FREDE: Anytime Graph Embeddings

Anton Tsitsulin
University of Bonn
tsitsulin@bit.uni-bonn.de

Marina Munkhoeva
Skoltech
marina.munkhoeva@skoltech.ru

Davide Mottin
Aarhus University
davide@cs.au.dk

Panagiotis Karras
Aarhus University
panos@cs.au.dk

Ivan Oseledets
Skoltech
ivan.oseledets@skoltech.ru

Emmanuel Müller
TU Dortmund
mueller@bit.uni-bonn.de

ABSTRACT

Low-dimensional representations, or *embeddings*, of a graph's nodes facilitate several practical data science and data engineering tasks. As such embeddings rely, explicitly or implicitly, on a similarity measure among nodes, they require the computation of a quadratic similarity matrix, inducing a tradeoff between space complexity and embedding quality. To date, no graph embedding work combines (i) linear space complexity, (ii) a nonlinear transform as its basis, and (iii) nontrivial quality guarantees. In this paper we introduce FREDE (*FREquent Directions Embedding*), a graph embedding based on matrix sketching that combines those three desiderata. Starting out from the observation that embedding methods aim to preserve the covariance among the rows of a similarity matrix, FREDE iteratively improves on quality while individually processing rows of a nonlinearly transformed PPR similarity matrix derived from a state-of-the-art graph embedding method and provides, *at any iteration*, column-covariance approximation guarantees in due course almost indistinguishable from those of the optimal approximation by SVD. Our experimental evaluation on variably sized networks shows that FREDE performs almost as well as SVD and competitively against state-of-the-art embedding methods in diverse data science tasks, even when it is based on as little as 10% of node similarities.

PVLDB Reference Format:
Anton Tsitsulin, Marina Munkhoeva, Davide Mottin, Panagiotis Karras, Ivan Oseledets, and Emmanuel Müller. FREDE: Anytime Graph Embeddings. PVLDB, 14(6): 1102 - 1110, 2021.
doi:10.14778/3447689.3447713

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/xgfs/frede.

1 INTRODUCTION

Low-dimensional representations, or *embeddings*, of graph's nodes provide a multi-purpose tool for performing data science tasks such as community detection, link prediction, and node classification. Neural embeddings [12, 25, 33, 36], computed by unsupervised representation learning over *nonlinear* transformations, outperform their linear counterparts [23, 47] in task performance, and achieve scalability via sampling a node similarity matrix, such as Personalized PageRank (PPR); however, such neural methods lack theoretically grounded error guarantees with respect to their objectives. The theoretically most well-grounded state-of-the-art method, NetMF [28], performs Singular Value Decomposition (SVD) on a dense matrix of nonlinear node similarities, and achieves the global optimum of its objective by virtue of the properties of SVD.

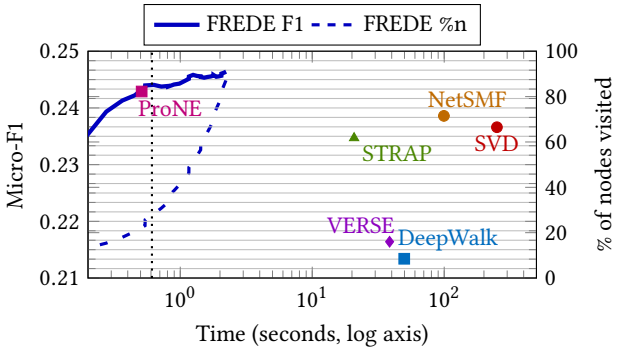

Figure 1: FREDE scalably produces an embedding *at any time*; at the dotted black line, it outperforms all contenders, including the SVD of a PPR-like similarity matrix, after processing about 20% of matrix rows. (PPI data)

However, this *optimality* comes to the detriment of *scalability*, as NetMF needs to precompute the similarity matrix and store it in memory at cost quadratic in the number of nodes. An ideal method should achieve both quality and scalability.

In this paper, we propose FREDE, the first, to our knowledge, linear-space algorithm that produces embeddings with *quality guarantees* from a nonlinear transform. We observe that factorization-based embeddings effectively strive to preserve the covariance of a similarity matrix, and that a few nodes acting as oracles approximate the distances among all nodes with guarantees [35]. Given these observations, we adapt a covariance-preserving matrix sketching algorithm, *Frequent Directions* (FD) [11, 18], to produce a graph embedding by factorizing, on a per-row basis, a PPR-like node similarity matrix derived by interpreting a state-the-art neural embedding, VERSE [36], as matrix factorization. FREDE can be distributed, as it inherits the *mergeability* property of FD: two embeddings can be computed independently on different node sets and merged to a single embedding, with quality guarantees that hold *anytime* [48], even after accessing a subset of similarity matrix rows. Figure 1 shows that FREDE outperforms state-of-the-art methods and SVD in a node classification task after processing about 20% of similarity matrix rows representing graph nodes.

We summarize our contributions as follows:

Proceedings of the VLDB Endowment, Vol. 14, No. 6 ISSN 2150-8097.
doi:10.14778/3447689.3447713

(1) we interpret a state-of-the-art graph embedding method, VERSE, as factorizing a transformed PPR similarity matrix;
(2) we propose FREDE, an *anytime* graph embedding algorithm that minimizes covariance error on that PPR-like matrix via *sketching*, with space complexity linear in the number of nodes and time linear in the number of processed rows;
(3) in a thorough experimental evaluation with real graphs we confirm that FREDE is competitive against the state of the art and scales to large networks.

2 PRELIMINARIES AND RELATED WORK

Our work builds on the know-how of matrix sketching to derive scalable, anytime graph embeddings for practical data science tasks. Here, we outline previous work on graph embeddings and the fundamentals of matrix sketching.

2.1 Problem setting

A *graph* is a pair $G = (V, E)$ with n vertices $V = (v_1, \ldots, v_n)$, $|V| = n$, and edges $E \subseteq V \times V$, $|E| = m$, represented by an *adjacency matrix* A for which $A_{ij} = 1$ if $(i, j) \in E$ is an edge between node i and node j, otherwise $A_{ij} = 0$. D is the diagonal matrix with the degree of node i as entry $D_{ii} = \sum_{j=1}^{n} A_{ij}$. The *normalized adjacency matrix*, $P = D^{-1}A$, represents the transition probability from a node to any of its neighbors. We represent interactions among nodes with a *similarity matrix* $S \in \mathbb{R}^{n \times n}$ [23, 36, 47]. The row i of A is denoted as A_i. The embedding problem is to find an $n \times d$ matrix W that retains most information in S.

2.2 Neural embeddings

Initial works on graph embeddings relied on a neural network to produce vector representations of a graph's nodes; DeepWalk [25] first transferred such methods from words [16, 20] to graphs, utilizing a corpus of random walks. LINE [33] extended DeepWalk by exploiting graph edges rather than walks; Node2vec [12] customized random walk generation; and VERSE [36] generalized this approach to a method that preserves any similarity measure among nodes, with Personalized PageRank (PPR) [24] as the default option. Such neural embeddings leverage paths around a node, reach scalability via sampling, and provide no closed-form solution and no quality guarantees; we call them *positional* embeddings. In another neural approach, *structural* embeddings [3, 29, 30] leverage complex graph structural patterns to improve quality at the expense of scalability; however, positional embeddings outperform their structural counterparts in both quality and scalability. For these reasons, we exclude structural embeddings from our discussion.

2.3 Factorization-based embeddings

Other works cast the problem of embedding a graph's nodes as one of exact or approximate factorization of the node similarity matrix, which is meant to minimize the *reconstruction error* [28]:

Definition 1 (Reconstruction error). *The* reconstruction error *between matrices* S *and* $\tilde{S}$ *is the Frobenius norm of the difference among the* S *and* $\tilde{S}$*, i.e.,* $\|S - \tilde{S}\|_F^2 = \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{n} (S_{ij} - \tilde{S}_{ij})^2}$.

