Bioinformatics, 39(1), 2023, btad047
https://doi.org/10.1093/bioinformatics/btad047
Advance Access Publication Date: 23 January 2023
Original Paper

Systems biology

Accurately modeling biased random walks on weighted networks using *node2vec*+

Renming Liu [1], **Matthew Hirn** [1,2,3] **and Arjun Krishnan** [1,4,*]

[1]Department of Computational Mathematics, Science & Engineering, Michigan State University, East Lansing, MI 48824, USA, [2]Department of Mathematics, Michigan State University, East Lansing, MI 48824, USA, [3]Center for Quantum Computing, Science & Engineering, Michigan State University, East Lansing, MI 48824, USA and [4]Department of Biomedical Informatics, University of Colorado Anschutz Medical Campus, Aurora, CO 80045, USA

*To whom correspondence should be addressed.

Associate Editor: Pier Luigi Martelli

Received on August 14, 2022; revised on January 16, 2023; editorial decision on January 18, 2023; accepted on January 20, 2023

Abstract

Motivation: Accurately representing biological networks in a low-dimensional space, also known as network embedding, is a critical step in network-based machine learning and is carried out widely using *node2vec*, an unsupervised method based on biased random walks. However, while many networks, including functional gene interaction networks, are dense, weighted graphs, *node2vec* is fundamentally limited in its ability to use edge weights during the biased random walk generation process, thus under-using all the information in the network.

Results: Here, we present *node2vec*+, a natural extension of *node2vec* that accounts for edge weights when calculating walk biases and reduces to *node2vec* in the cases of unweighted graphs or unbiased walks. Using two synthetic datasets, we empirically show that *node2vec*+ is more robust to additive noise than *node2vec* in weighted graphs. Then, using genome-scale functional gene networks to solve a wide range of gene function and disease prediction tasks, we demonstrate the superior performance of *node2vec*+ over *node2vec* in the case of weighted graphs. Notably, due to the limited amount of training data in the gene classification tasks, graph neural networks such as GCN and GraphSAGE are outperformed by both *node2vec* and *node2vec*+.

Availability and implementation: The data and code are available on GitHub at https://github.com/krishnanlab/node2vecplus_benchmarks. All additional data underlying this article are available on Zenodo at https://doi.org/10.5281/zenodo.7007164.

Contact: arjun@msu.edu or arjun.krishnan@cuanschutz.edu

Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 Introduction

Graphs and networks naturally appear in many real-world datasets, including social networks and biological networks. The graph structure provides insightful information about the role of each node in the graph, such as protein function in a protein–protein interaction network (Krishnan *et al.*, 2016; Liu *et al.*, 2020). To more efficiently and effectively mine information from large-scale graphs with thousands or millions of nodes, several node embedding methods have been developed (Cui *et al.*, 2018; Hamilton *et al.*, 2017). Among them, *node2vec* has been the top choice in bioinformatics due to its superior performance compared to many other methods (Ata *et al.*, 2021; Yue *et al.*, 2019). However, many biological networks, such as Greene *et al.* (2015) and Johnson and Krishnan (2022), are dense and weighted by construction, which we demonstrate to be undesirable conditions for *node2vec* that can lead to sub-optimal performance.

Node2vec (Grover and Leskovec, 2016) is a second-order random walk-based embedding method. It is widely used for unsupervised node embedding for various tasks, particularly in computational biology (Nelson *et al.*, 2019), such as for gene function prediction (Liu *et al.*, 2020), disease gene prediction (Ata *et al.*, 2018; Peng *et al.*, 2019), and essential protein prediction (Wang *et al.*, 2021a; Zeng *et al.*, 2021). Some recent works built on top of *node2vec* aim to adapt *node2vec* to more specific types of networks (Valentini *et al.*, 2021; Wang *et al.*, 2021b), generalize *node2vec* to higher dimensions (Hacker, 2021), augment *node2vec* with additional downstream processing (Chattopadhyay and Ganguly, 2020; Hu *et al.*, 2020), or to study *node2vec* theoretically (Davison and Austern, 2021; Grohe, 2020; Qiu *et al.*, 2018). Nevertheless, none of these follow-up works account for the fact that *node2vec* is less effective for weighted graphs, where the edge weights reflect the (potentially noisy) similarities between pairs of nodes. This failing is due to the inability of *node2vec* to differentiate between small and large edges connecting the previous vertex with a potential next vertex in the random walk, which subsequently causes less accurate modeling of the intended walk bias.

Meanwhile, another line of recent works on graph neural networks (GNNs) has shown remarkable performance in prediction tasks that involve graph structure, including node classification (Bronstein *et al.*, 2021; Wu *et al.*, 2021; Zhang *et al.*, 2021). Although GNNs and embedding methods like *node2vec* are related in that they both aim at projecting nodes in the graph to a feature space, two main differences set them apart. First, GNNs typically require labeled data, while embedding methods do not. This label dependency makes the embeddings generated by a GNN tied to the quality of the labels, which in some cases, like in biological networks, are noisy and scarce. Second, GNNs typically require node features as input to train, which are not always available. In the absence of given node features, one needs to generate them, and often GNN algorithms use trivial node features such as the constant features or node degree features. These two differences give node embedding methods a unique place in node classification, apart from the GNN methods.

Here, we propose an improved version of *node2vec* that is more effective for weighted graphs by taking into account the edge weight connecting the previous vertex and the potential next vertex. The proposed method *node2vec+* is a natural extension of *node2vec*; when the input graph is unweighted, the resulting embeddings of *node2vec+* and *node2vec* are equivalent in expectation. Moreover, when the bias parameters are set to neutral, *node2vec+* recovers a first-order random walk, just as *node2vec* does. Finally, we demonstrate the superior performance of *node2vec+* through extensive benchmarking on both synthetic datasets and network-based gene classification datasets using various functional gene interaction networks. *Node2vec+* is implemented as part of `PecanPy` (Liu and Krishnan, 2021) and is available on GitHub: https://github.com/krishnanlab/PecanPy.

2 Materials and methods

We start by briefly reviewing the *node2vec* method. Then, we illustrate that *node2vec* is less effective for weighted graphs due to its inability to identify *out* edges. Finally, we present a natural extension of *node2vec* that resolves this issue.

