REPEL - Random Embedding Perturbation for Enhanced Learning of Protein Function

Di Zhou[1], Lenore J. Cowen[1,2], Kaiyi Wu[2], Xiaozhe Hu[2], Donna K. Slonim[1]

[1] *Department of Computer Science, Tufts University, Medford, MA 02155, USA*
[2] *Department of Mathematics, Tufts University, Medford, MA 02155, USA*
Email: di.zhou@tufts.edu, cowen@cs.tufts.edu, kaiyiwu0124@gmail.com, xiaozhe.hu@tufts.edu, slonim@cs.tufts.edu

Protein function prediction from multiplex protein-protein association networks is a crucial approach to extending functional annotation. Current methods use embeddings of the heterogeneous network data that aim to place related proteins near each other in embedding space. However, such embeddings suffer from spurious protein proximity as well, reducing function prediction accuracy. Because heterogeneous input networks often have very different structures, it is hard to confidently declare proteins to be dissimilar using the network structure or the resulting embeddings. Here we address this problem with REPEL, a function prediction tool using a random graph augmentation method that applies a uniform weak force to push nodes apart. We assess this method on simulated networks with planted overlapping communities, as well as on real multiplex yeast and *E.coli* protein association networks. Surprisingly, we find that this method consistently improves protein function prediction over competing methods Mashup, deepNF, and BIONIC. The random repelling nature of the augmented graphs has a denoising effect on the learning process, distancing node pairs with spurious proximity while preserving true functional connections, thus increasing robustness. This graph augmentation principle may generalize to denoising and improving robustness in other graph-based learning algorithms.

Keywords: Protein Function Prediction; Protein-Protein Interaction, Denoising, Noise Injection for Regularization, Embedding, Multiplex Networks, Robustness

Code and data availability: http://bcb.cs.tufts.edu/REPEL
https://github.com/TuftsBCB/REPEL.git
Supplementary material: https://bcb.cs.tufts.edu/REPEL/repelsupplement.pdf

1. Introduction

Determining the functional roles of genes and proteins is crucial for better understanding biological processes and developing focused medical interventions. Given the cost and complexity of experimental assays of gene function in different cellular contexts, computational prediction of function is an important problem. Data from large-scale protein-protein interaction (PPI) networks has often been used for such purposes.[1]

Most early results in this area relied on the principle of "guilt by association," which assumes that protein functions are likely similar to those of other proteins with which they

interact.[2] Subsequent efforts grouped functions within highly connected graph communities or minimized interactions linking proteins with different functional annotations.[3–5] Improvements in predictive accuracy came from new measures of graph proximity that reflected random walk[6,7] or diffusion distances,[8] characterizing information flow in the local graph structures.

In 2016, Cho et al.[9] introduced the Mashup algorithm, integrating information across multiple biological networks reflecting heterogeneous sources of protein-protein interactions. Mashup consists of three main steps. First, Random Walk with Restart (RWR)[7] is run separately for each node for each network. Next, a low-dimensional embedding is constructed to minimize the distance to all the individual network-specific vectors. Finally, these global low-dimensional feature vectors are passed to classifiers such as k-nearest neighbors[8] or support vector machines[10] to predict functional labels. The aim of this low dimensional embedding is to induce proximity between nodes linked by network edges.

In this paper, we characterize the effects of extending the Mashup embedding paradigm by adding *negative* weight edges that push nodes further apart. Specifically, we augment the network with new artificial nodes and connect each original protein node uniformly, with negative weight "repelling" edges, to a *random* artificial node. The result of this random augmentation is to stretch out the integrated PPI network. Original protein nodes are still pulled closer together by the source network edges. However, the the new augmented nodes simultaneously are uniformly repelling, pushing everything further apart.

Surprisingly, we find that the resulting stretched network is far better for functional label prediction. We believe this improvement shows that the repelling process from the augmentation, though not strong enough to separate well-connected communities of truly related nodes, has sufficient strength to pull apart nodes placed in proximity just by chance. Accordingly, this repelling approach effectively regularizes the embedding, improving its utility for protein function prediction. We believe this represents another example of random perturbations effectively denoising and improving machine learning algorithms.[11–13]

2. Algorithms

2.1. *Preliminaries and Notation*

Let $G^i = (V, E^i)$ with $i \in \{1, \cdots, M\}$ be a multiplex network which shares a set of nodes V, with $|V| = n$. Each individual network i has its own set of edges E^i corresponding to different types of pairwise associations. The weighted adjacency matrix $A^i \in [0, 1]^{n \times n}$ contains the confidence scores of the association edge of type i between the node pairs being correct.

Let $L^{l \times n}$ denote a function annotation matrix with l functional labels and n proteins. We let $L_{ij} = 1$ if protein i has label j, otherwise $L_{ij} = 0$.

2.2. *The REPEL Algorithm*

REPEL de-noises the learned protein embeddings by augmenting the network before the embedding and function prediction phases. Figure 1 gives an overview of the procedure; details of the algorithm follow.

Augmentation. For the proteins with known labels, we randomly split them into 5 folds, and use 4 folds as function voting proteins and 1 fold as testing proteins. We further randomly split

