# Fast online node labeling with graph subsampling

**Anonymous authors**
**Paper under double-blind review**

## Abstract

Large data applications rely on storing data in massive, sparse graphs with millions to trillions of nodes. Graph-based methods, such as node prediction, aim for computational efficiency regardless of graph size. Techniques like localized approximate personalized page rank (APPR) solve sparse linear systems with complexity independent of graph size, but is in terms of the maximum node degree, which can be much larger in practice than the average node degree for real-world large graphs. In this paper, we consider an *online subsampled APPR method*, where messages are intentionally dropped at random. We use tools from graph sparsifiers and matrix linear algebra to give approximation bounds on the graph's spectral properties ($O(1/\epsilon^2)$ edges), and node classification performance (added $O(n\epsilon)$ overhead).

## 1 Introduction

Large data applications like search and recommendation systems, rely on data stored in the form of very large, sparse, irregular graphs, where the number of nodes can be on the order millions, billions, or even trillions. In such cases, graph-based methods, such as node prediction, must accomplish their tasks *exclusively using local operations*, e.g. where the memory complexity is independent of graph size. This is the intention of methods like the localized approximate Personalized Page Rank method (APPR) (Andersen et al., 2006; Page et al., 1999), which approximates solving a sparse linear system by truncating messages whenever the residual of that coordinate is small. However, the complexity for these methods are often in terms of the maximum node degree, and their benefits are tied to the assumption that the graph node degrees are relatively uniform.

In this work, we consider a simple solution to this problem, where high-degree nodes subsample their neighbors in message-passing methods. This solution can save considerable memory overhead when the graph's node degree distribution is heavily skewed. (See, for example, Figure 1.) However, the disadvantage of this strategy is that the stochasticity leads a high variance between each trial; thus, we implement present a mechanism for grounding the residual at each iteration, to reduce this variance.

We then evaluate this method on two APPR downstream tasks, one supervised and one unsupervised. In online node labeling, future node labels are predicted using the revealed labels of the past; the APPR method is used here to solve a linear system, with a carefully tuned right-hand-side as motivated by the graph regularization method of Belkin et al. (2004), and the relaxation method of Rakhlin et al. (2012). In unsupervised clustering, we use the APPR method to acquire an improved similarity matrix, which is then clustered using nearest neighbors.

**Contributions.** In this paper, we extend the work of (Andersen et al., 2006; Zhou et al., 2023) to graphs with unfavorable node degrees, using edge subsampling. We propose a simple approach: for a threshold $\bar{q}$, we identify all nodes with degree exceeding $\bar{q}$, and subsample their neighboring edges until they have $\leq \bar{q}$ neighbors. The remaining edges are reweighted such that the expected edge weights are held consistent. In offline graphs, sparsifications of this kind is $O(n)$ where $n$ is the number of nodes; however, the dependency on $n$ is removed when done online. Our contributions are

- We give a variance reduced subsampling APPR strategy which incrementally updates the primal variable (such as in iterative refinement), producing a stable higher-accuracy estimate for very little overhead.
- We give concentration bounds on the learning performance in offline sparsification, which are comparable to that of optimal sparsifiers in previous literature.
- In the case of online sparsification, we give high probability guarantees that the method will not stop early, and show a $O(1/T)$ convergence rate overall and a linear convergence rate in expectation.
- We show superior numerical performance of online node labeling and graph clustering when subsampled APPR is integrated.

### 1.1 Related works

#### 1.1.1 Applications.

We investigate two primary node labeling applications. In (supervised) online node labeling, the nodes are visited one at a time, and at time $t$, one infers the $t$th node using the revealed labels of nodes $y_1, ..., y_{t-1}$. (This models interactive applications such as online purchasing or web browsing.) In (unsupervised) graph clustering, the graph nodes are preemptively grouped, using some standard clustering method over node embeddings learned through APPR.

**Online node prediction.** This problem category has been investigated by Belkin et al. (2004); Zhu et al. (2003); Herbster & Pontil (2006); Rakhlin & Sridharan (2017), and many others. Using $\boldsymbol{z}_t$ as a vector containing the already revealed label, then the solution to the APPR system is an approximation of those offered in Belkin et al. (2004); Zhu et al. (2003), for choices of regularization and interpolation. Relatedly, node prediction via approximate Laplacian inverse is also related to mean field estimation using truncated discounted random walks (Li et al., 2019), with known performance guarantees. The series of papers (Herbster & Pontil, 2006; Herbster et al., 2005) directly attack the suboptimality in online learning when applied to graphs with large diameters. They show that for this class of problems, using direct interpolation, the worst-case rate cannot be sublinear, and suggest additional graph structures to assist in this regime. Finally, Rakhlin & Sridharan (2017) tackled the problem of computing an online learning bound by offering a method, which can be seen as a modifined right-hand-side of the linear system in Belkin & Niyogi (2003). The advantage of Rakhlin & Sridharan (2017) is that it provides a means of computing a learning regret bound. In Zhou et al. (2023), the analysis was tightened to $O(\sqrt{n})$ sublinear regret bounds, under the appropriate choice of kernel.

**Node embeddings and graph clustering.** The idea of learning node embeddings over large graphs has now been well studied, with tools like node2vec (Grover & Leskovec, 2016) and DeepWalk (Perozzi et al., 2014); see also Garcia Duran & Niepert (2017). Using the Laplacian as an eigenmap is also classical (Belkin & Niyogi, 2003; Weinberger et al., 2004). It is also the use of PPR vectors as node embeddings that drove the original APPR method (Andersen et al., 2006), and has been extended to more applications such as clustering (Guo et al., 2024) and improving GNNs (Bojchevski et al., 2020).

#### 1.1.2 Primary tool.

In both applications, the primary tool is to quickly and efficiently solve a linear system where the primary matrix is the sparse graph Laplacian. Specifically, in many past works, the cost of solving this linear system is not accounted for in the method's complexity analysis, with the justification that there are other existing methods for an offline linear system solve; for example, the combination of offline sparsifiers (Spielman & Srivastava, 2008) and fast iterative methods (Saad, 1980). Such a method reduces the $O(n^3)$ cost of direct linear systems solve to $O(n \log(n)/\epsilon^2) + \tilde{O}(n)$, which is a significant reduction. (Here, $n$ is the number of nodes in the graph). *However, for large enough $n$, any dependency on $n$ makes the method intractable.*

**Local methods for linear systems.** While Page et al. (1999) presented the infamous Personalized PageRank (PPR) method, Andersen et al. (2006) provided the analysis that showed that, for linear systems involv-

ing graph Laplacians, adding mild thresholding and a specific choice of weighting would ensure a bounded sparsity on *all* the intermediate variables used in computation; moreover, this bound was independent of graph size (e.g. number of nodes or edges), and depended only on node degree. Fountoulakis et al. (2019) also connected this method with $\ell_1$ penalization of a quadratic minimization problem, which offered a variational perspective on the sparsifying properties of APPR. The analysis was also applied to node prediction in Zhou et al. (2023). However, in all cases, complexity bounds depend on node degree, and are only optimal when node degree is independent of graph size.

**Subsampling and sparsification.** Our main contribution is in the further acceleration of APPR through "influencer subsampling", where high degree nodes are targetted for subsampling in order to reduce the memory complexity of single-step message propagation. The analysis of this follows from prior work in graph sparsification. Specifically, existing offline graph sparsification methods include Karger (1994), which showed that random subsampling maintained cut bounds, with complexity $\tilde{O}(m + n/\epsilon^3)$; Benczúr & Karger (2015), who reduced this to $\tilde{O}(m \log^2(n))$ complexity and $O(n \log n/\epsilon^2)$ edges by subsampling less edges with estimated smaller cut values; and Spielman & Srivastava (2008); Spielman (2010); Spielman & Teng (2014) who used the principle of effective resistance to define subsampling weights to obtain optimal spectral bounds. In practice, the last two approaches are not feasible without additional randomized approaches, as computing cuts and effective resistances exactly also involve computing a large matrix pseudoinverse. More recently, Saito & Herbster (2023) investigated estimating this resistance by creating a coordinate spanning set, which is closely related to our proposed scheme, and applied it to spectral clustering.

