Link prediction using low-dimensional node embeddings: the measurement problem

Nicolas Menand[a,1] **and C. Seshadhri**[b]

[a]Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA 19104. Most of this work was done when the author was at University of California, Santa Cruz.; [b]Department of Computer Science, University of California, Santa Cruz, CA 95064

This manuscript was compiled on January 5, 2024

Graph representation learning is a fundamental technique for machine learning on complex networks. Given an input network, these methods represent the vertices by low-dimensional real-valued vectors. These vectors can be used for a multitude of downstream machine learning tasks. We study one of the most important such task, link prediction. Much of the recent literature on graph representation learning has shown remarkable success in link prediction. On closer investigation, we observe that the performance is measured by the AUC (Area Under Curve), which suffers biases. Since the ground truth in link prediction is sparse, we design a new vertex-centric measure of performance, called the *VCMPR@k plots*. Under this measure, we show that link predictors using graph representations show poor scores. Despite having extremely high AUC scores, the predictors miss much of the ground truth. We discover a mathematical connection between this performance, the sparsity of the ground truth, and the low-dimensional geometry of the node embeddings. Under a formal theoretical framework, we prove that low-dimensional vectors cannot capture sparse ground truth using dot product similarities (the standard practice in the literature). Our results call into question existing results on link prediction and pose a significant scientific challenge for graph representation learning. The VCMPR plots identify specific scientific challenges for link prediction using low dimensional node embeddings.

Low Dimensional Embeddings | Link Prediction | Node Embeddings | Graph Representational Learning | Machine learning metrics | AUC

Measurement is central to any scientific endeavor. Informative measurements are crucial to guide experiments and interpret results. For machine learning (ML), measurements are central to evaluating performance and discovering better techniques. These measurements have a large impact in the deployment of real ML systems. As such systems become a large part of modern society, it becomes even more important to have sound measurements of machine learning methods.

Our focus is on the important field of graph machine learning, where ML is deployed on large complex networks. One of the recent advances in machine learning uses *graph representation learning* or low-dimensional node embeddings to tackle a large variety of tasks. The input is a graph G on n vertices. These methods map each vertex to a vector in $\mathbb{R}^d$, where d is typically much smaller than n. (So n may be in millions or more, while d is typically 128.) These embeddings are generated in an unsupervised or self-supervised manner. The aim is to generate embeddings where geometric proximity (often measured as dot product) maps to graph proximity. The design of low-dimensional node embeddings is a popular and timely research area. We point the reader to surveys (1, 2) and Chapter 23 in (3).

These embedding methods are evaluated by performing a downstream machine learning task, such as the classic link prediction problem (4, 5). Recall the formulation (4, 5). We are given a graph $G = (V, E)$ as part of the training data. One should think of a dynamic process generating edges, and G is the current snapshot of edges. Our aim is to predict the future edges amongst previously seen nodes. The link predictor trains on the current snapshot G, and does the following. Given a pair (i, j) of vertices, it predicts whether (i, j) will be an edge.

We discover a fundamental *measurement problem*. Most contemporary literature for link prediction using node embeddings measure performance by the AUC (Area Under the Curve) metric (6–19). But the AUC is a measure that is meaningful for dense signal and suffers from imbalance biases (20, 21). We discover that when link prediction performance is measured according to localized metrics, there is a significant drop in quality. This is a major scientific problem. AUC in link prediction is used to benchmark algorithms (11); used to evaluate new techniques (12); conclude properties of ML algorithms (13). At least eight of the cited papers were published in the past three years (12–19), and two appear in extremely high profile scientific venues (12, 19). It is of central importance to have measurements that lead us to correct scientific conclusions.

Our paper investigates the connections between alternate vertex-centric link prediction measures, the sparsity of ground truth, and low-dimensional node embeddings.

Our results. We empirically demonstrate that the AUC metric for link prediction by node embedding methods leads to incorrect conclusions on the quality. We define new vertex-centric measures, which clearly show poor performance on real-world datasets. These measures are used to construct VCMPR plots that quantitatively demonstrate the low quality on link prediction. To explain these results,

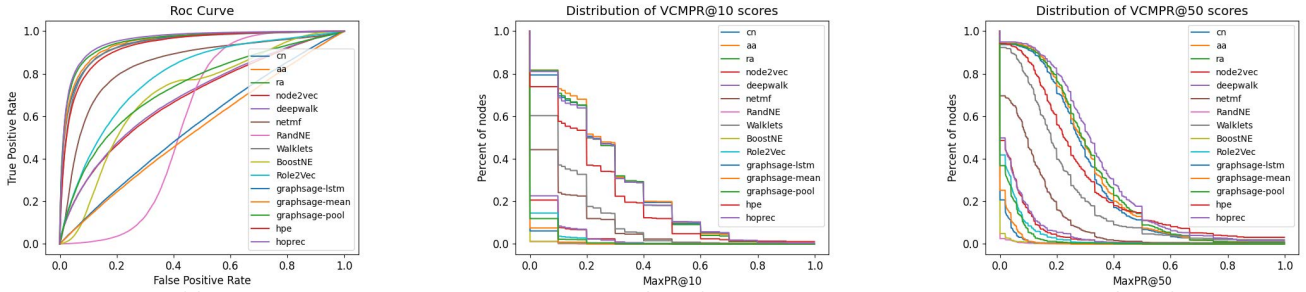
Significance Statement

Link prediction is a fundamental machine learning task on complex networks, used to evaluate the central technique of low-dimensional embeddings. Our results question the common wisdom that low-dimensional embeddings perform well in link prediction tasks. We show that this wisdom is based on faulty measurements (based on AUC) used to evaluate link prediction. We propose new vertex-centric local measures, under which existing low-dimensional embedding methods are shown to fail in link prediction. We discover a mathematical connection between this poor performance and the low-dimensional geometry of the node embeddings. Under a formal theoretical framework, we prove that low-dimensional vectors cannot capture sparse ground truth using dot product similarities (which is the standard practice in the literature).

N. M. and C.S. performed research, designed the experiments, analyzed the results, and wrote the paper.

The authors declare no competing interest.

[1] To whom correspondence should be addressed. E-mail: nmenand@seas.upenn.edu



(a) ROC curve **(b)** VCMPR@10 **(c)** VCMPR@50

Fig. 1. We train a number of important node embedding methods for transductive link prediction on the `blog-Catalog` dataset (22). In the left figure, we plot the standard ROC curves for each predictor. In the middle and right figures, we plot the distribution of VCMPR@k values, for $k = 10, 50$. Observe that ROC curves are typically high leading to large AUC values. On the other hand, the VCMPR curves are quite low. This is indicative of both low precision and recall among the top predictions. The average values are summarized in Tab. 1.

we design a theoretical framework formally showing that commonly used link prediction algorithms from low-dimensional embeddings are unlikely to get high local precision/recall values. Inspired by these results, we pose a concrete challenge problem for low-dimensional embeddings. We hope that our new measurements and challenge will inspire further research on this important topic.

The VCMPR scores and empirical setup. We train a large number of important node embedding methods on standard graph datasets (6–10, 18, 23, 24). We set up a transductive link prediction experiment, following the same setup as the above results. We are given a training graph $G = (V, E_{tr})$. Based on this training data G, we design a predictor for future edges, denoted $\chi : V^2 \to [0, 1]$. One can think of the predictor as giving a score for each pair (i, j), which measures the likelihood that this pair will form an edge. The performance of χ is tested against another set E_{test} of edges. In practice, the edges E_{tr} and E_{test} are generated by randomly partitioning the edges of an existing dataset. Note that E_{test} is the ground truth for our experiment.

We compute the standard ROC curve (Receiver-Operator Characteristic) (25), as shown in Fig. 1a for `Blog-Catalog` dataset. Most methods perform quite well according to this plot. Similar results for PR curve (Precision-Recall). These plots are often summarized using the "Area Under the Curve" (AUC) metric, whose maximum value is one. We show the ROC-AUC and PR-AUC values in Tab. 1; consistent with the literature, we see AUC scores more than 0.7, and a largest score of 0.94. This would suggest good performance on transductive link prediction.

For the *same* setup, we define the *VCMPR@k* plots, a vertex-centric metric. This measure requires a closer look into link prediction using node embeddings. For each pair of vertices (i, j), the classifier/predictor computes a score based on which it predicts an edge. For a given vertex i of non-zero degree d_i, we rank all other vertices j in decreasing order of their scores. Remove pairs from E_{tr}. Take the top k scores from this list, and let $t_i(k)$ denote the number of ground truth edges in this list. Meaning, there are $t_i(k)$ ground truth edges (i, j) in the top k-entries of the (downward) sorted list.

$$\text{VCMPR@k for vertex } i = \frac{t_i(k)}{\min(k, d_i)}$$

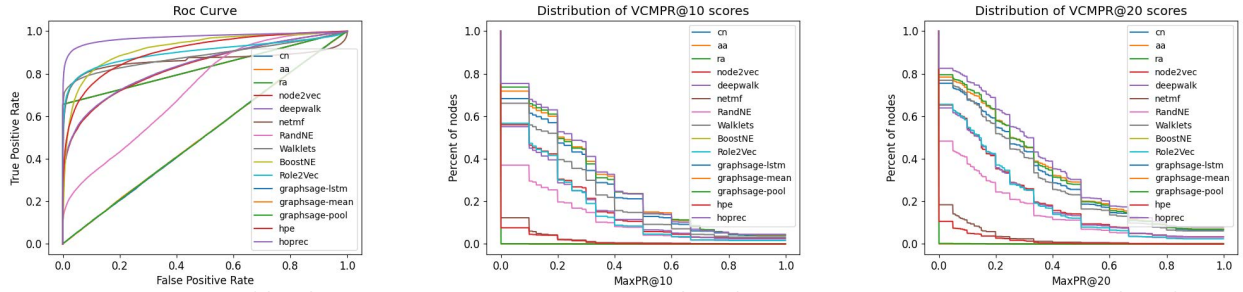
Note that $t_i(k)/k$ is the precision@k, and $t_i(k)/d_i$ is the recall@k. We take the larger of these values, so that low degree vertices are not penalized for poor precision (since $t_i(k) \leq d_i$). Roughly speaking, if the VCMPR@k for vertex i is δ, then a δ fraction of the top-k predictions (for i) are actually ground truth edges. Note that VCMPR is a local metric *and computed at a per-vertex basis*. Contrast with the AUC, which is a global measure. We also stress the difference from vanilla precision/recall@k, which is computed by ranking (a sample of) all edges in the graph. (We discuss more differences and variants of VCMPR subsequently.) The VCMPR plots give the complementary cumulative histogram for the VCMPR values. So for $x \in [0, 1]$, we plot the fraction of vertices with VCMPR at least x. These are shown in Fig. 1b and Fig. 1c.

	ROC AUC	PR AUC	Avg VCMPR @10	Avg VCMPR @50
Resource Allocation	0.95	0.95	0.26	0.30
Adamic Adar	0.95	0.94	0.27	0.30
Common Neighbors	0.94	0.94	0.26	0.29
HOP-Rec	0.96	0.95	0.27	0.33
Walklets	0.94	0.93	0.12	0.21
HPE	0.93	0.93	0.20	0.27
NetMF	0.85	0.82	0.09	0.12
Role2Vec	0.79	0.76	0.02	0.03
GraphSage-MP	0.73	0.73	0.01	0.03
BoostNE	0.70	0.63	0.00	0.00
Deepwalk	0.69	0.69	0.03	0.05
Node2Vec	0.69	0.69	0.03	0.05
RandNE	0.58	0.49	0.00	0.00
GraphSage-L	0.56	0.55	0.01	0.01
GraphSage-M	0.54	0.53	0.01	0.01

Table 1. This table complements Fig. 1, which has results from link prediction on the `blog-Catalog` dataset. We give the ROC-AUC, PR-AUC, and average VCMPR@k for $k = 10, 50$. We observe fairly large AUC values, with the largest being 0.96. But the average VCMPR@k values are quite low. For almost all methods, it is less than 0.2, which implies poor precision and recall at the individual vertex centric predictions.

We observe that the VCMPR scores are surprisingly low. For example, even for the best method on this dataset, on average only 18% of the top scores are edges. And most methods have an average VCMPR of less than 0.05. In some datasets, the VCMPR values become a little higher (0.2 - 0.3), but nowhere near the large AUC values. We observe this consistently across methods and datasets (§2). Rather surprisingly, an AUC score of more than 0.7 may still lead to a VCMPR of less than 0.01. This means, for an average vertex, for the top (say) 50 predictions, at most *one* of them is a true edge. The difference between the high AUC and poor VCMPR is quite dramatic. The global "dense metric" of AUC is not attuned to the sparse ground truth. The VCMPR@k, for small k, is an averaged "local" score, and is more appropriate for sparse ground truth.

This measurement problem is apparent in the VCMPR plots of Fig. 1b and Fig. 1c. The plots are quite low, as opposed to the high ROC curves in Fig. 1a. As an example, consider the Role2Vec algo-



(a) `ogbl-collab` ROC-AUC **(b)** `ogbl-collab` VCMPR@10 **(c)** `ogbl-collab` VCMPR@20

Fig. 2. We train a number of important node embedding methods for transductive link prediction on the `ogbl-collab` dataset, a current link prediction benchmark on the Open Graph Benchmark Leaderboard (26). In the left figure, we plot the standard ROC curves for each predictor. In the middle and right figures, we plot the distribution of VCMPR@k values, for $k = 10, 20$. The low VCMPR curves clearly shows the weaknesses of existing methods. For all methods, at least 40% of the vertices have VCMPR@20 value of less than 0.3, which is quite low.

rithm (curve in pink). The VCMPR plots are almost at the bottom, despite the ROC curve being third from the top in Fig. 1a. We see the ROC curves lead to misleading interpretations on prediction quality and should not be the basis for algorithm evaluation. We see identical issues with the PR-AUC.

The importance of being vertex-centric. We stress the difference such as (vanilla) precision/recall@k, as well as the hits@k metric. The latter is used in knowledge completion tasks (26, 27). The hits@k metric is global. It takes a decreasing sorted list of random negative instances according to model score, and checks the number of test true positive edges that score in the top k. On the other hand, VCMPR@k scores are computed for every vertex. We stress that the k parameters for both metrics are incomparable. For example, consider the `ogbl-collab` dataset from the Open Graph Benchmark Leaderboard that comes with a specific link prediction task (26) The recommended metric is hits@50. For VCMPR, a natural choice of k is around the average vertex degree (which is 9). The average degree is the average length of a ground truth list, for a vertex.