2.1 *Node2vec* overview

In the setting of node embeddings, we are interested in finding a mapping $f : V \to \mathbb{R}^d$ that maps each node $v \in V$ to a d-dimensional vector so that the mutual proximity between pairs of nodes in the graph is preserved. In particular, a random walk-based approach aims to maximize the probability of reconstructing the neighborhoods for any node in the graph based on some sampling strategy S. Formally, given a graph $G = (V, E)$ (the analysis generalizes to directed and/or weighted graphs), we want to maximize the log probability of reconstructing the sampled neighborhood $\mathcal{N}_S(v)$ for each $v \in V$:

$$\max_f \sum_{v \in V} \log \mathbb{P}(\mathcal{N}_S(v) | f(v))\,. \tag{1}$$

Under the conditional independence assumption, and the parameterization of the probabilities as the softmax normalized inner products (Grover and Leskovec, 2016; Mikolov *et al.*, 2013b), the objective function above simplifies to:

$$\max_f \sum_{v \in V} \left(\sum_{v' \in \mathcal{N}_S(v)} \langle f(v'), f(v) \rangle - \log Z_v \right). \tag{2}$$

In practice, the partition function $Z_v = \sum_{v' \in V} \langle f(v), f(v') \rangle$ is approximated by negative sampling (Mikolov *et al.*, 2013a) to save computational time. Given any sampling strategy S, Equation (2) can find the corresponding embedding f, which is achieved in practice by feeding the random walks generated to the skipgram with negative sampling (Mikolov *et al.*, 2013b).

Node2vec devises a second-order random walk as the sampling strategy. Unlike a first-order random walk (Perozzi *et al.*, 2014), where the transition probability of moving to the next vertex v_n, denoted as $\mathbb{P}(v_n | v_c)$, depends only on the current vertex v_c, a second-order random walk also depends on the previous vertex v_p, with transition probability $\mathbb{P}(v_n | v_c, v_p)$. It does so by applying a bias factor$\alpha_{pq}(v_n, v_p)$ to the edge $(v_c, v_n) \in E$ that connects the current vertex and a potential next vertex. This bias factor is a function that depends on the relation between the previous vertex and the potential next vertex, and is parameterized by the *return* parameter p and the *in–out* parameter q. In this way, the random walk can be generated based on the following transition probabilities:

$$\mathbb{P}(v_n | v_c, v_p) = \begin{cases} \dfrac{\alpha_{pq}(v_n, v_p) w(v_c, v_n)}{\sum_{v \in \mathcal{N}(v_c)} \alpha_{pq}(v, v_p) w(v_c, v)} & \text{if } (v_c, v_n) \in E \\ 0 & \text{otherwise} \end{cases}, \tag{3}$$

where the bias factor is defined as:

$$\alpha_{pq}(v_n, v_p) = \begin{cases} \dfrac{1}{p} & \text{if } v_p = v_n \\ 1 & \text{if } v_p \neq v_n \text{ and } (v_n, v_p) \in E\,. \\ \dfrac{1}{q} & \text{if } v_p \neq v_n \text{ and } (v_n, v_p) \notin E \end{cases} \tag{4}$$

According to this bias factor, *node2vec* differentiates three types of edges: (i) the *return* edge, where the potential next vertex is the previous vertex (Fig. 1a); (ii) the *out* edge, where the potential next vertex is *not* connected to the previous vertex (Fig. 1b); and (iii) the *in* edge, where the potential next vertex is connected to the previous vertex (Fig. 1c). Note that the first-order (or unbiased) random walk can be seen as a special case of the second-order random walk where both the *return* parameter and the *in–out* parameter are set to neutral ($p = 1$, $q = 1$).

We now turn our attention to weighted networks, where the edge weights are not necessarily zeros or ones. Consider the case where v_n is connected to v_p, but with a small weight (Fig. 1d), i.e. $(v_n, v_p) \in E$ and $0 < w(v_n, v_p) \ll 1$. According to the definition of the bias factor, no matter how small $w(v_n, v_p)$ is, (v_c, v_n) would always be considered as an *in* edge. Since in this case v_n and v_p are barely connected, (v_c, v_n) should in fact be considered as an *out* edge. In the extreme case of a fully connected weighted graph, where $(v, v') \in E$ for all $v, v' \in V$, *node2vec* completely loses its ability to identify *out* edges.

Thus, *node2vec* is less effective for weighted networks due to its inability to identify potential *out* edges where the terminal vertex v_n is loosely connected to a previous vertex v_p. Next, we propose an extension of *node2vec* that resolves this issue, by taking into account of the edge weight $w(v_n, v_p)$ in the bias factor.

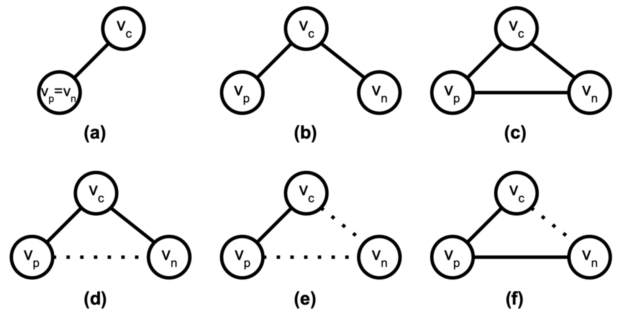

Fig. 1. Illustration of different settings of *return* and *in–out* edges. v_p, v_c and v_n indicate the previous, current, and next vertices. The solid and dotted lines represent edges with large and small edge weights, respectively. (**a–c**) *return*, *out* and *in* edges considered by *node2vec*. (**d–f**) Variations of (c) when taking into account of edge weights, where node2vec fail to distinguish from (c)

2.2 *Node2vec+*

The main idea of extending *node2vec* is to identify potential *out* edges $(v_c, v_n) \in E$ coming from v_p, where v_n is loosely connected to v_p. Intuitively, we can determine the 'looseness' of (v_c, v_n) based on some threshold edge value. However, given that the distribution of edge weights of any given node in the graph is not known a priori, it is hard to come up with a reasonable threshold value for all networks. Instead, we define the looseness of (v_c, v_n) based on the edge weight statistics for each node v.

$$\begin{aligned} \mu(v) &= \frac{\sum_{v' \in \mathcal{N}(v)} w(v, v')}{|\mathcal{N}(v)|} \\ \sigma(v) &= \sqrt{\frac{\sum_{v' \in \mathcal{N}(v)} (w(v, v') - \mu(v))^2}{|\mathcal{N}(v)|}} \, . \\ \tilde{w}_\gamma(v, u) &= \frac{w(v, u)}{\max\{\mu(v) + \gamma\sigma(v), \epsilon\}} \end{aligned} \tag{5}$$

Formally, we first define $\tilde{w}_\gamma(v, u)$, a normalized version of the edge weight $w(v, u)$, based on the mean $\mu(v)$ and the standard deviation $\sigma(v)$ of the edge weights connecting v, as in Equation (5). In practice, we clip the denominator of $\tilde{w}_\gamma(v, u)$ by a small number ϵ (1e−6 by default) to prevent divide by zero in some cases when γ is set to be negative. Then, we say $v \in V$ is γ *-loosely connected* (or simply *loosely connected* if $\gamma = 0$) to $u \in V$ if $\tilde{w}_\gamma(v, u) < 1$. Intuitively, we would like to treat an edge as being 'not connected' if it is 'small enough'. Finally, an edge (v, u) is γ *-loose* if v is γ-loosely connected to u, and otherwise it is γ *-tight*. Without loss of generality, we consider the case of $\gamma = 0$ in the subsequent sections to simplify the notion of *looseness*.

