Adaptive Diffusions for Scalable Learning over Graphs

Dimitris Berberidis, Athanasios N. Nikolakopoulos and Georgios B. Giannakis
Dept. of Electrical & Computer Engineering, Digital Technology Center, University of Minnesota
Minneapolis, MN, USA
{berb,anikolak,georgios}@umn.edu

ABSTRACT

Diffusion-based classifiers such as those relying on the Personalized PageRank and the Heat kernel, enjoy remarkable classification accuracy at modest computational requirements. Their performance however is affected by the extent to which the chosen diffusion captures a typically unknown label propagation mechanism, that can be specific to the underlying graph, and potentially different for each class. The present work introduces a disciplined, data-efficient approach to learning *class-specific diffusion functions adapted to the underlying network* topology. The novel learning approach leverages the notion of "landing probabilities" of class-specific random walks, which can be computed efficiently, thereby ensuring scalability to large graphs. This is supported by rigorous analysis of the properties of the model as well as the proposed algorithms. Classification tests on real networks demonstrate that adapting the diffusion function to the given graph and observed labels, significantly improves the performance over fixed diffusions; reaching—and many times surpassing—the classification accuracy of computationally heavier state-of-the-art competing methods, that rely on node embeddings and deep neural networks.

CCS CONCEPTS

• **Computing methodologies** → **Semi-supervised learning settings**; • **Mathematics of computing** → *Markov processes*;

KEYWORDS

Semi-supervised Classification, Random Walks, Diffusions

ACM Reference Format:
Dimitris Berberidis, Athanasios N. Nikolakopoulos and Georgios B. Giannakis. 2018. Adaptive Diffusions for Scalable Learning over Graphs. In *Proceedings of 2018 ACM SIGKDD conference (KDD18 (MLG Workshop))*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The task of classifying nodes of a graph arises frequently in several applications on real-world networks, such as the ones derived from social interactions and biological dependencies. Graph-based semi-supervised learning (SSL) methods tackle this task building on the premise that the true labels are distributed "smoothly" with respect to the underlying network, which then motivates leveraging the network structure to increase the classification accuracy [11]. Graph-based SSL has been pursued by various intertwined methods, including iterative label propagation [6, 24, 41], kernels on graphs [29], manifold regularization [5], graph partitioning [19, 38], transductive learning [37], competitive infection models [34], and bootstrapped label propagation [10]. SSL based on graph filters was discussed in [35], and further developed in [12] for bridge monitoring. Recently, approaches based on node-embeddings [18, 32, 40], as well as deep-learning architectures [2, 20] have gained popularity, and were reported to have state-of-the-art performance.

Many of the aforementioned methods are challenged by computational complexity and scalability issues that limit their applicability to large-scale networks. Random-walk-based diffusions present an efficient and effective alternative. Methods of this family diffuse probabilistically the known labels through the graph, thereby ranking nodes according to weighted sums of variable-length landing probabilities. Celebrated representatives include those based on the Personalized PageRank and the Heat Kernel that were found to perform remarkably well in certain application domains [21], and have been nicely linked to particular network models [3, 22, 23]. However, the effectiveness of diffusion-based classifiers can vary considerably depending on whether the chosen diffusion conforms with the latent label propagation mechanism that might be, (i) particular to the target application or underlying network topology; and, (ii) different for each class.

The present contribution alleviates these shortcomings and markedly improves the performance of random-walk-based classifiers by *adapting the diffusion functions* to both the network and the observed labels. The resulting novel classifier relies on the notion of landing probabilities of *short-length random walks* rooted at the observed nodes of each class. The small number of these landing probabilities can be extracted efficiently with a small number of sparse matrix-vector products, thus ensuring applicability to large-scale networks. Theoretical analysis establishes that short random walks are in most cases sufficient for reliable classification. We test our methods in terms of multiclass and multilabel classification accuracy, and confirm that it can achieve results competitive to state-of-the-art methods, while also being considerably faster.

The rest of the paper is organized as follows. Section 2 introduces random-walk based diffusions. Our novel approach along with relevant analytical results are the subjects of Section 3. Section 4 places our work in the context of related methods. Finally, Section 5 presents experiments, while Section 6 concludes the paper and discusses future directions.