In Tab. 2, we give the average AUC, VCMPR@10, and the recommended hits@50 metric scores for `ogbl-collab` dataset. The hits scores are significantly higher than the average VCMPR scores. HOP-REC, one of the embedding-based leaders on OGB, has a VCMPR@10 of just 0.28, while its hits score is 0.66. The hits score suggest reasonable performance, while the VCMPR scores are quite low. AUC scores are extremely high, as in Tab. 1. We get similar results on other examples, described in the SI. Overall, the experiments show that the VCMPR and hits metrics are fundamentally different.

More significant are the VCMPR plots in Fig. 2, which pinpoint weaknesses in a way that single scores cannot capture. For example, the VCMPR@20 plot shows that, for *all* methods, at least 40% of the vertices have a VCMPR@20 score less than 0.3. Metrics like average scores or hit@k do not reveal such problems. (A choice of 20 is quite generous, since the average degree is 9.) One can perform a deeper investigation into this set to understand for which vertices the algorithm is failing. Many sparse applications of link prediction are often for personalization in recommendation systems, where a vertex-centric view is closer to the application. (28, 29). Global metrics do not provide insights at a vertex level.

Theoretical explanation. We discover a theoretical connection between the poor VCMPR and the low dimensional aspect of embeddings. The standard scoring method for link prediction is the dot product (or Hadamard product) of the embedding vectors. We stress that almost all previous work follows this method (6–10, 18). We construct a framework that formalizes the notion of sparse ground truth that should be "detectable" by scores involving dot products of embedding vectors. Under this framework, we prove that the rank of the embedding vectors must be nearly-linear. This is a strong lower bound that counters the general notion that low-dimensional embeddings with dot product based scores is a good method for link prediction.

	ROC-AUC	PR-AUC	Avg VCMPR @10	Hits @50
Common Neighbors	0.83	0.91	0.26	0.53
Adamic Adar	0.83	0.91	0.28	0.65
Resource Allocation	0.83	0.91	0.28	0.65
Deepwalk	0.83	0.86	0.18	0.22
Node2Vec	0.83	0.86	0.18	0.22
NetMF	0.86	0.92	0.02	0.35
RandNE	0.72	0.73	0.11	0.09
Walklets	0.88	0.92	0.22	0.59
BoostNE	0.92	0.93	0.00	0.08
Role2Vec	0.90	0.93	0.17	0.44
GraphSage-M	0.51	0.51	0.00	0.00
GraphSage-MP	0.51	0.51	0.00	0.00
GraphSage-L	0.51	0.51	0.00	0.00
HPE	0.90	0.92	0.02	0.21
HOP-rec	0.97	0.98	0.28	0.66

Table 2. This table complements Fig. 2, which has results from link prediction on the `ogbl-collab` dataset. We give the ROC-AUC, PR-AUC, average VCMPR@k for $k = 10, 20$, and Hits@50.

For link prediction, the ground truth is fundamentally sparse, since the number of edges is proportional is to the number of vertices. If n is the number of vertices, the total number of edges we expect to see is $O(n)$. This is much smaller than $\binom{n}{2}$, the total number of *potential* edges. Alternately, the "yes class" is a tiny fraction of the size of the "no class" of non-edges (as observed by Lichtenwalter and Chawla (21)).

We explain the theoretical setup. Through some training process, a node embedding method outputs a vector $\vec{v}_i \in \mathbb{R}^d$ for each vertex i in the original graph. We represent these vectors by the $d \times n$ matrix V, where each column is an embedding vector. In the link prediction task, the final classifier/predictor uses the vectors $\vec{v}_i$ and $\vec{v}_j$ to determine whether (i, j) will become an edge. The most common score used for prediction is the dot product $\vec{v}_i \cdot \vec{v}_j$ or the cosine similarity $\vec{v}_i \cdot \vec{v}_j / (\|\vec{v}_i\|_2 \|\vec{v}_j\|_2)$*

Given a vertex i, we wish to predict some of the edges incident to

*In the latter case, we can just assume that the column vectors in V are normalized. So we will stick with the score being the dot product.

i. Our first step is to formalize what it means for the ground truth to be "sparse", and for the signal to be "strong".

Sparse, undirected ground truth: The total number of edges in the undirected graph is linear in n, the number of vertices. So, we expect an average vertex i to be linked with a constant number of other vertices. Moreover, if j is a good prediction for i, then i should be a good prediction for j.

Strong signal in the scores: The set of scores with respect to i is the set $\{\vec{v}_i \cdot \vec{v}_j | j \neq i\}$. We want the true positives (the edges) to "stand out" among these scores. By the sparsity discussed above, we do not want more than a constant number of candidates to stand out. For the vertex i, let $\mathcal{D}_i$ be the distribution over all other vertices, where the probability of j is $|\vec{v}_i \cdot \vec{v}_j| / \sum_{j \neq i} |\vec{v}_i \cdot \vec{v}_j|$. We can interpret the dot product as a relative likelihood that j is a potential neighbor of i.

For the signal to be strong, we expect the true positives to have significant probability mass in this distribution. Note that we may have false positives because of negative entries. (We take absolute values to ensure a distribution. Nonetheless, true positives should have high likelihood/probability in $\mathcal{D}_i$.) This discussion motivates a key definition. Let $\varepsilon > 0$ be a parameter; think of it as a small constant.

Definition 0.1. *A pair of vertices (i, j) is called* significant *if: both the probability of j in $\mathcal{D}_i$ and the probability of i in $\mathcal{D}_j$ are at least ε.*

This captures the notion that j is a good prediction for i and vice versa. Roughly speaking $1/\varepsilon$ samples from $\mathcal{D}_i$ are enough to detect j. Moreover, for a given i, there are at most $1/\varepsilon$ vertices j that form significant pairs.

We formally prove that any set of vectors that contain a linear number of significant pairs must have near-linear rank.

Theorem 0.2. *Consider a set of vectors $\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_n \in \mathbb{R}^d$ that are polynomially bounded in length. (So for some constant c, $\max_i \|\vec{v}_i\|_2 / \min_i \|\vec{v}_i\|_2 \leq n^c$.) Suppose there are at least δn significant pairs among these vectors. Then,* $\mathrm{rank}(V) \geq \mathrm{poly}(\varepsilon, \delta, \log^{-1} n) \times n$.

The condition on polynomial boundedness is a technicality; for most applications, n is extremely large (millions or more), so a polynomial bound in vector length is quite reasonable. The theorem is proven in §1.

Let us discuss the relevance of this theorem. The lower bound is quite strong with respect to the number of significant pairs. Even if (say) half the vertices participate in just one significant pair, the rank bound is still near-linear. By the contrapositive of Theorem 0.2, when the rank is small, then there exist $o(n)$ significant pairs. This means that that ground truth "signal" is drowned in the noise. Sampling according to the distributions $\mathcal{D}_i$ will not generate enough edges, resulting in many non-edges predicted.

This behavior will lead to poor precision/recall at the individual vertex level, exactly as we see in practice. In §A, we empirically see that the dot product scores between edges and non-edges are indistinguishable. In all our experiments, the sparse signal is hard to distinguish from the noise. On the other hand, an average edge has a high-score than an *average* non-edge, resulting in high AUC scores. But this is not relevant in an actual prediction task, where we need to report the pairs that are most likely to be edges.

The VCMPR challenge. *Can we design low-dimensional node embeddings that get high VCMPR plots for link prediction?*

Our theoretical results suggest some fundamental limitations, when using dot product based scores for prediction. Our empirical work shows that even generalizations like the Hadamard product do not give better results. Indeed, the overwhelming evidence is that a new idea is required. We believe that solution may need alternate geometries or kernels over low-dimensional vectors for the link prediction process. Recent work by Chanpuriya et al have suggested asymmetric factorizations and other methods to avoid weaknesses of low-dimensional embeddings (30). These techniques may help address this challenge.

We see our results as providing guidance for future work on graph representation learning. Theoretical frameworks that shed light on existing limitations direct us away for methods that might not hold promise. Our work underscores the importance for different kinds of measurements of performance for link prediction. We hope that the VCMPR challenge provides a concrete problem to tackle. *Regardless, we strongly advocate the use of VCMPR@k measures for link prediction performance in sparse graph settings.*

We note that there are link prediction settings where our analysis might not be applicable (more discussion in §C).

A. More on AUC and Local VCMPR-type measures. We give more insight into why AUC is a poor measure for link prediction. Recall that $\chi : V^2 \to [0, 1]$ denotes the predictor function for future edges, and E_{test} is the ground truth. For convenience, $(i, j) \sim E_{test}$ means that (i, j) is a uniform random sample from E_{test}. The following lemma gives an alternative definition of the ROC-AUC.

Lemma 0.3. *(Sec. 7 of (25)) The ROC-AUC of the predictor χ is*

$$\Pr_{(i,j)\sim E_{test}, (i',j')\sim \binom{V}{2}\setminus E_{test}} [\chi(i, j) > \chi(i', j')]$$

In plain English, the AUC is the probability that an average ground truth edge has a higher score than an average non-edge. (Henceforth, non-edge refers to a pair not in E_{test}.) But the ground truth set E_{test} has size $O(|V|)$, because real-world graphs are sparse. The complement (true negatives) has size size $\Omega(|V|^2)$, which is many orders of magnitude larger than the ground truth. Therein lies the fundamental problem. If ground truth edges have above average score, the AUC will be high. It is possible that there are (say) $10|E_{test}|$ non-edges with score higher than the ground truth edges, but the AUC is still $1 - o(1)$. The sparsity of ground truth makes it possible to have large AUC with a weak predictor. Indeed, this is what seems to happen in all our experiments. We stress that the weaknesses of AUC were known before (20). Lichtenwalter and Chawla specifically point out issues for link prediction (21). Despite that, it is the default method for evaluations of link predictors based on low-dimensional embeddings.

VCMPR variants: The main utility of VCMPR is that is a *local* measure, as opposed to a global measure such as Lemma 0.3. For link prediction, it is crucial to understand how a predictor behaves at a vertex level (how many of the top predictions for i are correct). The VCMPR was chosen for easy interpretability, and is a fairly permissive metric. If the ground truth degree of i is small, then the denominator for VCMPR is also small.

In general, one can go beyond VCMPR and compute other local metrics. For example, we can measure local NDCG (Normalized Discounted Cumulative Gain) at each vertex. Moreover, one can compute VCMPR using just the test edges or with both test and train edges, and define other variations of VCMPR@k. We discuss these variants in the SI and perform extensive experimentation on our datasets. Across *all* these measures, we consistently see poor

performance for link prediction. This is a strong indication that the embeddings are not capturing the structure of the graph.

B. Broader Context and Related Work. There is a rich literature showing that low-dimensional embeddings methods successfully perform the link prediction task (6–10, 18). Despite much empirical work on node embeddings methods, there are fewer principled theoretical results on their behavior (some recent papers address this topic (30–34)). There is growing evidence that hand-tuned algorithms can outperform embeddings methods for link prediction (11, 34). Recent work argues that certain low-dimensional embeddings cannot capture the cluster structure of real-world graphs (31).

The AUC is a fundamental metric using in machine learning and statistical applications. Fawcett has an excellent overview of AUC (25). Hand argues that AUC is deeply flawed because it creates a data-dependent reweighting of false positives vs false negatives (20). Deeply relevant to our own work, Lichtenwalter and Chawla perform an excellent study AUC in the context on link prediction (21). They show that the sparsity of the ground truth is a hindrance for using AUC. They also suggest precision and recall measures, but not in the localized manner of the VCMPR plots. Unlike previous work on AUC and link prediction, our work specifically connects the low values of VCMPR measures to low-dimensional embeddings. Existing work shows weaknesses of sampled metrics (like AUC) in ranking a single list (27). The crucial difference is that VCMPR focuses on the ranking of many lists, one for each vertex.

The hits@k (and precision/recall@k) metrics are fundamental measures in many knowledge completion tasks (35). We note the significant difference from personalized link prediction in the application. For knowledge completion, the goal is to find the potential edges, regardless of the endpoints. In personalized settings, we are given a specific vertex and want to know the potential edges incident to *that* vertex. Hence, the VCMPR metric more directly captures the goal.

Graph representation learning is an immensely large topic. We mention the surveys (1, 2) and Chapter 23 in (3). Our experiments use a large variety of contemporary and classic methods (4, 6–10, 18, 23, 24, 28, 36–39). We use the implementations of (40). The embedding methods span many categories, based on random walks, random projections, matrix factorizations, shallow embeddings, as well as Graph Neural Nets. (More details in §2.)

An increasing body of literature has attempted to build theoretical frameworks to study this wide array of embedding algorithms. Qi et al provide an overarching template of matrix factorization that subsumes many methods (10). Several results study the theoretical power of GNNs (32, 33, 41). Closer to our work, some results have tried to understand the limitations (and power) of low-dimensional embeddings. We borrow many theoretical tools from a result showing the inability of low-dimensional SVD to preserve the triangle structure of real-world networks (31). Chanpuriya et al give alternate geometric methods to circumvent these limitations (30). It is not clear how to use these asymmetric embeddings to perform (symmetric) link prediction for undirected graphs.

C. Limitations. Our mathematical analysis focuses on settings that are globally sparse, yet locally dense. (The local density appears in terms of having a strong signal at a vertex centric level.) This setting is widespread in most social network settings, often measured as low global density and high clustering coefficients. But for settings like link prediction in protein-protein interaction (PPI) networks, clustering coefficients are low, and our vertex-centric analysis might not be meaningful. We run link prediction experiments on PPI networks and observe that AUC scores are themselves quite low. In many of these experiments, VCMPR scores might not add more value.

Another important setting for link prediction is knowledge discovery. In such settings, the overall density may itself be high (seen in drug interaction datasets), or the vertex-centric measure might not be relevant. In knowledge discovery, we may be trying to predict any new edge, not just edges that are incident to a specific (set of) vertices. For such problems, our theoretical analysis is not informative and VCMPR metrics might not be meaningful. We perform experiments on a dense drug interaction dataset, and observe that hits@k metrics are probably more informative.