Based on the definition of looseness of edges, and assuming $v_p \neq v_n$, there are four types of (v_c, v_n) edges (see Fig. 1c–f). Following *node2vec*, we categorize these edge types into *in* and *out* edges. Furthermore, to prevent amplification of noisy connections, we added one more edge type called the *noisy* edge, which is always suppressed.

2.2.1 *Out* edge

As a direct generalization to *node2vec*, we consider (v_c, v_n) to be an *out* edge if (v_c, v_n) is tight and (v_n, v_p) is loose (Fig. 1b and d). The *in–out* parameter q then modifies the out edge to differentiate 'inward' and 'outward' nodes, and subsequently leads to Breadth First Search or Depth First Search like searching strategies (Grover and Leskovec, 2016). Unlike *node2vec*, however, we further parameterize the bias factor α based on $\tilde{w}_\gamma(v_n, v_p)$. Any choice of monotonic function should work, but we choose to use the linear interpolation in this study for simplicity and leave it as future work to explore more sophisticated interpolation functions such as the sigmoidal functions. Specifically, for an *out* edge (v_c, v_n), the bias factor is computed as $\alpha_{\gamma pq}(v_p, v_c, v_n) = \frac{1}{q} + (1 - \frac{1}{q})\tilde{w}_\gamma(v_n, v_p)$. Thus, the amount of modification to the *out* edge depends on the level of looseness of (v_n, v_p). When $w(v_n, v_p) = 0$, or equivalently $(v_n, v_p) \notin E$, the bias factor for (v_c, v_p) is $\frac{1}{q}$, same as that defined in *node2vec*.

2.2.2 *Noisy* edge

We consider (v_c, v_n) to be a *noisy* edge if both (v_c, v_n) and (v_n, v_p) are loose (Fig. 1e). Heuristically, the *noisy* edges are not very informative and thus should be suppressed regardless of the setting of q to prevent amplification of noise. Thus, the bias factor for a *noisy* edge is set to be $\min\left\{1, \frac{1}{q}\right\}$.

Notice that by introducing the noisy-edge term, we create discontinuity to the bias factor when $\tilde{w}_\gamma(v_n, v_p) > 1$ and $\tilde{w}_\gamma(v_c, v_n)$ switches from greater than one to less than one. We provide an alternative solution to *node2vec+* in the Supplementary material, which continuously extends the *out* edge term with the *noisy* edge term. However, we empirically show that the continuous version of *node2vec+* performs no better than *node2vec+*. Hence, in the main paper, we stick to the 'discontinuous' but simpler version of *node2vec+*.

2.2.3 *In* edge

Finally, we consider (v_c, v_n) to be an *in* edge if (v_n, v_p) is tight, regardless of $w(v_c, v_n)$ (Fig. 1c and f). The corresponding bias factor is set to neutral as in *node2vec*.

Combining the above, the bias factor for *node2vec+* is defined as follows:

$$\alpha_{\gamma pq}(v_p, v_c, v_n) = \begin{cases} \frac{1}{p} & \text{if } v_p = v_n \\ 1 & \text{if } \tilde{w}_\gamma(v_n, v_p) \geq 1 \\ \min\left\{1, \frac{1}{q}\right\} & \text{if } \tilde{w}_\gamma(v_n, v_p) < 1 \\ & \text{and } \tilde{w}_\gamma(v_c, v_n) < 1 \\ \frac{1}{q} + \left(1 - \frac{1}{q}\right)\tilde{w}_\gamma(v_n, v_p) & \text{if } \tilde{w}_\gamma(v_n, v_p) < 1 \\ & \text{and } \tilde{w}_\gamma(v_c, v_n) \geq 1 \end{cases} . \tag{6}$$

Note that the last two cases in Equation (6) include cases of $(v_n, v_p) \notin E$. Based on the biased random walk searching strategy using this bias factor, the embedding can be generated accordingly using (2). One can verify, by checking Equation (6), that this is indeed a natural extension of *node2vec* in the sense that

- For an unweighted graph, the *node2vec+* is equivalent to *node2vec*.
- When p and q are set to 1, *node2vec+* recovers a first-order random walk, same as *node2vec* does.

Finally, by design, *node2vec+* is able to identify potential *out* edges that would have been obliviated by *node2vec*.

3 Experiments

3.1 Synthetic datasets

We start by demonstrating the ability of *node2vec+* to identify potential *out* edges in weighted graphs using a barbell graph and the hierarchical cluster graphs. For simplicity, we fix $\gamma = 0$ for all experiments in this section.

3.1.1 Barbell graph

A barbell graph, denoted as B, is constructed by connecting two complete graphs of size 20 with a common bridge node (Fig. 2a). All edges in B are weighted 1. There are three types of nodes in B, (i) the bridge node; (ii) the peripheral nodes that connect the two modules with the bridge node; and (iii) the interior nodes of the two modules. By changing the *in–out* parameter q, *node2vec* could put the peripheral nodes closer to the bridge node or interior nodes in the embedding space.

When q is large, *node2vec* suppresses the *out* edges, e.g. an edge connecting a peripheral node to the bridge node, coming from an interior node. Consequently, the biased random walks are restricted to the network modules. In this case, the transition from the peripheral nodes to the bridge node becomes less likely compared to a first-order random walk, thus pushing the embeddings between the bridge node and the peripheral nodes away from each other. Conversely, when q is small, the transition between the peripheral nodes and the bridge node is encouraged. In this case, the embeddings of the bridge node and the peripheral nodes are pulled together. To see this, we run *node2vec* with fixed $p = 1$, and three different settings of $q = [1, 100, 0.01]$. Indeed, for $q = 100$, *node2vec* tightly clusters interior nodes and pushes the bridge node away from the peripheral nodes, and for $q = 0.01$, the peripheral nodes are pushed away from the interior nodes (Fig. 2b). Since *node2vec* and *node2vec+* are equivalent when the graph is unweighted (see Section 2), we omit the visualization of *node2vec+* embeddings for B in the main paper (see Supplementary material).

