathematics, Toronto Metropolitan University, Toronto, ON, Canada {ashkan.dehghan,pralat}@torontomu.ca
[2] Tutte Institute for Mathematics and Computing, Ottawa, ON, Canada theberge@ieee.org

Abstract. In this paper, we introduce **NEExT**(**N**etwork **E**mbedding **Ex**ploration **T**ool) for embedding collections of graphs via user-defined node features. The advantages of the framework are twofold: (i) the ability to easily define your own interpretable node-based features in view of the task at hand, and (ii) fast embedding of graphs provided by the **Vectorizers** library. In this exploratory work, we demonstrate the usefulness of **NEExT** on collections of synthetic and real-world graphs.

1 Introduction

Many real-world as well as artificial systems and processes can be represented as graphs. Such systems include social networks, financial transactions, supply chains and molecular structures, to name a few. In many cases, one needs to consider a collection of related graphs, whether they are the result of multiple systems (e.g., different proteins) or are produced by a dynamic process of the same network (e.g., evolution of a social network, graphs induced by different communities, or ego-nets around various nodes). A significant challenge in most scenarios is the absence of ground-truth labels for graphs and nodes, and therefore one needs to use unsupervised techniques to study various properties of such networks. To address these challenges, we introduce an **N**etwork **E**mbedding **Ex**ploration **T**ool (**NEExT**)[3], which can be used to analyze collection of graphs in an unsupervised fashion.

Node embedding is a transformation of nodes of a network into a set of vectors [1]. Due to their spectacular successes in various applications, they are becoming increasingly popular in the ML community. There are over 100 algorithms available to use and frameworks to evaluate them (such as [8]). Independently, many analytic tasks (such as classification, clustering, and regression) in various domains, including social networks, cybersecurity, bio- and chemo-informatics, require representing graphs as fixed-length feature vectors [1]. For example, embeddings of graphs representing program's calls could be used to detect malware [18], embeddings of graphs representing chemical compounds could be used to predict properties of the associated compounds such as solubility and anti-cancer activity [25, 19].

[3] https://pypi.org/project/NEExT/

Historically, graph kernels were considered to be a standard way to deal with the above graph analytics tasks. In this approach, the similarity (kernel value) between pairs of graphs is computed by recursively decomposing them into simpler substructures (such as random walks, shortest paths, graphlets) and defining similarity (kernel) between these substructures. After that, some standard kernel methods, such as Support Vector Machines (SVMs), can be used to classify or cluster graphs. Note that many algorithms of this nature do not explicitly produce graph embeddings and so they cannot be immediately used for general ML tasks.

To overcome this limitation, another powerful technique was introduced recently in the literature [22, 16]. We start with extracting features of nodes of a graph G via some node embedding algorithm. Such cloud of n points, corresponding to vectors of features of n nodes of G, can be easily normalized so that it can be viewed as the probability distribution on a metric space equipped with a distance, such as the Euclidean distance. Then, the Wasserstein distance can be used to measure the distance between two graphs by computing the distance between the two corresponding probability distributions. The Wasserstein distance is a metric and is linked to the optimal transport problem [23] which aims to find an optimal way to transport the probability mass associated with one graph to the one associated with another one. This distance is sometimes referred to as the earth mover's distance since in 2-dimensional case one can think of it as moving piles of dirt. Finally, some algorithm is used to embed graphs into k-dimensional space of vectors such that the Wasserstein distance between graphs matches the distance between the corresponding vectors as much as possible. In this paper, we introduce a framework that builds on ideas from [22, 16]. On top of providing an efficient and user-friendly exploratory networks analysis tool, our main contribution can be summarized as follows.

- The framework not only utilizes a number of standard classical and structural node embeddings but allows to include hand-crafted, user-defined feature vectors that, for example, measure the distribution of power (by including Pagerank or some other centrality measures) or expansion of ego-nets around nodes. This approach has a few immediate benefits: it is much faster to compute such features than embed all nodes, the results are interpretable and more robust.
- The framework utilizes various techniques and metrics for approximating the distances between graphs such as Wasserstein distance and Sinkhorn algorithms, in addition to a computationally efficient approximate Wasserstein vectorization approach. These tools are available and maintained in the **Vectorizers** package.
- For supervised learning (when labels for graphs are available), the framework selects (in an automated way) a subset of available node features and appropriately normalizes them for the best outcome of a given ML supervised task at hand such as classification or regression. (Work in progress, not discussed in this proceeding version of the paper.)