We also note that the term "density" may have different meanings. We treat it in terms of the total number of edges divided by the total number of possible edges. The vertex-centric measure we define focuses on typical vertices which usually have low degree.

Regardless, we believe that AUC has fundamental flaws and should not be used to measure link prediction performance.

1. Theoretical details

In this section, we prove Theorem 0.2. For the proof of our main theorem, we will use the following lemma stated first by Swanapoel (42).

Lemma 1.1. *[Rank lemma] Consider any square matrix $M \in \mathbb{R}^{n\times n}$. Then*

$$\mathrm{rank}(M) \geq \frac{|\sum_i M_{i,i}|^2}{\left(\sum_i \sum_j |M_{i,j}|^2\right)}$$

We prove our main theorem.

Proof. (of Theorem 0.2) Let us first bin the vectors based on their length (the Euclidean norm $\|\vec{v}_i\|_2$). For integer i, let B_i denote the set of vectors whose length is in $[2^i, 2^{i+1})$. By the polynomial length in bound, there are most $c \log_2 n$ bins of vectors, for some constant c.

Imagine labeling each vertex by the bin that its corresponding vector belongs to. We can label a pair (i, j) with the labels of i and j. Formally, the label of pair (i, j) is the label/bin pair (B_a, B_b) where $\vec{v}_i \in B_a$ and $\vec{v}_j \in B_b$. Observe that each vertex can have at most $c \log_2 n$ labels, hence the total number of pair labels is at most $c^2 \lg^2 n$. (We use lg to denote $\log_2$.) There are at least δn significant pairs. By averaging, there exists some pair label (B_a, B_b) (where a may be equal to b), such that there are at least $\delta n/c^2 \lg^2 n$ significant pairs with that label. In other words, for some values a, b, there are at least $\delta n/c^2 \lg^2 n$ significant pairs between B_a and B_b. Call these the *marked* significant pairs. Let $B'_a \subseteq B_a$ and $B'_b \subseteq B_b$ be the set of vectors in these bins that participate in the significant pairs between B_a and B_b.

By definition (Definition 0.1), each i can participate in at most $1/\varepsilon$ significant pairs. Hence, the number of marked significant pairs (which are all incident to B'_a) is at most $|B'_a|/\varepsilon$. As argued above, the number of marked significant pairs is at least $\delta n/c^2 \lg^2 n$. Hence, $|B'_a| \geq (\varepsilon\delta/c^2 \lg^2 n) \times n$. (Similarly, for $|B'_b|$.) For convenience, let C be $B'_a \cup B'_b$.

Let V' be the column matrix of the vectors in C. Let M be the Gram matrix $(V')^T(V')$. (For convenience, let us index M with respect to the original vertex labels. So $M_{ij} = \vec{v}_i \cdot \vec{v}_j$.). Observe that the rank of M is the rank of V', which is at most the original rank of V. By Lemma 1.1,

$$\mathrm{rank}(V) \geq \mathrm{rank}(V') = \mathrm{rank}(M) = \frac{|\sum_{i\in C} \vec{v}_i \cdot \vec{v}_i|^2}{\left(\sum_{i\in C} \sum_{k\in C} |\vec{v}_i \cdot \vec{v}_k|^2\right)} \quad [1]$$

Let us begin by lower bounding the numerator. Wlog, assume $a \leq b$. By definition, C contains B'_b. Moreover, $|B'_b| \geq (\varepsilon\delta/c^2 \lg^2 n) \times n$. For each vector in $B'_b \subseteq B_b$, the length is at least 2^b. Note that diagonal entry $\vec{v}_i \cdot \vec{v}_i$ is precisely the squared length. Hence, for any $i \in B'_b$, $\vec{v}_i \cdot \vec{v}_i \geq (2^b)^2$. Combining all our bounds,

$$\sum_{i \in C} |\vec{v}_i \cdot \vec{v}_i| \geq (\varepsilon\delta/c^2 \lg^2 n) \times n \times (2^b)^2 = (\varepsilon\delta/c^2 \lg^2 n) \times 2^{2b} \times n \quad [2]$$

We now upper bound the denominator of (1). Every vertex in C participates in a significant pair in C. Hence, for every $i \in C$, there exists a $j \in C$ such that j has probability at least ε in the distribution $\mathcal{D}_i$ (Definition 0.1). So,

$$\frac{|\vec{v}_i \cdot \vec{v}_j|}{\sum_{k \neq i} |\vec{v}_i \cdot \vec{v}_k|} \geq \varepsilon \implies |\vec{v}_i \cdot \vec{v}_j| \geq \varepsilon \sum_{k \in C} |\vec{v}_i \cdot \vec{v}_k|$$

By squaring and applying the l_1, l_2-inequality,

$$|\vec{v}_i \cdot \vec{v}_j|^2 \geq \varepsilon^2 (\sum_{k \in C} |\vec{v}_i \cdot \vec{v}_k|)^2 \geq \varepsilon^2 \sum_{k \in C} |\vec{v}_i \cdot \vec{v}_k|^2$$

By Cauchy-Schwartz, $|\vec{v}_i \cdot \vec{v}_j| \leq \|\vec{v}_i|_2 \|\vec{v}_j\|_2$. All the vectors are in C, where the length is at most the maximum length in B'_b. This bound is at most 2^{b+1}. Hence, we deduce that

$$(2^{b+1})^2 \geq \varepsilon^2 \sum_{k \in C} |\vec{v}_i \cdot \vec{v}_k|^2 \implies \sum_{k \in C} |\vec{v}_i \cdot \vec{v}_k|^2 \leq 4\varepsilon^{-2} 2^{2b}$$

We can now upper bound the denominator of (1). We have $\sum_{i \in C} \sum_{k \in C} |\vec{v}_i \cdot \vec{v}_k|^2 \leq 4\varepsilon^{-2} 2^{2b} n$. We combine this bound with the numerator bound of (2), and plug into (1)

$$\text{rank}(V) \geq \frac{[(\varepsilon\delta/c^2 \lg^2 n) \times 2^{2b} \times n]^2}{4\varepsilon^{-2} 2^{2b} n} \quad [3]$$

$$= \frac{(\varepsilon^4 \delta^2/c^4 \lg^4 n) \times 2^{2b} \times n^2}{4 \times 2^{2b} \times n} \quad [4]$$

We can cancel out $2^{2b} n$ to conclude that $\text{rank}(V) \geq (\varepsilon^4 \delta^2/4c^4 \lg^4 n) \times n$. □

A. On AUC and the rank bound. As a final point, we note that the rank lower bound (and poor reliability performance) is consistent with high AUC scores. We give a simple construction demonstrating a low dimensional embedding that gives nearly perfect AUC scores for an example sparse graph. This construction also highlights the primary weakness of AUC as a measure for link prediction.

Theorem 1.2. *There exists a connected bounded degree graph G with n vertices and a corresponding constant dimension embedding with the following properties. The AUC of edge prediction using the dot product score is $1 - o(1)$.*

Proof. Consider a graph G first formed by a disjoint collection of triangles. To make it connected, we add an arbitrary spanning tree that connects all the triangles. We will construct a randomized embedding with a high probability of having AUC $1 - o(1)$. By the probabilistic method, there must exist some embedding with such a large AUC.

For each triangle, let us assign an independent, Gaussian uniform random unit vector in $\mathbb{R}^d$. All vertices in the triangle are assigned to this vector. Hence for every edge (i, j), $\vec{v}_i \cdot \vec{v}_j = 1$. Now, consider i and k that are not part of a triangle. Observe that $\vec{v}_i \cdot \vec{v}_k$ is distributed as the dot product of independent Gaussian random vectors. This is distribution as $\mathcal{N}(0, 1/\sqrt{d})$, a standard Gaussian with $1/\sqrt{d}$ standard deviation. For sufficiently large constant c, the dot product is at most $1/4$ with high probability. Since a $3/4$-fraction of edges are in triangles, the average dot product of an edge is at least $3/4$. For a uniform random pair, with high probability, the score is at most $1/4$. Hence, the expected AUC is $1 - o(1)$. □

Note that in this construction there are $\Theta(n^2)$ dot products with value $\Theta(1/\sqrt{d})$. The distributions $\mathcal{D}_i$ will have a lot of noise, and the true signal (the dot product of 1) will have negligible mass.

2. Experimental results

We follow the standard setup for transductive link prediction experiments, as done in previous work. The experimental sections of the *node2vec* (7) or *role2vec* (18) papers provide good examples. The datasets are described in Tab. 3, taken from (22, 43). We also include two link prediction datasets from the Open Graph Benchmark (26), and two protein interaction datasets from (44, 45) We choose a dense dataset, `ogbl-ddi`, as a counterpoint to the other sparse datasets. In addition, we construct a small Stochastic Block Model. We create an SBM with a block size of 50 and 200 such blocks. Within each block, an edge is inserted with probability 0.3. Pairs across blocks are connected with probability $0.3/n$ (where n is the number of vertices). So blocks are extremely dense, and there are very few edges across blocks.

The edges of each dataset (graph) is split into a train and test set. We put 80% of the edges in the training set and 20% in the test set (called E_{test} earlier). We then apply an embedding method to embed each vertex into a 128-dimensional vector. We describe the embeddings methods in the next subsection.

We also follow the usual setting for transductive link prediction when computing the ROC-AUC and PR-AUC. First we randomly generate an equal number of negative edges to add to the test and train datasets. Now we train a logistic classifier over the training dataset that computes a Hadamard product to predict edges. (In most cases, the optimal Hadamard product is just the dot product.)

To compute the VCMPR plots, we sample 1000 uniform random vertices. For each sampled vertex i, we sort all other vertices in decreasing order of the Hadamard product used by the classifier. We remove all pairs from the training data. We then compute the precision and recall of the top k entries of this sorted list, where the ground truth is the neighborhood of i in E_{test}.

	Nodes	Edges	mean degree	clust. coeff.	Trans--itivity	Triangles
`amazon`	334K	925K	5	0.39	0.21	667K
`dblp`	317K	1.04M	6	0.63	0.31	2.24M
`blogcatalog`	10K	333K	64	0.46	0.09	5.61M
`sbm`	10K	75K	15	0.29	0.29	105K
`ogbl-collab`	235K	1.28M	9	0.71	0.34	3.66M
`ogbl-ddi`	4.27K	1.33M	625	0.63	0.57	27.3M
`bioplex`	11K	56.6K	10	0.10	0.06	29.3K
`HI-II-14`	4.3K	14K	6	0.05	0.03	6.62K

Table 3. Dataset Summary

Plots for `blogcatalog` are given in Fig. 1 and Tab. 1. Results for `ogbl-collab` are in Fig. 2 and Tab. 2, and for the SBM are in Tab. 4. All other plots and results are given in the SI.

Embedding Methods. We experiment on a large set of embeddings methods, that subsume random walk methods, factorization methods, and deep models. When available, we use the implementations

of these algorithms provided by the KarateClub library (40). We included HOP-Rec (37), a leading method for `ogbl-collab` on the OGB leaderboard (26). For completeness, we run some classic non-embedding methods, such as Common Neighbors (28), Adamic Adar (4) and Resource Allocation (36). These methods typically have good performance (within top 10) on the leaderboard datasets (26).

DeepWalk: DeepWalk (6) is a classic shallow embedding model that uses uniform random walks.

Node2Vec: Node2Vec (7) is another important shallow embedding model.

NetMF: NetMF (10) is a matrix factorization based model that approximates the DeepWalk matrix.

GraRep: GraRep (39) is a direct matrix factorization based model. We ran GraRep only on small graphs.

BoostNE: BoostNE (24) is an ensemble matrix factorization based embedding model.

RandNE: RandNE (23) is a random projection based embedding model.

Walklets: Walklets (8) is a random walk based embedding model that samples short random walks.

Role2Vec: Role2Vec (18) is a random walk based embedding model that embeds vertices based on attributed random walks.

GraphSage: GraphSage (9) is an deep learning based embedding algorithm. We use the unsupervised version of GraphSage, with the mean, LSTM and max-pooling aggregators.

HPE: HPE (38) is a random walk based embedding model that embeds vertices based on preference edges.

HOP-Rec: HOP-Rec (37) is an embedding model that combines random walks and matrix factorization.

A. Key observations.

High AUC values: Across all datasets and a majority of methods, the ROC-AUC and PR-AUC values are quite high (more than 0.8). For brevity, we only plot the ROC curves. We can see the characteristic "away from diagonal" trend that leads to high AUC. The AUC values are given in Tab. 1 and Tab. 4. Most methods give an AUC of at least 0.8 for all datasets; the maximum is at least 0.95. Consistently, the *Walklets* algorithm has strong performance in the AUC metric. The `dblp` dataset appears to be easier to learn, since all methods give high AUC scores.

Low VCMPR values: Across all methods and all datasets, the VCMPR values are quite low. We compute these values for $k = 10, 20, 50$ (that is, the top 10, 20, and 50 scores for each vertex). The low performance for $k = 10$ is quite surprising, suggesting that the top scores are almost always non-edges. Note that VCMPR should go up for larger k, since recall will always increase. The average degrees in all the graphs are well below 50. So the VCMPR@50 is basically measuring recall, with a list that is much larger than the degree. Despite that, the values are low. For the `amazon, dblp` datasets, the average degree is at most 6, which is quite small. For that reason, we only measure VCMPR@10 and VCMPR@20. (For $k = 50$, it would be measuring recall over a list of almost 10 times the size of ground truth list, which is not meaningful.)

For example, in the `blog-Catalog` dataset (Fig. 1, Tab. 1), the highest average VCMPR values is 0.21. Most methods have an average VCMPR value of less than 0.1. This means that, on the average, among the top 10 scores for a vertex i, at most *one* of them is a ground truth edge. We see the noise "drowning" out the signal, just as the theory in §1 suggests. For the `dblp` and `amazon` datasets, we see the same phenomenon, though the numbers are somewhat higher. The AUC scores for `amazon` are remarkably high, above 0.8. But the average VCMPR values are mostly at 0.3. Note that the average degree is 6, since VCMPR@20 is measuring how many of those neighbors are within the top 20 scores. This is rather permissive from a prediction standpoint, and yet the values are quite low. `Walklets` performs somewhat better, but nowhere near what the AUC would suggest. (An average VCMPR@10 of 0.38 means that, for an average vertex, at most 38% of the top 10 scores are actually edges.) We see a similar story with the `dblp` dataset.