In the case of *symmetric* S, there exists an eigendecomposition $S = U\Lambda U^\top$, and the optimal rank-k approximation of S is $[S]_k = WW^\top$, where the matrix $W = U_k\sqrt{\Lambda_k}$ is the product between the matrix of the first k eigenvectors, U_k, and a diagonal matrix of the square roots of the first k eigenvalues, Λ_k. On the other hand, in case S is *asymmetric*, the best rank-k approximation is obtained by the first k singular vectors and values of the Singular Value Decomposition (SVD) $S = U\Sigma V^\top$, i.e., $[S]_k = U_k\Sigma_k V_k^\top$, where U_k and V_k denotes the first k columns of U and V, respectively.

GraRep [8] applies SVD to factorize a concatenation of dense log-transformed DeepWalk transition probability matrices over different numbers of steps; yet it is neither scalable, nor provides quality guarantees. HOPE [23] overcomes the scalability drawback using a generalized form of SVD on special similarity matrices in the form AB^{-1}; it achieves optimality due to the guarantees of the Eckart–Young–Mirsky theorem, but its overall performance is hindered by the linearity of the underlying transform [36]. AROPE [47] applies spectral filtering on symmetric similarity matrices, forfeiting any guarantees. ApproxPPR [41] applies the randomized block Krylov SVD algorithm [22] on a truncated PPR matrix; NRP [41] iteratively reweights the resulting embedding vectors by coordinate descent; this post-hoc refinement boosts the performance of ApproxPPR embeddings [41]. STRAP [42] applies sparse factorization on a sparse-PPR-based proximity matrix; similarly, ProNE [45] uses sparse factorization on a weighted adjacency matrix and spectral propagation on the obtained embeddings.

2.4 The neural-factorization connection

Recent work has established a connection between neural and factorization-based embeddings. In NetMF [28], Qiu et al. extended an analysis of word embeddings [16] to connect matrix factorization and neural embeddings: under certain probability independence assumptions, DeepWalk, LINE, and Node2vec implicitly apply SVD on dense log-transformed similarity matrices. NetMF proposes novel closed-form solutions to compute such matrices with optimal error guarantees. For example, DeepWalk's objective is equivalent to SVD on the dense similarity matrix

$$S = \log\left(\frac{m}{bT}\left(\sum_{r=1}^{T} P^r\right)D^{-1}\right), \tag{1}$$

where T is the random walk window size and b is the number of negative samples [28]. The d-dimensional DeepWalk embedding is obtained as $U_d\sqrt{\Sigma_d}$, where U_d contains the d left singular vectors and Σ_d the first d singular values. However, this NetMF approach requires $O(n^2)$ space to store S, a prohibitive complexity that hinders its application to graphs with more than 100 000 nodes. In another attempt, NetSMF [27], Qiu et al. sought to mitigate NetMF's scalability drawback by sparsifying the similarity matrix; however, matrix sparsification forfeits optimality guarantees, causing performance deterioration for effectual sparsity levels [27]; besides, the sparsified matrix has $O(Tm \log n)$ nonzeros, hence it still yields quadratic growth.

2.5 Synoptic overview

Table 1 presents previous work, including a hierarchical-clustering-based heuristic, LouvainNE [5], and one sketching Self-Loop-Augmented adjacency vectors, NodeSketch [40], in terms of desiderata of the *solution*, its *computation*, and time/space requirements:

		Solution				Computation			Complexity	
type	*method*	Nonlinear	Closed-form	Error-bounded	Versatile	Frugal	Anytime	Mergeable	Space	Time
neural	DeepWalk	✔	✘	✘	✘	✔	✘	✘	$O(dn)$	$O(dn \log n)$
neural	Node2vec	✔	✘	✘	✘	✘	✘	✘	$O(n^3)$	$O(dnb)$
neural	LINE	✔	✘	✘	✘	✔	✘	✘	$O(dn)$	$O(dnb)$
neural	VERSE	✔	✘	✘	✔	✔	✘	✘	$O(dn)$	$O(dnb)$
factorization	HOPE	✘	✔	✔	✔	✔	✘	✘	$O(dn)$	$O(d^2m)$
factorization	AROPE	✘	✔	✘	✔	✔	✘	✘	$O(dn)$	$O(dm+d^2n)$
factorization	ApproxPPR	✘	✔	✘	✘	✔	✘	✘	$O(dn)$	$O((dm + d^2n)\log n)$
factorization	NRP		✘							
factorization	NetMF	✔	✔	✔	✔	✘	✘	✘	$O(n^2)$	$O(dn^2)$
factorization	NetSMF			✘					$O(n^2)$	$O(n^2)$
factorization	STRAP	✔	✘	✘	✔	✔	✘	✘	$O(n/\epsilon)$	$O(d^2n + m/\epsilon)$
factorization	ProNE	✔	✘	✘	✘	✔	✘	✘	$O(dn)$	$O(d^2n + Tm)$
clust	LouvainNE	✔	✘	✘	✘	✔	✘	✘	$O(m \log n)$	$O(m \log n + dn \log n)$
skch	NodeSketch	✔	✘	✘	✘	✘	✘	✘	$O(n^2)$	$O(n^2)$
skch	FREDE (ours)	✔	✔	✔	✔	✔	✔	✔	$O(dn)$	$O(dn^2)$

Table 1: Fulfilled (✔) and missing (✘) desiderata of related work; complexities in terms of number of nodes n and edges m, dimensionality d, context size T, and number of negative samples b; we assume a sparse graph with average degree $O(d)$.

- **nonlinear:** using nonlinear transforms; HOPE, AROPE, and ApproxPPR/NRP use linear transforms; nonlinearity is desirable, as linear dimensionality reduction methods fail to confer the advantages of their nonlinear counterparts in general [15, 36].
- **closed-form:** deriving the solution via an explicit, well-defined formula without relying on heuristic learning components; only NetMF and NetSMF are both closed-form and nonlinear; NRP loses the closed-form character of ApproxPPR due to its performance-boosting post-hoc heuristic reweighting; such reweighting may augment any embedding with additional node degree information, yet it was only applied on ApproxPPR in [41].
- **error-bounded:** affording nontrivial, end-to-end error guarantees with respect to a fixed objective; in principle, error-bounded methods, like HOPE and NetMF, are closed-form; the reverse is not always the case, as some closed-form methods abandon guarantees for sake of scalability: AROPE by spectral filtering, ApproxPPR by truncating, and NetSMF by sparsifying the similarity matrix.
- **versatile:** accommodating diverse similarity measures; HOPE, AROPE, VERSE, NetMF & NetSMF, and STRAP are versatile.
- **frugal** (space-efficient)**:** having worst-case space complexity subquadratic in the number of nodes.
- **anytime:** allowing the computation of a *partial* embedding whose quality improves as more nodes are processed.
- **mergeable:** allowing for a combination of embeddings on two node subsets that retains guarantees, hence enabling distributed computation [2].