## 2 Using graphs for node labeling

**Notation.** Denote a graph as $\mathcal{G}(\mathcal{V}, \mathcal{E}, \boldsymbol{A})$ where $\mathcal{V} = \{1, ..., n\}$ are the $n$ nodes, $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$ is the set of edges in $\mathcal{G}$, and $\boldsymbol{A}$ is the adjacency matrix for the unweighted graph (e.g. $\boldsymbol{A}_{u,v} > 0$ contain the edge weights whenever $(u, v) \in \mathcal{E}$). Denote $n = |\mathcal{V}|$ the number of nodes and $m = |\mathcal{E}|$ the number of edges. Denote $d_u = \sum_v \boldsymbol{A}_{u,v}$ the degree of node $u$ and $\boldsymbol{D} = \mathbf{diag}(d)$, and the neighbors of a node $u$ as $\mathcal{N}(u) = \{v : (u, v) \in \mathcal{E}\}$, and for a subset of nodes $\mathcal{S} \subseteq \mathcal{V}$, we express its volume as $\mathbf{vol}(\mathcal{S}) = \sum_{u \in \mathcal{S}} d_u$. For vectors, $\mathbf{supp}(\boldsymbol{x})$ is the set of indices $i$ where $x_i \neq 0$.

Now assume that each node has a ground truth binary label $y_i \in \{1, -1\}$, for $i \in \mathcal{V}$. For instance, nodes may represent customers, node label whether or not they will purchase a specific product. Consequently, the edges could reflect similarities or shared purchasing behaviors between customers. Thus, the likelihood of two connected nodes sharing the same label is high, making it possible to infer the label of an unobserved node based significantly on its neighbors.

**Simple baseline: Weighted majority algorithm.** Suppose that a small subset of the node labels are revealed, e.g. in a vector $\tilde{\boldsymbol{y}} = (\tilde{y}_1, ..., \tilde{y}_n) \in \mathbb{R}^n$ where $\tilde{y}_t \in \{-1, 1\}$ if node $t$'s label is revealed , and 0 otherwise. The revealed labels can be equal to, or approximate, the ground truth labels $\boldsymbol{y} = (y_1, ..., y_n) \in \mathbb{R}^n$, $y_t \in \{-1, 1\}$. Then, a simple method for inferring the label of an *unseen* node $t$ is to take the weighted average of its neighbors, e.g.

$$\hat{y}_t = \operatorname{sign}\left(\frac{\sum_{i \in \mathcal{N}(t)} \boldsymbol{A}_{i,t} \tilde{y}_i}{\sum_{j \in \mathcal{N}(t)} \boldsymbol{A}_{j,t}}\right) = \operatorname{sign}\left(\boldsymbol{e}_t^T \boldsymbol{A}\boldsymbol{D}^{-1}\tilde{\boldsymbol{y}}\right).$$

**Message passing and graph Laplacians.** We may generalize this further by expanding to the $K$-hop neighbors of $t$, using a discount factor $\beta^k$ where $k$ is the neighbor's hop distance to $t$, e.g. for $\hat{y}_t = \operatorname{sign}(\hat{y}_{t,\text{soft}})$, where for a transition matrix $\boldsymbol{T} = \boldsymbol{A}\boldsymbol{D}^{-1}$

$$\hat{y}_{t,\text{soft}} = \sum_{k=0}^{K} \sum_{i \in \mathcal{N}(t)} (\beta^{k-1}\boldsymbol{T}^k)_{i,t} \tilde{y}_i = \beta^{-1} \sum_{k=0}^{K} \boldsymbol{e}_t^T (\beta \boldsymbol{A}\boldsymbol{D}^{-1})^k \tilde{\boldsymbol{y}} \overset{K \to \infty}{=} \beta^{-1} \boldsymbol{e}_t^T (\boldsymbol{I} - \beta \boldsymbol{A}\boldsymbol{D}^{-1})^{-1} \tilde{\boldsymbol{y}}. \tag{1}$$

(Here, we take the sum from $k = 0$ without impunity since $\tilde{\boldsymbol{y}}_t = 0$.) That is to say, a simple baseline motivates that a *globally informative, discounted estimate* of the node labels can be written as the solution

to the linear system

$$\beta^{-1}(\boldsymbol{I} - \beta \boldsymbol{A}\boldsymbol{D}^{-1})\hat{\boldsymbol{y}} = \tilde{\boldsymbol{y}}. \tag{PPR}$$

This linear system is equivalent to the well-studied linear system of the personal page-rank (PPR) vectors, which first debued in web search engines (Page et al., 1999). A symmetrized version of the PPR system

$$\underbrace{(\boldsymbol{I} - \beta \boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2})}_{=:\boldsymbol{L}}\hat{\boldsymbol{y}} = \tilde{\boldsymbol{y}}. \tag{PPR-symm}$$

is more commonly studied in machine learning works (Rakhlin et al., 2012; Belkin et al., 2004). Here, the weighted Laplacian matrix $\boldsymbol{L}$ is symmetric positive semidefinite. Since the symmetrized linear system and the original PPR system are equivalent under a rescaling of the vectors $\hat{\boldsymbol{y}}$ and $\tilde{\boldsymbol{y}}$ by $\boldsymbol{D}^{1/2}$, one can view this symmetrized version as a further weighting of the seen labels, so that high-degree nodes have a tapered influence. In this work, we focus on solving this (PPR-symm) system.

### 2.1 Approximate Personalized Page Rank

The APPR method (Andersen et al., 2006), (Alg. 1) is a memory-preserving approximation of PPR. Specifically, by having proper termination steps in the PUSH substep, the method offers a better tradeoff between learnability and memory locality. It solves the linear system equivalent to that of (PPR-symm), with $\beta = \frac{1-\alpha}{1+\alpha} \in (0, 1)$ as the *teleportation parameter.*

$$\underbrace{\left(\boldsymbol{I} - \frac{1-\alpha}{1+\alpha}(\boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2})\right)}_{\boldsymbol{Q}}\boldsymbol{x} = \underbrace{\frac{2\alpha}{1+\alpha}\boldsymbol{D}^{-1/2}\boldsymbol{e}_s}_{\boldsymbol{b}}, \qquad \boldsymbol{z} = \frac{1+\alpha}{2\alpha}\boldsymbol{D}^{1/2}(\boldsymbol{b} - \boldsymbol{Q}\boldsymbol{x}). \tag{2}$$

where, given the solution $\boldsymbol{x}$, the variable $\pi = \boldsymbol{D}\boldsymbol{x}$ is the desired PPR vector in web applications. The matrix $\boldsymbol{Q}$ is symmetric positive semidefinite, with eigenvalues in the range $[2\alpha/(1+\alpha), 1]$. Note that Alg. 1 assumes that the right-hand side is $\boldsymbol{e}_s$; therefore solving a full linear system with a dense right-hand side requires $n$ APPR calls, accumulated through linearity.

In Alg. 1, Steps 4 to 7 can be summarized as a PUSH operation, because of its effect in "pushing" mass from the residual to the variables. The residual vector $\boldsymbol{z} = \frac{1+\alpha}{2\alpha}\boldsymbol{D}^{1/2}(\boldsymbol{b} - \boldsymbol{Q}\boldsymbol{x}^{(t)})$ is used to identify which node messages to "push forward", and which to terminate. In Fountoulakis et al. (2019), this method is shown to be closely related to the Fenchel dual of minimizing

$$f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T\boldsymbol{Q}\boldsymbol{x} + \boldsymbol{x}^T\boldsymbol{b} + \epsilon\|\boldsymbol{x}\|_1;$$

thus the steps of APPR can be viewed as directly promoting $\ell_1$-regularized sparsity (or, in the language of large graphs, memory locality).