Next, we perturb the barbell graph by adding loose edges with edge weights of 0.1, making the graph fully connected. This perturbed barbell graph is denoted $\tilde{B}$. As expected, *node2vec* failed to make use of

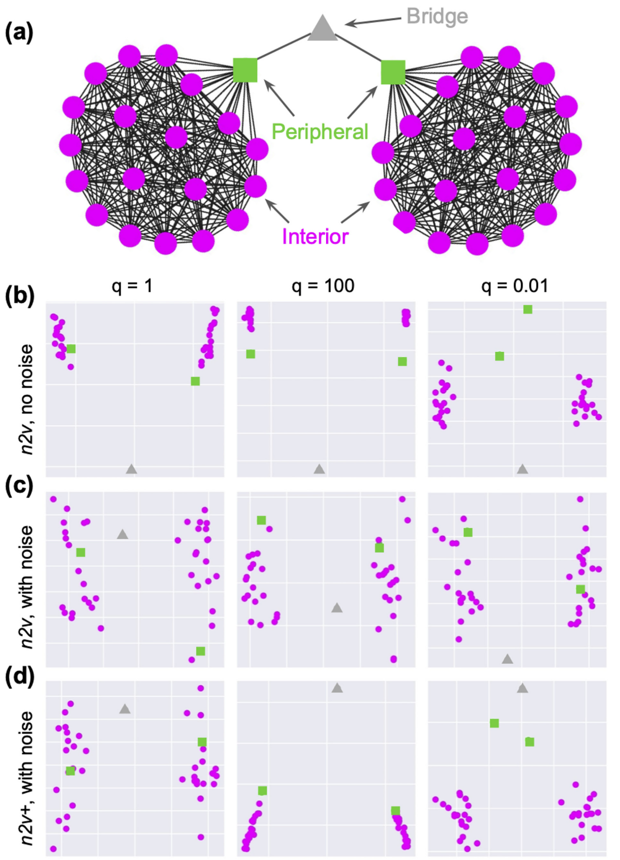

Fig. 2. Barbell graph. (**a**) Illustration of the barbell graph, and three different types of nodes indicated by the different marker styles. (**b**) Embedding of the barbell graph B using *node2vec*. (**c, d**) Embedding of the noisy barbell graph $\tilde{B}$ using *node2vec* and *node2vec+*, respectively. Each one of (b–d) contains three different settings of q: 1, 100 and 0.01

the q parameter (Fig. 2c), since none of the edges are identified as an *out* edge. On the other hand, *node2vec+* can pick up potential *out* edges and thus qualitatively recovers the desired outcome (Fig. 2d). Note that both *node2vec* and *node2vec+* have similar results for $\tilde{B}$ when $q = 1$. This confirms that *node2vec+* and *node2vec* are equivalent when p and q are set to neutral, corresponding to embedding with unbiased random walks. Finally, when using non-neutral settings of q, *node2vec+* is able to suppress some noisy edges, resulting in less scattered embeddings of the interior nodes (Fig. 2d).

3.1.2 Hierarchical CLUSTER graph

We use a modified version of the CLUSTER dataset (Dwivedi *et al.*, 2022) to further demonstrate the advantage of the *node2vec+* due to identifying potential *out* edges. Specifically, the hierarchical cluster graph K3L2 contains $L = 2$ levels (3 including the root level) of clusters, and each parent cluster is associated with $K = 3$ children clusters (Fig. 3a). There are 30 nodes in each cluster, resulting in a total of 390 nodes. To generate the hierarchical cluster graph, we first generate point clouds via a Gaussian process in a latent space so that the Euclidean distance between two points from two sibling clusters is about twice ($\sqrt{2}$ to be precise) the expected Euclidean distance from one of the two points to a point in the parent cluster, which is set to be 1. The noisiness of the clusters is controlled by the parameter σ, which is set to 0.1 by default. These data points are then turned into a fully connected weighted graph using a RBF kernel (see Supplementary material). We consider two different tasks (Fig. 3a), (i) *cluster classification*: identifying individual cluster identity of each node in the graph and (ii) *level classification*: identifying the level to which the clusters correspond to. We split the nodes into 10% training and 90% testing and use the multinomial logistic regression model with l2 regularization for prediction. The evaluation process, including the embedding generation, is repeated 10 times, and the final results are reported by Macro F1 scores.

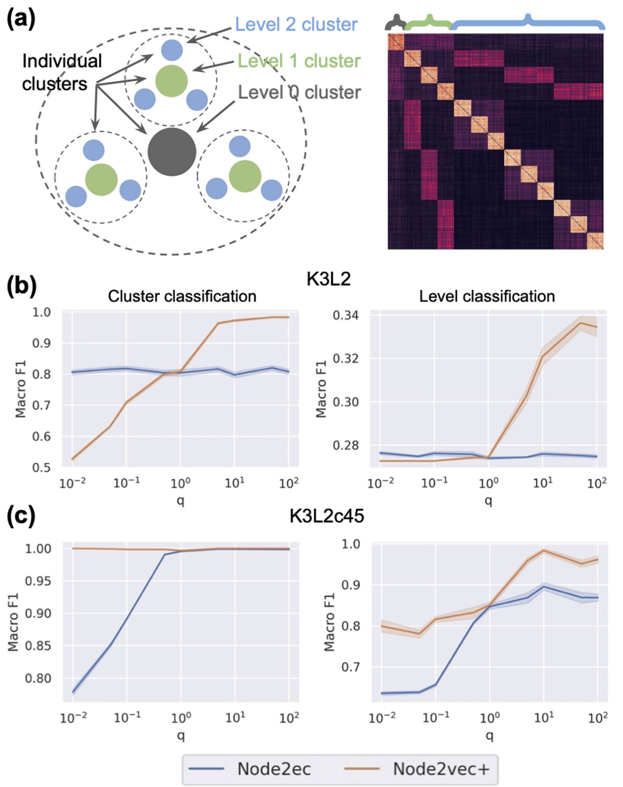

Fig. 3. Hierarchical CLUSTER graph classification task. (**a**) Illustrations of the K3L2 hierarchical clusters. Left: top-down view of the clusters. Right: adjacency matrix of K3L2; colored brackets indicate the corresponding cluster levels of the nodes. (**b**) Classification evaluation on K3L2. (**c**) Classification evaluation on K3L2c45

As shown in Figure 3b, the performance of *node2vec* is not affected by the q parameter because the graph is fully connected. Meanwhile, *node2vec+* achieves significantly better performance than *node2vec* for large q settings for both tasks, demonstrating the ability of *node2vec+* to identify potential *out* edges and use this information to perform localized biased random walks. Similar results are observed on a couple of different hierarchical cluster graphs K3L3, K5L1 and K5L2 (see Supplementary material).