These results are evidence that the low-dimensional node embedding methods are missing much of the graph structure. The high AUC scores are misleading indicators of link prediction performance. In numerous cases, a predictor with an AUC of more than 0.8 has an average VCMPR of less than 0.05. We also compute the local NDCG and consider metrics with both test and train data. The poor performance is consistent across all these measures (detailed discussion in the SI).

Quite surprisingly, we see this pattern even when evaluating non-embedding based link prediction methods, such as the high performing `Resource Allocation` method.

SBM experiments: The AUC values are nearly perfect, and one might think that prediction is extremely accurate. But the average VCMPR@10 values are quite low, typically around 0.25. Again, the high AUC scores hide the fact that the link prediction is not accurate. We see again the sharp contrast between AUC and the actual predictive power.

	ROC-AUC	PR-AUC	Avg VCMPR@10
Resource Allocation	0.96	0.98	0.26
Adamic Adar	0.96	0.98	0.26
Common Neighbors	0.96	0.98	0.26
HOP-Rec	0.99	0.99	0.26
HPE	0.99	0.99	0.25
NetMF	0.99	0.99	0.26
GraRep	0.99	0.99	0.24
Walklets	0.99	0.99	0.25
Node2Vec	0.98	0.98	0.23
Deepwalk	0.98	0.98	0.24
BoostNE	0.98	0.98	0.16
RandNE	0.98	0.98	0.24
Role2Vec	0.97	0.97	0.21
GraphSage-MP	0.76	0.82	0.11
GraphSage-M	0.72	0.77	0.06
GraphSage-L	0.69	0.71	0.02

Table 4. We run our experiments on a small SBM. Again we see the stark contrast between AUC and VCMPR scores. AUC scores are near perfect. But the VCMPR scores are quite low, under 0.25. See Fig. 7

On other metrics. We get analogous results for VCNDCG (vertex-centric Normalized Discounted Cumulative Gain). The VCNDCG plots are quite low, and average scores are lower than 0.2. This emphasizes that the vertex-centric view highlights the weaknesses of embedding-based link prediction. (Plots and more discussion in the SI.)

We also experimented with the classic hits@k metric, which is popular for knowledge completion tasks (26, 35, 46), . As discussed in the introduction, the hits@k is a global metric where a predicted score is compared against a sorted list of scores for ground truth edges and non-edges. We note that the parameter k is incomparable for hits@k and VCMPR@k, since the former considers a total list of edges/pair, while the latter considers a list of edges/pairs for each vertex. Tab. 2 and Fig. 2 highlight that hits@k can be quite large, while corresponding VCMPR scores can be low. We give more details for other datasets in the SI.

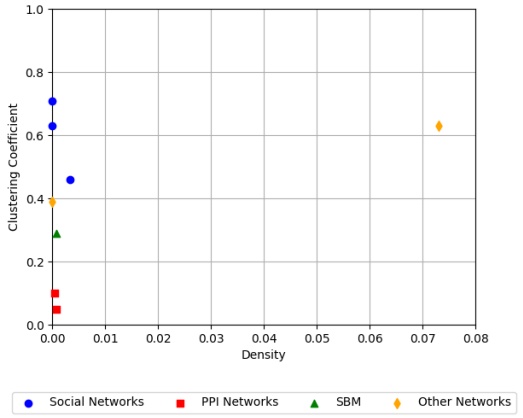


Fig. 3. We plot the network density vs clustering coefficient for all networks listed in Tab. 3 The social networks includes the `blogcatalog`, `ogbl-collab`, and `dblp` datasets. The PPI networks refers to `bioplex` and `HI-II-14`, while other networks refers to `amazon`, a product co-purchasing dataset and `ogbl-ddi`, a drug interaction dataset.

The connection to density. The theory and experiments related the weakness of AUC for sparse link prediction problems, where the VCMPR is able to identify weak performance. As a counterpoint, we do experiments on the `ogbl-ddi` link prediction benchmark, which is a dense graph. The average degree is more than one-tenth of the number of vertices. For this data, the VCMPR numbers are higher and comparable to the AUC scores. More details in the SI.

Results on PPI networks, and the clustering coefficient connection. In Fig. 3, we plot the network density vs clustering coefficient for all the datasets we experiment with. Observe that most of the social networks lie far above the PPI networks (which have low clustering). For the latter setting, our mathematical analysis is not as applicable. We perform the same collection of link prediction experiments on the PPI datasets. We observe that AUC scores tend to be lower (closer to 0.5). It is likely the AUC does not suffer from the same problem for other datasets. There are a few algorithms, like *Walklets*, *HPE*, and *HOP-rec* that have high AUC. Nonetheless, in all cases, the VCMPR scores are quite low, consistent with other datasets. More details in the SI.

Exploration of dot products

In this section, we do a deeper investigation of the actual dot product values, to corroborate the theory in §1. Theorem 0.2 implies that when the rank of the vectors is low, the sparse ground truth cannot have significantly higher dot products than the non-edges. As a case study, we focus on the `blogCatalog` dataset, though our results are consistent over other graphs.

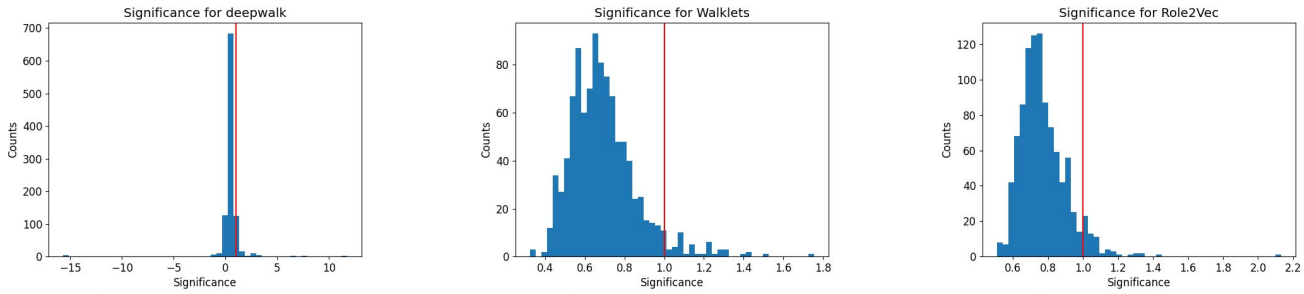
Consider the set of vectors $\{\vec{v}_i\}$ output by some embedding method for `blogCatalog`. For each vertex i, we first compute the average dot product $\vec{v}_i \cdot \vec{v}_j$, for all edges (i, j) in the test set. Call this quantity t_i. Then, we compute the average dot product $\vec{v}_i \cdot \vec{v}_j$ for the top 50 non-edges (i, j). Call this quantity f_i. If we want to predictor to have detected the ground truth, we would want $t_i \gg f_i$.

So let us define the *significance* of i to be t_i/f_i. This is the ratio of the score of an average ground truth edge and the score of an average top-50 non-edge. The distributions (for a uniform random sample of 1000 vertices) are given in Fig. 4 for the *Deepwalk*, *Walklets*, and *Role2Vec* embedding methods. We mark the significance of 1 as a red line. (For *Deepwalk*, there are some negative significance values, since the average dot product of test edges for some i is negative.)

For a predictor that captures the ground truth, we would expect the distribution to have mass much to the right of the red line. As our theory predicts for the low-dimensional vectors, this is *not* the case. The distributions is to the left of the red line, so most vertices have a significance less than 1. This means that among the top 50 scores, non-edges dominate edges.

ACKNOWLEDGMENTS. C. Seshadhri was supported by NSF DMS-2023495, CCF-1740850, 1839317, 1908384, 2245904, and ARO Award W911NF1910294.

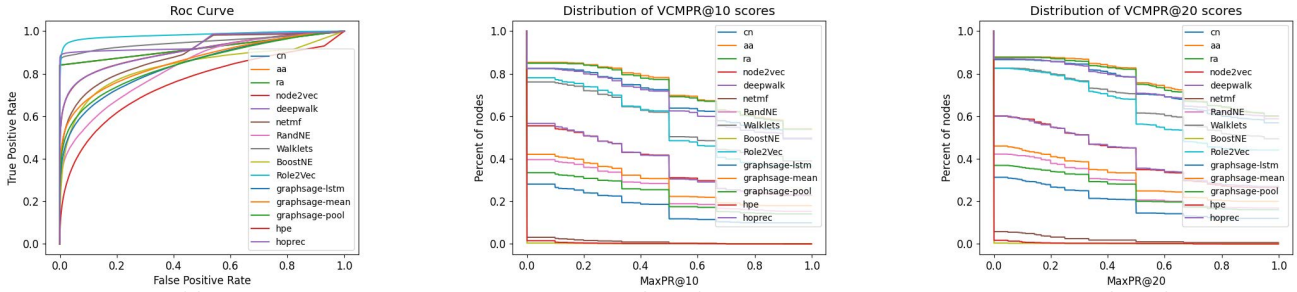
1. W Hamilton, Z Ying, J Leskovec, Inductive representation learning on large graphs in *Neural Information Processing Systems (NeurIPS)*. pp. 1024–1034 (2017).
2. I Chami, S Abu-El-Haija, B Perozzi, C Ré, K Murphy, Machine learning on graphs: A model and comprehensive taxonomy. *arXiv:2005.03675* (2020).
3. KP Murphy, *Probabilistic Machine Learning: An introduction*. (MIT Press), (2021).
4. L Adamic, E Adar, Friends and neighbors on the web. *Soc. Networks* **25** (2003).
5. D Liben-Nowell, J Kleinberg, The link prediction problem for social networks. *J. Am. Soc. for Inf. Sci. Technol.* **58**, 1019—-1031 (2007).
6. B Perozzi, R Al-Rfou, S Skiena, DeepWalk: Online learning of social representations in *Conference on Knowledge Discovery and Data Mining (KDD)*. (ACM Press), pp. 701–710 (2014).
7. A Grover, J Leskovec, node2vec: Scalable feature learning for networks in *Conference on Knowledge Discovery and Data Mining (KDD)*. pp. 855–864 (2016).
8. B Perozzi, V Kulkarni, H Chen, S Skiena, Don't walk, skip! online learning of multi-scale network embeddings in *Advances in Social Networks Analysis and Mining*. p. 258–265 (2017).
9. W Hamilton, Z Ying, J Leskovec, Inductive representation learning on large graphs in *Neural Information Processing Systems (NeurIPS)*. p. 11 (2017).
10. J Qiu, et al., Network embedding as matrix factorization: Unifying DeepWalk, LINE, PTE, and node2vec in *Conference on Web Science and Data Mining (WSDM)*. pp. 459–467 (2018).
11. S Gurukar, et al., Network representation learning: Consolidation and renewed bearing. *arXiv* **abs/1905.00987** (2019).
12. A Ghasemian, H Hosseinmardi, A Galstyan, A Clauset, Stacking models for nearly optimal link prediction in complex networks. *Proc. Natl. Acad. Sci. (PNAS)* **117**, 23393–23400 (2020).
13. A Mara, J Lijffijt, TD Bie, Benchmarking network embedding models for link prediction: Are we making progress? in *International Conference on Data Science and Advanced Analytics (DSAA)*. (2020).
14. L Torres, KS Chan, A Galstyan, T Eliassi-Rad, Glee: Geometric laplacian eigenmap embedding. *J. Complex Networks* **8** (2020).
15. W Huang, Y Li, Y Fang, J Fan, H Yang, Biane: Bipartite attributed network embedding in *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. (Association for Computing Machinery), Vol. 3, p. 149–158 (2020).
16. L Wang, C Huang, W Ma, X Cao, S Vosoughi, Graph embedding via diffusion-wavelets-based node feature distribution characterization in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, CIKM '21. p. 3478–3482 (2021).
17. J Qiu, L Dhulipala, J Tang, R Peng, C Wang, Lightne: A lightweight graph processing system for network embedding in *Proceedings of the 2021 International Conference on Management of Data*. (Association for Computing Machinery), p. 2281–2289 (2021).
18. NK Ahmed, et al., Role-based graph embeddings. *IEEE Transactions on Knowl. Data Eng.* **34**, 2401–2415 (2022).
19. L Cappelletti, et al., Grape for fast and scalable graph processing and random-walk-based embedding. *Nat. Comput. Sci.* **3**, 552–568 (2023).
20. DJ Hand, Measuring classifier performance: a coherent alternative to the area under the ROC curve. *Mach. Learn.* **77**, 103–123 (2009).
21. R Lichtenwalter, NV Chawla, Link prediction: Fair and effective evaluation in *Advances in Social Networks Analysis and Mining*. pp. 376–383 (2012).
22. R Zafarani, H Liu, Social computing data repository at ASU (2009).
23. Z Zhang, P Cui, H Li, X Wang, W Zhu, Billion-scale network embedding with iterative random projection. pp. 787–796 (2018).
24. J Li, L Wu, H Liu, Multi-level network embedding with boosted low-rank matrix approximation in *Advances in Social Networks Analysis and Mining*. pp. 49–56 (2019).
25. T Fawcett, An introduction to roc analysis. *Pattern Recognit. Lett.* **117**, 861—-874 (2006).
26. W Hu, et al., Open graph benchmark: Datasets for machine learning on graphs in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS'20. (Curran Associates Inc., Red Hook, NY, USA), (2020).
27. W Krichene, S Rendle, On sampled metrics for item recommendation in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20. (Association for Computing Machinery, New York, NY, USA), p. 1748–1757 (2020).
28. D Liben-Nowell, J Kleinberg, The link prediction problem for social networks in *Proceedings of the Twelfth International Conference on Information and Knowledge Management*, CIKM '03. (Association for Computing Machinery, New York, NY, USA), p. 556–559 (2003).
29. MA Hasan, MJ Zaki, *A Survey of Link Prediction in Social Networks*, ed. CC Aggarwal. (Springer US, Boston, MA), pp. 243–275 (2011).
30. S Chanpuriya, C Musco, K Sotiropoulos, CE Tsourakakis, Node embeddings and exact low-rank representations of complex networks in *Neural Information Processing Systems (NeurIPS)*. (2020).
31. C Seshadhri, A Sharma, A Stolman, A Goel, The impossibility of low-rank representations for triangle-rich complex networks. *Proc. Natl. Acad. Sci.* **117**, 5631–5637 (2020).
32. A Loukas, What graph neural networks cannot learn: depth vs width in *International Conference on Learning Representations*. (2020).
33. VK Garg, S Jegelka, T Jaakkola, Generalization and representational limits of graph neural networks. *arXiv:2002.06157* (2020).