**Algorithm 1** APPR($\mathcal{G}$, $s$, $\epsilon$)(Andersen et al., 2006)

**Require:** $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \boldsymbol{A})$, starting node $s$, tolerance $\epsilon$
1: **Init**: $\boldsymbol{x}^{(0)} \leftarrow \boldsymbol{0}$, $\boldsymbol{z}^{(0)} \leftarrow \boldsymbol{e}_s$, $t \leftarrow 1$, $\mathcal{S}^{(0)} = \{s\}$
2: **while** $\mathcal{S}^{(t)} \neq \emptyset$ **do**
3: Pick $u \in \mathcal{S}^{(t-1)}$
4: $\boldsymbol{x}_u^{(t)} \leftarrow \boldsymbol{x}_u^{(t-1)} + \alpha \cdot \frac{\boldsymbol{z}_u^{(t-1)}}{\sqrt{d_u}}$
5: **for** $v \in \mathcal{N}(u)$ **do**
6: $\boldsymbol{z}_v^{(t)} \leftarrow \boldsymbol{z}_v^{(t-1)} + \frac{(1-\alpha)\boldsymbol{A}_{u,v}}{2d_u}\boldsymbol{z}_u^{(t-1)}$
7: $\boldsymbol{z}_u^{(t)} \leftarrow \frac{(1-\alpha)}{2}\boldsymbol{z}_u^{(t-1)}$
8: $\mathcal{S}^{(t)} = \{v : v \in \mathcal{S}^{(t-1)} \cup \mathcal{N}(u),\ |\boldsymbol{z}_v| \geq d_v\epsilon\}$
9: $t \leftarrow t + 1$
10: **Return** $\boldsymbol{x}^{(t)}$

The following Lemma from Andersen et al. (2006) form the basis of the memory locality guarantee of this method.

**Lemma 2.1** (Monotonicity and conservation (Andersen et al., 2006)). *For all $t$, $\boldsymbol{x}^{(t)} \geq 0$, $\boldsymbol{r}^{(t)} \geq 0$. Moreover,*

$$\|\boldsymbol{x}^{(t+1)}\|_1 \geq \|\boldsymbol{x}^{(t)}\|_1, \quad \|\boldsymbol{r}^{(t+1)}\|_1 \leq \|\boldsymbol{r}^{(t)}\|_1, \|\boldsymbol{z}^{(t)}\|_1 + \|\boldsymbol{D}^{1/2}\boldsymbol{x}^{(t)}\|_1 = 1 \tag{3}$$

*for $\boldsymbol{z}^{(t)} = \frac{1+\alpha}{2\alpha}\boldsymbol{D}^{1/2}\boldsymbol{r}^{(t)}$. And, denoting $\mathbf{supp}(\boldsymbol{x})$ as the support of $\boldsymbol{x}$ (e.g. the set of indices $i$ where $x_i \neq 0$),*

$$\mathbf{supp}(\boldsymbol{x}^{(t)}) \subseteq \mathbf{supp}(\boldsymbol{x}^{(t+1)}) \subseteq \mathbf{supp}(\boldsymbol{x}^*).$$

In other words, *the memory complexity of the intermediate variables* $\boldsymbol{x}^{(t)}$ *is bounded by that of its final solution* $\boldsymbol{x}^*$. Note that this is not the typical case for sparse methods, such as the proximal gradient method, whose intermediate variables can be dense even if the final solution is sparse. Rather, it is the subtle interplay of $\alpha$, $\epsilon$, and the PUSH operation that maintains this quality.

**Bound on $\mathbf{supp}(\boldsymbol{r}^{(t)})$.** This auxiliary variable is also an important memory-using component. Because of the nature of the PUSH method, one can infer that

$$\mathbf{supp}(\boldsymbol{r}^{(t)}) \subseteq \mathbf{supp}(\boldsymbol{x}^{(t)}) \cup \{v : v \in \mathcal{N}(u),\ u \in \mathbf{supp}(\boldsymbol{x}^{(t)})\}.$$

In other words, for a graph with unweighted edges, the complexity of the auxiliary variable $\boldsymbol{r}^{(t)}$ is bounded by the complexity of the main variable $\boldsymbol{x}^{(t)}$

$$|\mathbf{supp}(\boldsymbol{r}^{(t)})| \leq |\mathbf{supp}(\boldsymbol{x}^{(t)})| + \mathbf{vol}(\mathbf{supp}(\boldsymbol{x}^{(t)})).$$

However, this bound is sensitive to the node degrees, especially for those active in the main variable.

## 3 Influencer-targeted sparsification

Degree distributions over real world graphs are often very heavy-tailed, as demonstrated in Figure 1. For message passing methods, this presents a practical challenge: if a node transmits messages to all its neighbors at each step, truncating the number of steps might not substantially alleviate the computational burden. This is due to the significant memory requirements when propagating messages from "influencer" nodes, e.g. those with exceptionally high degrees, which serve as major hubs in the network.

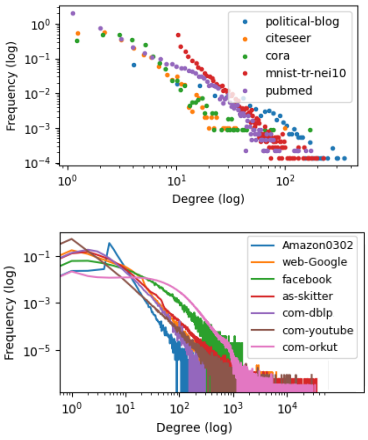


Figure 1: Degree distributions of small (top) vs large (bottom) graphs

To combat this, we propose two schemes: *offline sparsification*, where a new, sparsified graph is produced and used in place of the original one (in similar spirit as Karger (1994); Spielman & Srivastava (2008), etc.); and *online sparsification*, where the APPR method itself subsamples neighbors at each step. *A major goal of this paper is to show this method's consistency in learning tasks, and improved computational efficiency in practice.*

First, we ask a question: is it better to remove the edge of an "influencer" (high-degree node), or will such an action disproportionately degrade performance? To offer some preliminary intuition, we compare different sparsifiers in terms of correlation with resistive distance, a measure proposed by Spielman & Srivastava (2008) for spectrally optimal graph sparsification; and by analyzing the edge ratios of sparsified graphs to assess their alignment with node learnability.

### 3.1 Resistive distance

For an unnormalized graph Laplacian $\boldsymbol{L}_{\text{un}} = \boldsymbol{D} - \boldsymbol{A}$, the resistive distance between nodes $i$ and $j$ is

$$\boldsymbol{R}_{i,j} = \boldsymbol{L}^{\dagger}_{i,i} + \boldsymbol{L}^{\dagger}_{j,j} - 2\boldsymbol{L}^{\dagger}_{i,j}.$$

In electrical engineering, this measures the effective resistance between two junctions $i$ and $j$ in a network of resistors, whose weights form the weights of the graph ($\boldsymbol{A}_{i,j} = 1/r_{i,j}$). Importantly, resistive distance is known to be the optimal metric for offline sparsification for preserving spectral properties (Spielman & Srivastava, 2008). However, in general, it is challenging to compute, as it requires a full Laplacian pseudoinverse. Figure 2 compares a graph edge's "influencer status" (e.g. highest degree connected node) by comparing the correlation between an influencer-connecting edge, and the edge's inverse resistive distance (conductance). There is a positive correlation, hinting that indeed this is a good (and cheap) metric for subsampling.

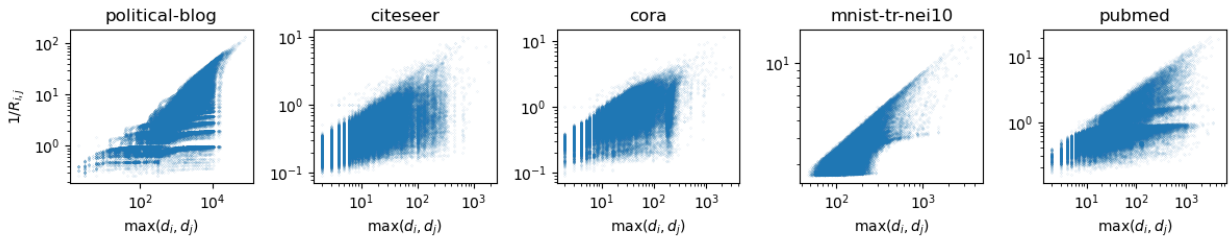


Figure 2: **Is it better to subsample infleuncers?** Correlation of inverse resistive distance (conductance, spectrally optimal) with high degree connecting edge is largely positive, across several graphs.

**Non-spectral properties.** Figure 3 investigates the performance of edge subsampling as it pertains to node labeling, our desired downstream task. This suggests that influencer-based sparsity not only preserves but can sometimes improve the edge ratio despite a reduction in the number of edges (even moreso than resistive subsampling).