On the other hand, one might suspect that the issue with the fully connected graph can be alleviated by sparsifying the graph based on certain edge weight thresholds. Such an approach is widely adopted as a post-processing step for constructing functional gene interaction networks. Here, we show that even after sparsifying the graph aggressively, *node2vec+* still outperforms *node2vec*. In particular, we sparsify the K3L2 graph using the edge weight threshold 0.45, which is the largest value that keeps the graph connected. We then perform the same evaluation analysis above on this sparsified graph K3L2c45. In this case, *node2vec* indeed performs significantly better than before the sparsification for both tasks. Nonetheless, *node2vec+* achieves even better performance, still out-competing node2vec (Fig. 3c).

Finally, we conduct a fine-grained evaluation analysis, showing that node2vec+ consistently outperforms node2vec under a wide range of conditions, including edge threshold, train-test ratio and noise level (see Supplementary material).

3.2 Real-world datasets

Our primary motivation for developing *node2vec+* stems from the fact that many functional gene interaction networks are dense and weighted. To systematically evaluate the ability of *node2vec+* to embed such biological networks, we consider various challenging gene classification tasks, including gene function and disease gene predictions. Furthermore, we devise experiments with previously benchmarked datasets BlogCatalog and Wikipedia (Grover and Leskovec, 2016) and confirm that *node2vec+* performs equal to or better than *node2vec*, depending on whether the network is weighted (see Supplementary material).

3.2.1 Datasets

Human functional gene interaction networks: We consider functional gene interaction networks, which is a broader class of gene interaction networks that are routinely used to capture gene functional relationships.

- **STRING** (Szklarczyk *et al.*, 2021) is an integrative gene interaction network that combines evidence of protein interactions from various sources, such as text-mining, high-throughput experiments, etc.
- **HumanBase-global** is a tissue-naive version of the HumanBase (Greene *et al.*, 2015) tissue-specific networks (previously known as GIANT), which are constructed by integrating hundreds of thousands of publicly available gene expression studies, protein–protein interactions and protein–DNA interactions via a Bayesian approach, calibrated against high-quality known functional gene interactions.
- **HumanBaseTop-global** is a sparsified version of HumanBase-global that eliminates all edges below the prior of 0.1.

Multi-label gene classification tasks: We follow the procedure detailed in Liu *et al.* (2020) to prepare the multi-label gene classification datasets. More specifically, we prepare two collections of gene classification tasks (each is called a gene set collection):

- **GOBP**: Gene function prediction tasks derived from the Biological Processes gene sets from The Gene Ontology Consortium (2018).
- **DisGeNET**: Disease gene prediction tasks derived from the disease gene sets from the DisGeNET database (Piñero *et al.*, 2016).

After filtering and cleaning up the raw gene set collections, we end up with ∼45 functional gene prediction tasks and ∼100 disease gene prediction tasks (Table 1). These gene classification tasks are challenging primarily due to the scarcity of the labeled examples, with on average 100 and 200 positive examples per task for GOBP and DisGeNET, respectively, relative to the (order of) tens of thousands of nodes in the networks.

We split the genes into 60% training, 20% validation and 20% testing according to the level at which they have been studied in the literature (based on the number of PubMed publications associated with each gene). In particular, the top 60% most well-studied genes are used for training; the 20% least-studied genes are used for testing, and the rest are used for validation. For GNNs, we report the test scores at the epoch where the best validation score is achieved.

3.2.2 Baseline methods

We exclude several popular node embedding methods, such as DeepWalk (Perozzi *et al.*, 2014), LINE (Tang *et al.*, 2015) and GraRep (Cao *et al.*, 2015), from our main analysis, as it has been shown previously in various contexts (Ata *et al.*, 2021; Grover and Leskovec, 2016; Yue *et al.*, 2019) that *node2vec* is superior.

On the other hand, we include two popular GNNs, GCN (Kipf and Welling, 2016) and GraphSAGE (Hamilton *et al.*, 2017) in our comparison. Both methods have shown exceptional performance on many node classification tasks, but their performance on the gene classification tasks here still needs to be better studied. For GraphSAGE, we consider the full-batch training strategy with mean pooling aggregation following the Open Graph Benchmark (Hu *et al.*, 2021).

Table 1. Number of tasks (i.e. gene sets or node classes) for each combination of network and gene set collection

	GOBP	DisGeNET
HumanBase	46 (98.3)	103 (225.9)
STRING	41 (100.0)	97 (221.5)

Note: The number in the parenthesis is the average number of positive examples.

3.2.3 Experiment setup

Evaluation metric: Following (Liu *et al.*, 2020), we use the $\log_2 \frac{\text{auPRC}}{\text{prior}}$ as our evaluation metric, which represents the $\log_2$ fold change of the average precision compared to the prior. This metric is more suitable than other commonly used metrics like AUROC as it corrects for the class imbalance issue that is prevalent in the gene classification tasks here, as well as emphasizes the correctness of top predictions.

Tuning embeddings parameters: For *node2vec* and node2vec+, we train a one versus rest logistic regression with l2 regularization using the embeddings learned. The parameters for embeddings including dimension, window-size, walk-length and number of walks per node are set to 128, 10, 80 and 10, respectively, by default. We tune the hyperparameters for *node2vec* (p, q) and for *node2vec+* (p, q, γ) via grid search using the validation sets. To keep the grid search budget comparable, we search p and q over $\{0.01, 0.05, 0.1, 0.5, 1, 5, 10, 50, 100\}^2$ for *node2vec* $(n = 81)$; we search p and q over $\{0.01, 0.1, 1, 10, 100\}^2$, together with $\gamma \in \{0, 1, 2\}$ for *node2vec+* $(n = 75)$.