2.6 Matrix sketching

SVD applied on a matrix $\mathrm{M} \in \mathbb{R}^{s \times t}$ (s elements, t features) produces $[\mathrm{M}]_k = \mathrm{U}_k \Sigma_k \mathrm{V}_k^\top$ that minimizes reconstruction error; from the same SVD we can also obtain $\mathrm{W} = \Sigma_k \mathrm{V}_k^\top$, which minimizes *column covariance error*, dependent on the singular value decay of M:

Definition 2 (Covariance error). *The* column covariance error *is the normalized difference between the covariance matrices:*

$$\mathsf{ce}_k(\mathrm{M}, \mathrm{W}) = \frac{\|\mathrm{M}^\top \mathrm{M} - \mathrm{W}^\top \mathrm{W}\|_2}{\|\mathrm{M} - [\mathrm{M}]_k\|_F^2} \geq \frac{\|\mathrm{M}^\top \mathrm{M} - \mathrm{W}^\top \mathrm{W}\|_2}{\|\mathrm{M}\|_F^2} = \mathsf{ce}(\mathrm{M}, \mathrm{W})$$

Matrix sketching [6, 7, 18, 38] is an alternative to the computationally heavy matrix reconstruction by SVD grounded on the connection between SVD and covariance error; it finds a low-dimensional matrix, or *sketch*, $\mathrm{W} \in \mathbb{R}^{d \times t}$ of M, by *row-wise* streaming and with guarantees on *column covariance* error, which accounts for variance loss in each dimension. The correct k for the best rank k approximation $[\mathrm{M}]_k$ is not known and often requires grid search. Hence, we use the lower bound $\mathsf{ce}(\mathrm{M}, \mathrm{W})$ in lieu of $\mathsf{ce}_k(\mathrm{M}, \mathrm{W})$.

A desirable sketch property is *mergeability*:

Definition 3. **Mergeability.** *A sketching algorithm* `sketch` *is* mergeable *if there exists an algorithm* `merge` *that, applied on the $d \times t$ sketches, $\mathrm{W}_1 = \mathtt{sketch}(\mathrm{M}_1)$ and $\mathrm{W}_2 = \mathtt{sketch}(\mathrm{M}_2)$, of two $\frac{s}{2} \times t$ matrices, $\mathrm{M}_1, \mathrm{M}_2$, with $\mathsf{ce}(\mathrm{M}_1, \mathrm{W}_1) \leq \epsilon$ and $\mathsf{ce}(\mathrm{M}_2, \mathrm{W}_2) \leq \epsilon$, produces a $d \times t$ sketch W of the concatenated matrix $\mathrm{M} = [\mathrm{M}_1; \mathrm{M}_2]$, $\mathrm{W} = \mathtt{merge}(\mathrm{W}_1, \mathrm{W}_2) = \mathtt{sketch}(\mathrm{M})$, that preserves the covariance error bound ϵ, i.e., $\mathsf{ce}(\mathrm{M}, \mathrm{W}) \leq \epsilon$.*

We now discuss some representative sketching algorithms.

Hashing. We construct a 2-universal hash function $h : [s] \to [d]$ and a 4-universal hash function $g : [s] \to \{-1, +1\}$. Starting with a zero-valued sketch matrix W, each row M_i is added to the $h(i)$-th sketch matrix row with sign $g(i)$: $\mathrm{W}_{h(i)} = g(i) * \mathrm{M}_i$, with complexity linear in matrix size, $O(st)$. In practice, random assignment of rows is used instead of a hash function. Setting $d = O(t^2/\epsilon^2)$, hashing achieves $\mathsf{ce} \leq \epsilon$ [38]. This sketch is trivially mergeable: $\mathtt{merge}(\mathrm{W}_1, \mathrm{W}_2) = \mathrm{W}_1 + \mathrm{W}_2$.

Random Projections are a fundamental data analysis tool [38]. Boutsidis et al. [6] propose a row-streaming matrix sketching algorithm that randomly combines rows of the input matrix. In matrix form, $\tilde{\mathrm{M}} = \mathrm{RM}$, where the elements R_{ij} of the $d \times s$ matrix R are uniformly from $\{-1/\sqrt{d}, 1/\sqrt{d}\}$. For each row M_i, the algorithm samples a random vector $\mathbf{r}_i \in \mathbb{R}^d$ with entries in $\{-1/\sqrt{d}, 1/\sqrt{d}\}$ and updates $\mathrm{W} = \mathrm{W} + \mathbf{r}_i \mathrm{M}_i^\top$. This sketch achieves $\mathsf{ce} \leq \epsilon$ with $d = O(t/\epsilon^2)$, with practical performance exceeding the guarantee [17], and is mergeable with $\mathtt{merge}(\mathrm{W}_1, \mathrm{W}_2) = \mathrm{W}_1 + \mathrm{W}_2$.

Sampling. The Column Subset Selection Problem (CSSP) [7] is to select a column subset of an entire matrix. In the row-update model,

a solution is found by sampling scaled rows $M_i/\sqrt{dp_i}$ with probability $p_i = \|M_i\|^2/\|M\|_F^2$. While the norm $\|M\|_F^2$ is usually unknown in advance, the method can work with d reservoir samplers, where d is the sketch size. This sketch achieves $\mathsf{ce} \leq \epsilon$ with $d = O(t/\epsilon^2)$, yet the cost of maintaining reservoir samples is non-negligible. The sketch is mergeable if we use distributed reservoir sampling.

Frequent Directions (FD) [18], the current state of the art in sketching, extends the Misra-Gries algorithm [21] from frequent items to matrices and outperforms other methods [6, 7, 9] in quality. FD sketches a matrix by iteratively filling the sketch with incoming rows, performing SVD on it when it cannot add more rows, and shrinking the accumulated vectors with a low-rank SVD approximation. The complexity is $O(dts)$, due to s/d iterations of computing the $O(d^2t)$ SVD decomposition of a $2d \times t$ matrix W with $d \ll t$. This sketch achieves $\mathsf{ce} \leq \epsilon$ when $d = O(t/\epsilon)$ and is mergeable with

$$\mathtt{merge}(W_1, W_2) = \mathtt{FD}(\mathtt{concatenate}(W_1, W_2)). \tag{2}$$

The table below lists the embedding dimension d required to attain error bound $\mathsf{ce} \leq \epsilon \leq 1$ for different algorithms.

Algorithm	Hashing	RP	Sampling	FD
Dimension d	$O(t^2/\epsilon^2)$	$O(t/\epsilon^2)$	$O(t/\epsilon^2)$	$O(t/\epsilon)$

We observe that, by putting the node similarity matrix S in the role of the sketched matrix M, we can effectively turn a sketching technique to an embedding method. Indeed, recent work [46] has adapted a sketching algorithm [6, 37] to graph embeddings, yet forfeited[1] its error guarantees. We apply the know-how of state-of-the-art matrix sketching to serve graph embedding purposes, leading to *anytime graph embeddings* with error guarantees.

3 ANYTIME GRAPH EMBEDDINGS

We observe that SVD-based graph embeddings, such as HOPE and NetMF, use only one of the two unitary matrices SVD produces, U and V. For example, NetMF returns $W = U_{:d}\sqrt{\Sigma_{:d}}$, with Σ truncated to d singular values. Therefore, such methods cannot reconstruct matrix S; SVD products U and Σ *may only* reconstruct the *row covariance matrix* $SS^\top = U\Sigma^2U^\top$, as $WW^\top = U\Sigma^2U^\top$, where $W = U\Sigma$; thus, such methods are better understood as implicitly minimizing the covariance error, rather than the reconstruction error [28], in relation to a similarity matrix among graph nodes.

Serendipitously, *sketching algorithms* aim to reconstruct the *column covariance* $S^\top S = V\Sigma^2V^\top$. Given this relationship, we apply a state-of-the-art *matrix sketching algorithm* in lieu of SVD to construct a graph embedding in *anytime* fashion, by *row updates* of any partially materializable similarity matrix S. Unfortunately, the matrix form of DeepWalk (Eq. 1) cannot be partially materialized. Next, we propose a partially materializable matrix based on Personalized PageRank (PPR), inspired from the VERSE [36] similarity-based embeddings. As we show in the experiments, this choice attains good quality and time performance. However, our method carries no prejudice with regard to the partially materializable matrix used; other choices are possible, such as, for example, the Node-Reweighted PageRank (NRP) [41]. Our aim is to illustrate the advantageous application of sketching for embedding purposes, while our framework supports any way of deriving the primary input matrix.