### 3.2 Graph sparsification

We next consider a general (offline) graph sparsification scheme, where $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \boldsymbol{A})$ is sparsified into $\mathcal{G}' = (\mathcal{V}, \mathcal{E}', \boldsymbol{A}')$ where $\mathcal{E}' \subset \mathcal{E}$. This is given in Alg. 2. We consider three main forms of sparsification: uniformly removing edges (Karger, 1994), removing edges based on resistive distance (Spielman & Srivastava, 2008), and removing edges based on node degree (influencer). We first give high-concentration bounds influencer-based sparsifications, to show asymptotic consistency.

**Algorithm 2** General offline sparsifier

**Require:** Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{A})$,

1: edge probabilities $p_{u,v}$ for $(u, v) \in \boldsymbol{E}$
2: Initialize $\tilde{\mathcal{E}} = \emptyset$, $\tilde{\boldsymbol{A}} = \mathbf{0}$, $\tilde{\mathcal{G}} = (\mathcal{V}, \tilde{\mathcal{E}}, \tilde{\boldsymbol{A}})$
3: **for** $i = 1, ..., m$ **do**
4: Randomly pick edge $(u, v)$.
5: With probability $p_{u,v}$,
6: $\tilde{\boldsymbol{A}}_{u,v} = \tilde{\boldsymbol{A}}_{v,u} = \frac{1}{p_{u,v}}$, $\tilde{\mathcal{E}} = \tilde{\mathcal{E}} \bigcup \{(u, v)\}$
**return** $\tilde{\mathcal{G}} = (\mathcal{V}, \tilde{\mathcal{E}}, \tilde{\mathbf{A}})$

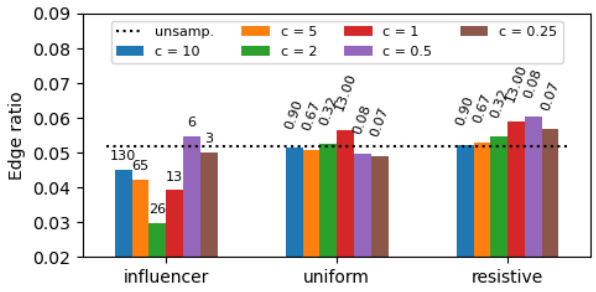


Figure 3: **Edge ratio.** This measures the proportion of edges that connect different-labeled nodes over same-labeled nodes. (Smaller is better.) Sparsifications: U = uniform, R = resistive, I = influencer. The labels show $\bar{q}$ for (I), and the corresponding sparsification rate for (U) and (R).

**Theorem 3.1** (Offline sparsification)**.** *Consider $\boldsymbol{L} = \boldsymbol{I} - \boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}$, $\tilde{\boldsymbol{L}}$ the Laplacian matrices corresponding to a graph and its sparsified version, and $\bar{q}$ the subsampling threshold. Then, for any $\boldsymbol{x} \in \mathbb{R}^n$, the term $\boldsymbol{x}^T\tilde{\boldsymbol{L}}_I\boldsymbol{x}$ is subgaussian with parameter*

$$\textbf{subgauss}(\boldsymbol{x}^T\tilde{\boldsymbol{L}}_I\boldsymbol{x}) = \frac{1}{4}\frac{d_{\max}^2}{\bar{q}^2}\|\boldsymbol{x}\|_\infty^4|\mathcal{S}_I|.$$

*As a consequence,*

$$\mathbf{Pr}(\boldsymbol{x}^T\tilde{\boldsymbol{L}}_I\boldsymbol{x} - \boldsymbol{x}^T\boldsymbol{L}\boldsymbol{x} \geq \epsilon) \leq \exp\left(-\frac{\epsilon^2\bar{q}^2}{d_{\max}^2\|\boldsymbol{x}\|_\infty^4|\mathcal{S}_I|}\right).$$

For a fixed probability, $\bar{q} = O(1/\epsilon)$. In total, the sampling complexity is $O(n/\epsilon)$. In comparison, optimal samplers have complexity $O(1/\epsilon^2)$ (Karger, 1994; Benczúr & Karger, 2015; Spielman & Srivastava, 2008). For $n = O(1/\epsilon)$, the two rates are thus comparable. This lemma is most meaningful when $d_{\max} \gg \bar{q}$. For example, this is the regime if we set $\bar{q}$ to be a small multiple of the median node degree (see Table 1).

## 4 APPR with online sparsification

**Algorithm 3** DUALCORRECT($\mathcal{G}$, $\boldsymbol{x}$, $\bar{q}$)

Identify
$$\mathcal{U} \subseteq \mathbf{supp}(\boldsymbol{x}), \qquad |\mathcal{U}| \leq \bar{q}.$$
$\tilde{\boldsymbol{z}} = \mathbf{0}$
**for** $u \subset \mathcal{U}$ **do**
  Sample neighbors $\mathcal{S} \subset \mathcal{N}(u)$
  Update
$$\Delta \boldsymbol{z} = (x_u \cdot \boldsymbol{A}\boldsymbol{D}^{-1}\boldsymbol{e}_k)_{(\mathcal{S})}$$
  Push operation
$$\begin{aligned} \tilde{z} &\leftarrow \tilde{z} + \frac{(1-\alpha)\sqrt{d_u}\Delta \boldsymbol{z}}{2\alpha} \\ \tilde{z}_u &\leftarrow \tilde{z}_u - \frac{(1+\alpha)\sqrt{d_k}}{2\alpha} \end{aligned}$$
**Return** $\tilde{\boldsymbol{z}}$

**Algorithm 4** PUSHAPPR($\mathcal{G}$, $\boldsymbol{x}$, $\boldsymbol{z}$, $\bar{q}$, $\mathcal{A}$)

Set $\boldsymbol{x}^{(0)} = \boldsymbol{x}$, $\boldsymbol{z}^{(0)} = \boldsymbol{z}$
**for** $u_i \in \mathcal{A} \neq \emptyset$ **do**
  Update $\boldsymbol{x}^{(i+1)} \leftarrow \boldsymbol{x}^{(i)} + \frac{\alpha}{\sqrt{d_{u_i}}} \boldsymbol{z}_{u_i}^{(i)} \boldsymbol{e}_{u_i}$
  Sample neighbors $\mathcal{S} \subset \mathcal{N}(u_i)$, $|\mathcal{S}| \leq \bar{q}$
  Update
$$\Delta \boldsymbol{z}^{(i)} = (\boldsymbol{z}_{u_i}^{(i)} \cdot \boldsymbol{A}\boldsymbol{D}^{-1}\boldsymbol{e}_{u_i})_{(\mathcal{S})}$$
  Push operation
$$\begin{aligned} \boldsymbol{z}^{(i+1)} &\leftarrow \boldsymbol{z}^{(i)} + \tfrac{1-\alpha}{2}\Delta \boldsymbol{z}^{(i)}, \\ \boldsymbol{z}_{u_i}^{(i+1)} &\leftarrow \tfrac{1-\alpha}{2}\boldsymbol{z}_{u_i}^{(i)}, \end{aligned}$$
**Return** $\boldsymbol{x}^{(|\mathcal{S}^{(t)}|)}$, $\boldsymbol{z}^{(|\mathcal{S}^{(t)}|)}$

We now consider the problem of improving Alg. 1 via *online sparsification*; e.g., the APPR method itself is modified to perform subsampling at each step. Here, the notation is defined for unbiased sampling

$$(\boldsymbol{w}_{(\mathcal{S})})_i = \frac{\boldsymbol{w}_i n}{|\mathcal{S}|} \text{ if } i \in \mathcal{S}, \qquad \tilde{\boldsymbol{w}}_i = 0 \text{ otherwise.}$$

The RANDOM-APPR algorithm (Alg 5) aims to compute an approximation of the vector $\mathbf{x}$ for a given graph $\mathcal{G}$, accuracy $\epsilon$, and start node $s$. Specifically, for a query node $u$, we subsample $\bar{q}$ of its neighbors $\mathcal{N}(u)$ whenever $|\mathcal{N}(u)| > \bar{q}$, for some threshold $\bar{q}$ we set. This is to eliminate any extra memory requirements in forming an offline sparsified graph. Additionally, an online implementation allows for more exploration. If the starting node $e_s$ is only connected to influencers, for example, an offline sparsification will likely disconnect it from the graph altogether; this is not necessarily true in online subsampling.