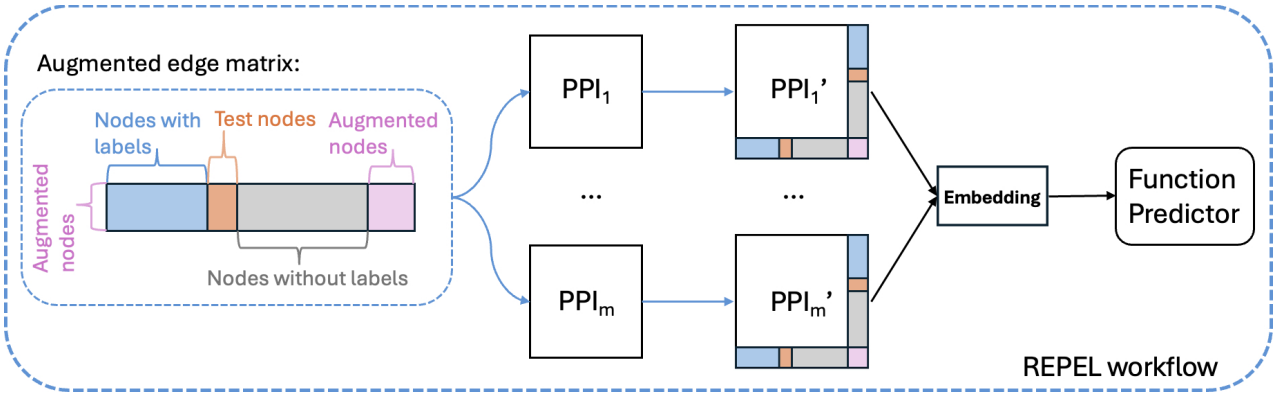

Fig. 1. REPEL Workflow. The blue arrows illustrate how we augment auxiliary nodes and edges. The blue rectangle area represents the augmented edge matrices for proteins with functional labels. The orange and gray rectangles represent the all edge 0 matrix between test proteins (orange) and unlabeled proteins (gray). The pink square illustrates the repelling negative weight matrix. Using these augmented networks, REPEL follows the feature representation learning procedures of Mashup to learn a protein embedding and predict functions.

the function voting proteins into x *equal sized* non-overlapping groups, to achieve a uniformly "repelling" power. For each group, we create one auxiliary node, and connect each protein that belongs to that group to the auxiliary node with an edge with weight 1. We augment all networks with the same set of edges between the auxiliary nodes and proteins. Then we add a "repel" edge with weight -1 between each pair of the auxiliary nodes to push them apart. We denote the augmented graph as $\overline{G}^i$. The resulting symmetrical weighted signed adjacency matrix $\overline{A}^i \in [-1,1]^{(n+x)\times(n+x)}$ is defined as:

$$\overline{A}^i_{uv} = \begin{cases} A^i_{uv} & \text{if } u,v \in V, \\ 1 & \text{if } edge(u,v) \in E_{aug}, \\ -1 & \text{if } u,v \in V_{aug}, u \neq v, \\ 0 & \text{otherwise,} \end{cases}$$

where V_{aug} is the augmented node set with $|V_{aug}| = x$, and E_{aug} is the augmented edge set. In addition, due to how we augment the graph, we ensure

$$\sum_v \overline{A}^i_{uv} = c \quad \text{for all } u \in V_{aug} \text{ and } v \in V, \ \sum_u \overline{A}^i_{uv} = 1 \quad \text{for all } u \in V_{aug} \text{ and } v \in V.$$

The constant c is the closest integer to the number of voting proteins divided by x, the total number of groups. Each group contains c proteins, except the last one which may contain more or fewer to account for the remainder.

Diffusion. On each augmented graph $\overline{G}^i$, a diffusion process using RWR on the signed graph is run which creates a matrix representation denoted as $\overline{\boldsymbol{W}}^i$ of the augmented network. Since we augment the links with negative weights to the graph, the original RWR won't work, because it requires positive transition probabilities. Thus, we need to generalize the diffusion state matrix $\boldsymbol{W}^i$ to signed graphs.

A natural generalization to a signed graph is obtained by replacing the weighted degree

matrix $\boldsymbol{D}$ in the normal RWR diffusion state equation with the signed weighted degree matrix $\overline{\boldsymbol{D}}$. Letting α represent the restart rate, the diffusion state matrix for signed graphs is defined:

$$\overline{\boldsymbol{W}} = \alpha\overline{\boldsymbol{D}}\left(\overline{\boldsymbol{D}} - (1-\alpha)\overline{\boldsymbol{A}}\right)^{-1}$$

Embedding. A shared low-dimensional embedding $\boldsymbol{X} \in \mathbb{R}^{d\times n}$ is created using the matrix representation $\overline{\boldsymbol{W}}^i$ where $i \in \{1, \cdots, M\}$. This is achieved via a singular value decomposition (SVD). Ultimately this results in a d-dimensional vector representation of every node in the dataset. We adopted Mashup's approach to combine these matrix representations to obtain a low-dimensional embedding.

Once each set of M networks has been represented by its RWR matrix $\overline{\boldsymbol{W}}^i$, the low-dimensional embedding $\boldsymbol{X}$ is formed by the scaled largest d left singular vectors of the concatenated matrix $\log(\boldsymbol{S})$ where $\boldsymbol{S} = [(\overline{\boldsymbol{W}}^1)^T, \cdots, (\overline{\boldsymbol{W}}^M)^T]^T$ and $\log(\cdot)$ denotes the element-wise logarithm. As suggested in the Mashup paper,[9] to optimize the implementation, the SVD-based embedding can be computed by the eigenvalue decomposition of the $n \times n$ matrix $\boldsymbol{R}$,

$$\boldsymbol{R} = \sum_{i=1}^{M} \log(\overline{\boldsymbol{W}}^i)^T \log(\overline{\boldsymbol{W}}^i).$$

A small constant is added to $\overline{\boldsymbol{W}}^i$ to avoid taking the logarithm of zero. Taking the top d eigenvalues $\boldsymbol{\Lambda} = \mathrm{diag}(\lambda_1, \cdots, \lambda_d)$ and eigenvectors $\boldsymbol{U} = (\boldsymbol{u}_1, \cdots, \boldsymbol{u}_d) \in \mathbb{R}^{n\times d}$, the low-dimensional embedding is $\boldsymbol{X} = \boldsymbol{\Lambda}^{1/4}\boldsymbol{U}^T$.

Learning. Once the embedding $\boldsymbol{X}$ for every node in the dataset is learned, any classical function prediction method can be applied. We adopted a weighted majority voting classifier by k-nearest neighbors (kNN) (similar to that used by Cao, *et al.*[8]) to predict functional labels using the embedding and the available annotations.