Tuning GNN parameters: For both GNNs, we train one model for each combination of a network and a gene set collection in an end-to-end fashion. The architectures are fixed to five hidden layers with a hidden dimension of 128. Since the gene interaction networks here do not come with node features, we use the constant feature for GCN and the degree feature for GraphSAGE, respectively. We use the Adam optimizer (Kingma and Ba, 2014) to train the GNNs with 100 000 max number of epochs. The learning rates are tuned via grid search from 10^{-5} to 10^{-1} based on the validation performance. The optimal learning rates that result in a decent convergence rate without diverging are 0.01 and 0.0005 for GCN and GraphSAGE, respectively (see Supplementary material).

3.2.4 Experimental results

Tuning γ significantly improves performance for dense graph: The γ parameter in *node2vec+* (see Section 2.2) controls the threshold of distinguishing *in* edges and *out* edges. A small or negative valued γ considers most non-zero edges as *out* edges. Conversely, a large valued γ identifies less *out* edges. When the input graph is noisy and dense, assigning a larger γ (e.g. 1) can act as a stronger denoiser to suppress spurious *out* edges. Figure 4 compares the gene classification test performance between $\gamma = 0$ and $\gamma = \{1, 2\}$ with optimally tuned p, q using the HumanBase-global network. Higher testing scores are achieved by larger γ settings, illustrating that, to properly 'denoise' the fully connected weighted graph HumanBase-global, we need to increase the noisy edge thresholds. On the contrary, the difference in performance due to the γ settings is less pronounced for sparse networks like HumanBaseTop-global and STRING (see Supplementary material).

GNN methods performs worse than *node2vec(+)*: In all settings, *node2vec+* significantly outperforms both GNN methods (Fig. 5). Notably, for the STRING network, both *node2vec* and *node2vec+* outperform the two GNNs by a large margin. The sub-optimal GNN performance here illustrates that, despite being powerful neural network architectures that can leverage the graph structures, GNNs alone cannot learn effectively given a limited number of labeled examples. On the contrary, the embedding processes of *node2vec(+)* are task agnostic and can be carried out effectively without labels. These results indicate that gene classification tasks based on gene interaction network are more effectively solved by unsupervised shallow embedding methods than GNNs.***nod2vec+* matches or outperforms *node2vec*:** *node2vec+* significantly outperforms *node2vec* [Wilcoxon paired test (Wilcoxon, 1945) $P < 0.05$] except for the DisGeNET tasks using HumanBaseTop-global and STRING networks, in which cases the two methods perform equally (Fig. 5). The performance differences are especially pronounced when using the fully connected and noisy HumanBase-global network, demonstrating *node2vec+*'s ability to learn robust node representations in the presence of noise. Nevertheless, when the network is less dense (e.g. HumanBaseTop-global), *node2vec+* is still able to perform at least as well as *node2vec*, indicating that *node2vec+* is overall a good replacement of *node2vec*.

3.2.5 Tissue-specific functional gene classification

A key feature of functional gene interaction networks constructed using gene expression data is capturing biological context specificity, such as tissue-specificity provided by the HumanBase networks. Thus, we further demonstrate the use case of *node2vec+* using tissue-specific functional gene classification tasks derived from Zitnik and Leskovec (2017). After processing, there are 25 tissue-specific functional gene classification tasks, with 12 different tissues found in the HumanBase database. We follow a similar experimental setup as above, and for each tissue-specific functional gene classification task, we report the followings: (i) *matched*: the prediction performance using the corresponding tissue-specific network; (ii) *other*: the average prediction performance using tissue-specific networks other than the corresponding tissue; (iii) *global*: the prediction performance using the tissue-naive network.

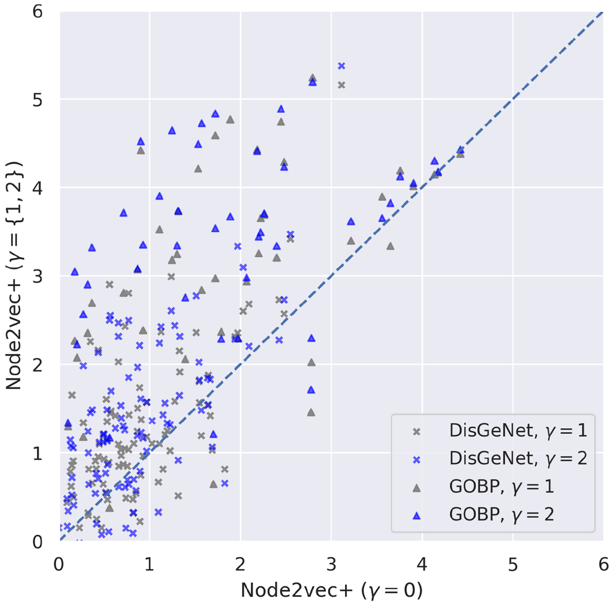

Fig. 4. Comparison of different γ settings in *node2vec+* using HumanBase-global. Each dot represents the testing performance ($\log_2 \frac{\text{auPRC}}{\text{prior}}$) of a specific gene set, with optimally tuned p and q settings

Figure 6 shows that *node2vec+* outperforms *node2vec* in most scenarios, especially when using the full HumanBase networks. In particular, *node2vec+*, using the *matched* tissue-specific full networks for the given functional gene classification tasks, results in significantly better performance than using *other* (unrelated) tissue-specific networks, as well as the *global* (tissue-naive) network. On the contrary, *node2vec* cannot fully utilize the tissue-specific networks, as indicated by the lack of difference in performance between *matched* and *global* networks.

We observe similar results using another collection of tissue-specific co-expression networks, GTExCoExp, that are generated using a benchmarked co-expression network generation workflow by Johnson and Krishnan (2022) (see Supplementary material).

4 Discussion and conclusion

In this article, we proposed *node2vec+* that improves upon the second-order random walk in *node2vec* for weighted graphs by considering edge weights. Consequently, the corresponding node embeddings are improved whenever the *in–out* walks positively influence the task (meaning that the optimal q setting is not 1).