2 The Framework

Consider a collection of graphs $G_1, G_2, \ldots, G_m$. Our approach follows the following steps (details are provided in the following subsections):

1. Pre-process the collection of graphs.
2. For each graph G_i with n_i nodes, build k-dimensional vector representations for all the nodes.
3. Given m collections of k-dimensional vectors, one collection for each graph, compute d-dimensional embedding of the graphs via either the `Wasserstein`, `Sinkhorn`, or `ApproximateWasserstein` distance.

Note: In the following description, individual components of the graph collection ($G_i s$), will be referred to as *subgraphs*.

2.1 Pre-processing

In the pre-processing layer, **NEExT** loads each subgraph into a **GraphCollection** object, assigning to each subgraph appropriate details such as graph labels, graph statistics, etc. In this layer, we can also filter each subgraph for its largest connected component.

2.2 Vectorizing the Nodes

Several methods to obtain vector representations for the nodes of each graph are available in the framework. However, one advantage of **NEExT** is that it is easy to add other vector representations for the nodes, which can be interpretable and designed specifically for the types of graphs one wants to analyze and downstream Machine Learning task using the generated graph embedding. It can also be used to easily test various families of node-based features in order to select suitable ones. In the final implementation of **NEExT** (work in progress), if labels are available for some graphs, then such selection can be done in a supervised way. Node features can be computed recursively up to some maximum value k (resulting a k dimensional feature vector) by averaging its values for neighbours up to j hops away for $1 \leq j \leq k$. Moreover, features can be concatenated to obtained even higher dimensional representations.

LSME. One of the built-in structural embedding algorithms is called **Local Signature Matrix Embedding (LSME)**. This technique uses a random-walk algorithm to capture local structural properties of nodes. The algorithm results in a k-dimensional vector, where each element measures the transition probability between various neighbourhoods around a given node[4].

Centrality measures. We compute various commonly used centrality measures such as **PageRank**, **Closeness Centrality**, **Degree Centrality**, and

[4] `https://github.com/ashdehghan/LSME`

Eigenvector Centrality. Details for those measures can be found in, for example, [11].

Expansion properties. For each node v, let $\hat{m}_i$ be the number of neighbours at distance i from v, for $i \in \{1, 2, \ldots, k\}$. Our goal is to embed vertices of possibly different degrees that expand in a similar way close to each other. Hence, we consider the following feature vector for a node v:

$$E(v) = \left(\frac{\hat{m}_1}{1 \cdot \bar{d}}, \frac{\hat{m}_2}{n_1 \cdot \bar{d}}, \ldots, \frac{\hat{m}_k}{n_{k-1} \cdot \bar{d}} \right),$$

where $\bar{d} = \frac{1}{N} \sum_{v \in V} \deg(v) = \frac{2|E|}{N}$ is the average degree. For good expanders, one would get a collection of vectors that are close to $(1, 1, \ldots, 1)$.

2.3 Embedding of the Graphs

Assume that we have a collection of graphs $G_i, 1 \leq i \leq m$, with respectively n_i nodes, and a k-dimensional vector representation for each node. Then, each graph can be seen as a distribution of points over k-dimensional space, and we can use some measure of distance between distributions to embed the graphs in some vector space. One possible approach is to use the Wasserstein distance, which is obtained by finding the optimal transport plan between distributions; this is also known as the earth mover's distance between distributions (i.e. measure the amount of "work" to move mass from one distribution to the other); see, for example, [22]. Embedding graphs this way is similar to the context of document embedding[5], where each word is represented by a vector (obtained via some word embedding algorithm), and each document is a "bag of word vectors".

Given m graphs, computing all such distances requires estimating $O(m^2)$ pairwise distances, which has a high computational cost. One solution to this issue is to define some reference distribution (for example via averaging the vectors), and find the optimal transport plan from each graph's "bag of vectors" to this reference distribution. This is know as linear optimal transport (LOT) [23], which is used, for example, in [16]. We use the implementation of this approach from the easy to use and frequently maintained `Vectorizers`[6] Python package, which solves the LOT and computes embeddings by computing the SVD (Singular Value Decomposition) of the optimal transport plans.

Computing the Wasserstein distances, even using a reference distribution, can still be prohibitive for some large problems. We therefore consider two faster methods which are also implemented in `Vectorizers`. The first one uses the Sinkhorn distance which is based on entropic regularization of the transport plans; see [5]. The other one, ApproximateWasserstein, also solves the LOT but using a single-point reference distribution obtained via averaging, as described in [2]. Embeddings are obtained using SVD with scaling according to the singular values.

[5] `https://vectorizers.readthedocs.io`
[6] `https://pypi.org/project/vectorizers/`

As a rule of thumb, when $k \ll m$, one can use the Wasserstein or Sinkhorn approach while for larger k, the ApproximateWasserstein can be used for better performance.