2.3. *Prior Work and Review of Comparator Algorithms*

In the experiments in this paper, we compare REPEL to Mashup; DeepNF, which uses a multimodal deep autoencoder; and BIONIC, which integrates biological networks using a Graph Convolutional Network based model. Each of these is reviewed briefly below.

2.3.1. *Mashup*

The original Mashup procedure consists of three core steps. First a **Diffusion** step that creates a RWR matrix representation $\boldsymbol{W}^i \in \mathbb{R}^{n\times n}$ for each input of the network G^i, followed by an **Embedding** step where a shared embedding is created using the matrix representations generated in the diffusion step. This is achieved via singular value decomposition or dictionary learning techniques. Ultimately, this gives a d-dimensional vector representation of every node in the dataset. Finally the **Learning** step: once every node in the dataset is represented by a vector, the existing function prediction methods can be applied using the embedding and available annotations. The original Mashup used a support vector machine (SVM) for the final learning step. However, it also carefully filtered the set of GO labels that it considered to discard GO labels whose set of labeled genes overlapped (with a Jaccard threshold greater than 0.1), advantaging methods such as SVM that search for a separating hyperplane. Since

we are mainly interested in the functional enrichment of the embedding itself and in the more realistic setting where GO labels are allowed to overlap, we instead use kNN as the classifier to compare the different embedding methods in this paper.

2.3.2. *deepNF*

DeepNF[14] tries to learn a useful low-dimensional embedding of proteins with a multimodal deep autoencoder (MDA)[15] that preserves non-linear network structure across multiple networks characterized by diverse connectivity patterns.[14] It is shown to preserve the non-linear network structure with its deep neural network (DNN) architecture in an efficient and scalable manner, and at the same time it denoises the links in the networks.

It first utilizes RWR to create its matrix representation $\boldsymbol{W}^i \in \mathbb{R}^{n \times n}$ for each input network G^i. Then each RWR matrix is converted into a Positive Pointwise Mutual Information (PPMI) matrix $\boldsymbol{Q}^i \in \mathbb{R}^{n \times n}$ that captures the structural information of the network. Then an MDA is trained that takes the PPMI matrices as input. A canonical d-dimensional feature representation across the networks is extracted from the middle layer of the MDA, which will be fed into the downstream function predictors.

2.3.3. *BIONIC*

BIONIC[16] is a Graph Convolutional Network[17] based deep learning model to learn a unified, combined gene feature representation using multiple biological networks. It reflects the underlying network topologies while capturing the gene functional information.

BIONIC first encodes each biological networks through a 3-layer GAT[18] model, then aggregates the learned features through a weighted, stochastically masked summation. It has both unsupervised and semi-supervised learning modes depending on availability of the gene functional information. For the unsupervised learning mode, the model will be trained with network reconstruction loss as the objective. For the semi-supervised learning mode, an additional label classification loss term, introduced by incorporating the known label information, will be added along with the reconstruction loss as the training objective. Here, to match the learning modes of REPEL, Mashup, and deepNF, and to make as fair a comparison as possible, we adopted the unsupervised approach and retrieved the learned gene embeddings using the recommended parameters.

2.3.4. *Other methods*

In addition to the methods discussed here and in the Introduction, there have been other efforts to predict function from heterogeneous networks using modern neural network approaches. However, because most also require other input data, these could not be directly compared on the data used by Mashup and REPEL. For example, NetGO[19] and its subsequent incarnations integrate data across multiple species and predict cross-species interactions, a separate and challenging problem. GTPLM-GO[20] and SEGT-GO[21] are related graph transformer methods that both require additional protein sequence data. It would, however, also be interesting to test REPEL against DropEdge[22] variants of DeepNF or BIONIC, or other GCN augmenta-

tion strategies for BIONIC that, like REPEL, are intended to prevent over-squashing of the embedding space.[23]

3. Materials and Methods

3.1. *Biological Interaction Data*

3.1.1. *Protein-protein Interaction Networks*

Yeast Networks The set of yeast (*Saccharomyces cerevisiae*) networks we consider in this paper are the ones used in the original Mashup paper[9] and are also described in detail there. They come from the STRING database v9.1,[24] excluding links derived from text-mining. In particular, we consider six heterogeneous networks featuring neighborhood, fusion, co-occurrence, co-expression, experimental, and database interaction types with edge counts ranging from $1,361$ to $314,013$ over the same set of $6,400$ genes (see Table S1 in the Supplement).

***E. coli* Networks** We downloaded analogous *E. coli* (*Escherichia coli*) networks from STRING database v12.0.[25] We use the same interaction types as in yeast, with edge counts ranging from $9,152$ to $219,483$ on the same set of $4,140$ genes (see Table S1 in the Supplement).

3.1.2. *Protein annotation*

We use functional annotation from the Gene Ontology (GO)[26] for yeast and *E. coli*; functional labels are separated according to their three distinct hierarchies: Biological Process (BP), Molecular Function (MF), and Cellular Component (CC). As in the Mashup paper, we filter the GO terms to only retain functional labels of intermediate specificity, terms labeling more than 10 and fewer than 301 protein nodes. This label set is further divided into levels of varying specificity, each containing labels that annotate either 11-30, 31-100, or 101-300 proteins, respectively.[27] Thus for each organism we perform 9 different function prediction experiments, parameterized by one of the three hierarchies (BP, MF, or CC) and one of the three levels of GO-term specificity (11-30, 31-100, and 101-300). Number of GO labels in each bin appear in Table S2 in the Supplement.

3.2. *Synthetic Data*

To clarify the role of negative weight edges, we generate synthetic data that we use to specifically compare Mashup to REPEL in cases where there are known, correct annotations.

To create synthetic datasets with a realistic multi-network structure, we begin by randomly generating one graph referred to as the ***base graph***, denoted $\mathcal{G}_{\mathcal{B}}$. The graph $\mathcal{G}_{\mathcal{B}}$ contains multiple communities, characterized by a higher density of intra-community edges and a lower density of inter-community edges; these communities are each assigned a different label that is considered to be the ground truth. Each node is assigned to exactly one community.