(a) Significance plot for deepwalk **(b)** Significance plot for Walklets **(c)** Significance plot for role2vec

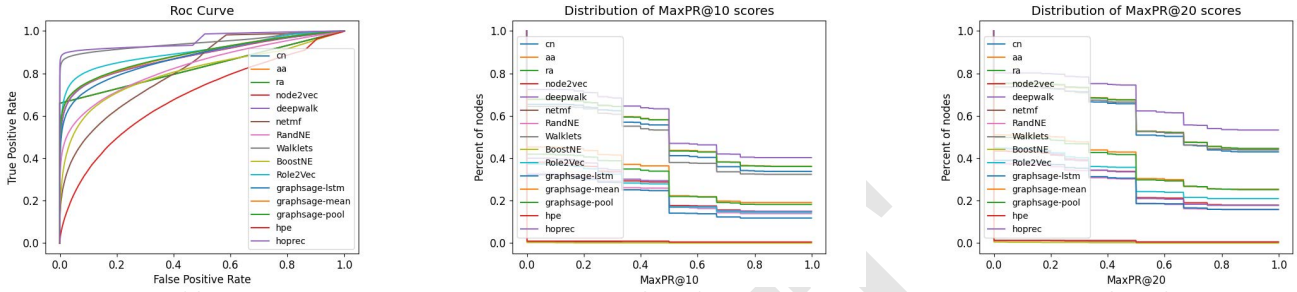
Fig. 4. We plot the distributions of significance values for the embedding vectors generated by different methods, for the `blog-Catalog` dataset. Roughly speaking, the significance value for a vertex i is the average dot product with its neighbor vectors divided the average of the top 50 $\vec{v}_i \cdot \vec{v}_j$ values (varying over j). The plot gives the distribution (over vertices) of significance values. For ground truth to be captured, we would need the significance to be much higher than 1. Our theory predicts that this will *not* happen, and most significance values will be much smaller. This is exactly what we see in practice. The distribution of significance values is much lower than the red line, which is at 1.

34. A Stolman, C Levy, C Seshadhri, A Sharma, Classic graph structural features outperform factorization-based graph embedding methods on community labeling in *SIAM Conference on Data Mining (SDM)*. pp. 388–396 (2022).
35. A Bordes, N Usunier, A Garcia-Duran, J Weston, O Yakhnenko, Translating embeddings for modeling multi-relational data in *Advances in Neural Information Processing Systems*, eds. C Burges, L Bottou, M Welling, Z Ghahramani, K Weinberger. (Curran Associates, Inc.), Vol. 26, (2013).
36. LLYCZ Tao Zhou, Predicting missing links via local information. *The Eur. Phys. J. B* **25**, 623–630 (2009).
37. JH Yang, CM Chen, CJ Wang, MF Tsai, Hop-rec: High-order proximity for implicit recommendation in *Proceedings of the 12th ACM Conference on Recommender Systems*, RecSys '18. (Association for Computing Machinery, New York, NY, USA), p. 140–144 (2018).
38. CM Chen, MF Tsai, YC Lin, YH Yang, Query-based music recommendations via preference embedding in *Proceedings of the 10th ACM Conference on Recommender Systems*, RecSys '16. (Association for Computing Machinery, New York, NY, USA), p. 79–82 (2016).
39. S Cao, W Lu, Q Xu, GraRep: Learning graph representations with global structural information in *Conference on Information and Knowledge Management (CIKM)*. pp. 891–900 (2015).
40. B Rozemberczki, O Kiss, R Sarkar, Karate Club: An API Oriented Open-source Python Framework for Unsupervised Learning on Graphs in *Conference on Information and Knowledge Management (CIKM)*. (ACM), (2020).
41. K Xu, W Hu, J Leskovec, S Jegelka, How powerful are graph neural networks? in *International Conference on Learning Representations*. (2019).
42. K Swanapoel, The rank lemma (https://konradswanepoel.wordpress.com/2014/03/04/the-rank-lemma/) (2014).
43. J Yang, J Leskovec, Defining and evaluating network communities based on ground-truth in *Proceedings of the ACM SIGKDD Workshop on Mining Data Semantics*, MDS '12. (Association for Computing Machinery, New York, NY, USA), (2012).
44. T Rolland, et al., A proteome-scale map of the human interactome network. *Cell* **159**, 1212–1226 (2014).
45. EL Huttlin, et al., Architecture of the human interactome defines protein communities and disease networks. *Nature* **545**, 505–509 (2017).
46. M Zhang, P Li, Y Xia, K Wang, L Jin, Revisiting graph neural networks for link prediction in *Internation Conference on Learning Representations*. (2021).
47. K Järvelin, J Kekäläinen, Cumulated gain-based evaluation of ir techniques. *ACM Trans. Inf. Syst.* **20**, 422–446 (2002).



(a) ROC curve **(b)** VCMPR@10 **(c)** VCMPR@20

Fig. 5. We show results for `dblp` dataset (43). The left figure has the standard ROC curves for each predictor. In the middle and right figures, we plot the distribution of VCMPR@k values, for $k = 10, 20$. The `dblp` dataset has a small degree of 6, which is smaller than the choice of k. The ROC curves are quite high, consistent with the literature. But the VCMPR values show that the predictor performance is not strong. The data is summarized in Tab. 5.



(a) ROC curve **(b)** VCMPR@10 **(c)** VCMPR@20

Fig. 6. We show results for `amazon` dataset (43). The `amazon` dataset also has a small average degree of 5, so $k = 20$ is significantly large for this dataset. The AUC scores are high, more than 0.8 for almost all cases. The VCMPR values are quite low. Even for $k = 20$, the plots are quite low. There are very few vertices with VCMPR values above 0.7. The data is summarized in Tab. 6.

Supporting Information for "Link prediction using low-dimensional node embeddings: the measurement problem"

A. Details on experiments

We give details on all the embedding methods used in our experiments. We have included HOP-REC, an embedding method that performs highly on the Open Graph Benchmark link prediction leaderboards (26). We explore general node embedding models, such as Node2Vec (7), NetMF (10), Role2Vec (18), as well as node embedding models specific designed for recommendations, such as Heterogenous Preference Embedding (HPE) (38) and HOP-Rec (37). Also, for the sake of completeness, we show results on some classic non-embedding methods, such as Common Neighbors (28), Adamic Adar (4), and Resource Allocation (36).

DeepWalk: DeepWalk (6) is a classic shallow embedding model that uses uniform random walks paired with a skip-gram language model to learn a low dimensional embedding. DeepWalk aims to embed nodes with similar neighborhoods (nodes with high second order proximity) close together. We ran DeepWalk with the following parameters, dimension $d = 128$, window size $w = 5$, walks per vertex $\gamma = 10$, walk length $t = 80$.

Node2Vec: Node2Vec (7) is another important shallow embedding model that expands on the random walks used in DeepWalk by introducing a transition probability associated with the 2nd order random walks. These two parameters correspond to the random walk behaving more like BFS or DFS, and is meant to help balance between the walk staying near the start vertex and exploring the graph. We ran Node2Vec with the following parameters, dimension $d = 128$, window size $w = 5$, walks per vertex $\gamma = 10$, walk length $t = 80$, $p = 1$, $q = 1$.

NetMF: NetMF (10) is a matrix factorization based model that approximates the DeepWalk matrix for a graph. The NetMF paper also showed that the previous random walk based models can be expressed in terms of matrix factorization with closed forms. We ran NetMF with the following parameters, dimension $d = 128$, 10 SVD iterations, and 2 PMI matrix powers.

BoostNE: BoostNE (24) is an ensemble matrix factorization based embedding model. It iteratively factorizes the residual of the connectivity matrix found by NetMF to produce multiple weak embedding representations. These are combined with a gradient boosting technique to produce a final embedding. We ran BoostNE with the following parameters $d = 128$, 16 boosting iterations, and for the NetMF calls, we use 2 PMI matrix powers.

RandNE: RandNE (23) is a random projection based embedding model that embeds the graph by using a Gaussian random projection. We ran RandNE with dimension $d = 128$, iterations $q = 2$ and an unweighted average $\alpha_0 = \alpha_1 = 0.5$.

Walklets: Walklets (8) is a random walk based embedding model that generates multi-scale relationships of vertices by sub sampling short random walks on the vertices of the graph. The parameters we choose for Walklets are dimension $d = 128$, window size $w = 4$, walks per vertex $\gamma = 10$, walk length $t = 80$.

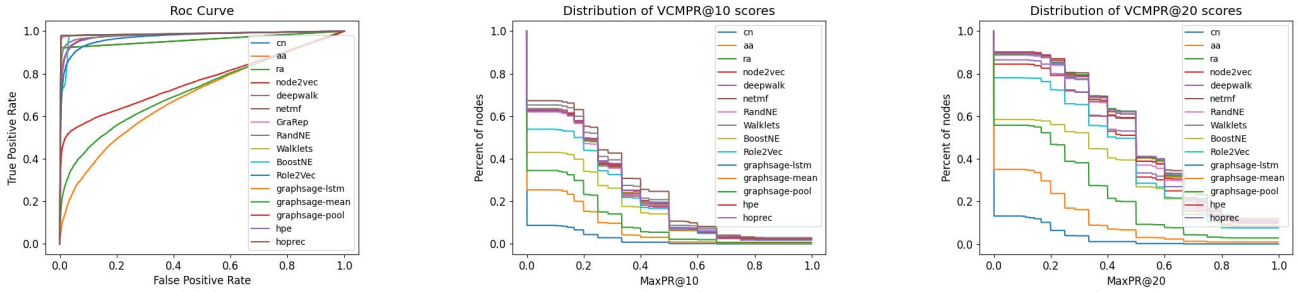
Role2Vec: Role2Vec (18) is a random walk based embedding model that embeds vertices based on attributed random walks. An attributed random walk is a random walk on adjacent vertex types, where a type is defined by Weisfeiler-Lehman structural features. We set the parameters of Role2Vec to be $d = 128$, window size $w = 2$, walks per vertex $\gamma = 10$, walk length $t = 80$.

GraphSage: GraphSage (9) is an deep learning based embedding algorithm. We use the unsupervised version of GraphSage, with the mean, LSTM and max-pooling aggregators. We set the parameters of GraphSage to have output dimension $d = 128$, number of iterations 10,000, and the identity dimension to be 128.

HPE: HPE (38) is an embedding method specifically designed for recommender systems. It first constructs a preference matrix, then creates an embedding of this preference matrix using random walks. We set the parameters of HPE to be dimension to be 128, number of negative samples per positive to be 5, window size 5, and learning rate 0.025.

HOP-REC: HOP-REC (37) is a embedding method designed specifically for recommendations. It combines factorizations of the adjacency matrix and higher order matrices approximated by random walks. We set the parameters of HOP-REC to be dimension to be 128, number of updates to be 500, number of negative samples per positive to be 5, window size 5, and learning rate 0.025.

A. Results on different datasets. We analyze the above models on the following datasets. The `amazon` product co-purchasing network represents products that are frequently purchased together as edges between vertices in the graph (43). The `dblp` collaboration network is an unweighted graph where edges represents two authors publishing 1 or more papers together (43). The `blog-Catalog` social network represents bloggers and friendships between them (22). The `ogbl-ddi` network represents interactions between different drugs (26). The `ogbl-collab` network represents authors and collaborations between them (26). These graphs vary in size, average degree,



(a) ROC Curve **(b)** VCMPR@10 **(c)** VCMPR@deg20

Fig. 7. We show results for a small SBM. The SBM has a block size of 50 and 200 blocks. The AUC scores are exceptional, over 0.97 for almost all cases. However the VCMPR values are quite low. The VCMPR@10 scores are quite low, under 0.3. Even for $k = 20$, there are no methods that score above 0.5, despite the fact that at this threshold is more than half of the size of the block with the training edges removed. The data is summarized in Tab. 4.

	ROC AUC	PR AUC	Avg VCMPR @10	Avg VCMPR @20
Common Neighbors	0.92	0.96	0.65	0.71
Adamic Adar	0.92	0.96	0.70	0.75
Resource Allocation	0.92	0.96	0.70	0.74
Role2Vec	0.98	0.99	0.52	0.60
Walklets	0.95	0.97	0.54	0.63
HOP-Rec	0.95	0.97	0.64	0.71
Deepwalk	0.90	0.93	0.37	0.41
Node2Vec	0.90	0.93	0.40	0.43
NetMF	0.89	0.91	0.02	0.03
GraphSage-M	0.86	0.89	0.27	0.30
GraphSage-MP	0.84	0.88	0.23	0.26
GraphSage-L	0.84	0.87	0.17	0.20
BoostNE	0.83	0.87	0.00	0.00
RandNE	0.83	0.85	0.26	0.28
HPE	0.75	0.79	0.00	0.01

Table 5. This table complements Fig. 5, which has results from link prediction on the `dblp` dataset. We give the ROC-AUC, PR-AUC, and average VCMPR@k for $k = 10, 20$. The AUC numbers are extremely large for real data, close to 0.9, with all methods showing good scores. Comparatively, the average VCMPR@10 numbers are low. `Walklets` gets a score of 0.56, but other methods are below 0.4.