Notation. Bold lower-case letters denote column vectors (e.g., $\mathbf{v}$); bold upper-case letters denote matrices (e.g., $\mathbf{Q}$). Vectors $\mathbf{q}_j$ and $\mathbf{q}_i^\mathsf{T}$ denote the j^{th} column and the i^{th} row of $\mathbf{Q}$, respectively; whereas Q_{ij} (or sometimes for clarity $[\mathbf{Q}]_{ij}$) denotes the ij^{th} entry of $\mathbf{Q}$. Vector $\mathbf{e}_K$ denotes the K^{th} canonical column vector; and $\|\cdot\|$ denotes the Euclidean norm, unless stated otherwise. Calligraphic upper-case letters denote sets (e.g., $\mathcal{U}, \mathcal{V}$); and finally, symbol := is used in definition statements.

KDD18 (MLG Workshop), August 2018, London, UK
2018. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 PROBLEM STATEMENT AND MODELING

Consider a graph $\mathcal{G} := \{\mathcal{V}, \mathcal{E}\}$, where $\mathcal{V}$ is the set of N nodes, and $\mathcal{E}$ the set of edges. Connectivity is captured by the weight matrix $\mathbf{W}$ having entries $W_{ij} > 0$ if $(i,j) \in \mathcal{E}$. Associated with each node $i \in \mathcal{V}$ there is a discrete label $y_i \in \mathcal{Y}$. In SSL classification over graphs, a subset $\mathcal{L} \subset \mathcal{V}$ of nodes has available labels $\mathbf{y}_{\mathcal{L}}$, and the goal is to infer the labels of the unlabeled set $\mathcal{U} := \mathcal{V} \setminus \mathcal{L}$. Given a measure of influence, a node most influenced by labeled nodes of a certain class is deemed to also belong to the same class. Thus, label-propagation on graphs boils down to quantifying the influence of $\mathcal{L}$ on $\mathcal{U}$, see, e.g. [11, 24, 39]. An intuitive yet simple measure of node-to-node influence relies on the notion of random walks on graphs.

A simple random walk on a graph is a discrete-time Markov chain with state space the set of nodes, and transition probabilities

$$[\mathbf{H}]_{ij} := \Pr\{X_k = i | X_{k-1} = j\} = W_{ij}/d_j = [\mathbf{W}\mathbf{D}^{-1}]_{ij}$$

where $X_k \in \mathcal{V}$ denotes the position of the random walker (state) at the k^{th} step; $d_j := \sum_{k \in \mathcal{N}_j} W_{kj}$ is the degree of node j; and, $\mathcal{N}_j$ its neighborhood. Since we consider undirected graphs the steady-state distribution of $\{X_k\}$ always exists if it is connected, and non-bipartite. It is given by the dominant right eigenvector of the column-stochastic transition probability matrix $\mathbf{H} := \mathbf{W}\mathbf{D}^{-1}$, where $\mathbf{D} := \text{diag}\,(d_1, d_2, \ldots, d_N)$ [26]. The steady-state distribution $\boldsymbol{\pi}$ can be shown to have entries

$$\pi_i := \lim_{k\to\infty} \sum_{j \in \mathcal{V}} \Pr\{X_k = i | X_0 = j\} \Pr\{X_0 = j\} = \frac{d_i}{2|\mathcal{E}|}$$

that are clearly not dependent on the initial "seeding" distribution $\Pr\{X_0\}$; and $\boldsymbol{\pi}$ is thus unsuitable for measuring influence among nodes. Instead, for graph-based SSL, we will utilize the $k-$step *landing probability* per node i given by

$$p_i^{(k)} := \sum_{j \in \mathcal{V}} \Pr\{X_k = i | X_0 = j\} \Pr\{X_0 = j\} \tag{1}$$

that in vector form $\mathbf{p}^{(k)} := [p_1^{(k)} \ \ldots \ p_N^{(k)}]^\mathsf{T}$ satisfies $\mathbf{p}^{(k)} = \mathbf{H}^k \mathbf{p}^{(0)}$, where $p_i^{(0)} := \Pr\{X_0 = i\}$. In words, $p_i^{(k)}$ is the probability that a random walker with initial distribution $\mathbf{p}^{(0)}$ is located at node i after k steps. Therefore, $p_i^{(k)}$ is a valid measure of the influence that $\mathbf{p}^{(0)}$ has on any node in $\mathcal{V}$.