We create $\mathcal{G}_{\mathcal{B}}$ with 1,000 nodes using the LFR_benchmark_graph function[28] from the NetworkX library.[29] This function produces a simulated graph in which both the node degrees and community sizes follow power-law distributions, where we set the power-law exponent to 2.5 and 1.5, respectively, resembling the real-world biological networks we study. The mixing

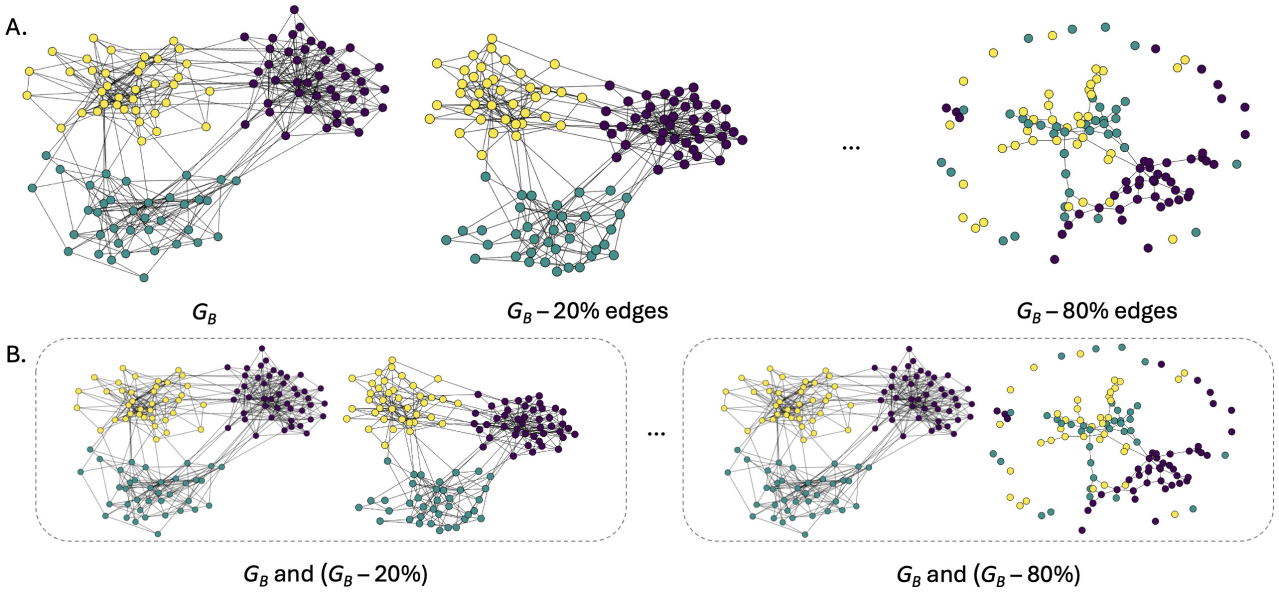

Fig. 2. Illustration of sparsity robustness analysis experiment settings. (A.) An example of $\mathcal{G}_\mathcal{B}$ generated with the LFR algorithm but with just 120 nodes to improve visualization. $\mathcal{G}_\mathcal{B}$ is shown on the left. The next two graphs illustrate $\mathcal{G}_\mathcal{B}$ with a random 20% and 80% of the edges removed, respectively. For the first round of sparsity experiments, we input each individual graph alone to both Mashup and REPEL. (B.) In the second experimental setting, both the Mashup and REPEL algorithms receive as input a pair of graphs: $\mathcal{G}_\mathcal{B}$ plus one of the graphs with edges randomly removed.

parameter μ, which controls the fraction of each node's edges that connect to nodes outside its own community, is set to 0.15, reflecting relatively well-separated communities. We set the average node degree to 20 and the maximum degree to 100, while constraining community sizes to range from 15 to 100 nodes.

From $\mathcal{G}_\mathcal{B}$ we then generate multiple graphs, each of which captures different (yet consistent and overlapping) parts of this base graph community structure. To mimic the noisy, incomplete, and biased nature of our real input networks, we create these simulated networks such that the communities among multiple networks have some overlap, but we allow different levels of sparsity and noise.

Sparsity Available PPI network data are typically sparse,[30] motivating an investigation of how sparsity impacts model performance. To introduce varying levels of sparsity, we randomly remove 20%, 40%, 60% and 80% of the edges from $\mathcal{G}_\mathcal{B}$ to generate increasingly sparse versions. We then evaluate model performance under two input settings: first, using only the sparse graph, and second, using both the sparse graph and $\mathcal{G}_\mathcal{B}$ as input, as shown in Figure 2. We note that although these methods are all designed to learn from *multiple* networks, in some simulations, we assess performance when learning from just a single network as a control.

Noise Similar to the sparsity analysis, and motivated by the fact that PPI networks are often noisy and biased,[30] we also conducted a noise robustness analysis. To simulate variation and introduce noise while preserving the community information, we construct a new empty graph with the same set of nodes as $\mathcal{G}_\mathcal{B}$ and add its edges as follows: for each community, we randomly select a subset comprising p percent of its nodes. Within this subset, we add intra-

community edges with higher probabilities to ensure cluster formation. Also, we randomly add edges between these selected nodes and nodes outside their community (inter-community) with lower probabilities, thereby introducing a degree of noise. This process yields a noisy graph derived from $\mathcal{G}_{\mathcal{B}}$ that maintains at least some of its core community structure.