	ROC AUC	PR AUC	Avg VCMPR @10	Avg VCMPR @20
Adamic Adar	0.83	0.91	0.51	0.60
Resource Allocation	0.83	0.91	0.52	0.60
Common Neighbors	0.82	0.91	0.49	0.58
HOP-Rec	0.96	0.97	0.56	0.67
Walklets	0.95	0.96	0.47	0.59
Role2Vec	0.90	0.93	0.24	0.31
Deepwalk	0.88	0.91	0.26	0.29
Node2Vec	0.88	0.91	0.25	0.29
GraphSage-M	0.88	0.92	0.31	0.37
GraphSage-MP	0.88	0.92	0.29	0.37
GraphSage-L	0.87	0.90	0.21	0.26
RandNE	0.83	0.87	0.22	0.26
NetMF	0.81	0.82	0.00	0.01
BoostNE	0.80	0.85	0.00	0.00
HPE	0.69	0.71	0.01	0.01

Table 6. This table complements Fig. 6, which has results from link prediction on the `amazon` dataset. We give the ROC-AUC, PR-AUC, and average VCMPR@k for $k = 10, 20$. The AUC numbers are quite high, with a highest of 0.97. But the average VCMPR number is quite low. Even the highest for `Walklets` is at 0.58, while other methods are lower than 0.4. The average degree is `amazon` is 5, so a choice of $k = 20$ is quite large.

and clustering coefficient as shown in Tab. 3. We train each model on 90% of the edges of the graph and withhold 10% for testing. The results for `amazon` can be found in Fig. 6 and Tab. 6, for `dblp` in Fig. 5 and Tab. 5, for `blog-Catalog` in Fig. 1 and Tab. 1, for `ogbl-collab` in Fig. 2 and Tab. 2, and for `ogbl-ddi` in Fig. 8 and Tab. 7.

We do some small scale experiments with simple Stochastic Block Models (SBMs) to make our point more compelling. We create an SBM with a block size of 50 and 200 such blocks. Within each block, an edge is inserted with probability 0.3. Pairs across blocks are connected with probability $0.3/n$ (where n is the number of vertices). So blocks are extremely dense, and there are few edges across blocks. We show the results in Tab. 4 and Fig. 7.

B. The connection to graph density. We show results on the dense `ogbl-ddi` dataset. In Tab. 7, we can see that VCMPR scores are quite high, and in many cases, almost the same as AUC scores. For the leading `HOP-rec` and `Walklets` algorithms, the scores are quite close to each other. This is an empirical converse of our main result theory that connects poor link prediction performance for low-dimensional embeddings to sparsity of the ground truth data. When the ground truth is dense, then both AUC and VCMPR suggest that the algorithms are performing well in link prediction.

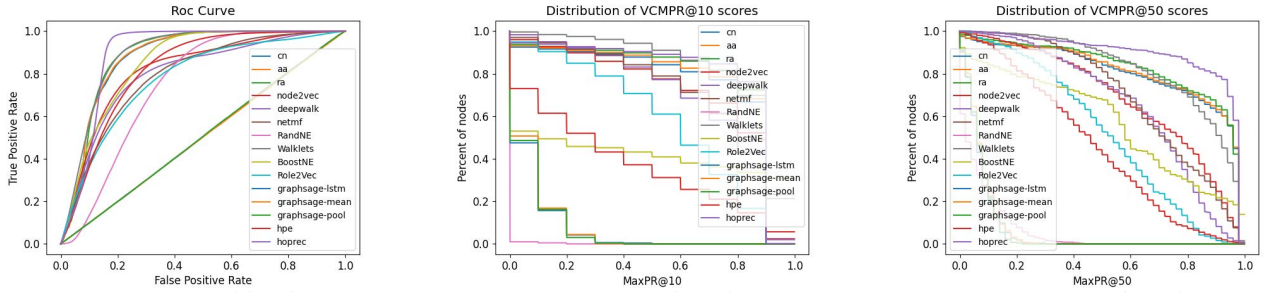
C. Results on PPI datasets. Unlike other datasets, PPI networks tend to have low clustering coefficients. Hence, they are not covered by our theoretical analysis. We perform the same link prediction experiments on the PPI datasets, `Bioplex` (45) and `HI-II-14` (44), described by Fig. 9 and Fig. 10. We observe that AUC scores are already quite low, typically between 0.6 and 0.8. There are few notable examples like *Walklets* and *HOP-rec* on the `Bioplex` where AUC scores are above 0.85, despite having VCMPR@10 scores below 0.2 and VCMPR@50 scores below 0.35. Observe that *NetMF* has higher VCMPR scores than *Walklets*, despite having lower ROC-AUC and PR-AUC scores. But in all cases, the VCMPR scores are extremely low. Overall, AUC is typically sufficient to demonstrate poor performance of link prediction algorithms for such datasets. Hence, VCMPR may have limited utility for such settings.

D. Exploration of Normalized Discounted Cumulative Gain. We also compute a vertex-centric Normalized Discounted Continuous Gain (NDCG). NDCG is a metric that analyzes how well a ranking method ranks relevant documents (47). Similarly to VCMPR, we compute VCNDCG@k for some threshold k. Formally, VCNDCG@k is computed as follows. For a given vertex i of non-zero degree, we rank all other vertices j in decreasing order of their scores. Let L be a list of the binary ground truth values ordered by their ranking. Let I, the ideal ranking, be a list of binary ground truth values in sorted order. We consider all ground-truth lists only contain values of 0 or 1, i.e. relevant or not. Then VCNDCG@k for the vertex i is defined as

$$\text{VCDCG@k} = \sum_{s=1}^{k} \frac{L_s}{\log_2(s+1)} \qquad \text{VCIDCG@k} = \sum_{s=1}^{k} \frac{I_s}{\log_2(s+1)}$$

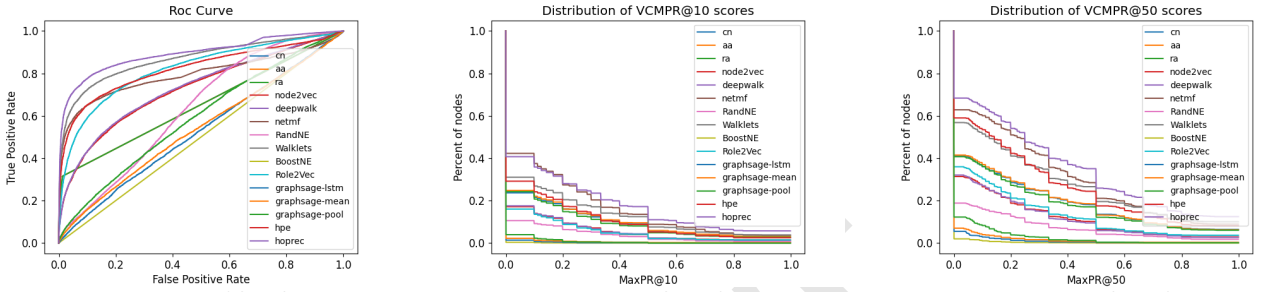
$$\text{VCNDCG@k} = \frac{VCDCG@k}{VCIDCG@k}$$

We compute VCNDCG for all data-sets and plot their scores in Fig. 12. Just as with VCMPR, the scores are quite low, which indicates that the low-dimensional embeddings have poor overall performance.



(a) `ogbl-ddi` ROC-AUC **(b)** `ogbl-ddi` VCMPR@10 **(c)** `ogbl-ddi` VCMPR@50

Fig. 8. We compute VCMPR curves for the `ogbl-ddi` dataset.



(a) `bioplex` ROC-AUC **(b)** `bioplex` VCMPR@10 **(c)** `bioplex` VCMPR@50

Fig. 9. We compute VCMPR curves for the `bioplex` dataset.

	ROC-AUC	PR-AUC	Avg VCMPR @10	Avg VCMPR @50
Common Neighbors	0.88	0.80	0.77	0.82
Adamic Adar	0.88	0.80	0.78	0.82
Resource Allocation	0.88	0.81	0.80	0.84
Deepwalk	0.80	0.75	0.71	0.66
Node2Vec	0.82	0.78	0.72	0.68
NetMF	0.74	0.74	0.74	0.70
RandNE	0.75	0.63	0.00	0.06
Walklets	0.89	0.83	0.84	0.83
BoostNE	0.85	0.79	0.41	0.57
Role2Vec	0.78	0.74	0.59	0.52
GraphSage-M	0.50	0.50	0.08	0.08
GraphSage-MP	0.50	0.50	0.07	0.07
GraphSage-L	0.50	0.50	0.07	0.07
HPE	0.82	0.76	0.38	0.46
HOP-rec	0.89	0.81	0.82	0.90

Table 7. This table complements Fig. 8, which has results from link prediction on the `ogbl-ddi` dataset. We give the ROC-AUC, PR-AUC average VCMPR@k for $k = 10, 50$.

	ROC-AUC	PR-AUC	Avg VCMPR @10	Avg VCMPR @50
Common Neighbors	0.65	0.81	0.10	0.19
Adamic Adar	0.65	0.81	0.10	0.19
Resource Allocation	0.65	0.81	0.10	0.18
Deepwalk	0.73	0.76	0.06	0.12
Node2Vec	0.73	0.76	0.06	0.11
NetMF	0.80	0.85	0.16	0.31
RandNE	0.63	0.61	0.04	0.07
Walklets	0.86	0.90	0.13	0.27
BoostNE	0.50	0.75	0.00	0.00
Role2Vec	0.82	0.84	0.05	0.08
GraphSage-M	0.54	0.55	0.01	0.02
GraphSage-MP	0.59	0.59	0.01	0.02
GraphSage-L	0.53	0.53	0.00	0.01
HPE	0.82	0.86	0.11	0.26
HOP-rec	0.89	0.92	0.18	0.34

Table 8. This table complements Fig. 9, which has results from link prediction on the `Bioplex` dataset. We give the ROC-AUC, PR-AUC, average VCMPR@k for $k = 10, 50$.

E. Comparison with Hits@k. As mentioned in the main body, the hits@k is a global metric appropriate for knowledge completion tasks. The natural choice of k for VCMPR is the average degree (or maybe twice average degree). For the hits metric, the k can vary depending on the instance. On the OGBL leaderboard, common choices are $k = 20, 50$. We performed a comparison of all methods on the `ogbl-collab` dataset, where we see that VCMPR scores are much lower than then hits scores.

We also perform the same experiments on the `sbm` datasets we generated. We simply set the k parameter to 20, for both VCMPR and hits. We see that again, hits scores are significantly higher than VCMPR scores. The results are summarized in Tab. 11. For example, the HOP-rec method has a hits@20 value of 0.83, but the average VCMPR@20 is 0.49. There are numerous hits values above 0.65, where the average VCMPR@20 is less than 0.5.

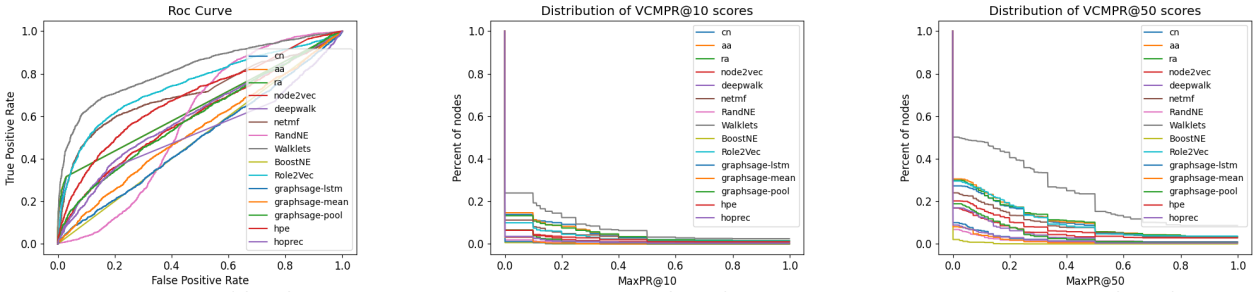
For the dense `ogbl-ddi` dataset, the recommended metric on OGBL is hits@20. Here, we see the opposite: VCMPR scores are large, but the hits scores are low. This is another indication that the hits@k and VCMPR metrics are fundamentally different.

F. Results on entire datasets. For the same setup as described in the paper, we compute VCMPR@k plots for the entire dataset, not just the graph consisting of E_{test}. This includes edges seen in training. This allows us to closely examine the local structure of the embedding. Under this setting, VCMPR@k is defined as follows.

$$\text{VCMPR@k for vertex } i = \frac{t_i(k)}{\min(k, D_i)}$$

where D_i is the degree of vertex i in the entire graph $G = (V, E)$. We report the scores in Fig. 13 and Fig. 14. As expected the scores are much higher. However, we see that in general, across all data-sets, thresholds, and methods, very few vertices have a VCMPR of 0.8, despite the embedding having seen 80% of the graph's edges. As before VCMPR curves drop quite steeply. This shows that the low-dimensional node embedding methods are not capturing much of the graph structure.

G. Results using a variable threshold for VCMPR. We also investigate the use of a variable threshold in our evaluation. We compute VCMPR as follows. Let D_i be the degree of vertex i in the original graph $G = (V, E)$



(a) HI-II-14 ROC-AUC **(b)** HI-II-14 VCMPR@10 **(c)** HI-II-14 VCMPR@50

Fig. 10. We compute VCMPR curves for the HI-II-14 dataset.