The landing probabilities per class $c \in \mathcal{Y}$ are (cf. (1))

$$\mathbf{p}_c^{(k)} = \mathbf{H}^k \mathbf{v}_c \tag{2}$$

where for $\mathcal{L}_c := \{i \in \mathcal{L} : y_i = c\}$, we select as $\mathbf{v}_c$ the normalized class-indicator vector with $i-$th entry

$$[\mathbf{v}_c]_i = \begin{cases} 1/|\mathcal{L}_c|, & i \in \mathcal{L}_c \\ 0, & \text{else} \end{cases} \tag{3}$$

acts as initial distribution. Using (2), we model diffusions per class c over the graph driven by $\{\mathbf{p}_c^{(k)}\}_{k=1}^K$ as

$$\mathbf{f}_c(\boldsymbol{\theta}) = \sum_{k=1}^K \theta_k \mathbf{p}_c^{(k)} = \mathbf{P}_c^{(K)} \boldsymbol{\theta} \tag{4}$$

where $\mathbf{P}_c^{(K)} := \begin{bmatrix} \mathbf{p}_c^{(1)} & \cdots & \mathbf{p}_c^{(K)} \end{bmatrix}$, and θ_k denotes the importance assigned to the k^{th} hop neighborhood. By constraining $\boldsymbol{\theta} \in \mathcal{S}^K$, where $\mathcal{S}^K := \{\mathbf{x} \in \mathbb{R}^K : \ \mathbf{x} \geq \mathbf{0}, \ \mathbf{1}^\mathsf{T}\mathbf{x} = 1\}$ is the $K-$dimensional probability simplex, $\mathbf{f}_c(\boldsymbol{\theta})$ becomes a valid nodal probability mass function (pmf) for class c.

Given $\boldsymbol{\theta}$ and upon obtaining $\{\mathbf{f}_c(\boldsymbol{\theta})\}_{c \in \mathcal{Y}}$, our diffusion-based classifiers will predict labels over $\mathcal{U}$ as

$$\hat{y}_i(\boldsymbol{\theta}) := \arg\max_{c \in \mathcal{Y}} [\mathbf{f}_c(\boldsymbol{\theta})]_i \tag{5}$$

where $[\mathbf{f}_c(\boldsymbol{\theta})]_i$ is the i^{th} entry of $\mathbf{f}_c(\boldsymbol{\theta})$.

The upshot of (4) is a *unifying form* of superimposed diffusions allowing even tunable simplex weights, taking up to K steps per class to come up with an influence metric for the semi-supervised classifier (5) over graphs. Next, we outline two notable members of the family of diffusion-based classifiers that can be viewed as special cases of (4).

2.1 Personalized PageRank Classifier

Inspired by its celebrated network centrality metric [9], the Personalized PageRank (PPR) algorithm has well-documented merits for label propagation; see, e.g. [27]. PPR is a special case of (4) corresponding to $\boldsymbol{\theta}_{\text{PPR}} = (1-\alpha)\left[\alpha, \alpha^2, \ldots, \alpha^K\right]^\mathsf{T}$, where $0 < \alpha < 1$, and $1-\alpha$ can be interpreted as the "restart" probability of random walks with restarts.

The PPR-based classifier relies on (cf. (4))

$$\mathbf{f}_c(\boldsymbol{\theta}_{\text{PPR}}) = (1-\alpha) \sum_{k=0}^K \alpha^k \mathbf{p}_c^{(k)} \tag{6}$$

satisfying asymptotically in the number of random walk steps

$$\lim_{K\to\infty} \mathbf{f}_c(\boldsymbol{\theta}_{\text{PPR}}) = (1-\alpha)(\mathbf{I} - \alpha\mathbf{H})^{-1}\mathbf{v}_c$$

which implies that $\mathbf{f}_c(\boldsymbol{\theta}_{\text{PPR}})$ approximates the solution of a linear system. Indeed, as shown in [3], PPR amounts to solving a weighted regularized least-squares problem over $\mathcal{V}$; see also [22] for a PPR interpretation as an approximate geometric discriminant function defined in the space of landing probabilities.