Table 1. Configurations for Noisy Graph Generation

Configurations	num graph	$G_{gen}1$ param set	$G_{gen}2$ param set	$G_{gen}3$ param set
Config 1	3	[0.8, 0.2, 0.005]	[0.8, 0.2, 0.005]	[0.8, 0.2, 0.005]
Config 2	3	[0.8, 0.2, 0.005]	[0.6, 0.2, 0.005]	[0.4, 0.2, 0.005]
Config 3	3	[0.2, 0.005, 0.005]	[0.2, 0.005, 0.005]	[0.2, 0.005, 0.005]
Config 4	1	[0.2, 0.005, 0.005]	NA	NA

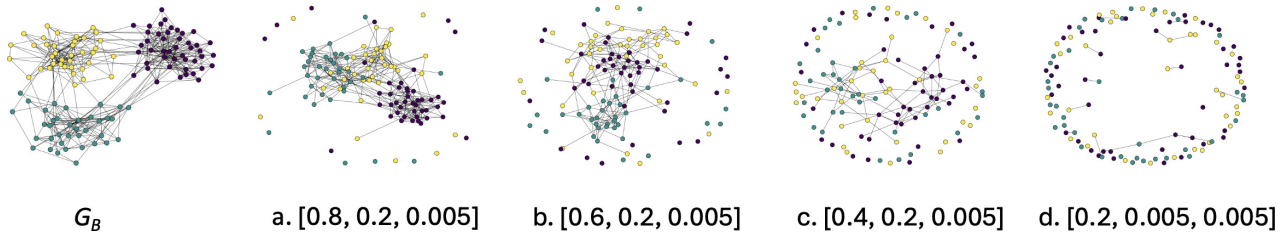

Fig. 3. Illustration of noise robustness analysis experiment settings. The base graph $\mathcal{G}_{\mathcal{B}}$ shown is generated using the LFR algorithm with 120 nodes for clarity of visualization. Noisy graphs *a.* to *d.* are randomly generated using the methods in Section 3.2 under different sets of parameters (Table 1). Each parameter list denotes the percentage of preserved cluster core nodes, the probability of edges existing intra-cluster, and the probability of edges existing inter-cluster. From graph *a.* to *d.*, the structure of $\mathcal{G}_{\mathcal{B}}$ becomes more disrupted.

We construct multiple noisy graphs under varying configurations to simulate different noise scenarios and assess model robustness. The parameters for the four chosen configurations are shown in Table 1, where the parameters for each of three simulated graphs are listed in order: percent of preserved cluster core nodes, probability of within-cluster edges, and probability of between cluster edges.

In Configuration 1, we generate three random graphs with similar structural characteristics to $\mathcal{G}_{\mathcal{B}}$. In these graphs, the core structure of each cluster is largely preserved and remains connected, while inter-cluster edges are kept sparse. In Configuration 2, we also generate three random graphs with the same overall probability of intra-cluster and inter-cluster edges as in Configuration 1. However, in this case, the core structures within the clusters are gradually disrupted, simulating a scenario where community integrity is degraded. In Configuration 3, we generate three graphs in which the cluster cores are deliberately disrupted and the clusters are only loosely connected. In Configuration 4, we follow the same procedure as in Configuration 3 but generate only a single corrupted graph.

Thus, Configurations 3 and 4 together serve as a negative control, providing a baseline for evaluating model performance in the absence of a meaningful community structure. In each case, we evaluate the performance by providing the algorithms either with only the simulated

graphs, or with the simulated graphs derived graphs together with $\mathcal{G}_{\mathcal{B}}$.

3.3. *Evaluation Methods*

We evaluate the performance of Mashup, deepNF, BIONIC and our REPEL approaches for predicting GO functional labels across the three GO hierarchies, BP, MF, and CC, using the yeast and *E. coli* multi-network collections. On the simulated networks, we evaluate the performance using only Mashup and REPEL, to directly analyze how the augmented uniform repel strength denoises the learned embeddings. Evaluation is conducted via five-fold cross-validation, by randomly splitting the nodes into five folds. In each iteration, the labels from one fold are withheld and treated as test data.

For each test fold, after obtaining the low-dimensional embedding of each node, we identify its k nearest neighbors based on the pairwise Euclidean distance in the embedding space. A weighted majority vote is then performed, where each neighbor's vote is weighted by the reciprocal of its distance to the test node. The GO terms receiving the highest total votes are assigned as the predicted functions.

In each simulation setting, following the evaluation protocol described in the Mashup paper,[9] we report the performance as accuracy, measured as the proportion of correctly annotated nodes. We conduct 10 independent runs of five-fold cross-validation. The averaged prediction accuracy and the standard deviation of that accuracy across the 10 runs are reported.

In experiments on real data, we conduct 5 independent runs of five-fold cross-validation and report the accuracy, F1 score, and area under the precision–recall curve (AUPRC), again following the definitions in the Mashup paper.[9]

3.4. *Algorithm Parameters*

For all experiments, REPEL was run with default parameters (summarized also in Tables S3 and S4) as follows. We set the dimensionality of the embeddings to 400, consistent with the recommendation of Cho et al[9] for yeast and *E.coli*, which suggests retaining the dimensionality of $5\% - 10\%$ of the original number of nodes. We also adopt the restart rate α of 0.5 from the Mashup implementation. For label prediction, we only consider the nodes with labels within the nearest 10 neighbors. Positive and negative edge weights are set to 1 and -1. The number of auxiliary nodes was set to 15 based on a hyperparameter search, in which we observed that when the number of augmented nodes is below 5, the results are noisy. The performance becomes and remains stable with 10 to 20 augmented nodes, then starts to drop very gradually as more augmented nodes are added (see Figure S1 in the Supplement).

4. Results and Discussion

4.1. *REPEL is Robust on Synthetic Networks*

In our synthetic data experiments, we generated "ground-truth" base graphs consisting of $1,000$ nodes. Using the procedures described in Section 3.2, we constructed several sets of additional sparse and noisy graphs. We evaluated REPEL and Mashup performance based on the accuracy in predicting community labels for the test nodes to directly compare the impact of random graph augmentation in controlled settings.

The results of the sparsity simulations are reported in Table S6 in the Supplement. When provided with only the single network $\mathcal{G}_\mathcal{B}$ as input, both Mashup and REPEL perfectly recover the community assignments of all nodes. Removal of up to 40% of edges does not significantly impact prediction accuracy for either method. However, a noticeable decline in accuracy is observed at a 60% edge removal rate. With removal of 80% of the edges, performance deteriorates substantially for both methods, with Mashup being more severely affected than REPEL.