	ROC-AUC	PR-AUC	Avg VCMPR @10	Avg VCMPR @50
Common Neighbors	0.64	0.79	0.05	0.10
Adamic Adar	0.65	0.79	0.05	0.11
Resource Allocation	0.65	0.79	0.05	0.11
Deepwalk	0.61	0.62	0.02	0.04
Node2Vec	0.60	0.62	0.02	0.04
NetMF	0.71	0.75	0.03	0.09
RandNE	0.58	0.50	0.01	0.01
Walklets	0.82	0.85	0.08	0.24
BoostNE	0.50	0.75	0.00	0.00
Role2Vec	0.75	0.77	0.03	0.05
GraphSage-M	0.53	0.54	0.00	0.01
GraphSage-MP	0.60	0.62	0.01	0.04
GraphSage-L	0.50	0.51	0.00	0.02
HPE	0.69	0.70	0.02	0.07
HOP-rec	0.53	0.58	0.01	0.02

Table 9. This table complements Fig. 10, which has results from link prediction on the HI-II-14 dataset. We give the ROC-AUC, PR-AUC, average VCMPR@k for $k = 10, 50$.

and d_i be the degree of i in the test graph $G_{test} = (V, E_{test})$. Then VCMPR@Deg(v) is defined as follows:

$$\text{VCMPR@Deg(i) for vertex } i = \frac{t_i(D_i)}{\min(D_i, d_i)}$$

We set D_i and d_i to not be the same to ensure each threshold is sufficiently large. This experiment is motivated by the fact that in some graphs, the degree of vertices tends to obey a power law distribution. Under such a distribution, a small portion of vertices are incident to a large portion of the edges. Thus, variable thresholds may be more appropriate. The results of these experiments are in Fig. 11 and Tab. 12, where we see low scores as before. The performance of all methods is extremely low in comparison to the AUC scores.

	blogcatalog	amazon	dblp	sbm	collab	ddi	bioplex	HI-II-14
	Avg VCNDCG @50	Avg VCNDCG @10	Avg VCNDCG @10	Avg VCNDCG @10	Avg VCNDCG @10	Avg VCNDCG @50	Avg VCNDCG @10	Avg VCNDCG @10
Common Neighbors	0.23	0.30	0.49	0.15	0.21	0.82	0.07	0.03
Adamic Adar	0.24	0.31	0.53	0.15	0.23	0.82	0.07	0.03
Resource Allocation	0.24	0.31	0.2	0.15	0.23	0.84	0.06	0.03
Deepwalk	0.04	0.15	0.26	0.14	0.14	0.66	0.04	0.01
Node2Vec	0.03	0.15	0.27	0.14	0.14	0.68	0.04	0.00
NetMF	0.10	0.00	0.03	0.18	0.02	0.70	0.13	0.03
RandNE	0.00	0.13	0.18	0.14	0.09	0.07	0.03	0.00
Walklets	0.14	0.34	0.42	0.17	0.20	0.05	0.09	0.05
BoostNE	0.00	0.00	0.00	0.12	0.00	0.53	0.00	0.00
Role2Vec	0.02	0.17	0.43	0.15	0.15	0.52	0.04	0.02
GraphSage-M	0.01	0.18	0.17	0.04	0.00	0.08	0.00	0.00
GraphSage-MP	0.02	0.17	0.15	0.07	0.00	0.07	0.01	0.01
GraphSage-L	0.01	0.12	0.11	0.01	0.00	0.07	0.00	0.00
HPE	0.21	0.00	0.00	0.17	0.02	0.44	0.07	0.01
HOP-rec	0.26	0.39	0.50	0.15	0.25	0.85	0.12	0.01

Table 10. This table complements Fig. 12, which plots the Normalized Discounted Cumulative Gain of each method over multiple datasets. We give the average NDCG. We see that across all datasets, the scores are very low, typically below 0.2. Since blog-Catalog and ogbl-ddi have high datasets, we set the parameter to be 50.

Comparison of VCMPR to Hits

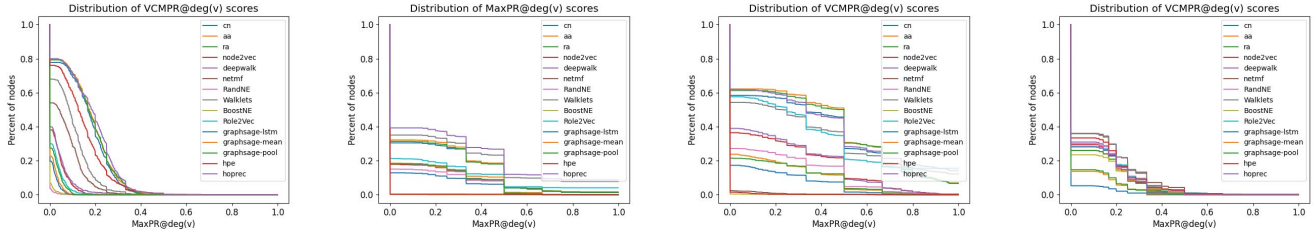
	sbm		collab		ddi		bioplex	
	VCMPR @20	Hits @20	VCMPR @20	Hits @50	VCMPR @50	Hits @20	VCMPR @50	Hits @50
Common Neighbors	0.50	0.52	0.31	0.53	0.82	0.18	0.19	0.32
Adamic Adar	0.50	0.74	0.34	0.65	0.82	0.18	0.19	0.32
Resource Allocation	0.50	0.74	0.33	0.65	0.84	0.05	0.18	0.32
Deepwalk	0.44	0.53	0.21	0.22	0.66	0.01	0.12	0.31
Node2Vec	0.44	0.59	0.21	0.22	0.68	0.02	0.11	0.28
NetMF	0.50	0.73	0.03	0.35	0.70	0.00	0.31	0.58
RandNE	0.50	0.67	0.15	0.09	0.06	0.05	0.07	0.09
Walklets	0.49	0.75	0.29	0.59	0.83	0.02	0.27	0.64
BoostNE	0.31	0.64	0.00	0.08	0.57	0.04	0.00	0.00
Role2Vec	0.42	0.52	0.21	0.44	0.52	0.00	0.08	0.47
GraphSage-M	0.10	0.11	0.00	0.00	0.08	0.00	0.02	0.09
GraphSage-MP	0.23	0.44	0.00	0.00	0.07	0.00	0.02	0.10
GraphSage-L	0.03	0.04	0.00	0.00	0.07	0.00	0.01	0.07
HPE	0.50	0.69	0.02	0.21	0.46	0.00	0.26	0.54
HOP-rec	0.49	0.83	0.35	0.66	0.90	0.00	0.34	0.71

Table 11. This table compares VCMPR to Hits of each method over multiple data-sets.

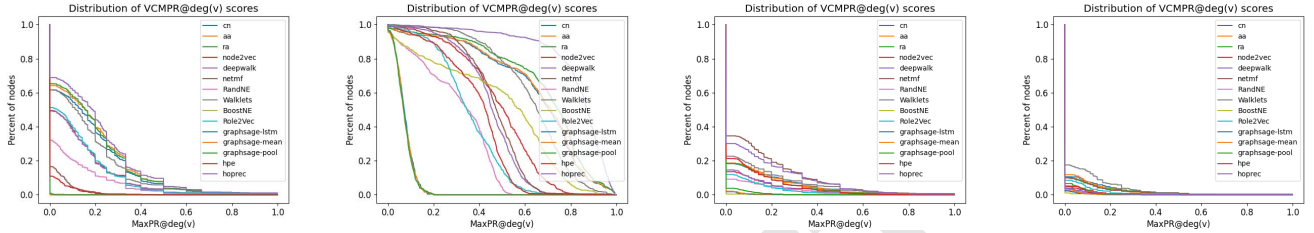
VCMPR@Deg(V)

	blog-catalog	amazon	dblp	sbm	collab	ddi	bioplex	HI-II-14
Common Neighbors	0.16	0.14	0.35	0.08	0.17	0.65	0.05	0.02
Adamic Adar	0.16	0.15	0.38	0.08	0.18	0.66	0.05	0.02
Resource Allocation	0.16	0.14	0.38	0.08	0.19	0.69	0.05	0.02
Deepwalk	0.03	0.07	0.18	0.08	0.12	0.45	0.03	0.01
Node2Vec	0.02	0.07	0.18	0.08	0.11	0.49	0.03	0.01
NetMF	0.06	0.00	0.01	0.10	0.02	0.46	0.10	0.02
RandNE	0.00	0.06	0.12	0.08	0.06	0.30	0.02	0.00
Walklets	0.09	0.19	0.29	0.10	0.15	0.63	0.06	0.04
BoostNE	0.00	0.00	0.00	0.07	0.00	0.49	0.00	0.00
Role2Vec	0.01	0.11	0.31	0.08	0.11	0.35	0.02	0.01
GraphSage-M	0.01	0.08	0.09	0.03	0.00	0.07	0.00	0.00
GraphSage-MP	0.01	0.07	0.09	0.03	0.00	0.07	0.00	0.01
GraphSage-L	0.01	0.05	0.06	0.01	0.00	0.07	0.00	0.00
HPE	0.12	0.00	0.00	0.09	0.01	0.40	0.05	0.01
HOP-rec	0.18	0.22	0.36	0.08	0.19	0.40	0.09	0.01

Table 12. This table complements Fig. 11, which plots the VCMPR scores using a variable threshold based on the deg(v). We give the average scores here. For amazon and dblp, we see a slight drop in scores when compared to the fixed thresholds.

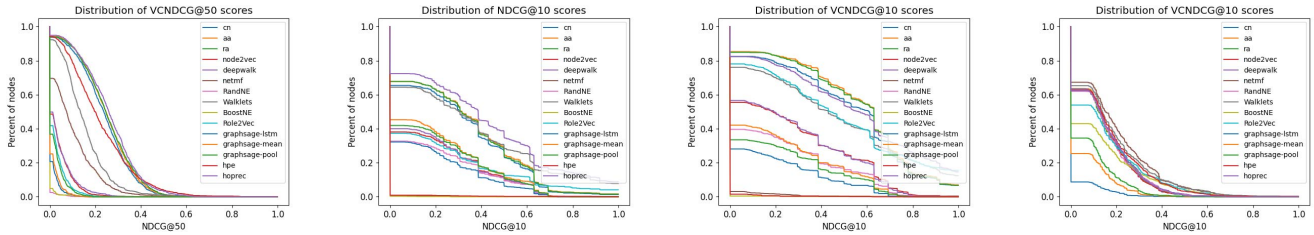


(a) `blog-Catalog` VCMPR@deg(v) **(b)** `amazon` VCMPR@deg(v) **(c)** `dblp` VCMPR@deg(v) **(d)** `SBM` VCMPR@deg(v)

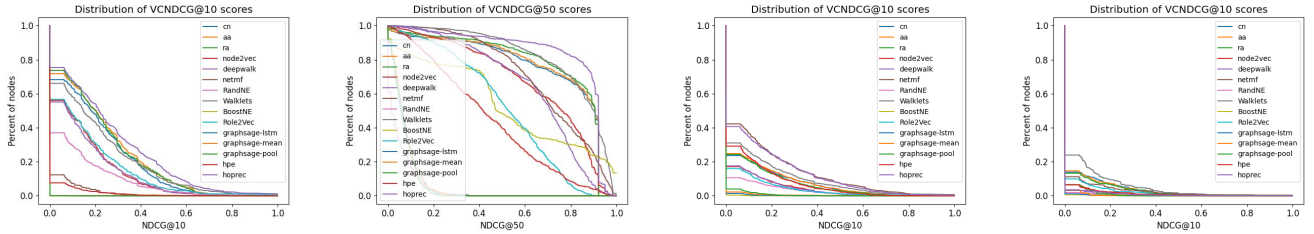


(e) `ogbl-collab` VCMPR@deg(v) **(f)** `ogbl-ddi` VCMPR@deg(v) **(g)** `bioplex` VCMPR@deg(v) **(h)** `HI-II-14` VCMPR@deg(v)

Fig. 11. We compute VCMPR curves with a variable threshold set to the degree of each vertex v in the ground-truth graph. In the more sparse `amazon` and `dblp` data-sets, the VCMPR values are lower than in the fixed threshold plots. In the `blog-Catalog` dataset, the VCMPR values are slightly higher.

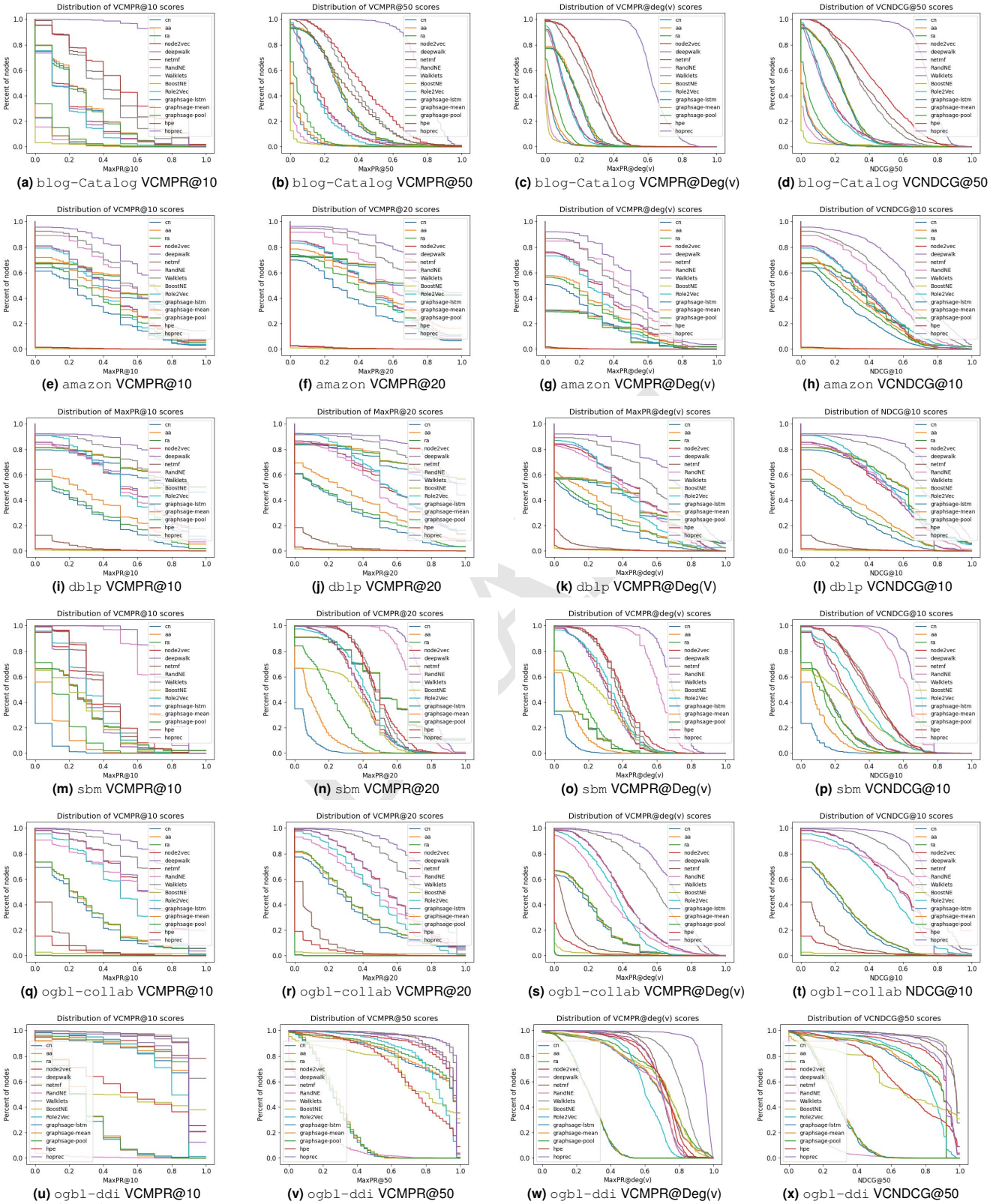


(a) `blog-Catalog` VCNDCG@50 **(b)** `amazon` VCNDCG@10 **(c)** `dblp` VCNDCG@10 **(d)** `sbm` VCNDCG@10



(e) `ogbl-collab` VCNDCG@10 **(f)** `ogbl-ddi` VCNDCG@50 **(g)** `Bioplex` VCNDCG@10 **(h)** `HI-II-14` VCNDCG@10

Fig. 12. We compute VCNDCG for all datasets. Similarly to VCMPR, the VCNDCG scores are quite low, indicating that the top predictions are of poor quality. Thus most edges are not in the top k predictions or are not ranked highly within said predictions.