**Algorithm 5** RANDOMAPPR($\mathcal{G}$, $\boldsymbol{s}$, $\epsilon$, $\bar{q}$) with variance reduced debiasing

**Require:** $c < 1$, tol. $\epsilon > 0$, parameter $\alpha$
**Init** $\bar{\boldsymbol{x}} = \boldsymbol{x}^{(0)} = 0$, $\bar{\boldsymbol{z}} = \boldsymbol{z}^{(0)} = \boldsymbol{e}_s$
**for** $t = 1, \ldots$ **do**
  **if** Dual correct **then**
    % Estimate unbiased $\boldsymbol{z}^{(t)} = \boldsymbol{D}^{-1}\boldsymbol{Q}\boldsymbol{x}^{(t)}$
    $\tilde{\boldsymbol{z}} \leftarrow$ DUALCORRECT($\mathcal{G}, \bar{q}, \boldsymbol{x}^{(t)}$)
    $\bar{\boldsymbol{x}} \leftarrow \boldsymbol{x}^{(t)}$, $\boldsymbol{x}^{(t)} \leftarrow \mathbf{0}$
    $\bar{\boldsymbol{z}} \leftarrow \bar{\boldsymbol{z}} - \tilde{\boldsymbol{z}}$, $\boldsymbol{z}^{(t)} \leftarrow \bar{\boldsymbol{z}}$
  Find set $\mathcal{A}^{(t)} = \{k : |z_k^{(t)}| > cd_k\epsilon\}$
  **if** $\mathcal{A}^{(t)} = \emptyset$ **then**
    **break**
  $\boldsymbol{x}^{(t+1)}, \boldsymbol{z}^{(t+1)} =$ PUSHAPPR($\mathcal{G}, \boldsymbol{x}^{(t)}, \boldsymbol{z}^{(t)}, \bar{q}, \mathcal{A}^{(t)}$).
**Return** $\bar{\boldsymbol{x}} + \boldsymbol{x}^{(T)}$

**Resampling the dual variable.** Consider Alg. 5 in which the dual correction step is completely skipped. Then, note that the PUSH steps in Alg. 5 are the same as those in Alg. 1. While in expectation, the solution $\boldsymbol{x}^{(T)}$ will be that of the original, unsampled solution, in each run the dual variable $\boldsymbol{z}^{(t)}$ will drift, causing high variance in $\boldsymbol{x}$. Therefore, the primary purpose of DUALCORRECT is to provide an unbiased estimate of the dual variable $\boldsymbol{z}^{(t)}$ at each step, offering an important dual correction. This is critical to achieve similar bounds despite subsampling.

However, simply resampling the dual variable at each step is computationally expensive, since using a small sampling rate for this specific step can result in mismatch between the primal iterate $\boldsymbol{x}^{(t)}$ and the dual $\boldsymbol{z}^{(t)}$, which causes algorithmic instability. Therefore, we borrow ideas of "iterative refinement" from large-scale convex optimization solvers, and increment the primal and dual variables at each dual correction step. In other words, at each such step, the linear system is shifted from $\boldsymbol{Q}\bar{\boldsymbol{x}} = \boldsymbol{b}$ to $\boldsymbol{Q}\boldsymbol{x}^{(t)} = \boldsymbol{r}$ where $\boldsymbol{r}$ is the current computed residual. Then by accumulating $\bar{\boldsymbol{x}} = \boldsymbol{x}^{(t_1)} + \boldsymbol{x}^{(t_2)} + \cdots$ at each correction step, we return $\bar{\boldsymbol{x}}$ the

sum of the incrementally computed primal variables. In practice, this method is much more stable, and achieves very little extra complexity overhead.

### 4.1 Analysis

All proofs are given in Appendix A, B, and C.

**Theorem 4.1.** *In online APPR, for all $t$, $i \in \mathcal{S}_t$, $\mathbb{E}[\tilde{\boldsymbol{z}}^{(t,i)}] \geq 0$, $\mathbb{E}[\boldsymbol{x}^{(t,i)}] \geq 0$ and*

$$\mathbb{E}[\tilde{\boldsymbol{z}}^{(t,i+1)}] \leq \mathbb{E}[\tilde{\boldsymbol{z}}^{(t,i)}],\ \mathbb{E}[\tilde{\boldsymbol{z}}^{(t+1)}] \leq \mathbb{E}[\tilde{\boldsymbol{z}}^{(t)}], \quad \mathbb{E}[\boldsymbol{x}^{(t,i+1)}] \geq \mathbb{E}[\boldsymbol{x}^{(t,i)}],\ \mathbb{E}[\boldsymbol{x}^{(t+1)}] \geq \mathbb{E}[\bar{\boldsymbol{x}}^{(t)}]. \tag{monotonicity}$$

*Moreover,*

$$\|\mathbb{E}[\boldsymbol{D}^{1/2}\boldsymbol{x}^{(t)}]\|_1 + \|\mathbb{E}[\tilde{\boldsymbol{z}}^{(t)}]\|_1 = \|\boldsymbol{D}^{-1/2}\boldsymbol{b}\|_1 \tag{conservation}$$

*and*

$$\|\mathbb{E}[\boldsymbol{D}^{1/2}\boldsymbol{x}^{(t+1)}]\|_1 - \|\mathbb{E}[\boldsymbol{D}^{1/2}\boldsymbol{x}^{(t)}]\|_1 = \|\mathbb{E}[\tilde{\boldsymbol{z}}^{(t)}]\|_1 + \|\mathbb{E}[\tilde{\boldsymbol{z}}^{(t+1)}]\|_1 \leq |\mathcal{S}_t|\alpha\epsilon. \tag{descent}$$

Next, we give convergence results.

**Assumption 4.1.** *There exists constants $\sigma^{(t)}$ and $\sigma^{(t,i)}$ for $t = 1, ..., T$ and $i = 1, ..., |\mathcal{S}^{(t)}|$ such that*

- *the random variable $\tilde{\boldsymbol{z}}_j^{(t)}|\boldsymbol{x}^{(t)}$ is subgaussian with parameter $(\sigma^{(t)})^2$, for all $j$*
- *the random variable $\tilde{\boldsymbol{z}}_j^{(t,i+1)}|\tilde{\boldsymbol{z}}^{(t,i)}$ is subgaussian with parameter $(\sigma^{(t,i)})^2$, for all $j$*

**Assumption 4.2.** *There exists a constant $R$ upper bounding each residual term*

$$\max\{\|\tilde{\boldsymbol{z}}^{(t)}\|_\infty, \|\tilde{\boldsymbol{z}}^{(t,i)}\|_\infty\} \leq R,$$

*for all $t = 1, ..., T,\ i = 1, ..., |\mathcal{S}_t|$.*

Note that from monotonicity, each PUSH action cannot increase $\|\tilde{\boldsymbol{z}}^{(t,i)}\|_\infty$. However, the unbiased estimation of $\|\tilde{\boldsymbol{z}}^{(t)}\|_\infty$ is not theoretically bounded; in practice, we observe $\|\boldsymbol{r}^{(t)}\|_\infty$ to be small (usually $< 1$), so this assumption is quite reasonable. Under these assumptions, we have the following conclusions.

**Theorem 4.2** (Chance of stopping too early.)**.** *Consider the online version of the algorithm. The probability that for some $i$, $\boldsymbol{D}^{-1}\boldsymbol{z}_i^{(t)} > \epsilon$ but $\boldsymbol{D}^{-1}\tilde{\boldsymbol{z}}_i^{(t)} < c\epsilon$ is bounded by*

$$\mathbf{Pr}(|\boldsymbol{D}^{-1}\boldsymbol{z}_i^{(t)}| > \epsilon,\ |\boldsymbol{D}^{-1}\tilde{\boldsymbol{z}}_i^{(t)}| < c\epsilon) \leq \exp\left(-\frac{(1-c)^2\epsilon^2}{2\sigma_t^2}\right).$$

That is to say, the chance of stopping too early is diminishingly small; even more so if we reduce the user parameter $0 < c < 1$.