Incorporating $\mathcal{G}_\mathcal{B}$ with the sparse graph consistently improves prediction performance as expected. When the edge removal rate is 60% or lower, both methods achieve accuracy above 0.99. Notably, at an 80% edge removal rate, if provided with $\mathcal{G}_\mathcal{B}$, REPEL's accuracy rebounds to above 0.99, while Mashup's performance only returns to 0.92. These results indicate that Mashup is robust to some degree of sparsity, but becomes more sensitive as sparsity increases. In contrast, REPEL not only achieves comparable performance when more community information is available, but also demonstrates greater robustness under high levels of sparsity.

Performance in the noise robustness simulations is reported in Table S7 in the Supplement. Here, the results of Configuration 1 indicate that Mashup is more sensitive to noise, while REPEL demonstrates greater robustness. Moreover, when $\mathcal{G}_\mathcal{B}$ is incorporated as additional input, REPEL shows a larger performance gain compared to Mashup, suggesting its better ability in leveraging the underlying structures. As more noise is introduced in Configuration 2, both Mashup and REPEL experience a decline in performance. However, REPEL consistently demonstrates greater robustness, maintaining higher accuracy. While incorporating $\mathcal{G}_\mathcal{B}$ as a fourth input network leads to improved performance for Mashup, it remains substantially affected by the noise. In contrast, REPEL is able to more effectively incorporate the structural information provided by $\mathcal{G}_\mathcal{B}$, restoring its accuracy to over 0.95.

The results from Configurations 3 and 4 further support the above observations. When only severely corrupted graphs are provided as input, both Mashup and REPEL perform poorly as expected, with similarly low accuracy. When $\mathcal{G}_\mathcal{B}$ is included, Mashup performs more variably given different numbers of degraded graphs. Specifically, additional corrupted graphs more strongly degrade Mashup's performance, suggesting it struggles to distinguish noise from real graph structure. This highlights a limitation in Mashup's ability to filter out spurious information. In contrast, although the addition of more corrupted graphs also leads to a decrease in REPEL's performance, the impact is smaller. This suggests that REPEL is better equipped to extract and preserve underlying structure in the presence of noise.

4.2. *Evaluation on Real Networks*

We observed that REPEL outperforms Mashup, deepNF and BIONIC on accuracy, F1 score and AUPRC in all experiments on the yeast networks (Figures 4 and 5, Tables S5A and S5B). On the *E.coli* networks, REPEL also outperforms other methods in all BP and MF experiment settings. The methods show similar performance in CC settings on terms labeling at least 31 nodes, because there are very few (≤ 30) terms in these categories, so the numbers of labels and test genes are too low to reliably distinguish the methods.

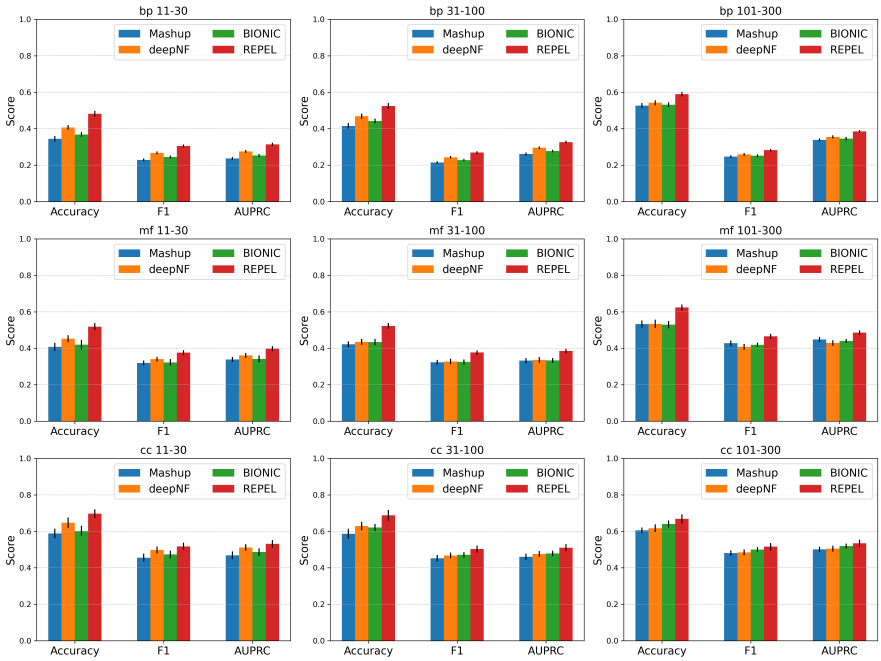

Fig. 4. Cross-validation performance on **yeast** protein networks with GO functional labels for BP, MF, and CC hierarchies compared to Mashup, deepNF and BIONIC. Performance measured by accuracy, F1 score and AUPRC, averaged across 5 independent runs of 5-fold cross-validation. The black lines denote the averaged standard deviation. Full numerical results in Table S5A.

4.3. *REPEL Finds Better Functional Clusters*

We analyzed the learned embeddings for both REPEL and Mashup. Given a protein, we find its nearest 10 neighbor proteins using L2 distance according to the learned protein feature vectors. Then we queried the protein along with its neighbors in the STRING v12.[25] database. We filter out the text mining interactions, and only keep the interactions with combined confidence scores higher than 0.9. Figure 6 shows the interactions among some example proteins and their neighbors, with line thickness indicating the strength of data support. We observe more consistent and more functionally closely connected proteins using REPEL.

For example, OLI1 is a mitochondrial ATP synthase. Mashup is able to find some functionally related genes, including ATP1, ATP2, COX4 and CYT1, however, the rest of its neighbor genes are not obviously functionally closely related. REPEL is able to find more ATP related genes including ATP1, ATP2, ATP3, ATP5, ATP6, ATP8, ATP16, and ATP17.