We showed that *node2vec+* better identifies potential out edges on weighted graphs than *node2vec* using two synthetic datasets, including the barbell graph and the hierarchical cluster graphs. Furthermore, evaluations on various challenging gene classification tasks demonstrated that embedding methods like *node2vec(+)* are superior to GNNs. GNNs learn how to orient the nodes in a low-dimensional space to maximize the separation between nodes of different classes in an end-to-end fashion. The suboptimal GNN performance here highlights their need for a much larger labeled training dataset to fully exploit the expressive power of their architectures. Unfortunately, many real-world biological applications, such as the function or disease gene classification problems here, still lack large amounts of labeled data. For these applications, an unsupervised approach like *node2vec(+)* may be more suitable as it arranges the latent space purely based on the underlying graph

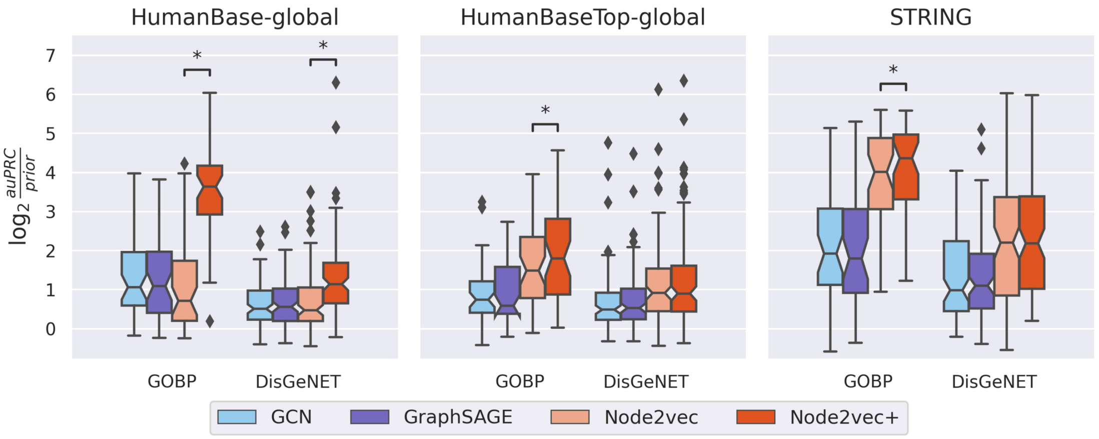

Fig. 5. Gene classification tasks using protein–protein interaction networks. Each panel corresponds to a specific protein–protein interaction network (HumanBase-global, HumanBaseTop-global and STRING). Each point in a boxplot represents the final test score for a specific task (gene set) in the gene set collection (GOBP or DisGeNET). Starred (*) pairs indicate that the performance between *node2vec* and *node2vec+* are significantly different (Wilcoxon $P < 0.05$)

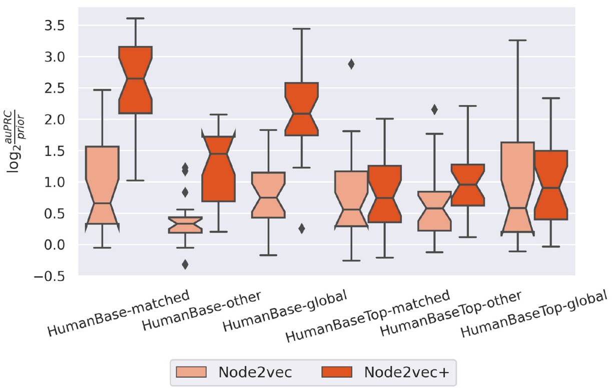

Fig. 6. Tissue-specific functional gene classification performance comparison between node2vec and node2vec+ using HumanBase and HumanBaseTop tissue-specific networks

structure, after which a less data-hungry model, such as logistic regression, can be applied to perform the classifications.

Dense weighted graphs are common in biology, directly based on the experiment [e.g. genetic interactions (Costanzo *et al.*, 2016)], by construction [e.g. co-expression (Zhang and Horvath, 2005)] or by integrating multiple network datasets sources [Greene *et al.*, 2015; Szklarczyk *et al.*, 2021]. Network embedding has recently found applications in studying co-expression networks, e.g. in the context of evolutionary and cross-species network alignment (Ovens *et al.*, 2021a,b), cancer prognostic gene identification (Choi *et al.*, 2018) and gene functional interaction prediction (Du *et al.*, 2019). These applications, especially the ones that leverage dense weighted graphs, are likely to benefit from using *node2vec+*.

Sparsification using a hard threshold is a common technique for dealing with fully connected weighted graphs like co-expression (Du *et al.*, 2019; Zhang and Horvath, 2005). However, finding the optimal cut threshold could be quite challenging [usually relying on heuristics (Ovens *et al.*, 2021a)], and such thresholding may change the graph significantly in terms of its spectrum (Spielman and Teng, 2010). *Node2vec+*, on the other hand, can be seen as a soft thresholding approach that suppresses transitions over *noisy* edges.

Overall, *node2vec+* is a natural extension of *node2vec* for weighted graphs and has several desirable properties. With its general procedure for biased random walks, *node2vec+* can be easily adapted into other methods such as KG2Vec (Wang *et al.*, 2021b) and HetNode2vec (Valentini *et al.*, 2021) that are built on top of *node2vec*. *Node2vec+* is available as an open-source software as part of the PecanPy package: https://github.com/krishnanlab/PecanPy.

Funding

This work was supported by the US National Institutes of Health (NIH) [R35 GM128765 to A.K.] and [R01 GM135929 to M.H.]; It was also supported by the National Science Foundation (NSF) CAREER [1845856 to M.H.].

Conflict of Interest: none declared.

References

Ata,S.K. *et al.* (2018) Integrating node embeddings and biological annotations for genes to predict disease-gene associations. *BMC Syst. Biol.*, **12**, 138.

Ata,S.K. *et al.* (2021) Recent advances in network-based methods for disease gene prediction. *Brief. Bioinform.*, **22**, bbaa303.

Bronstein,M.M. *et al.* (2021) Geometric deep learning: grids, groups, graphs, geodesics, and gauges. arXiv, arXiv:2104.13478 [cs, stat], preprint: not peer reviewed.

Cao,S. *et al.* (2015) GraRep: learning graph representations with global structural information. In: Proceedings of the 24th ACM International on Conference on Information and Knowledge Management, CIKM '15, pp. 891–900. Association for Computing Machinery, New York, NY, USA.

Chattopadhyay,S. and Ganguly,D. (2020) Community structure aware embedding of nodes in a network. arXiv, arXiv:2006.15313 [physics], preprint: not peer reviewed.

Choi,J. *et al.* (2018) G2vec: distributed gene representations for identification of cancer prognostic genes. *Nat. Sci. Rep*.

Costanzo,M. *et al.* (2016) A global genetic interaction network maps a wiring diagram of cellular function. *Science*, **353**, aaf1420.

Cui,P. *et al.* (2018) A survey on network embedding. *IEEE Trans. Knowl. Data Eng.*, **31**(5), 833–852.

Davison,A. and Austern,M. (2021) Asymptotics of network embeddings learned via subsampling. arXiv, arXiv:2107.02363 [cs, math, stat], preprint: not peer reviewed.