3 Experiments

We illustrate the use of our framework to analyse synthetic as well as real-life networks. The goal is to explore various capabilities of the framework for both unsupervised and supervised applications. In the first subsection, we use the ***A**rtificial **B**enchmark for **C**ommunity **D**etection* (**ABCD**) framework [10], to generate synthetic graphs. The **ABCD** framework is a random graph model framework with community structure and power-law distribution for both degrees and community sizes. The goal here is to explore and highlight various properties of our framework in a controlled environment and showcase its use from a practitioners point of view. Therefore, we consider idealized and synthetically generated cases, while considering more real-world scenarios in the following subsection.

3.1 Synthetic Graphs

The **ABCD** model, an alternative approach to the **LFR** model [17], allows us to generate random graphs with control over power-law distribution of both node degrees and of community sizes, fraction of outlier nodes, noise, and other parameters. We leverage the Julia implementation[7] to generate the synthetic graphs used in this section. However, let us mention that for generating enormous graphs there exists a faster implementation[8] available that uses multiple threads (**ABCDe**) [15].

Undirected variant of **LFR** and **ABCD** produce graphs with comparable properties but **ABCD**/**ABCDe** is faster than **LFR** and can be easily tuned to allow the user to make a smooth transition between the two extremes: pure (disjoint) communities and random graph with no community structure. Moreover, it is easier to analyze theoretically—for example, in [9, 3] various theoretical asymptotic properties of the **ABCD** model are investigated including the modularity function and self-similarities of the ground-truth communities. More importantly, the model is extremely flexible and allows to include outliers [12] (**ABCD+o**) or generate hypergraphs [13] (**h–ABCD**).

To explore various properties of our framework, we designed three experiments. The parameters used in each experiment are outlined in Table 1.

Experiment 1 – Varying Level of Noise. In the first experiment, we explore the effect of noise that is controlled by parameter ξ in the **ABCD** model. We designed this experiment to mimic a dynamic property of a graph in which

[7] `https://github.com/bkamins/ABCDGraphGenerator.jl`
[8] `https://github.com/tolcz/ABCDeGraphGenerator.jl/`

Parameter	Experiment 1	Experiment 2	Experiment 3
n	200	$\{200, 250, \ldots, 400\}$	200
γ	3	3	3
δ	5	5	5
Δ	10	10	10
β	2	2	2
c	10	10	10
C	20	20	20
ξ	$\{0.1, 0.101, \ldots, 0.9\}$	$\{0.1, 0.2, \ldots, 0.5\}$	0.2
o	0	0	(150 x 10) + (150 x 50)

Table 1: **ABCD** Synthetic Graph Parameters. (The number of nodes is equal to n. The degree distribution follows power-law with exponent γ, minimum δ and maximum Δ. The distribution of community sizes follows power-law with exponent β, minimum c and maximum C. The level of noise is controlled by ξ. Finally, the number of outliers is equal to o. For a more in depth description of each parameter and how it is utilized, we refer the reader to the GitHub repository.)

an incremental change in a property of a parameter in the graph results in the change in underlying graph structure. In a real world network, this could resemble the change in the polarization of a network in which the boundaries between communities slowly vanish. Of course, we have to note that our synthetic experiment is an idealized version of such system.

We construct 801 graphs, with ξ ranging from 0.1 to 0.9 in steps of 0.001. In the **ABCD** model, ξ controls the fraction of edges that fall into the background graph (almost all of these edges are between nodes from different communities). A sample of four graphs from the 801 generated ones are shown in Figure 1. We then use ξ values as a label for each graph to be used for a regression task. Steps of this experiment are as follows:

- Generate a collection of graphs with varying level of noise controlled by ξ.
- Generate feature vectors of size k=2 to k=8 for each graph.
- Use the features vectors to construct graph embeddings of dimension $d = 2$.
- Use the graph embedding vectors as features for a regression task to predict the ξ values for each graph.

Once the graph collection is generated and loaded into our framework, we can compute various graph properties on each graph in the collection. For this experiment, we compute the following graph features: **Expansion**, **LSME**, **PageRank**, **Closeness Centrality**, **Degree Centrality**, and **Eigenvector Centrality**. For each feature, we construct a k dimensional vector. For example, for **PageRank** as a feature with $k = 3$, for each node v we calculate the **PageRank** value of v as well as the average **PageRank** values of neighbours at distance i from v, where $i \in \{1, 2, \ldots, k-1\}$. We also construct larger feature vectors

by concatenating the vectors from multiple features. For example, a feature vector of **Expansion** + **LSME** with $k = 3$ is a concatenation of a 3 dimensional **LSME** feature vector and 3 dimensional **Expansion** feature vector, resulting in a 6 dimensional global feature vector.