**Theorem 4.3** ($1/T$ rate)**.** *Consider $f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T\boldsymbol{Q}\boldsymbol{x} - \boldsymbol{b}^T\boldsymbol{x}$ where $\boldsymbol{b} = \frac{2\alpha}{1+\alpha}\boldsymbol{D}^{-1/2}\boldsymbol{e}_s$. Initialize $\boldsymbol{x}^{(0)} = 0$. Define $M^{(t)} = \sum_{\tau=1}^t |\mathcal{S}_\tau|$ the number of push calls at epoch $t$. Then,*

$$\min_{t,j} \|\nabla f(\boldsymbol{x}^{(t,j)})\|_2^2 \leq \frac{1}{\alpha M^{(t)}} + \frac{\alpha\sigma_{\max}^2}{2}$$

*where $\sigma_{\max} = \max_{i,t}\{\sigma^{(i,t)}, \sigma^{(t)}\}$. Here, $\alpha$ can be chosen to mitigate the tradeoff between convergence rate and final noise level.*

**Theorem 4.4** (Linear rate in expectation)**.** *Using $\tilde{\boldsymbol{z}}^{(t)} = \frac{1+\alpha}{2\alpha}\boldsymbol{D}^{1/2}(\boldsymbol{b} - \boldsymbol{Q}\boldsymbol{x}^{(t)})$, in expectation,*

$$\|\mathbb{E}[\tilde{\boldsymbol{z}}^{(t+1)}]\|_1 \leq \exp\left(-\frac{M_t\alpha\epsilon}{R}\right).$$

*Moreover, since $\|\boldsymbol{z}^{(t)}\|_1 \leq \sqrt{n}\|\boldsymbol{z}^{(t)}\|_2$,*

$$\mathbf{Pr}\left(\|\boldsymbol{z}^{(t)}\|_1 \geq \exp\left(-\frac{M_t\alpha\epsilon}{R}\right)^{M_t} + \epsilon\right) \leq \exp\left(-\frac{\epsilon^2}{2\alpha^2\sqrt{n}\left(\sum_{\tau=1}^t\sum_{i\in\mathcal{S}_t}\sigma^{(t,i)} + \sum_{\tau=1}^t\sigma^{(t)}\right)}\right).$$

These two Theorems (4.4 and 4.3) offer two points of view of the convergence behavior: linear in expectation, and at least $O(1/T)$ deterministically.

**Algorithm 6** RELAXATION($\boldsymbol{L}, \gamma$)(Rakhlin & Sridharan (2017))

$n =$ size of $\boldsymbol{L}$ (num of rows or columns)
Compute $\boldsymbol{M} = \left(\frac{\boldsymbol{L}}{2\gamma} + \frac{\boldsymbol{I}_n}{2n}\right)^{-1}$
$\tau_1 = \mathbf{tr}(\boldsymbol{M}), a_1 = 0, \boldsymbol{G} = \mathbf{0} \in \mathbb{R}^{K\times n}$
**for** $t = 1, \ldots, n$ **do**
  $z_t = -2\boldsymbol{G}\boldsymbol{M}_{:,t}/\sqrt{a_t + D^2 \cdot \tau_t}$
  Predict $\hat{y}_t \sim \omega_t(\boldsymbol{z}_t)$, $\nabla_t = \nabla\phi_{\boldsymbol{z}_t}(\cdot, y_t)$
  Update $\boldsymbol{G}_{:,t} = \nabla_t$
  $a_{t+1} = a_t + 2\nabla_t^\top \boldsymbol{G}\boldsymbol{M}_{:,t} + \boldsymbol{M}_{tt} \cdot \|\nabla_t\|_2^2$
  $\tau_{t+1} = \tau_t - \boldsymbol{M}_{tt}$
**return** $\boldsymbol{z}_t$, $t = 1, ..., n$

**Algorithm 7** REGULARIZE($\mathbf{L}, \gamma$)(Belkin et al. (2004))

$n =$ size of $\boldsymbol{L}$ (num of rows or columns)
Compute $\boldsymbol{M} = \left(\frac{\mathbf{L}}{2\gamma} + \frac{\boldsymbol{I}_n}{2n}\right)^{-1}$
**for** $t = 1, \ldots, n$ **do**
  $z_t = -2\boldsymbol{G}\boldsymbol{M}_{:,t}$
  Predict $\hat{y}_t \sim \omega_t(\boldsymbol{z}_t)$
  Update $\boldsymbol{G}_{:,t} = y_t$
**return** $\boldsymbol{z}_t$, $t = 1, ..., n$

## 5 Applications

### 5.1 Online node labeling (ONL)

In this framework (Belkin et al., 2004; Herbster et al., 2005; Zhou et al., 2023), node labels are revealed after visiting. Formally, we traverse through the nodes in some order $t = 1, ..., n$, we infer the label $\hat{y}_t \in \{1, -1\}$, and incur some loss $\ell(y_t, \hat{y}_t)$. Then the true label $y_t$ is revealed (e.g. the customer bought the item or left the page), and we predict $\hat{y}_{t+1}$ using the now seen labels $y_1, ..., y_t$.

We integrate RANDOMAPPR with two well-studied methods for ONL:

- REGULARIZE (Alg. 6), where (PPR-symm) is solved using $\mathbf{y}_t = [y_1, ..., y_{t-1}, 0, ..., 0]^T \in \mathbb{R}^{n\times K}$.
- RELAXATION(Alg. 7), which follows the process outlined in Rakhlin & Sridharan (2017).

In both cases, the methods also depend on the graph smoothness parameter $\gamma_{\text{sm}} = \boldsymbol{y}^T\boldsymbol{L}\boldsymbol{y}$. [1] Note that $\gamma$ is then integrated into the method, such that it matches $\beta = \frac{n}{\gamma}$ as in our discount factor in Section 2.

We quantify the success of a set of predictions $\hat{\boldsymbol{y}} = [\hat{y}_1, ..., \hat{y}_n]^T \in \mathbb{R}^{n\times K}$ in terms of the solution's suboptimality, compared against the best solution that is $\gamma$-smooth, e.g.

$$\text{regret}(t) = \ell_t(\boldsymbol{y}, \hat{\mathbf{y}}) - \inf_{\boldsymbol{y}' \in \mathcal{F}_\gamma} \ell_t(\boldsymbol{y}, \boldsymbol{y}'), \quad \text{where} \quad \ell_t(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \sum_{i=1}^{t} \mathbf{1}_{y_i \neq \hat{y}_i},\ \mathcal{F}_\gamma = \{\boldsymbol{y} : \mathbf{tr}(\boldsymbol{y}^T\boldsymbol{L}\boldsymbol{y}) \leq \gamma_{\text{sm}}\}.$$

In both methods, $\omega_t(z_t)$ transforms a positive vector into a probability vector through *waterfilling*, e.g. $\hat{y}_t = \max\{0, z_t - \tau\}$ such that $\hat{y}_t$ sums to 1. The function $\phi$ is a specially designed convex loss function, introduced in Rakhlin & Sridharan (2017). Overall, the RELAXATION method, originally introduced in Rakhlin & Sridharan (2017), gives a regret bound, which is then tightened and shown to be sublinear in Zhou et al. (2023):

**Theorem 5.1** (Rakhlin & Sridharan (2017); Zhou et al. (2023))**.** *For some (graph-independent) constant $D = O(K)$, and $\rho = \frac{\log(\gamma)}{\log(n)}$, Alg. 6 has the regret bound where $\gamma_{\text{sm}} = \boldsymbol{y}^T\boldsymbol{L}\boldsymbol{y}$:*

$$\text{regret}(n) \overset{\textit{Rakhlin \& Sridharan (2017)}}{\leq} \sqrt{\mathbf{tr}\Big(\Big(\frac{\boldsymbol{L}}{2\gamma_{\text{sm}}} + \frac{\boldsymbol{I}}{2n}\Big)^{-1}\Big)} \overset{\textit{Zhou et al. (2023)}}{\leq} D\sqrt{2n^{1+\rho}}.$$

[1] In practice, $\gamma_{\text{sm}}$ is not a parameter known ahead of time. However, an incorrect guess of $\gamma_{\text{sm}}$ usually does not affect the practical performance significantly, and a correct guess allows us to align the numerical performance more closely with the available regret bounds.

*Furthermore, integrating APPR into* RELAXATION *adds overhead to result in*

$$\text{regret}(n) \leq D\sqrt{(1+K^2)n^{1+\rho}}.$$

*where $K$ is the number of classes.*

Specifically, in Rakhlin & Sridharan (2017), the assumption is that the matrix inversion steps are done explicitly, using full matrix linear system solvers. To tighten the analysis under practical methods, Zhou et al. (2023) computes a new regret bound where the matrix inversion steps are replaced with vanilla APPR. This theorem suggests that small perturbations in the numerics of the learning method will not greatly deteriorate the overall regret bound.