For another example protein, COB, both Mashup and REPEL find overlapping sets of close functionally related genes. However, the protein set found by REPEL shows slightly higher connectivity, with an average local clustering coefficient of 0.952, versus 0.846 for the neighbors according to Mashup.

5. Discussion

We have introduced the simple but effective REPEL algorithm for protein function prediction. Our experiments on both synthetic data and biological data demonstrate the power of

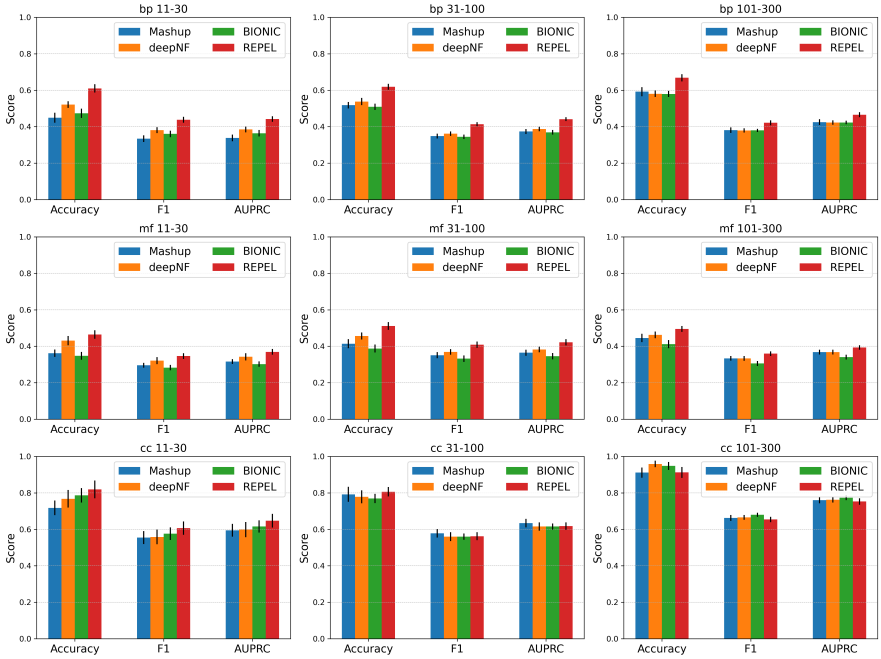

Fig. 5. Cross-validation performance on ***E. coli*** protein networks with GO functional labels for BP, MF, and CC hierarchies compared to Mashup, deepNF and BIONIC. Performance measured by accuracy, F1 score and AUPRC, averaged across 5 independent runs of 5-fold cross-validation. The black lines denote the averaged standard deviation. Full numerical results in Table S5B.

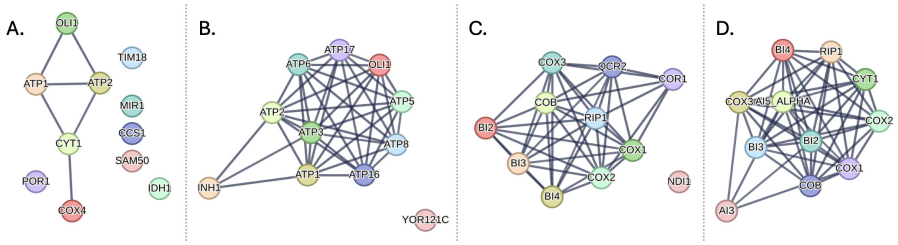

Fig. 6. OLI1, COB proteins and their 10 nearest proteins learned by Mashup and REPEL. A.) Mashup learned neighbors for OLI1. B) REPEL learned neighbors for OLI1. C.) Mashup learned neighbors for COB. D) REPEL learned neighbors for COB

injecting uniform random repelling strength in regularizing protein embeddings across multiplex networks. The simulation experiments highlight that this approach appears to increase robustness, effectively de-noising the learned embeddings. In the current version of REPEL, we apply the node augmentation before embedding, but it would be interesting to see what happens if instead we did it afterwards. Also, we have kept the focus on the information content of the embedding by having the learning module simply perform k-nearest neighbors to predict the functional labels. However, the best way to combine the REPEL embedding with more sophisticated classifiers could be an interesting topic of future work. In addition, we would like to add some degree of uncertainty quantification to the REPEL framework, so that in addition to predicting functions, REPEL gives confidence levels for its predictions.

Funding

Early work in this paper was partially supported by the National Science Foundation under grant CCF-1934553. LC also thanks the National Institute Of General Medical Sciences of the National Institutes of Health under Award Number R01GM163241. The content is solely the responsibility of the authors and does not necessarily represent the official views of the National Institutes of Health.