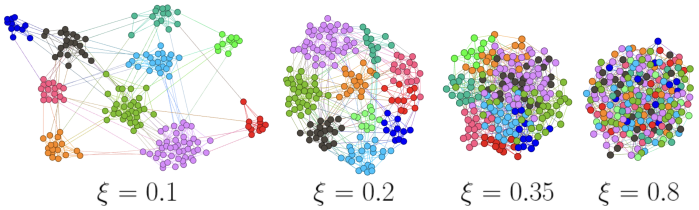

Fig. 1: Examples of graphs generated using the **ABCD** synthetic graph for Experiment 1, as detailed in Table 1, for $\xi \in \{0.1, 0.2, 0.35, 0.8\}$.

Having defined the feature generation process, we construct features of lengths k from 2 to 8 based on the above list. We then use the approximate Wasserstein technique to embed each graph in the collection into a two dimensional embedding. We chose embedding dimension $d = 2$, since the approximate technique has an upper limit of k for the dimension of the embedded space and the smallest feature vector size is of dimension $k = 2$. Moreover, since our graph embeddings are used in a downstream supervised regression task, we wanted to keep the dimensionality of the embeddings the same to standardize the comparison of the models.

In Figure 2 we show the two dimensional embedding of graphs build using the **Expansion**, **LSME**, and **PageRank** features. In all three cases, the underlying feature vectors have length $k = 5$. Each data point (graph embedding) is then coloured based on the available label (ξ). It is clear that in all three cases there is a relation between the value of ξ and the graph embedding vector. To explore this relationship further, we use a regression model for graph embeddings that are built using various graph features.

Using the two dimensional graph embeddings of the graphs, we train regression models using XGBoost[9] to predict the value of ξ for the unlabeled graphs. In our experiment, the train/test split is set to 70/30 and we repeat each experiment 100 times to arrive at the average mean-absolute-error and the standard deviation over the runs, shown as error bars in Figure 3. Here, the x-axis corresponds to k, the length of feature vectors computed for nodes of each graph, and different colours correspond to combination of various types of features.

We start by highlighting the fact that the overall performance of the models increases (the mean-absolute-error decreases) as the length of the underlying feature vectors increases. This is expected, since increase in k corresponds to a larger window for capturing structural properties of the underlying graph. We

[9] `https://xgboost.readthedocs.io/en/stable/`

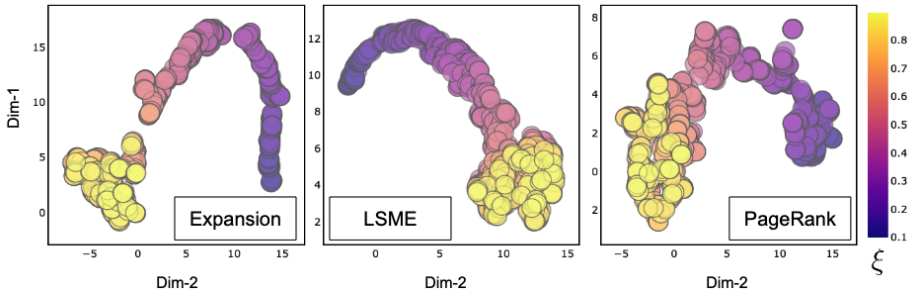

Fig. 2: Two dimensional graph embeddings built using the approximate Wasserstein technique and graph features built using the **Expansion**, **LSME**, and **PageRank** metrics. The dimension for all the above node embeddings is set to $k = 5$.

show that this trend continues until the length of the feature vectors reaches the diameter of the graphs. At this length scale, the feature vectors are capturing global structural properties of the graph. We note that the diameter for our synthetic graphs is relatively small, since we consider graphs of size $n = 200$ that are relatively good expanders. Lastly, we note that models built on combination of features perform the best, since each feature captures a different structural property of the graph. We note that in this experiment we are not interested in fine-tuning the XGBoost models to achieve the best performance, but rather to illustrate the predictive power of various graph features and associated graph embeddings.