Now we consider the vanilla APPR method learned over a graph perturbation resulting in weighted Laplacian $\tilde{\boldsymbol{L}}$ has label smoothness $\tilde{\gamma}_{\text{sm}} = \mathbf{tr}(\boldsymbol{y}^T \tilde{\boldsymbol{L}} \boldsymbol{y})$.

**Theorem 5.2.** *RELAXATION over $\tilde{\boldsymbol{L}}$ using $\sigma'_{\text{sm}} = \mathbf{tr}(\boldsymbol{y}^T \tilde{\boldsymbol{L}} \boldsymbol{y})$ achieves*

$$\text{regret}(n) \leq D\sqrt{2n^{1+\rho}} + \sqrt{\frac{\epsilon n}{1-\beta}}.$$

The proof is in Appendix D. The additional error maintains a sublinear regret.

### 5.2 Unsupervised clustering

Clustering is crucial for community detection, social network analysis, and other applications where the inherent structure of the data must be discovered. In this context, we use the node embeddings as the rows to the matrix inverse $\boldsymbol{Z} = (\boldsymbol{L} + \beta \boldsymbol{I})^{-1}$ via RANDOMAPPR. Then, clustering is done by first identifying the highest degree nodes as seeds $\mathcal{C} \subset \mathcal{V}$, and then assigning the cluster based on largest value in $\boldsymbol{Z}$:

$$\mathbf{cluster}(u) = \text{PICKONE}(\arg\max_{j \in \mathcal{C}} \{\boldsymbol{Z}_{t,j}\})$$

The evaluation score is the normalized sum purity of the ground truth labels, over each cluster

$$\mathbf{score} = \frac{\sum_{j \in \mathcal{C}} \mathbf{purity}(\mathcal{S}_j)}{n}, \qquad \mathcal{S}_j = \{u : \mathbf{cluster}(u) = j\}.$$

This method is reminiscent of latent semantic analysis in document retrieval; for example, by using a co-occurance matrix as a proxy for similarity (Deerwester et al., 1990).

## 6 Numerical experiments

We evaluate the various methods across several tasks. The total number of nodes visited is the main complexity measure. In each case, we explore the tradeoff between task performance, and complexity.

**Baseline.** Figure 4 compares the performance of APPR (without any subsampling) to that of simple message passing (generalized WMA, denoted WMA*, where we use (1) for finite $K \geq 1$). Generally, while WMA is more computationally efficient, APPR consistently delivers superior performance across large graphs.

**Offline sparsification.** Figure 5 demonstrates the performance of APPR applied to an offline sparsified graph. The first row gives the residual over the original linear system (smaller is better.) The second gives the residual over the new, biased linear system. The bias in each run is evident; the residual of the original system obtains a noise floor when subsampling is used; this is lowered first by online sparsification, and then by dual correction. Note that when measuring performance over linear systems, there is not much benefit to subsampling; the main advantage comes in learning tasks.

**ONL performance.** Figure 6 gives the performance of offline and online (with and without dual correction) methods for online node labeling. Note that for small graphs, the baseline (WMA) is very strong; however, for larger graphs, the tradeoff for reduced misclasification rate becomes more apparent. It is also clear that the RELAXATION method, although providing strong rates, is not as strong in practice as the REGULARIZATION method. Nonetheless, both methods are closely related, and their respective advantages and disadvantages stem from subtle adjustments in procedures and hyperparameters.

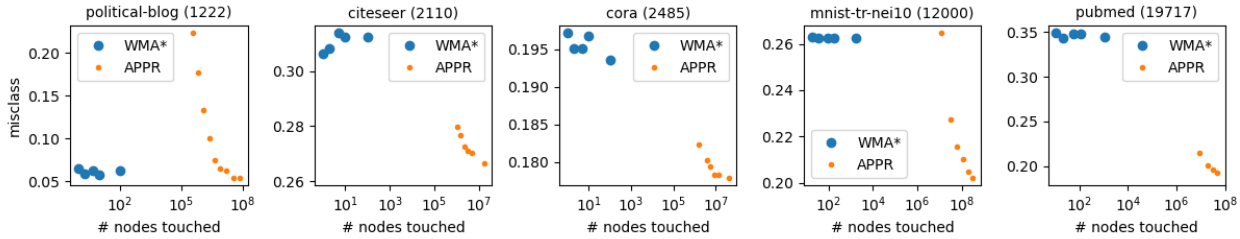


Figure 4: **Baseline comparisons.** Standard APPR versus extended WMA on ONL (WMA*). The x-axis represents memory complexity (total number of nodes queried) and the y-axis shows performance in terms of misclassification rate. While WMA operates faster, APPR is superior at higher performance levels.

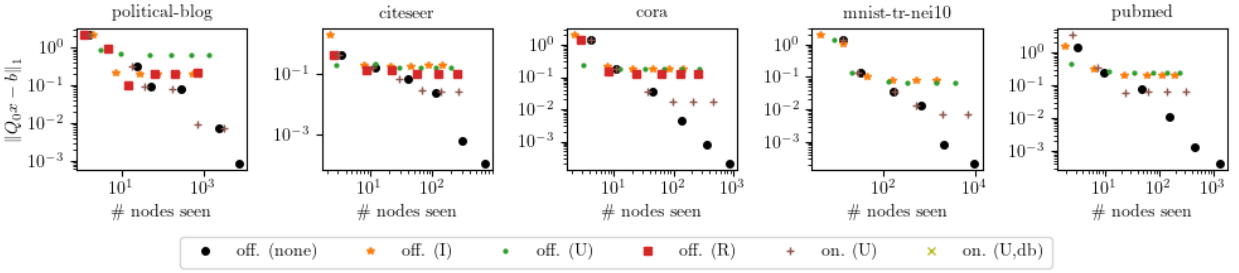


Figure 5: **APPR.** The residual of the original linear system is $\|\boldsymbol{Q}_0 x - \boldsymbol{b}\|_2$. off = offline sparsification, on = online sparsification. O = original (unsparsified). For offline, U = uniform, R = resistive, I = influencer. All online sparsifications are influencer, and further subsampled via U = uniform, W = edge weighted, D = degree weighted. c = with dual correction.

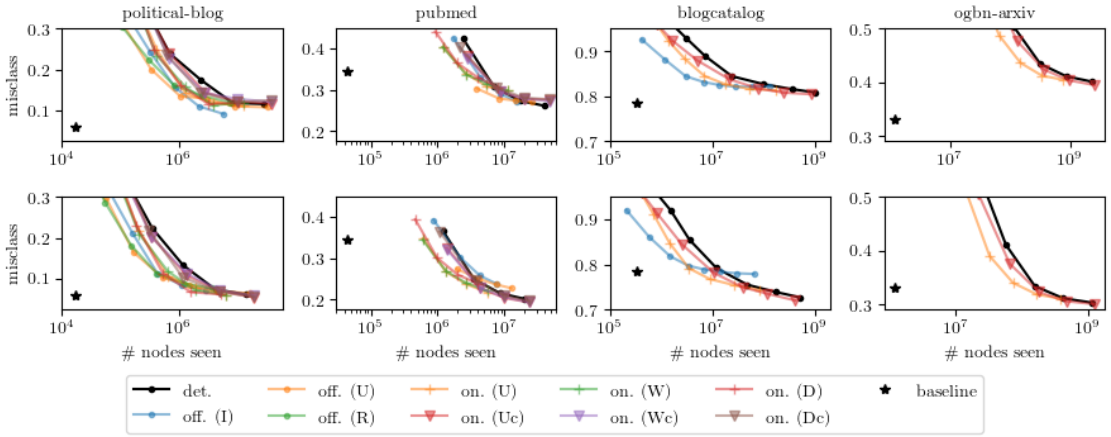


Figure 6: **ONL performance.** Top: relaxation, bottom: regularization. det = deterministic APPR, off = offline, on = online. U = uniform, W = weighted by edge, D = weighted by degree, R = weighted by resistive distance. Baseline = WMA (1-hop). All experiments were run for the same set of $\epsilon$ values. Offline graph sparsification was too memory-intensive for large graphs. For online, we only subsample influencer nodes. c = with dual correction. Lower is better.