Du,J. *et al.* (2019) Gene2vec: distributed representation of genes based on co-expression. *BMC Genomics*, **20**.

Dwivedi,V.P. *et al.* (2022) Benchmarking graph neural networks. *JMLR*, **24**(43), 1–48.

Greene,C.S. *et al.* (2015) Understanding multicellular function and disease with human tissue-specific networks. *Nat. Genet.*, **47**, 569–576.

Grohe,M. (2020) word2vec, node2vec, graph2vec, X2vec: towards a theory of vector embeddings of structured data. In: *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS*'20, pp.1–16. AMC, New York, NY, USA; Portland, Oregon, USA.

Grover,A. and Leskovec,J. (2016) Node2Vec: scalable feature learning for networks. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '16*, pp. 855–864. ACM, New York, NY, USA; San Francisco, California, USA.

Hacker,C. (2021) k-simplex2vec: a simplicial extension of node2vec. arXiv, arXiv:2010.05636 [cs, math], preprint: not peer reviewed.

Hamilton,W.L. *et al.* (2017) Inductive representation learning on large graphs. In: *31st Conference on Neural Information Processing Systems, Long Beach, CA, USA*.

Hu,F. *et al.* (2020) Community detection in complex networks using Node2vec with spectral clustering. *Physica A*, **545**, 123633.

Hu,W. *et al.* (2021) Open graph benchmark: datasets for machine learning on graphs. In: *34th Conference on Neural Information Processing Systems, Vancouver, Canada*.

Johnson,K.A. and Krishnan,A. (2022) Robust normalization and transformation techniques for constructing gene coexpression networks from RNA-seq data. *Genome Biol.*, **23**, 1–26.

Kingma,D.P. and Ba,J. (2014) Adam: a method for stochastic optimization. In: *3rd International Conference on Learning Representations, San Diego, CA, USA*.

Kipf,T.N. and Welling,M. (2016) Semi-supervised classification with graph convolutional networks. In: *5th International Conference on Learning Representations, Toulon, France*.

Krishnan,A. *et al.* (2016) Genome-wide prediction and functional characterization of the genetic basis of autism spectrum disorder. *Nat. Neurosci.*, **19**, 1454–1462.

Liu,R. and Krishnan,A. (2021) PecanPy: a fast, efficient and parallelized python implementation of node2vec. *Bioinformatics*, **37**, 3377–3379.

Liu,R. *et al.* (2020) Supervised learning is an accurate method for network-based gene classification. *Bioinformatics*, **36**, 3457–3465.

Mikolov,T. *et al.* (2013a) Distributed representations of words and phrases and their compositionality. In: *Proceedings of the 26th International Conference on Neural Information Processing Systems, NIPS'13; Lake Tahoe, Nevada, USA*.

Mikolov,T. *et al.* (2013b) Efficient estimation of word representations in vector space. In: *1st International Conference on Learning Representations; Scottsdale, Arizona, USA*.

Nelson,W. *et al.* (2019) To Embed or not: network embedding as a paradigm in computational biology. *Front. Genet.*, **10**, 381.

Ovens,K. *et al.* (2021a) Comparative analyses of gene co-expression networks: implementations and applications in the study of evolution. *Front. Genet.*, **12**, 695399.

Ovens,K. *et al.* (2021b) Juxtapose: a gene-embedding approach for comparing co-expression networks. *BMC Bioinformatics*, **22**, 125.

Peng,J. *et al.* (2019) Predicting parkinson's disease genes based on node2vec and autoencoder. *Front. Genet.*, **10**, 226.

Perozzi,B. *et al.* (2014) DeepWalk: online learning of social representations. *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '14*, pp. 701–710. arXiv: 1403.6652.

Piñero,J. *et al.* (2016) DisGeNET: a comprehensive platform integrating information on human disease-associated genes and variants. *Nucleic Acids Res.*, **45**, D833–D839.

Qiu,J. *et al.* (2018) Network embedding as matrix factorization: unifying DeepWalk, LINE, PTE, and node2vec. In: Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining, WSDM '18, pp. 459–467. Association for Computing Machinery, New York, NY, USA.

Spielman,D.A. and Teng,S.-H. (2010) Spectral sparsification of graphs. *SIAM Journal on Computing*, **40**(4), 981–1025.

Szklarczyk,D. *et al.* (2021) The STRING database in 2021: customizable protein–protein networks, and functional characterization of user-uploaded gene/measurement sets. *Nucleic Acids Res.*, **49**, D605–D612.

Tang,J. *et al.* (2015) LINE: large-scale information network embedding. In: *Proceedings of the 24th International Conference on World Wide Web*, WWW' 15, pp. 1067–1077. Republic and Canton of Geneva, CHE. International World Wide Web Conferences Steering Committee.; Florence, Italy

The Gene Ontology Consortium. (2018) The Gene Ontology Resource: 20 years and still GOing strong. *Nucleic Acids Res.*, **47**, D330–D338.

Valentini,G. *et al.* (2021) Het-node2vec: second order random walk sampling for heterogeneous multigraphs embedding. arXiv, arXiv:2101.01425 [physics], preprint: not peer reviewed.

Wang,N. *et al.* (2021a) Essential protein prediction based on node2vec and XGBoost. *J. Comput. Biol.*, **28**, 687–700.

Wang,Y. *et al.* (2021b) KG2Vec: a node2vec-based vectorization model for knowledge graph. *PLoS One*, **16**, e0248552.

Wilcoxon,F. (1945) Individual comparisons by ranking methods. *Biometrics Bull.*, **1**, 80–83.

Wu,Z. *et al.* (2021) A comprehensive survey on graph neural networks. *IEEE Trans. Neural Netw. Learn. Syst.*, **32**, 4–24.

Yue,X. *et al.* (2019) Graph embedding on biomedical networks: methods, applications, and evaluations. *Bioinformatics*, **36**(4), 1241–1251.

Zeng,M. *et al.* (2021) A deep learning framework for identifying essential proteins by integrating multiple types of biological information. *IEEE/ACM Trans. Comput. Biol. Bioinform.*, **18**, 296–305.

Zhang,B. and Horvath,S. (2005) A general framework for weighted gene co-expression network analysis. *Stat. Appl. Genet. Mol. Biol.*, **4**.

Zhang,X.-M. *et al.* (2021) Graph neural networks and their current applications in bioinformatics. *Front. Genet.*, **12**, 1073.

Zitnik,M. and Leskovec,J. (2017) Predicting multicellular function through multi-layer tissue networks. *Bioinformatics*, **33**, i190–i198.