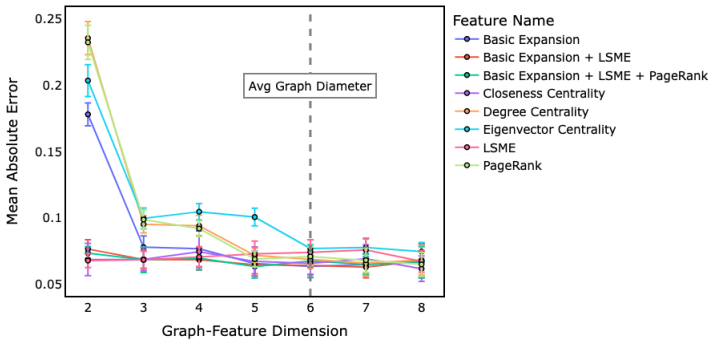

Fig. 3: Mean-absolute-error measured for a regression model built to predict ξ in Experiment 1, as defined in Table 1. The x-axis is the length of the feature vectors computed on each graph. The final graph embedding is uniformly set to $d = 2$.

Experiment 2 – Varying Network Size. In the second experiment, we explore the ability of our framework to capture structural similarities in collections of graphs. In real systems, it is often important to identify structurally similar graphs, regardless of the size of the network. This is often seen in self-similar systems, such as social networks, where particular property presents itself at different scales [20, 3]. To study this effect, we use the **ABCD** model to generate structurally similar networks of various sizes. We achieve this goal by tuning two parameters: the level of noise (ξ) and the number of nodes in each graph (n). As highlighted in Table 1, we build a collection of 25 graphs with $\xi \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$ and $n \in \{200, 250, 300, 350, 400\}$. As it was done in Experiment 1, we compute node features for each graph and use the approximate Wasserstein technique to build an embedding vector for each graph. We fix the feature vector size to $k = 4$ and graph embedding size to $d = 4$. To visualize the final embeddings, we use UMAP[10] to map the final graph embedding into two dimensions.

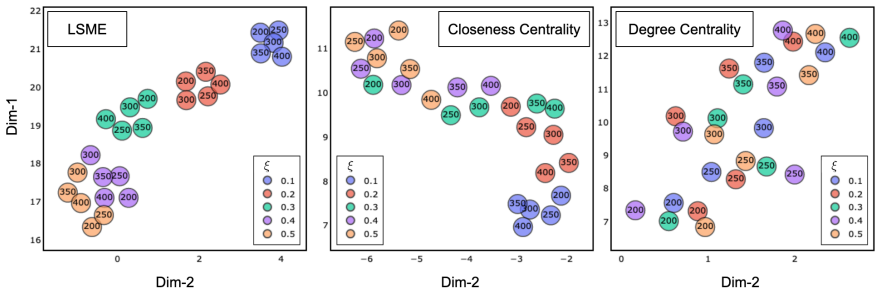

Fig. 4: Two dimensional representations of the approximate Wasserstein graph embeddings built using **LSME**, **Closeness Centrality**, and **Degree Centrality** graph features. Colours correspond to different values of noise (ξ) and the size of the underlying graphs (n) are shown inside each data point.

In Figure 4, we show the two dimensional representations of the graph embeddings, coloured and annotated using ξ and n, respectively. In the first chart (left), the underlying feature vector was computed using the **LSME** structural embedding algorithm. This technique captures structural properties of nodes within each graph. Using **LSME** as features, the graph embeddings captures similarities between structural properties of each graph. This can be seen in the final two dimension representation of the embeddings, since graphs with similar structure (ξ value, colour-coded) are clustered together. It is interesting to observe that the quality of the clusters (tightness) decreases as the noise (ξ) increases.

[10] `https://umap-learn.readthedocs.io/en/latest/`

Similar behaviour is also captured by a more simple structural feature, **Closeness Centrality**. We can see in the middle chart in Figure 4 that **Closeness Centrality** also groups graphs of similar property together, but with lower quality compared to **LSME**. The decrease in clustering quality as a function of the level of noise is more evident in this case. The important observation in these cases points at the fact that the approximate Wasserstein technique using **LSME** or **Closeness Centrality** preserves structural properties of the embedded graphs such as level of noise.

Experiment 3 – Outlier Detection. In this experiment, we control the fraction of outlier nodes in the graph. We construct two sets of graphs. In the first one, 5% of nodes are outliers and in the second group there are more outliers, namely, 25%. For each group, we generate 150 graphs. It is important to note that, since the **ABCD** model is a *randomized* graph generator, each graph in their respective groups are different due to a random nature of the model. This is achieved by setting different random seeds while generating the graphs.

We compute various graph features on each subgraph and use them to construct graph embeddings using the approximate Wasserstein techniques. Here, we consider 9 different sub-groups of features, as defined in Table 2. Each feature type has a dimension of $k = 4$ and we combine different features by concatenating their feature vectors. We note that in this experiment, we train binary classifiers using XGBoost classifier with 70/30 train/test split, and repeat each experiment 100 times to collect enough statistics for model performance.