(a) `blog-Catalog` VCMPR@10 **(b)** `blog-Catalog` VCMPR@50 **(c)** `blog-Catalog` VCMPR@Deg(v) **(d)** `blog-Catalog` VCNDCG@50

(e) `amazon` VCMPR@10 **(f)** `amazon` VCMPR@20 **(g)** `amazon` VCMPR@Deg(v) **(h)** `amazon` VCNDCG@10

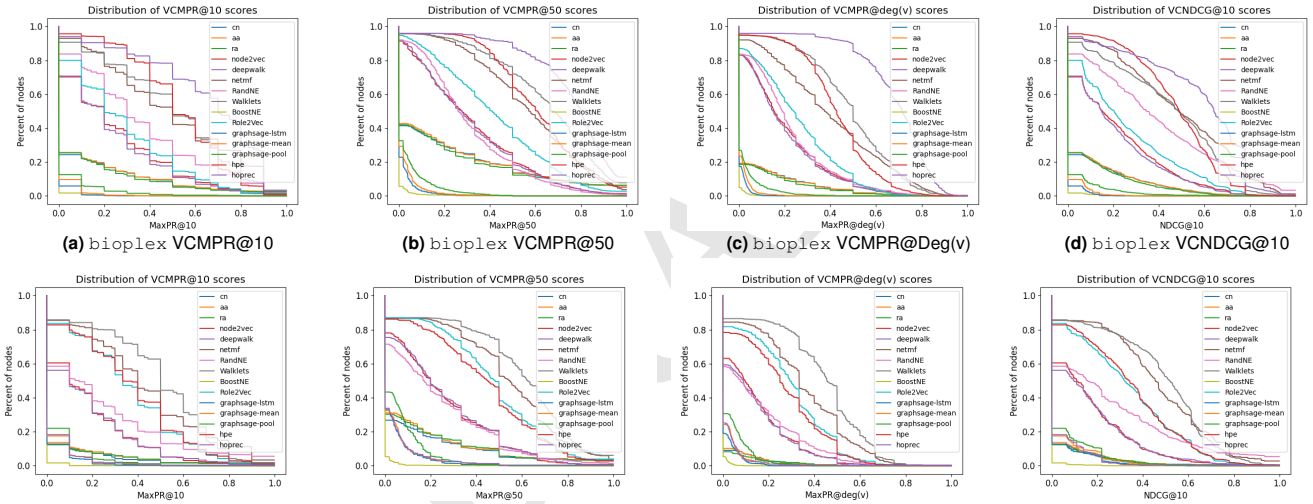
(i) `dblp` VCMPR@10 **(j)** `dblp` VCMPR@20 **(k)** `dblp` VCMPR@Deg(V) **(l)** `dblp` VCNDCG@10

(m) `sbm` VCMPR@10 **(n)** `sbm` VCMPR@20 **(o)** `sbm` VCMPR@Deg(v) **(p)** `sbm` VCNDCG@10

(q) `ogbl-collab` VCMPR@10 **(r)** `ogbl-collab` VCMPR@20 **(s)** `ogbl-collab` VCMPR@Deg(v) **(t)** `ogbl-collab` NDCG@10

(u) `ogbl-ddi` VCMPR@10 **(v)** `ogbl-ddi` VCMPR@50 **(w)** `ogbl-ddi` VCMPR@Deg(v) **(x)** `ogbl-ddi` VCNDCG@50

Fig. 13. VCMPR@k scores for multiple data-sets over *both the test and train edges*. Despite the fact that the majority of edges have been seen by the model in training, the VCMPR curves show that the models often do not reflect those results. `blog-Catalog` still has quite poor scores, but even for the other datasets only a small percentage of nodes have high VCMPR scores.



(a) `bioplex` VCMPR@10 **(b)** `bioplex` VCMPR@50 **(c)** `bioplex` VCMPR@Deg(v) **(d)** `bioplex` VCNDCG@10

(e) `HI-II-14` VCMPR@10 **(f)** `HI-II-14` VCMPR@50 **(g)** `HI-II-14` VCMPR@Deg(v) **(h)** `HI-II-14` VCNDCG@10

Fig. 14. VCMPR@k scores for PPI data-sets over *both the test and train edges*. Despite the fact that the majority of edges have been seen by the model in training, the VCMPR curves show that the models often do not reflect those results.

	Avg VCMPR @10	Avg VCMPR @50	Avg VCMPR @Deg(V)	VCNDCG @50
Common Neighbors	0.25	0.29	0.15	0.22
Adamic Adar	0.25	0.29	0.15	0.23
Resource Allocation	0.20	0.18	0.12	0.17
Deepwalk	0.19	0.17	0.12	0.16
Node2Vec	0.20	0.18	0.12	0.17
NetMF	0.43	0.34	0.25	0.38
RandNE	0.03	0.02	0.02	0.03
Walklets	0.39	0.38	0.27	0.37
BoostNE	0.01	0.01	0.01	0.01
Role2Vec	0.17	0.15	0.11	0.15
GraphSage-M	0.03	0.04	0.03	0.04
GraphSage-MP	0.06	0.06	0.05	0.06
GraphSage-L	0.03	0.03	0.03	0.03
HPE	0.49	0.43	0.29	0.44
HOP-rec	0.82	0.77	0.64	0.73

Table 13. This table complements Fig. 13, which has results from link prediction on the *entire* `blogcatalog` dataset. We give the average VCMPR@k for $k = 10, 50$, VCMPR@Deg(v), and VCNDCG@50.

	Avg VCMPR @10	Avg VCMPR @20	Avg VCMPR @Deg(V)	VCNDCG @10
Common Neighbors	0.48	0.57	0.15	0.29
Adamic Adar	0.52	0.59	0.15	0.31
Resource Allocation	0.52	0.58	0.14	0.31
Deepwalk	0.40	0.45	0.31	0.33
Node2Vec	0.42	0.47	0.31	0.34
NetMF	0.00	0.01	0.00	0.00
RandNE	0.53	0.56	0.43	0.46
Walklets	0.55	0.69	0.40	0.51
BoostNE	0.00	0.00	0.00	0.00
Role2Vec	0.37	0.45	0.28	0.36
GraphSage-M	0.38	0.47	0.25	0.29
GraphSage-MP	0.33	0.42	0.22	0.25
GraphSage-L	0.28	0.34	0.18	0.21
HPE	0.01	0.01	0.00	0.01
HOP-rec	0.67	0.78	0.86	0.63

Table 14. This table complements Fig. 13, which has results from link prediction on the *entire* `amazon` dataset. We give the average VCMPR@k for $k = 10, 20$, VCMPR@Deg(v), and VCNDCG@10.

	Avg VCMPR @10	Avg VCMPR @20	Avg VCMPR @Deg(V)	VCNDCG @10
Common Neighbors	0.64	070	0.34	0.48
Adamic Adar	0.68	0.73	0.36	0.51
Resource Allocation	0.68	0.73	0.35	0.50
Deepwalk	0.53	0.55	0.44	0.46
Node2Vec	0.53	0.54	0.43	0.45
NetMF	0.03	0.03	0.02	0.02
RandNE	0.50	0.4	0.40	0.47
Walklets	0.67	0.75	0.55	0.65
BoostNE	0.00	0.00	0.00	0.00
Role2Vec	0.52	0.57	0.39	0.51
GraphSage-M	0.32	0.35	0.24	0.27
GraphSage-MP	0.26	0.28	0.21	0.22
GraphSage-L	0.23	0.25	0.17	0.19
HPE	0.01	0.01	0.01	0.01
HOP-rec	0.77	0.82	0.66	0.74

Table 15. This table complements Fig. 13, which has results from link prediction on the *entire* `dblp` dataset. We give the average VCMPR@k for $k = 10, 20$, VCMPR@Deg(v), and VCNDCG@10.

	Avg VCMPR @10	Avg VCMPR @20	Avg VCMPR @Deg(V)	VCNDCG @10
Common Neighbors	0.25	0.50	0.08	0.15
Adamic Adar	0.24	0.50	0.08	0.14
Resource Allocation	0.24	0.50	0.08	0.15
Deepwalk	0.29	0.39	0.30	0.25
Node2Vec	0.29	0.39	0.30	0.26
NetMF	0.38	0.46	0.38	0.38
GraRep	0.40	0.49	0.39	0.39
RandNE	0.67	0.70	0.60	0.60
Walklets	0.40	0.49	0.38	0.41
BoostNE	0.22	0.27	0.21	0.22
Role2Vec	0.33	0.40	0.32	0.33
GraphSage-M	0.09	0.10	0.08	0.08
GraphSage-MP	0.14	0.19	0.15	0.12
GraphSage-L	0.03	0.03	0.03	0.03
HPE	0.41	0.48	0.39	0.40
HOP-rec	0.76	0.76	0.69	0.68

Table 16. This table complements Fig. 13, which has results from link prediction on the *entire* `sbm` dataset. We give the average VCMPR@k for $k = 10, 20$, VCMPR@Deg(v), and VCNDCG@10.

	Avg VCMPR @10	Avg VCMPR @20	Avg VCMPR @Deg(V)	VCNDCG @10
Common Neighbor	0.27	0.31	0.17	0.22
Adamic Adar	0.30	0.34	0.19	0.24
Resource Allocation	0.30	0.34	0.19	0.24
Deepwalk	0.62	0.56	0.40	0.56
Node2Vec	0.62	0.56	0.41	0.56
NetMF	0.08	0.08	0.07	0.08
RandNE	0.61	0.50	0.33	0.65
Walklets	0.71	0.71	0.58	0.62
BoostNE	0.02	0.02	0.02	0.02
Role2Vec	0.50	0.48	0.34	0.49
GraphSage-M	0.00	0.00	0.00	0.00
GraphSage-MP	0.00	0.00	0.00	0.00
GraphSage-L	0.00	0.00	0.00	0.00
HPE	0.04	0.03	0.03	0.04
HOP-rec	0.79	0.79	0.67	0.74

Table 17. This table complements Fig. 13, which has results from link prediction on the *entire* `collab` dataset. We give the average VCMPR@k for $k = 10, 20$, VCMPR@Deg(v), and VCNDCG@10.

	Avg VCMPR @10	Avg VCMPR @50	Avg VCMPR @Deg(V)	VCNDCG @50
Common Neighbors	0.76	0.81	0.64	0.76
Adamic Adar	0.77	0.82	0.65	0.77
Resource Allocation	0.79	0.83	0.68	0.79
Deepwalk	0.86	0.90	0.68	0.88
Node2Vec	0.84	0.89	0.70	0.86
NetMF	0.95	0.92	0.69	0.93
RandNE	0.01	0.12	0.62	0.60
Walklets	0.94	0.95	0.80	0.95
BoostNE	0.47	0.70	0.66	0.65
Role2Vec	0.76	0.83	0.60	0.79
GraphSage-M	0.27	0.27	0.27	0.27
GraphSage-MP	0.28	0.28	0.27	0.28
GraphSage-L	0.27	0.27	0.27	0.27
HPE	0.55	0.72	0.68	0.67
HOP-rec	0.86	0.94	0.94	0.90

Table 18. This table complements Fig. 13, which has results from link prediction on the *entire* `ddi` dataset. We give the average VCMPR@k for $k = 10, 50$, VCMPR@Deg(v), and VCNDCG@50.

	Avg VCMPR @10	Avg VCMPR @50	Avg VCMPR @Deg(V)	VCNDCG @10
Common Neighbors	0.10	0.18	0.05	0.07
Adamic Adar	0.10	0.19	0.05	0.07
Resource Allocation	0.09	0.18	0.05	0.07
Deepwalk	0.23	0.31	0.19	0.20
Node2Vec	0.23	0.31	0.19	0.20
NetMF	0.49	0.57	0.40	0.49
RandNE	0.36	0.31	0.22	0.39
Walklets	0.49	0.64	0.48	0.45
BoostNE	0.00	0.00	0.00	0.00
Role2Vec	0.28	0.42	0.25	0.25
GraphSage-M	0.01	0.02	0.01	0.01
GraphSage-MP	0.02	0.03	0.02	0.02
GraphSage-L	0.01	0.01	0.01	0.01
HPE	0.53	0.60	0.42	0.47
HOP-rec	0.64	0.74	0.67	0.57

Table 19. This table complements Fig. 14, which has results from link prediction on the *entire* `Bioplex` dataset. We give the average VCMPR@k for $k = 10, 50$, VCMPR@Deg(v), and VCNDCG@10.

	Avg VCMPR @10	Avg VCMPR @50	Avg VCMPR @Deg(V)	VCNDCG @10
Common Neighbors	0.04	0.10	0.02	0.03
Adamic Adar	0.05	0.11	0.02	0.03
Resource Allocation	0.04	0.11	0.02	0.03
Deepwalk	0.17	0.25	0.13	0.16
Node2Vec	0.17	0.25	0.14	0.16
NetMF	0.42	0.52	0.34	0.45
RandNE	0.23	0.23	0.14	0.25
Walklets	0.48	0.57	0.42	0.46
BoostNE	0.00	0.00	0.00	0.00
Role2Vec	0.35	0.45	0.28	0.32
GraphSage-M	0.03	0.04	0.02	0.03
GraphSage-MP	0.04	0.06	0.03	0.04
GraphSage-L	0.02	0.03	0.01	0.02
HPE	0.36	0.42	0.26	0.33
HOP-rec	0.03	0.04	0.02	0.03

Table 20. This table complements Fig. 14, which has results from link prediction on the *entire* `HI-II-14` dataset. We give the average VCMPR@k for $k = 10, 50$, VCMPR@Deg(v), and VCNDCG@10.