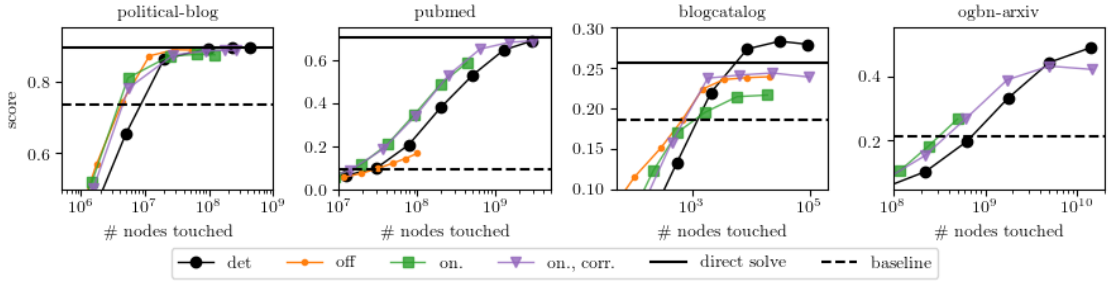


Figure 7: **Clustering performance.** det = deterministic APPR, off = offline, on = online. U = uniform. Baseline = WMA (1-hop). All experiments were run for the same set of $\epsilon$ values. Offline graph sparsification and direct solve were too memory-intensive for large graphs. For online, we only subsample influencer nodes. c = with dual correction. direct solve is shown as an upper bound, when it is computable. Higher is better.

**Unsupervised clustering.** Finally, in Figure 7 we evaluate the performance of RandomAPPR in producing node embeddings, which are then used in clustering, and evaluated based on their ground truth labels. Only influencer-based uniform sampling is used. An ablative study is in Appendix E.

## 7 Discussion

Our results demonstrate a tradeoff between performance and memory utilization when random sparsification is used. For very large graphs, some form of memory alleviation is mandatory, and thus such methods must be considered even if performance degrades. While offline sparsification produces significant noise in solving linear systems, it is still robust in the downstream online prediction task. Moreover, online sparsification produces good results in both linear solving and learning.

**Relationship to other locality sampling methods.** The idea of subsampling edges to reduce memory complexity in graph applications is not new; in particular, in Shin et al. (2024); Huang & Zitnik (2020); Wu et al. (2023), it is used to alleviate message passing for GNNs. However, in many of those methods, the sampling strategies focus on a hop-radius, and alleviate small-world effects via aggressive subsampling, weighted by edge weights or outgoing degrees. We argue that using an APPR-inspired approach offers a more powerful tradeoff between performance and sparsity, as it inherently does not restrict the hop-radius in fusing neighboring nodes in prediction, but rather weights them according to their residual.

**Extension to general quadratic minimization.** It is interesting to ask if this scheme can be used to minimize more general classes of quadratic problems. For example, in kernel SVM, the dual problem mimics a quadratic problem, and the kernel matrix is sparse, where nonzeros indicate training samples that are similar. By re-assigning the diagonal values, the kernel matrix becomes a (dense and signed) graph Laplacian – however, there are very few values that are very large (indicating similar training samples).

However, the question of whether this property holds *regardless* of diagonal rescaling is an open one. We find in practice that monotonicity does not hold, but the property that $\mathbf{supp}(\mathbf{x}^{(t)}) \subset \mathbf{supp}(\mathbf{x}^*)$ often does, in practice. Showing this theoretically would be interesting and powerful, but does not seem straightforward.

## References


Reid Andersen, Fan Chung, and Kevin Lang. Local graph partitioning using PageRank vectors. In *47th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2006.

Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural computation*, 15(6):1373–1396, 2003.

Mikhail Belkin, Irina Matveeva, and Partha Niyogi. Regularization and semi-supervised learning on large graphs. In *Learning Theory: 17th Annual Conference on Learning Theory, COLT 2004, Banff, Canada, July 1-4, 2004. Proceedings 17*, pp. 624–638. Springer, 2004.

András A Benczúr and David R Karger. Randomized approximation schemes for cuts and flows in capacitated graphs. *SIAM Journal on Computing*, 44(2):290–319, 2015.

Aleksandar Bojchevski, Johannes Klicpera, Bryan Perozzi, Amol Kapoor, Martin Blais, Benedek Rózemberczki, Michal Lukasik, and Stephan Günnemann. Scaling graph neural networks with approximate pagerank. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2464–2473, 2020.

Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391–407, 1990.

Kimon Fountoulakis, Farbod Roosta-Khorasani, Julian Shun, Xiang Cheng, and Michael W Mahoney. Variational perspective on local graph clustering. *Mathematical Programming*, 174(1):553–573, 2019.

Alberto Garcia Duran and Mathias Niepert. Learning graph representations with embedding propagation. *Advances in neural information processing systems*, 30, 2017.

Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 855–864, 2016.

Xingzhi Guo, Silong Wang, Baojian Zhou, Yanghua Xiao, and Steven Skiena. Fast and robust contextual node representation learning over dynamic graphs. *arXiv preprint arXiv:2411.07123*, 2024.

Mark Herbster and Massimiliano Pontil. Prediction on a graph with a perceptron. *Advances in neural information processing systems*, 19, 2006.

Mark Herbster, Massimiliano Pontil, and Lisa Wainer. Online learning over graphs. In *Proceedings of the 22nd international conference on Machine learning*, pp. 305–312, 2005.

Kexin Huang and Marinka Zitnik. Graph meta learning via local subgraphs. *Advances in neural information processing systems*, 33:5862–5874, 2020.

David R Karger. Random sampling in cut, flow, and network design problems. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pp. 648–657, 1994.

Pan Li, I Chien, and Olgica Milenkovic. Optimizing generalized pagerank methods for seed-expansion community detection. *Advances in Neural Information Processing Systems*, 32, 2019.

Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, November 1999.

Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 701–710, 2014.

Alexander Rakhlin and Karthik Sridharan. Efficient online multiclass prediction on graphs via surrogate losses. In *Artificial Intelligence and Statistics*, pp. 1403–1411. PMLR, 2017.

Sasha Rakhlin, Ohad Shamir, and Karthik Sridharan. Relax and randomize: From value to algorithms. *Advances in Neural Information Processing Systems*, 25, 2012.

Yousef Saad. On the rates of convergence of the lanczos and the block-lanczos methods. *SIAM Journal on Numerical Analysis*, 17(5):687–706, 1980.

Shota Saito and Mark Herbster. Multi-class graph clustering via approximated effective $p$-resistance. In *International Conference on Machine Learning*, pp. 29697–29733. PMLR, 2023.

Seiyun Shin, Ilan Shomorony, and Han Zhao. Efficient learning of linear graph neural networks via node subsampling. *Advances in Neural Information Processing Systems*, 36, 2024.

Daniel A Spielman. Algorithms, graph theory, and linear equations in laplacian matrices. In *Proceedings of the International Congress of Mathematicians 2010 (ICM 2010) (In 4 Volumes) Vol. I: Plenary Lectures and Ceremonies Vols. II–IV: Invited Lectures*, pp. 2698–2722. World Scientific, 2010.

Daniel A Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pp. 563–568, 2008.

Daniel A Spielman and Shang-Hua Teng. Nearly linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems. *SIAM Journal on Matrix Analysis and Applications*, 35 (3):835–885, 2014.

Martin J Wainwright. *High-dimensional statistics: A non-asymptotic viewpoint*, volume 48. Cambridge university press, 2019.

Kilian Q Weinberger, Fei Sha, and Lawrence K Saul. Learning a kernel matrix for nonlinear dimensionality reduction. In *Proceedings of the twenty-first international conference on Machine learning*, pp. 106, 2004.

Wenchao Wu, Xuanhua Shi, Ligang He, and Hai Jin. TurboGNN: Improving the end-to-end performance for sampling-based gnn training on gpus. *IEEE Transactions on Computers*, 2023.

Baojian Zhou, Yifan Sun, and Reza Babanezhad Harikandeh. Fast online node labeling for very large graphs. In *International Conference on Machine Learning*, pp. 42658–42697. PMLR, 2023.

Xiaojin Zhu, Zoubin Ghahramani, and John D Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. In *Proceedings of the 20th International conference on Machine learning (ICML-03)*, pp. 912–919, 2003.