[1] In our experiments, we use a variant of [46] with error guarantees as a baseline.

3.1 A row-wise computable similarity matrix

VERSE [36] is the first similarity-based embedding method that does *not* require the entire matrix as input, as it allows for efficient row-wise computation; in its default version, it uses the PPR similarity measure:

Definition 4. *Given a starting node distribution s, damping factor α, and the transition probability matrix* P*, the PPR vector* PPR_i *is defined by the recursive equation:*

$$\mathrm{PPR}_i = \alpha s + (1-\alpha)\mathrm{PPR}_i^\top \mathrm{P} \tag{3}$$

To compute PPR_i, we leverage the fact that the probability distribution of a random walk with restart converges to PPR_i vector [4, 24]. Following [16, 28] we show that, under mild assumptions, VERSE with PPR similarity virtually factorizes the log(PPR) matrix up to an additive constant.

Theorem 1. *Let* X *be the matrix of VERSE embeddings. If the terms $z_{ij} = \mathbf{x}_i^\top \mathbf{x}_j$ are independent, then VERSE factorizes the matrix* $\mathrm{Y} = \log(\mathrm{PPR}) + \log n - \log b = \mathrm{XX}^\top$.

Proof. *Consider the VERSE objective function for the uniform sampling distribution and PPR similarity:*

$$\mathcal{L} = \sum_{i=1}^{n}\sum_{j=1}^{n}\left[\mathrm{PPR}_{ij}\log\sigma(\mathbf{x}_i^\top\mathbf{x}_j) + b\mathbb{E}_{j\prime\sim Q_i}\log\sigma(-\mathbf{x}_i^\top\mathbf{x}_{j\prime})\right],$$

where $\sigma(x) = (1+e^{-x})^{-1}$ is the sigmoid, Q_i is the noise sample distribution, and b the number of noise samples. Since PPR *is right-stochastic and Q_i is uniform, i.e.,* $\Pr(Q_i = j) = \frac{1}{n}$*, we can separate the two terms as follows:*

$$\mathcal{L} = \sum_{i=1}^{n}\sum_{j=1}^{n}\mathrm{PPR}_{ij}\log\sigma(\mathbf{x}_i^\top\mathbf{x}_j) + \frac{b}{n}\sum_{i=1}^{n}\sum_{j\prime=1}^{n}\log\sigma(-\mathbf{x}_i^\top\mathbf{x}_{j\prime}).$$

An individual loss term for vertices i and j is:

$$\mathcal{L}_{ij} = \mathrm{PPR}_{ij}\log\sigma(\mathbf{x}_i^\top\mathbf{x}_j) + \frac{b}{n}\log\sigma(-\mathbf{x}_i^\top\mathbf{x}_j).$$

We substitute $z_{ij} = \mathbf{x}_i^\top\mathbf{x}_j$, use our independence assumption, and solve for $\frac{\partial\mathcal{L}_{ij}}{\partial z_{ij}} = \mathrm{PPR}_{ij}\sigma(-z_{ij}) - \frac{b}{n}\sigma(z_{ij}) = 0$ to get $z_{ij} = \log\frac{n\cdot\mathrm{PPR}_{ij}}{b}$, hence $\mathrm{XX}^\top = \log(\mathrm{PPR}) + \log n - \log b = \mathrm{Y}$.

Even though this solution is algebraically impossible, as it implies approximating a non-symmetric matrix by a symmetric one, it provides a matrix whose covariance we can sketch. Similarly, STRAP [42] applies Sparse Randomized SVD to the logarithm of a similarity matrix based on Sparse PPR.

3.2 FREDE algorithm

Since the matrix $\mathrm{Y} = \mathrm{XX}^\top$ has equal row and column ranks, we rewrite the decomposition commutatively, as $\mathrm{Y} = \log(\mathrm{PPR}) + \log n - \log b = \mathrm{X}^\top\mathrm{X}$. We keep the bias parameter b equal to 1, as in NetMF, and apply Frequent Directions (Section 2.6) to obtain a $d \times n$ sketch-based embedding W by processing rows of Y. Algorithm 1 presents

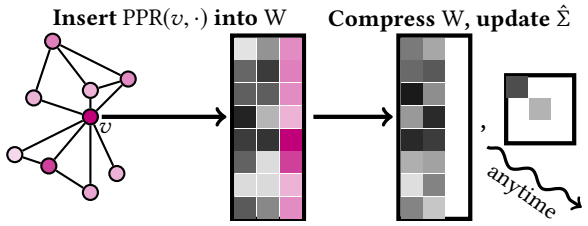

Figure 2: FREDE samples log(PPR) **rows, periodically compresses the derived sketch and gets singular values by SVD, and yields an embedding with error guarantees at any time.**

the details of FREDE and Figure 2 shows its workflow; it computes rows of the PPR matrix, and hence of the transformed Y, by sampling or power iterations, applies the SVD-based Frequent Directions sketching process periodically with each d rows it processes (Lines 8–12), and returns embeddings with guarantees at any time (Lines 14–15). We keep track of singular values in $\hat{\Sigma}$ alongside the sketch so as to avoid performing SVD upon a request for output; as in [28], we multiply by $\sqrt{\hat{\Sigma}}$ at output time (Line 15), whereas a covariance-oriented sketcher would use $\hat{\Sigma}$. The time to process *all* n nodes with $O(n/d)$ SVD iterations costing $O(d^2 n)$ is $O(dn^2)$.

Sketch-based embeddings inherit the covariance error bounds of sketching (Section 2.6), which hold *anytime*, even after processing only an arbitrary subset of rows. Thus, FREDE embeddings inherit the *anytime error guarantees* of Frequent Directions, which are valid after materializing only part of the similarity matrix, and superior to those of other sketch-based embeddings; it achieves $\mathsf{ce} \leq \epsilon$ on the submatrix $S_{[s]}$ built from any size-s subset of processed rows (nodes) when $d = O(n/\epsilon)$ [11], independently of s. In Section 4.8 we show that FREDE outperforms other sketch-based embeddings in anytime node classification.

Algorithm 1 FREDE algorithm

1: **function** FREDE(G, n, d)
2: $\quad W \leftarrow \mathtt{zeros}(2d, n)$ ▷ all zeros matrix $W \in R^{2d \times n}$
3: $\quad \hat{\Sigma} \leftarrow I(2d)$ ▷ diagonal identity matrix $\hat{\Sigma} \in R^{2d \times 2d}$
4: $\quad$ **for** $v \in V$ **do**
5: $\quad\quad x \leftarrow \mathtt{PersonalizedPageRank}(v)$
6: $\quad\quad y \leftarrow \log x + \log n$ ▷ PPR-like similarity row
7: $\quad\quad$ Insert y into the last zero valued row of W
8: $\quad\quad$ **if** W has no zero valued rows **then**
9: $\quad\quad\quad U, \Sigma, V^\top \leftarrow \mathtt{SVD}(\hat{\Sigma}W), \sigma \leftarrow \Sigma_{d,d}$
10: $\quad\quad\quad \hat{\Sigma}_{:d} \leftarrow \sqrt{\max(\Sigma^2_{:d} - \sigma^2 I_d, 0)}$ ▷ set d^{th} row of $\hat{\Sigma}$ to 0
11: $\quad\quad\quad \hat{\Sigma}_{d:} \leftarrow I_d$ ▷ set last d entries of $\hat{\Sigma}$ to 1
12: $\quad\quad\quad W_{:d} \leftarrow V^\top_{:d}, W_{d:} \leftarrow \mathbf{0}_{d \times n}$ ▷ zero last d rows of W
13: $\quad$ **return** $\hat{\Sigma}, W_{:d}$
14: **function** GetEmbedding($k \leq d$) ▷ Anytime
15: $\quad$ **return** $\sqrt{\hat{\Sigma}} W_{:k}$ ▷ first k rows

3.3 Parallelization and distribution

The steps of Algorithm 1 are parallelizable. Line 5 could employ approximate PPR [41, 43], and Line 9 efficient SVD calculations [13]. Such speedups trade quality for scalability. Furthermore, FREDE can be efficiently *distributed* across machines for the sake of scalability, with very small communication overhead and *preserving* its quality guarantees. This appealing characteristic, unique among related works on embeddings, follows from the mergeability property that FREDE inherits from Frequent Directions. In each machine m, we may create a partial embedding matrix W based on the subset of the nodes available to m, and then `merge` partial embeddings from t servers, i.e., iteratively sketch their concatenations by Equation 2 in hierarchical fashion, incurring a $\log_2 t$ time complexity factor.