2.2 Heat Kernel Classifier

The heat kernel (HK) is another popular diffusion that has recently been employed for SSL [29] and community detection on graphs [21]. HK is also a special case of (4) with $\boldsymbol{\theta}_{\text{HK}} = e^{-t}\left[t, \frac{t^2}{2}, \ldots, \frac{t^K}{K!}\right]^\mathsf{T}$, yielding class distributions (cf. (4))

$$\mathbf{f}_c(\boldsymbol{\theta}_{\text{HK}}) = e^{-t} \sum_{k=0}^K \frac{t^k}{k!} \mathbf{p}_c^{(k)}. \tag{7}$$

Furthermore, it can be readily shown that

$$\lim_{K\to\infty} \mathbf{f}_c(\boldsymbol{\theta}_{\text{HK}}) = e^{-t(\mathbf{I}-\mathbf{H})}\mathbf{v}_c$$

allowing HK to be interpreted as an approximation of a heat diffusion process, where heat is flowing from $\mathcal{L}_c$ to the rest of the graph; and $\mathbf{f}_c(\boldsymbol{\theta}_{\text{HK}})$ is a snapshot of the temperature after time t has elapsed. HK provably yields low conductance communities, while also converging faster to its asymptotic closed-form expression than PPR [15].

3 ADAPTIVE DIFFUSIONS

Besides the unifying view of (4), the main contribution here is on efficiently designing $\mathbf{f}_c(\boldsymbol{\theta}_c)$ in (4), by learning the corresponding $\boldsymbol{\theta}_c$ per class. Thus, unlike PPR and HK, the methods introduced here can afford class-specific label propagation that is *adaptive* to the graph structure, and also *adaptive* to the labeled nodes; see Fig. 1 for a high-level illustration of the proposed adaptive diffusion framework.

Consider for generality a goodness-of-fit loss $\ell(\cdot)$, and a regularizer $R(\cdot)$ promoting e.g., smoothness over the graph. Using these, the sought class distribution will be

$$\hat{\mathbf{f}}_c = \arg\min_{\mathbf{f}\in\mathbb{R}^N} \ell(\mathbf{y}_{\mathcal{L}_c}, \mathbf{f}) + \lambda R(\mathbf{f}) \tag{8}$$

where λ tunes the degree of regularization, and

$$[\mathbf{y}_{\mathcal{L}_c}]_i = \begin{cases} 1, & i \in \mathcal{L}_c \\ 0, & \text{else} \end{cases}$$

is the indicator vector of the nodes belonging to class c. Using our diffusion model in (4), the $N-$dimensional optimization problem (8) reduces to solving for the $K-$dimensional vector ($K \ll N$)

$$\hat{\boldsymbol{\theta}}_c = \arg\min_{\boldsymbol{\theta}\in\mathcal{S}^K} \ell(\mathbf{y}_{\mathcal{L}_c}, \mathbf{f}_c(\boldsymbol{\theta})) + \lambda R(\mathbf{f}_c(\boldsymbol{\theta})). \tag{9}$$

Although many choices of $\ell(\cdot)$ may be of interest, we will focus for simplicity on the quadratic loss, namely

$$\begin{aligned} \ell(\mathbf{y}_{\mathcal{L}_c}, \mathbf{f}) &:= \sum_{i\in\mathcal{L}} \frac{1}{d_i}([\bar{\mathbf{y}}_{\mathcal{L}_c}]_i - f_i)^2 \\ &= (\bar{\mathbf{y}}_{\mathcal{L}_c} - \mathbf{f})^\mathsf{T} \mathbf{D}_{\mathcal{L}}^\dagger (\bar{\mathbf{y}}_{\mathcal{L}_c} - \mathbf{f}) \end{aligned} \tag{10}$$

where $\bar{\mathbf{y}}_{\mathcal{L}_c} := (1/|\mathcal{L}|)\,\mathbf{y}_{\mathcal{L}_c}$ is the class indicator vector after normalization to avoid overfitting and numerical instabilities, and $\mathbf{D}_{\mathcal{L}}^\dagger = \text{diag}(\mathbf{d}_{\mathcal{L}}^{(-1)})$ with entries

$$[\mathbf{d}_{\mathcal{L}}^{(-1)}]_i = \begin{cases} 1/d_i, & i \in \mathcal{L} \\ 0, & \text{else} \end{cases}.$$