Model	**Details**
M-0	Expansion
M-1	LSME
M-2	PageRank
M-3	Degree Centrality
M-4	Closeness Centrality
M-5	Eigenvector Centrality
M-6	Expansion, LSME
M-7	Expansion, LSME, PageRank
M-8	Expansion, LSME, PageRank, Degree Centrality, Closeness Centrality, Eigenvector Centrality

Table 2: **ABCD** outlier classification model.

In Figure 5 we show, on the left, the performance of binary-classifiers built for each model (M-0 to M-8) measured using accuracy and (right) a two dimensional clustering of the graph embeddings built using M-8 features. Starting with the right figure, it is clear that the two dimensional representations of graph embedding vectors form two well separated clusters. In an unsupervised setting, one

could use a technique such as DBSCAN [7] to identify these two clusters, even if the underlying classes are not known. In this experiment however, we know that the underlying graphs are generated using random graph technique with two outlier settings. We show the fraction of outliers for each group as different colours in this figure.

Next, we analyze the performance of binary classifiers trained on graph embedding vectors built using different combinations of feature vectors (Table 2). In Figure 5 (left), we show the model accuracy for each feature set. Focusing on single feature models (M-0 to M-5), we can see that graph embeddings built on top of **Closeness Centrality** (M-4) perform the best, while models built using **Eigenvector Centrality** (M-5) perform poorly. It is also worth mentioning that **Expansion** (M-0), an easy and fast to compute node feature, does very well. The predictive power of **Closeness Centrality** as a node feature comes from the fact that outlier nodes are, on average, closer to other nodes in the network, since they do not belong to any of the communities but rather randomly connected to the entire network. Therefore, this can be a distinctive factor for graphs with higher number of outlier nodes.

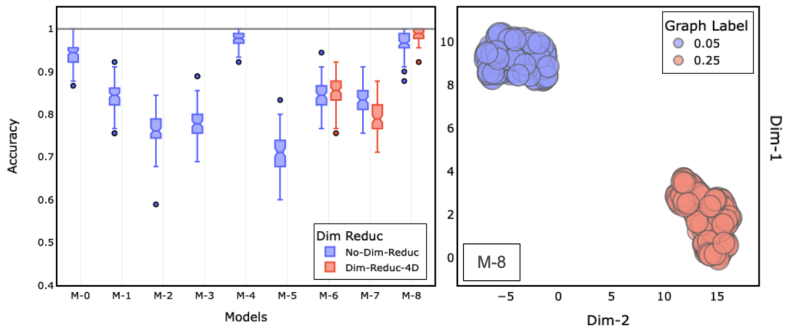

Fig. 5: Left: Accuracy of binary-classifiers built for models M-0 to M-8. Right: two dimensional representation of graph embedding vectors built using features in M-8.

Lastly, we consider composite models, where embeddings are generated from a combination of feature vectors. In Figure 5 (left), models M-6, M-7, and M-8 are composite models (as defined in Table 2). For each one these models, we run two sets of experiments. One in which we do not apply any dimensionality reduction to the composite feature vectors, before passing them to the graph embedding layer. And, another where we apply dimensionality reduction using UMAP, to reduce the dimension of the composite feature vector to $k = 4$. We can see that in all three cases, dimensionality reduction does not have a significant effect on the performance of the models. One thing worth highlighting is that the composite model (M-8) built using all features with $k = 4$ performs the best.

This hints at the fact that one could capture a wide variety of features in the feature computation layer, then reducing the feature space using a technique such as UMAP to allow for a better performance of a given machine learning model at hand.

3.2 Real-World Networks

In this section, we explore the performance of our framework on a collection of real-world networks. This collection was acquired from the Benchmark Data Set provided by the department of computer science of TU Dortmund[11]. A summary of networks used for our experiments are provided in Table 3. Here, we consider five real-world networks (**IMDB** [25], **MUTAG** [6], **NCI1** [24], **BZR** [21], and **PROTEINS** [4]) with various sizes and source, which have sub-networks that can be categorized into two classes. Therefore, a natural type of analysis would be to investigate the performance of binary-classifiers trained on graph embeddings generated by **NEExT**. In our analysis, we use the performance of publicity available models as a benchmark for comparing **NEExT** to other techniques. In this exploratory stage of our project, the goal is not to seek the best performing model at all cost, but rather provide a framework that can easily create models with reasonable performance compared to state-of-the-art techniques, while keeping model explainability. Moreover, note that some models are trained on additional metadata available for nodes as well es edges. We do not do it at present but it would be easy to incorporate such additional information in our model. It is expected that after appropriate selection of node features and fine-tuning the model, the accuracy of the corresponding models should increase.