4 EXPERIMENTS

The primary advantage of FREDE is its *anytime* character, i.e., its ability to derive embeddings by processing only a fraction of similarity matrix rows. On that front, it may only be compared against other sketch-based embeddings. Here, we also compare FREDE on qualitative performance in data science tasks against other graph embeddings to corroborate its practical impact.

4.1 Compared methods

We evaluate FREDE against three sketching baselines, exact matrix factorization by SVD, and all methods in Table 1 bar those that (i) use linear transforms, which previous work [36] has established underperform nonlinear ones (i.e., HOPE, AROPE, ApproxPPR, NRP), (ii) require heavy hyperparameter tuning (i.e., Node2vec and NodeSketch), or (iii) underperform DeepWalk (i.e., LINE):

- Sketching baselines, i.e., **Hashing**, **Random Projections** and **Sampling** (Section 2.6), compute the sketch and filter singular values as in FREDE. Our **Random Projections** baseline is a refined variant of [46], substituting a crude higher-order matrix approximation with the row-update sketching algorithm applied on the transformed PPR matrix, and **hashing** is a variant of [26].
- SVD is the **exact SVD decomposition** of the nonlinearly tranformed PPR matrix Y with the same parameters as in FREDE, against which we were able to compare on the three smallest datasets.
- DeepWalk[2] [25] learns an embedding by sampling fixed-length random walks from each node and applying word2vec-based learning on those walks; despite intensive research on graph embeddings, DeepWalk remains competitive when used with time-tested default parameters [36]: walk length $t = 80$, number of walks per node $\gamma = 80$, and window size $T = 10$; we use these values.
- VERSE[3] [36] trains a single-layer neural network to learn the PPR similarity measure via sampling, with default parameters $\alpha = 0.85$ and $\mathtt{nsamples} = 10^6$.
- NetMF[4] [28] performs SVD on the closed-form DeepWalk matrix. We use the optimal method, NetMF-small; as it is not scalable, we evaluate it on our three smallest datasets, using the same parameters as in DeepWalk, and bias $b = 1$ as in the original paper.
- NetSMF[5] [28] sparsifies the NetMF similarity matrix to attain scalability forfeiting optimality; we run it with default $M = 10^3$.
- LouvainNE[6] [5] learns embeddings in a hierarchical fashion using the Louvain hierarchical clustering method; the final node

[2] https://github.com/xgfs/deepwalk-c
[3] https://github.com/xgfs/verse
[4] Code in the supplementary material.
[5] https://github.com/xptree/NetSMF
[6] https://github.com/maxdan94/LouvainNE

embeddings are a concatenations of cluster embeddings, with default parameters parameters $\alpha = 0.01$ and no restriction on h_{max}, the maximum number of levels.

- STRAP[7] [42] obtains embeddings through sparse factorization of approximate PPR vectors computed with a backward-push algorithm. We use the default setting with $\epsilon = 10^{-5}$.
- ProNE[8] [45] applies spectral propagation on embeddings obtained by sparsely factorizing a weighted adjacency matrix. We use the default parameters $k = 10$, $\mu = 0.2$, and $\theta = 0.5$.

4.2 Datasets

We experiment on 8 publicly available real[9,10] datasets.

- PPI [12, 32]: a protein-protein interaction dataset, where labels represent hallmark gene sets of specific biological states.
- POS [12, 19]: a word co-occurrence network built from Wikipedia data. Labels tag parts of speech induced by Stanford NLP parser.
- BlogCatalog [34, 44]: a social network of bloggers from the blogcatalog website. Labels represent self-identified topics of blogs.
- CoCit [1, 36]: a paper citation graph generated from the Microsoft Academic graph, featuring papers published in 15 major data mining conferences. We use conference identifiers as labels.
- CoAuthor [1, 36]: a coauthorsip graph from Microsoft Academic. We use snapshots from 2014 and 2016 for link prediction.
- VK [36]: a Russian all-encompassing social network. Labels represent user genders. We use snapshots from November 2016 and May 2017 for link prediction.
- Flickr [34, 44]: a photo-sharing social network, where labels represent user interests, and edges messages between users.
- YouTube [34, 44]: a video-based network; labels are genres.

	Size			Statistics	
dataset	$\lvert V\rvert$	$\lvert E\rvert$	$\lvert\mathcal{L}\rvert$	Avg. deg.	Density
PPI	4k	77k	50	19.9	5.1×10^{-3}
POS	5k	185k	40	38.7	8.1×10^{-3}
BlogCatalog	10k	334k	39	64.8	6.3×10^{-3}
CoCit	44k	195k	15	8.86	2.0×10^{-4}
CoAuthor	52k	178k	—	6.94	1.3×10^{-4}
VK	79k	2.7M	—	34.1	8.7×10^{-4}
Flickr	80k	12M	195	146.55	1.8×10^{-3}
YouTube	1.1M	3M	47	5.25	9.2×10^{-6}

Table 2: Dataset characteristics: number of vertices $|V|$, number of edges $|E|$; number of node labels $|\mathcal{L}|$; average node degree; density defined as $|E|/\binom{|V|}{2}$.

Table 2 summarises the data characteristics. All algorithms are implemented in Python[11] and ran on a 2×20-core Intel E5-2698 v4 CPU machine with 384Gb RAM and a 64Gb memory constraint.

4.3 Parameter settings

We set embedding dimension $d = 128$ unless indicated otherwise. For SVD, we use the `gesdd` routine in the Intel MKL library. For classification we use LIBLINEAR [10]. We repeat each experiment 10 times and evaluate each embedding 10 times.

[7] https://github.com/yinyuan1227/STRAP-git
[8] https://github.com/THUDM/ProNE
[9] https://github.com/xgfs/verse/tree/master/data
[10] http://leitang.net/code/social-dimension/data/flickr.mat
[11] https://github.com/xgfs/FREDE

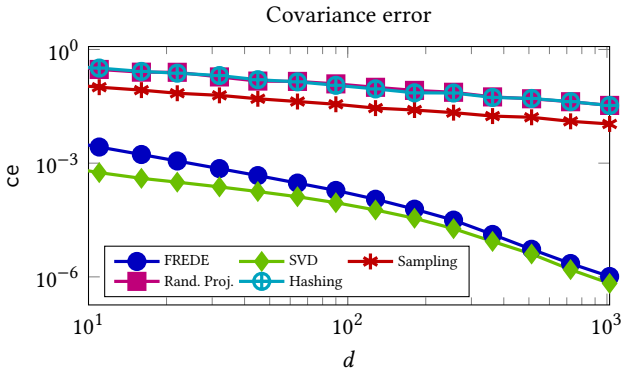

Figure 3: Covariance error vs. dimensionality d; FREDE approaches SVD, which yields optimal covariance error.