References

1. B. Schwikowski, P. Uetz and S. Fields, A network of protein-protein interactions in yeast, *Nature Biotechnology* , 1257 (Dec 2000).
2. P. Uetz, L. Giot, G. Cagney, T. A. Mansfield, R. S. Judson, J. R. Knight, D. Lockshon, V. Narayan, M. Srinivasan, P. Pochart *et al.*, A comprehensive analysis of protein–protein interactions in Saccharomyces cerevisiae, *Nature* **403**, 623 (2000).
3. A. Vazquez, A. Flammini, A. Maritan and A. Vespignani, Global protein function prediction from protein-protein interaction networks, *Nature Biotechnology* **21**, 697 (2003).
4. U. Karaoz, T. Murali, S. Letovsky, Y. Zheng, C. Ding, C. R. Cantor and S. Kasif, Whole-genome annotation by using evidence integration in functional-linkage networks, *Proceedings of the National Academy of Sciences* **101**, 2888 (2004).
5. E. Nabieva, K. Jim, A. Agarwal, B. Chazelle and M. Singh, Whole-proteome prediction of protein function via graph-theoretic analysis of interaction maps, *Bioinformatics* **21**, i302 (2005).
6. L. Cowen, T. Ideker, B. J. Raphael and R. Sharan, Network propagation: a universal amplifier of genetic associations, *Nature Reviews Genetics* **18**, 551 (2017).
7. H. Tong, C. Faloutsos and J.-Y. Pan, Fast random walk with restart and its applications, in *Sixth international conference on data mining (ICDM'06)*, 2006.
8. M. Cao, H. Zhang, J. Park, N. M. Daniels, M. E. Crovella, L. J. Cowen and B. Hescott, Going the distance for protein function prediction: a new distance metric for protein interaction networks, *PLOS One* **8**, p. e76339 (2013).
9. H. Cho, B. Berger and J. Peng, Compact integration of multi-network topology for functional analysis of genes, *Cell Systems* **3**, 540 (2016).
10. A. Ben-Hur, C. S. Ong, S. Sonnenburg, B. Schölkopf and G. Rätsch, Support vector machines and kernels for computational biology, *PLOS Computational Biology* **4**, p. e1000173 (2008).
11. G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever and R. R. Salakhutdinov, Improving neural networks by preventing co-adaptation of feature detectors (2012), arXiv 1207.0580.
12. J. Ho, A. Jain and P. Abbeel, Denoising diffusion probabilistic models, *Advances in Neural Information Processing Systems* **33**, 6840 (2020).
13. P. Vincent, H. Larochelle, Y. Bengio and P.-A. Manzagol, Extracting and composing robust features with denoising autoencoders, in *Proceedings of the 25th International Conference on Machine Learning*, 2008.
14. V. Gligorijević, M. Barot and R. Bonneau, deepNF: deep network fusion for protein function prediction, *Bioinformatics* **34**, 3873 (2018).
15. P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, P.-A. Manzagol and L. Bottou, Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion., *Journal of Machine Learning Research* **11** (2010).
16. D. T. Forster, S. C. Li, Y. Yashiroda, M. Yoshimura, Z. Li, L. A. V. Isuhuaylas, K. Itto-Nakama, D. Yamanaka, Y. Ohya, H. Osada, B. Wang, G. D. Bader and C. Boone, BIONIC: Biological network integration using convolutions, *Nature Methods* **19**, p. 1250–1261 (October 2022).
17. T. N. Kipf and M. Welling, Semi-supervised classification with graph convolutional networks

(2016), arXiv:1609.02907.
18. P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò and Y. Bengio, Graph attention networks (2017), arXiv: 1710.10903.
19. R. You, S. Yao, Y. Xiong, X. Huang, F. Sun, H. Mamitsuka and S. Zhu, NetGO: improving large-scale protein function prediction with massive network information, *Nucleic Acids Research* **47**, W379 (2019).
20. H. Zhang, Y. Sun, Y. Wang, X. Luo, Y. Liu, B. Chen, X. Jin and D. Zhu, GTPLM-GO: Enhancing protein function prediction through dual-branch graph transformer and protein language model fusing sequence and local–global PPI information, *International Journal of Molecular Sciences* **26**, p. 4088 (2025).
21. Y. Wang, Y. Sun, B. Lin, H. Zhang, X. Luo, Y. Liu, X. Jin and D. Zhu, SEGT-GO: a graph transformer method based on PPI serialization and explanatory artificial intelligence for protein function prediction, *BMC Bioinformatics* **26**, p. 46 (2025).
22. Y. Rong, W. Huang, T. Xu and J. Huang, Dropedge: Towards deep graph convolutional networks on node classification, *arXiv:1907.10903* (2019).
23. S. Akansha, Over-squashing in graph neural networks: A comprehensive survey, *Neurocomputing* , p. 130389 (2025).
24. A. Franceschini, D. Szklarczyk, S. Frankild, M. Kuhn, M. Simonovic, A. Roth, J. Lin, P. Minguez, P. Bork, C. von Mering and L. J. Jensen, STRING v9.1: protein-protein interaction networks, with increased coverage and integration, *Nucleic Acids Research* **41**, p. D808–D815 (November 2012).
25. D. Szklarczyk, R. Kirsch, M. Koutrouli, K. Nastou, F. Mehryary, R. Hachilif, A. L. Gable, T. Fang, N. T. Doncheva, S. Pyysalo, P. Bork, L. J. Jensen and C. von Mering, The STRING database in 2023: protein–protein association networks and functional enrichment analyses for any sequenced genome of interest, *Nucleic Acids Research* **51**, p. D638–D646 (November 2022).
26. M. Ashburner, C. A. Ball, J. A. Blake, D. Botstein, H. Butler, J. M. Cherry, A. P. Davis, K. Dolinski, S. S. Dwight, J. T. Eppig, M. A. Harris, D. P. Hill, L. Issel-Tarver, A. Kasarskis, S. Lewis, J. C. Matese, J. E. Richardson, M. Ringwald, G. M. Rubin and G. Sherlock, Gene ontology: tool for the unification of biology, *Nature Genetics* **25**, p. 25–29 (May 2000).
27. A. Ruepp, A. Zollner, D. Maier, K. Albermann, J. Hani, M. Mokrejs, I. Tetko, U. Guldener, G. Mannhaupt, M. Munsterkotter and H. W. Mewes, The FunCat, a functional annotation scheme for systematic classification of proteins from whole genomes, *Nucleic Acids Research* **32**, 5529 (2004).
28. A. Lancichinetti, S. Fortunato and F. Radicchi, Benchmark graphs for testing community detection algorithms, *Phys. Rev. E Stat. Nonlin. Soft Matter Phys.* **78**, p. 046110 (October 2008).
29. A. Hagberg, P. Swart and D. S Chult, *Exploring network structure, dynamics, and function using NetworkX*, tech. rep., Los Alamos National Lab.(LANL), Los Alamos, NM (United States) (2008).
30. N. Safari-Alighiarloo, M. Taghizadeh, M. Rezaei-Tavirani, B. Goliaei and A. A. Peyvandi, Protein-protein interaction networks (PPI) and complex diseases, *Gastroenterol. Hepatol. Bed Bench* **7**, 17 (2014).