Name	**# of Graphs**	**# of Classes**	**Avg. # of Nodes/Edges**
IMDB	1000	2	19.77/96.53
MUTAG	188	2	17.93/19.79
NCI1	4110	2	29.87/32.30
BZR	405	2	35.75/38.36
PROTEINS	1113	2	39.06/72.82

Table 3: Summary of Real-World Networks.

In our experiments on the real-networks listed in Table 3, we build $d = 24$ dimensional graph embeddings using the approximate Wasserstein technique on top of $k = 24$ dimensional feature vectors computed on each sub-network. Here, we use 4-dimensional concatenated **LSME**, **Expansion**, **Degree Centrality**, **Closeness Centrality**, **Load Centrality**, and **Eigenvector Centrality** as our feature vectors. We then train the XGBoost binary classifier on top of the

[11] `https://ls11-www.cs.tu-dortmund.de/staff/morris/graphkerneldatasets`

$d = 24$ dimensional graph embedding feature vectors. Similarly to the approach taken before, we keep a 70/30 train/test split, and repeat each experiment 100 times to build the statistics for our model performance. Lastly, we point out that we do not balance the datasets in our models, to allow the classifiers to capture statistical imbalances in the underlying data distribution.

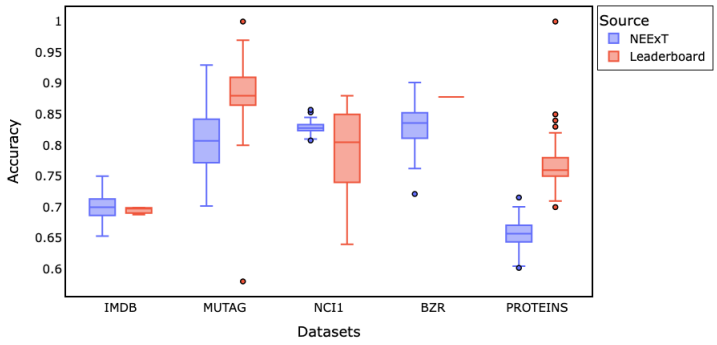

Fig. 6: Accuracy of models built using **NEExT** framework (blue) and publicly available models (red), for various real-world networks. The performance of other models is collected from leaderboard chart available on-line.

Name	Accuracy	Precision	Recall	F1-Score	LB Accuracy
IMDB	0.70 ± 0.02	0.71 ± 0.03	0.69 ± 0.04	0.69 ± 0.02	0.52-0.96
MUTAG	0.81 ± 0.05	0.70 ± 0.08	0.70 ± 0.08	0.69 ± 0.08	0.58-1.00
NCI1	0.83 ± 0.01	0.85 ± 0.02	0.79 ± 0.01	0.82 ± 0.01	0.64-0.88
BZR	0.83 ± 0.03	0.85 ± 0.03	0.95 ± 0.02	0.90 ± 0.02	0.87
PROTEINS	0.66 ± 0.02	0.59 ± 0.05	0.47 ± 0.04	0.52 ± 0.03	0.70-0.85

Table 4: Summary of Real-World Networks Classification Results. Here, LB Accuracy refers to the range of accuracy of the models from the leaderboard.

In Figure 6 and Table 4, we show the performance of classifiers trained using **NEExT** and benchmark models collected from leaderboard chart available on-line[12]. Note that the **LB** accuracies are shown as the range of accuracies from various models submited for each dataset. We see that the accuracy of models

[12] `https://paperswithcode.com/task/graph-classification`

built using **NEExT** is similar to other models, even without performing any fine-tuning of our models.

4 Conclusion

In this paper, we introduce **NEExT**, the **N**etwork **E**mbedding **Ex**ploration **T**ool and show that it can be easily used for feature engineering toward embedding of graphs. This is the beginning of a larger project but the initial experiments we performed so far, some of them reported in this paper, are positive and encouraging to do more work in this space. Here are some natural next steps that we plan to do in the near future.

- Include and test more of our own, carefully designed and explainable, node features. In particular, based on our own personal interests and applications in mind, we plan to add various community-aware node features [14].
- Include more classical node features (such as other centrality measures, degree-degree correlations) and node embeddings (especially structural node embeddings).
- Design and implement an algorithm that automatically selects features and normalizes them in an supervised process, provided that labels for graphs are available.
- Do a grand study comparison with the state-of-the-art methods for graph classification tasks (comparing both the quality of generated embeddings as well as speed). Analyze which types of node features work best for graphs from various domains (explainability).
- Design and implement sampling method which might be useful to embed a large collection of large networks.