4.4 Sketching quality

As a preliminary test, we assess our choice of sketching backbone against other sketching algorithms and the optimal rank-k covariance approximation obtained by SVD on the full similarity matrix, $\tilde{S}^\top \tilde{S} = V_d \Sigma_d^2 V_d^\top$. Figure 3 reports the covariance error ce on PPI data, vs. the dimensionality d. FREDE outperforms the other sketching algorithms (Section 2.6) by at least 2 orders of magnitude and, as d grows, it converges to the optimal SVD solution. This result reconfirms that the advantages of Frequent Directions versus other sketching methods transfer well to the domain of graph embeddings. For the sake of completeness, we keep comparing to other sketching methods in the rest of our study, as performance may vary depending on the downstream data science task.

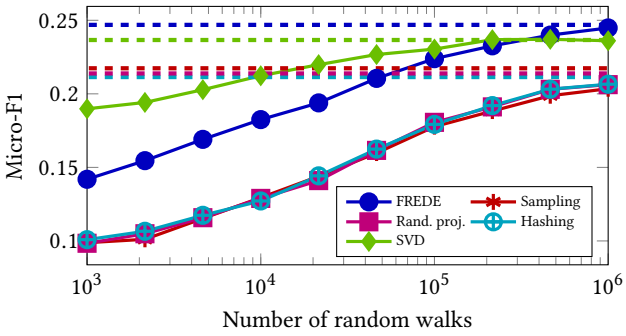

Figure 4: Classification performance of sketching algorithms on PPI data wrt. number of walks to compute PPR.

4.5 PPR approximation

Figure 4 shows the performance of sketching algorithms on a node classification task (predicting correct labels) vs. the number of random walks for PPR approximation. FREDE consistently outperforms sketching baselines and reaches the exact-PPR solution with 10^6 walks. This result indicates that we can achieve performance obtained using the exact PPR values in downstream tasks even without computing such PPR values with high precision. In the rest of our experiments, we compute PPR values by power iterations, as that is feasible with the data we use.

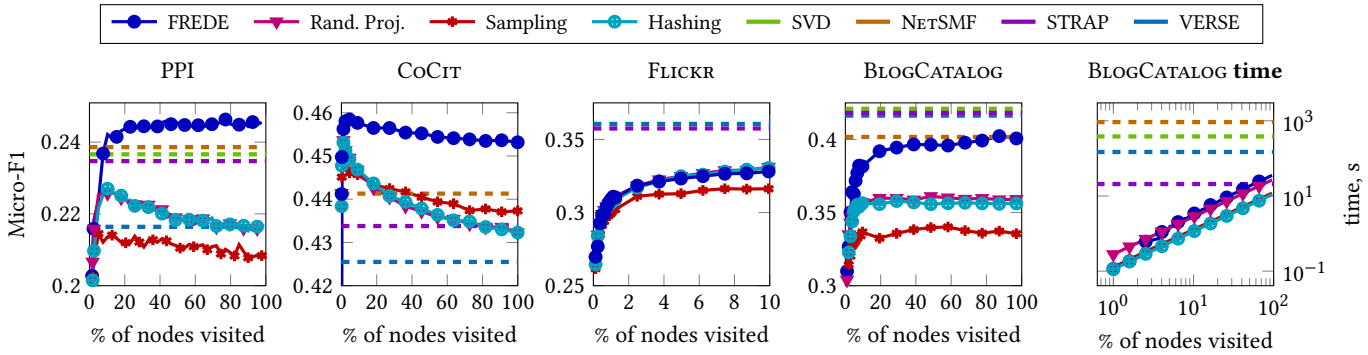

Figure 5: Classification performance of FREDE with varying percentage of the graph as input on three datasets.

	labelled nodes, %				
method	10%	30%	50%	70%	90%
DeepWalk	16.33	19.74	21.34	22.39	23.38
NetMF	18.58	22.01	23.87	24.65	25.30
NetSMF	18.45	22.38	23.86	24.59	25.56
VERSE	16.45	19.89	21.64	23.08	23.84
LouvainNE	13.8	15.91	16.59	16.83	17.06
STRAP	18.25	21.94	23.47	24.15	24.91
ProNE	17.56	22.53	24.29	25.03	26.02
FREDE	19.56	23.11	24.38	25.11	25.52
SVD	18.31	22.12	23.66	25.03	25.78
Rand. Proj.	16.80	19.99	21.45	22.38	23.14
Sampling	16.25	19.55	20.93	21.85	22.68
Hashing	16.73	19.97	21.51	22.43	23.44

Table 3: Micro-F1 classification, PPI data.

	labelled nodes, %				
method	10%	30%	50%	70%	90%
DeepWalk	43.42	47.12	48.96	49.86	50.18
NetMF	43.42	46.98	48.52	49.23	49.72
NetSMF	42.06	44.25	45.17	45.60	46.25
VERSE	40.80	44.70	46.60	47.65	48.24
LouvainNE	41.19	41.62	41.72	41.94	41.84
STRAP	47.26	50.85	52.08	52.70	53.60
ProNE	47.41	52.18	53.84	54.57	55.10
FREDE	46.59	49.23	50.45	51.02	51.30
SVD	44.69	48.86	50.57	51.53	52.20
Rand. Proj.	40.24	43.87	45.65	46.43	47.18
Sampling	40.35	43.80	45.39	46.30	46.69
Hashing	40.17	43.88	45.44	46.35	46.79

Table 4: Micro-F1 classification, POS data.

4.6 Node classification

Tables 3–8 report classification results in terms of the popular Micro-F1 measure [25, 33]; Macro-F1 results are similar. SVD is featured where it runs within 64Gb. For each dataset, we repeat the experiment 10 times and report the average. Surprisingly, on PPI and POS, FREDE outperforms its exact counterpart, SVD, and consistently supersedes its sketching counterparts across all datasets.

4.7 Link prediction

Link prediction is the task of predicting the appearance of a link between pairs of nodes in a graph. Tables 10 and 11 report link prediction accuracy (predicting the appearance of a link) on CoAuthor and VK by a logistic regression classifier on features derived from embeddings by the rules in Table 9. As a baseline, we use common link prediction features (node degree, number of common neighbors, Adamic-Adar index, Jaccard coefficient, and preferential attachment). We represent absent links in the training data by

	labelled nodes, %				
method	10%	30%	50%	70%	90%
DeepWalk	36.22	39.84	41.22	42.06	42.53
NetMF	36.62	39.80	41.05	41.70	42.17
NetSMF	35.74	39.16	40.17	40.82	41.06
VERSE	35.82	40.06	41.63	42.63	43.14
LouvainNE	18.40	19.99	20.68	21.17	21.40
STRAP	37.48	40.74	41.82	42.40	42.88
ProNE	36.74	40.19	41.27	41.75	41.99
FREDE	35.69	38.88	39.98	40.54	40.75
SVD	37.60	40.99	42.10	42.66	43.47
Rand. Proj.	30.82	34.43	35.81	36.52	37.16
Sampling	29.44	32.32	33.41	34.04	34.29
Hashing	30.81	34.36	35.82	36.65	37.28

Table 5: Micro-F1 classification, BlogCatalog data.

	labelled nodes, %				
method	1%	3%	5%	7%	9%
DeepWalk	37.22	40.34	41.72	42.59	43.16
NetSMF	41.07	43.29	44.13	44.61	44.99
VERSE	38.95	41.20	42.55	43.41	44.01
LouvainNE	35.29	36.56	37.14	37.49	37.67
STRAP	35.86	41.93	43.38	43.80	44.12
ProNE	35.82	39.54	40.72	41.46	41.97
FREDE	42.46	44.56	45.39	45.84	46.17
Rand. Proj.	40.89	42.63	43.63	44.32	44.78
Sampling	40.84	42.97	43.93	44.49	44.91
Hashing	40.86	42.66	43.65	44.29	44.83

Table 6: Micro-F1 classification, CoCit data.

	labelled nodes, %				
method	1%	3%	5%	7%	9%
DeepWalk	32.39	36.02	37.41	38.15	38.70
VERSE	30.08	34.22	36.06	37.11	37.83
LouvainNE	22.19	22.60	22.73	22.89	23.03
STRAP	30.13	34.26	35.75	36.60	37.14
ProNE	30.90	35.11	36.63	37.47	38.04
FREDE	30.90	32.98	33.86	34.48	34.88
Rand. Proj.	28.92	32.21	33.82	34.76	35.49
Sampling	28.46	30.97	32.08	32.75	33.24
Hashing	29.07	32.23	33.77	34.75	35.48

Table 7: Micro-F1 classification, Flickr data.

	labelled nodes, %				
method	1%	3%	5%	7%	9%
DeepWalk	37.96	40.54	41.75	42.60	43.37
VERSE	38.04	40.50	41.72	42.59	43.33
LouvainNE	31.41	32.28	32.65	33.00	33.29
STRAP	32.82	38.02	40.03	41.13	41.75
ProNE	38.40	42.20	43.09	43.69	44.04
FREDE	34.51	37.37	38.78	39.40	39.95
Rand. Proj.	33.88	36.10	37.23	37.94	38.38
Sampling	33.97	35.66	36.37	37.19	37.71
Hashing	32.64	35.64	36.92	37.46	38.13

Table 8: Micro-F1 classification, YouTube data.

Operator	Result
Average	$(\mathbf{a} + \mathbf{b})/2$
Concat	$[\mathbf{a}_1, \ldots, \mathbf{a}_d, \mathbf{b}_1, \ldots, \mathbf{b}_d]$
Hadamard	$[\mathbf{a}_1 * \mathbf{b}_1, \ldots, \mathbf{a}_d * \mathbf{b}_d]$
Weighted L1	$[\lvert\mathbf{a}_1 - \mathbf{b}_1\rvert, \ldots, \lvert\mathbf{a}_d - \mathbf{b}_d\rvert]$
Weighted L2	$[(\mathbf{a}_1 - \mathbf{b}_1)^2, \ldots, (\mathbf{a}_d - \mathbf{b}_d)^2]$

Table 9: Edge embedding strategies for link prediction, nodes $u, v \in V$ and corresponding embeddings $\mathbf{a}, \mathbf{b} \in \mathbb{R}^d$.

method	Average	Concat	Hadamard	L1	L2
DeepWalk	68.97	68.43	66.61	78.80	77.89
NetSMF	74.59	74.24	81.82	64.73	64.57
VERSE	79.62	79.25	86.27	75.15	75.32
LouvainNE	67.88	67.45	67.85	69.66	69.91
STRAP	79.08	78.72	80.32	71.81	72.05
ProNE	74.15	73.88	74.74	74.80	73.89
FREDE	81.28	80.95	86.83	81.70	82.37
Rand. Proj.	80.81	80.54	86.73	80.79	81.42
Sampling	80.98	80.74	86.45	79.53	79.51
Hashling	80.84	80.48	86.66	80.59	81.33
Baseline			77.53		

Table 10: Link prediction accuracy, CoAuthor data.

method	Average	Concat	Hadamard	L1	L2
DeepWalk	69.98	69.83	69.56	78.42	77.42
NetSMF	72.63	72.51	68.17	74.52	74.05
VERSE	74.56	74.42	80.94	77.16	77.47
LouvainNE	66.87	66.78	67.74	67.42	67.44
STRAP	74.35	74.23	76.93	67.84	66.10
ProNE	71.29	71.14	74.92	73.54	74.41
FREDE	74.68	74.59	77.63	74.25	73.60
Rand. Proj.	74.41	74.27	77.01	74.33	74.56
Sampling	74.38	74.27	76.82	72.26	71.95
Hashing	74.36	74.27	76.86	74.30	74.56
Baseline			78.84		

Table 11: Link prediction accuracy, VK data.

negative sampling, and use 50% of links for training and remaining 50% for testing. FREDE outperforms all methods on CoCit, and all sketching baselines on VK. Surprisingly, sketching baselines perform better than state-of-the-art graph embeddings on CoCit.

4.8 Anytime classification

We study anytime operation (Section 3.2) on node classification using 50% of nodes for training and processing PPR rows in random order. Figure 5 presents results for PPR-based methods and NetSMF on three datasets. FREDE outperforms all others on PPI, as in Table 3, after processing only 1% of similarities. Remarkably, on CoCit data, all sketchers outperform all other methods after processing about 3% of nodes; their downstream performance drops thereafter, a fact indicating that more information, revealing inter-cluster connections, harms the classification outcomes. A similar effect appears for sketching baselines with PPI data, yet not for FREDE, which outperforms all others after processing less than 20% of nodes and keeps growing thereafter. FREDE also performs competitively on Flickr (we examine up to 10% of nodes, as Random Projections was inefficient; SVD and NetSMF did not run within 64Gb) and BlogCatalog, as in Tables 5 and 7. The rightmost plot in Figure 5 shows runtime on BlogCatalog; while sketchers' runtime grows linearly, those of one-off methods, except STRAP, stand apart. These results illustrate that embedding merging preserves the downstream quality; as Equation 2 shows, merging two embeddings amounts to sketching their concatenation; therefore, the sketch operation Algorithm 1 periodically performs with each new d similarity matrix rows it processes can also be viewed as a `merge` operation.

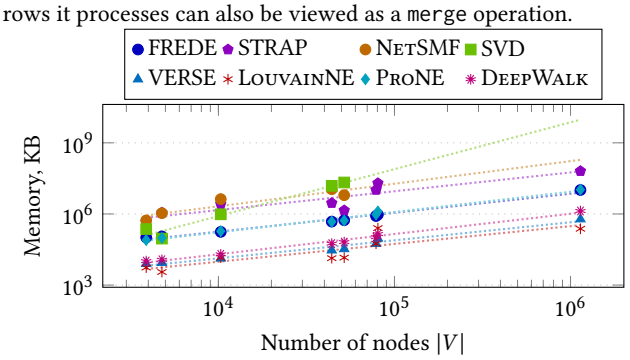

Figure 6: Memory consumption per datasets and regression lines; SVD and NetSMF could not fit YouTube in 64Gb RAM.

4.9 Memory consumption

Lastly, Figure 6 shows the memory consumption for each method and dataset, with linear regression on data sizes. All methods bar SVD and NetSMF embed the largest YouTube graph (10^6 nodes) on a 64Gb RAM commodity machine. FREDE consumes memory comparable to that of neural methods and never exceeds that of other factorization-based embeddings.

5 CONCLUSION

Since graph embeddings implicitly aim to preserve similarity matrix covariance, row-wise sketching techniques are suited therefor. We applied a state-of-the-art sketcher, Frequent Directions, on a matrix-factorization interpretation of a state-of-the-art embedding, VERSE, to craft FREDE: a linear-space graph embedding that allows for scalable data science operations on graph data, as well as for anytime and distributed computation *with error guarantees*. Besides its anytime character, FREDE achieves almost as low covariance error as the exact SVD solution and stands its ground against previous graph embeddings even after processing as little as 10% of similarity matrix rows; therefore, it promises significant practical impact. In the future, we plan to examine the feasibility of anytime local embeddings [26], applications of graph embeddings in the context of graph summarization [31] and anonymization [39], and augment FREDE using recent enhancements on Frequent Directions [14].

REFERENCES

[1] 2016. Microsoft Academic Graph - KDD cup 2016. https://kddcup2016.azurewebsites.net/Data.

[2] Pankaj K. Agarwal, Graham Cormode, Zengfeng Huang, Jeff M. Phillips, Zhewei Wei, and Ke Yi. 2013. Mergeable summaries. *TODS* (2013), 26:1–26:28.