References

1. Aggarwal, M., Murty, M.N.: Machine learning in social networks: embedding nodes, edges, communities, and graphs. Springer Nature (2020)
2. Arora, S., Liang, Y., Ma, T.: A simple but tough-to-beat baseline for sentence embeddings. In: International Conference on Learning Representations (2017)
3. Barrett, J., Kamiński, B., Pankratz, B., Prałat, P., Théberge, F.: Self-similarity of communities of the abcd model. preprint, arXiv (2023)
4. Borgwardt, K.M., Ong, C.S., Schönauer, S., Vishwanathan, S., Smola, A.J., Kriegel, H.P.: Protein function prediction via graph kernels. Bioinformatics **21**(suppl_1), i47–i56 (2005)
5. Cuturi, M.: Sinkhorn distances: Lightspeed computation of optimal transport. In: Burges, C., Bottou, L., Welling, M., Ghahramani, Z., Weinberger, K. (eds.) Advances in Neural Information Processing Systems. vol. 26. Curran Associates, Inc. (2013)
6. Debnath, A.K., Lopez de Compadre, R.L., Debnath, G., Shusterman, A.J., Hansch, C.: Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. Journal of medicinal chemistry **34**(2), 786–797 (1991)

7. Ester, M., Kriegel, H.P., Sander, J., Xu, X., et al.: A density-based algorithm for discovering clusters in large spatial databases with noise. In: kdd. vol. 96, pp. 226–231 (1996)
8. Kamiński, B., Kraiński, Ł., Prałat, P., Théberge, F.: A multi-purposed unsupervised framework for comparing embeddings of undirected and directed graphs. Network Science **10**(4), 323–346 (2022)
9. Kamiński, B., Pankratz, B., Prałat, P., Théberge, F.: Modularity of the abcd random graph model with community structure. Journal of Complex Networks **10**(6), cnac050 (2022)
10. Kamiński, B., Prałat, P., Théberge, F.: Artificial benchmark for community detection (abcd)—fast random graph model with community structure. Network Science **9**(2), 153–178 (2021)
11. Kamiński, B., Prałat, P., Théberge, F.: Mining Complex Networks. CRC Press (2022)
12. Kamiński, B., Prałat, P., Théberge, F.: Artificial benchmark for community detection with outliers (abcd+o). Applied Network Science **8**(1), 25 (2023)
13. Kamiński, B., Prałat, P., Théberge, F.: Hypergraph artificial benchmark for community detection (h–abcd). Journal of Complex Networks **11**(4), cnad028 (2023)
14. Kamiński, B., Prałat, P., Théberge, F., Zajac, S.: Predicting properties of nodes via community-aware features. arXiv preprint **2311.04730** (2023), `doi.org/10.48550/arXiv.2311.04730`
15. Kamiński, B., Olczak, T., Pankratz, B., Prałat, P., Théberge, F.: Properties and performance of the abcde random graph model with community structure. Big Data Research **30**, 100348 (2022)
16. Kolouri, S., Naderializadeh, N., Rohde, G.K., Hoffmann, H.: Wasserstein embedding for graph learning. In: International Conference on Learning Representations (2021)
17. Lancichinetti, A., Fortunato, S., Radicchi, F.: Benchmark graphs for testing community detection algorithms. Physical review E **78**(4), 046110 (2008)
18. Narayanan, A., Chandramohan, M., Chen, L., Liu, Y., Saminathan, S.: subgraph2vec: Learning distributed representations of rooted sub-graphs from large graphs. arXiv preprint arXiv:1606.08928 (2016)
19. Narayanan, A., Chandramohan, M., Venkatesan, R., Chen, L., Liu, Y., Jaiswal, S.: graph2vec: Learning distributed representations of graphs. arXiv preprint arXiv:1707.05005 (2017)
20. Song, C., Havlin, S., Makse, H.A.: Self-similarity of complex networks. Nature **433**(7024), 392–395 (2005)
21. Sutherland, J.J., O'brien, L.A., Weaver, D.F.: Spline-fitting with a genetic algorithm: A method for developing classification structure- activity relationships. Journal of chemical information and computer sciences **43**(6), 1906–1915 (2003)
22. Togninalli, M., Ghisu, E., Llinares-López, F., Rieck, B., Borgwardt, K.: Wasserstein weisfeiler-lehman graph kernels. Advances in neural information processing systems **32** (2019)
23. Villani, C., et al.: Optimal transport: old and new, vol. 338. Springer (2009)
24. Wale, N., Watson, I.A., Karypis, G.: Comparison of descriptor spaces for chemical compound retrieval and classification. Knowledge and Information Systems **14**, 347–375 (2008)
25. Yanardag, P., Vishwanathan, S.: Deep graph kernels. In: Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining. pp. 1365–1374 (2015)