[3] Nesreen Ahmed, Ryan Anthony Rossi, John Lee, Theodore Willke, Rong Zhou, Xiangnan Kong, and Hoda Eldardiry. 2020. Role-based Graph Embeddings. *TKDE* (2020).

[4] Bahman Bahmani, Abdur Chowdhury, and Ashish Goel. 2010. Fast Incremental and Personalized PageRank. *PVLDB* 4, 3 (2010), 173–184.

[5] Ayan Kumar Bhowmick, Koushik Meneni, Maximilien Danisch, Jean-Loup Guillaume, and Bivas Mitra. 2020. LouvainNE: Hierarchical Louvain Method for High Quality and Scalable Network Embedding. In *WSDM*. 43–51.

[6] Christos Boutsidis, Petros Drineas, and Malik Magdon-Ismail. 2011. Near Optimal Column-Based Matrix Reconstruction. *FOCS* (2011), 305–314.

[7] Christos Boutsidis, Michael W. Mahoney, and Petros Drineas. 2009. An Improved Approximation Algorithm for the Column Subset Selection Problem. In *SODA*.

[8] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2015. GraRep: Learning Graph Representations with Global Structural Information. In *CIKM*.

[9] Kenneth L. Clarkson and David P. Woodruff. 2013. Low rank approximation and regression in input sparsity time. In *STOC*.

[10] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. 2008. LIBLINEAR: A library for large linear classification. *Journal of machine learning research* (2008).

[11] Mina Ghashami, Edo Liberty, Jeff M. Phillips, and David P. Woodruff. 2016. Frequent Directions : Simple and Deterministic Matrix Sketching. *SIAM J. Comput.* 45 (2016), 1762–1792.

[12] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable Feature Learning for Networks. In *KDD*.

[13] Michael P Holmes, Jr Isbell, Charles Lee, and Alexander G Gray. 2009. QUIC-SVD: Fast SVD using cosine trees. In *NIPS*. 673–680.

[14] Zengfeng Huang. 2018. Near Optimal Frequent Directions for Sketching Dense and Sparse Matrices. In *ICML*. 2048–2057.

[15] John A. Lee and Michel Verleysen. 2007. *Nonlinear Dimensionality Reduction* (1st ed.). Springer Publishing Company, Incorporated.

[16] Omer Levy and Yoav Goldberg. 2014. Neural Word Embedding as Implicit Matrix Factorization. In *NIPS*.

[17] Ping Li, Trevor J. Hastie, and Kenneth Ward Church. 2006. Very sparse random projections. In *KDD*.

[18] Edo Liberty. 2013. Simple and deterministic matrix sketching. In *KDD*.

[19] Matt Mahoney. 2011. Large text compression benchmark. http://www.mattmahoney.net/text/text.html.

[20] Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *NIPS*.

[21] Jayaved Misra and David Gries. 1982. Finding repeated elements. *Science of Computer Programming* (1982).

[22] Cameron Musco and Christopher Musco. 2015. Randomized Block Krylov Methods for Stronger and Faster Approximate Singular Value Decomposition. In *NeurIPS*. 1396–1404.

[23] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. 2016. Asymmetric Transitivity Preserving Graph Embedding. In *KDD*.

[24] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. The PageRank citation ranking: bringing order to the web. (1999).

[25] Bryan Perozzi, Rami Al-Rfou', and Steven Skiena. 2014. DeepWalk: online learning of social representations. In *KDD*.

[26] Stefan Postavaru, Anton Tsitsulin, Filipe Miguel Gonçalves de Almeida, Yingtao Tian, Silvio Lattanzi, and Bryan Perozzi. 2020. InstantEmbedding: Efficient Local Node Representations. *CoRR* abs/2010.06992 (2020).

[27] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Chi Wang, Kuansan Wang, and Jie Tang. 2019. NetSMF: Large-Scale Network Embedding as Sparse Matrix Factorization. In *WWW*.

[28] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. 2018. Network Embedding as Matrix Factorization: Unifying DeepWalk, LINE, PTE, and node2vec. In *WSDM*.

[29] Leonardo FR Ribeiro, Pedro HP Saverese, and Daniel R Figueiredo. 2017. struc2vec: Learning node representations from structural identity. In *KDD*. 385–394.

[30] Ryan A Rossi, Nesreen K Ahmed, Eunyee Koh, Sungchul Kim, Anup Rao, and Yasin Abbasi-Yadkori. 2020. A structural graph representation learning framework. In *WSDM*. 483–491.

[31] Tara Safavi, Caleb Belth, Lukas Faber, Davide Mottin, Emmanuel Müller, and Danai Koutra. 2019. Personalized Knowledge Graph Summarization: From the Cloud to Your Pocket. In *ICDM*. 528–537.

[32] Chris Stark, Bobby-Joe Breitkreutz, Teresa Reguly, Lorrie Boucher, Ashton Breitkreutz, and Mike Tyers. 2006. BioGRID: a general repository for interaction datasets. *Nucleic Acids Research* (2006).

[33] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. LINE: Large-scale Information Network Embedding. In *WWW*.

[34] Lei Tang and Huan Liu. 2009. Scalable learning of collective behavior based on sparse social dimensions. In *CIKM*.

[35] Mikkel Thorup and Uri Zwick. 2005. Approximate distance oracles. *JACM* 52, 1 (2005), 1–24.

[36] Anton Tsitsulin, Davide Mottin, Panagiotis Karras, and Emmanuel Müller. 2018. VERSE: Versatile Graph Embeddings from Similarity Measures. In *WWW*.

[37] Santosh S Vempala. 2005. *The random projection method.* American Math. Soc.

[38] David P Woodruff. 2014. Sketching as a Tool for Numerical Linear Algebra. *Theoretical Computer Science* (2014).

[39] Mingqiang Xue, Panagiotis Karras, Chedy Raïssi, Panos Kalnis, and Hung Keng Pung. 2012. Delineating social network data anonymization via random edge perturbation. In *CIKM*. 475–484.

[40] Dingqi Yang, Paolo Rosso, Bin Li, and Philippe Cudré-Mauroux. 2019. NodeSketch: Highly-Efficient Graph Embeddings via Recursive Sketching. In *KDD*. 1162–1172.

[41] Renchi Yang, Jieming Shi, Xiaokui Xiao, Yin Yang, and Sourav S. Bhowmick. 2020. Homogeneous Network Embedding for Massive Graphs via Reweighted Personalized PageRank. *PVLDB* 13, 5 (2020), 670–683.

[42] Yuan Yin and Zhewei Wei. 2019. Scalable graph embeddings via sparse transpose proximities. In *KDD*. 1429–1437.

[43] Minji Yoon, Jinhong Jung, and U Kang. 2018. TPA: Fast, scalable, and accurate method for approximate random walk with restart on billion scale graphs. In *ICDE*. IEEE, 1132–1143.

[44] R. Zafarani and H. Liu. 2009. Social Computing Data Repository at ASU. http://socialcomputing.asu.edu

[45] Jie Zhang, Yuxiao Dong, Yan Wang, Jie Tang, and Ming Ding. 2019. ProNE: Fast and Scalable Network Representation Learning.. In *IJCAI*. 4278–4284.

[46] Ziwei Zhang, Peng Cui, Haoyang Li, Xiao Wang, and Wenwu Zhu. 2018. Billion-scale Network Embedding with Iterative Random Projection. In *ICDM*. 787–796.

[47] Ziwei Zhang, Peng Cui, Xiao Wang, Jian Pei, Xuanrong Yao, and Wenwu Zhu. 2018. Arbitrary-Order Proximity Preserved Network Embedding. In *KDD*.

[48] Shlomo Zilberstein. 1996. Using Anytime Algorithms in Intelligent Systems. *AI Magazine* 17, 3 (1996), 73–83.