For a smoothness-promoting regularization, we will employ the following (normalized) Laplacian-based metric

$$R(\mathbf{f}) = \frac{1}{2}\sum_{i\in\mathcal{V}}\sum_{j\in\mathcal{N}_i}\left(\frac{f_i}{d_i} - \frac{f_j}{d_j}\right)^2 = \mathbf{f}^\mathsf{T}\mathbf{D}^{-1}\mathbf{L}\mathbf{D}^{-1}\mathbf{f}. \tag{11}$$

Intuitively speaking, (10) favors vectors $\mathbf{f}$ having non-zero ($|1/|\mathcal{L}|$) values on nodes that are known to belong to class c, and zero values on nodes that are known to belong to other classes ($\mathcal{L}\setminus\mathcal{L}_c$), while (11) promotes similarity of the entries of $\mathbf{f}$ that correspond to neighboring nodes. In (10) and (11), each entry f_i is normalized by $d_i^{-\frac{1}{2}}$ and d_i^{-1} respectively. This normalization counterbalances the tendency of random walks to concentrate on high-degree nodes, thus placing equal importance to all nodes.

Substituting (10) and (11) into (9), and recalling from (4) that $\mathbf{f}_c(\boldsymbol{\theta}) = \mathbf{P}_c^{(K)}\boldsymbol{\theta}$, yields the convex quadratic program

$$\hat{\boldsymbol{\theta}}_c = \arg\min_{\boldsymbol{\theta}\in\mathcal{S}^K} \boldsymbol{\theta}^\mathsf{T}\mathbf{A}_c\boldsymbol{\theta} + \boldsymbol{\theta}^\mathsf{T}\mathbf{b}_c \tag{12}$$

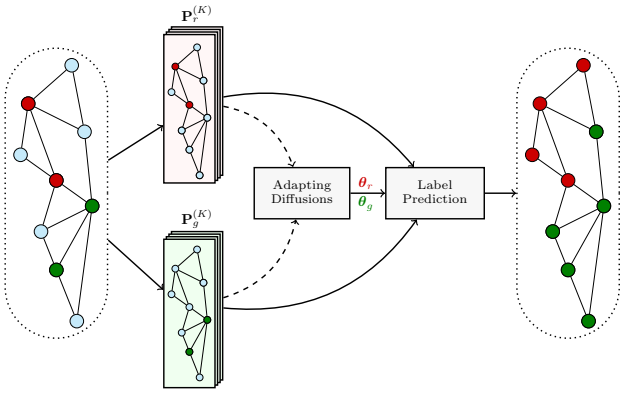

Figure 1: High-level illustration of adaptive diffusions. The nodes belong to two classes (red and green). The per-class diffusions are learned by exploiting the landing probability spaces produced by random walks rooted at the sample nodes (second layer: up for red; down for green).

with $\mathbf{b}_c$ and $\mathbf{A}_c$ given by

$$\mathbf{b}_c = -\frac{2}{|\mathcal{L}|}(\mathbf{P}_c^{(K)})^\mathsf{T}\mathbf{D}_{\mathcal{L}}^\dagger\mathbf{y}_{\mathcal{L}_c} \tag{13}$$

$$\begin{aligned} \mathbf{A}_c &= (\mathbf{P}_c^{(K)})^\mathsf{T}\mathbf{D}_{\mathcal{L}}^\dagger\mathbf{P}_c^{(K)} + \lambda(\mathbf{P}_c^{(K)})^\mathsf{T}\mathbf{D}^{-1}\mathbf{L}\mathbf{D}^{-1}\mathbf{P}_c^{(K)} \\ &= (\mathbf{P}_c^{(K)})^\mathsf{T}\left[\left(\mathbf{D}_{\mathcal{L}}^\dagger + \lambda\mathbf{D}^{-1}\right)\mathbf{P}_c^{(K)} - \lambda\mathbf{D}^{-1}\mathbf{H}\mathbf{P}_c^{(K)}\right] \end{aligned} \tag{14}$$

$$= (\mathbf{P}_c^{(K)})^\mathsf{T}\left(\mathbf{D}_{\mathcal{L}}^\dagger\mathbf{P}_c^{(K)} + \lambda\mathbf{D}^{-1}\tilde{\mathbf{P}}_c^{(K)}\right) \tag{15}$$

where

$$\begin{aligned} \mathbf{H}\mathbf{P}_c^{(K)} &= \left[\mathbf{H}\mathbf{p}_c^{(1)} \quad \mathbf{H}\mathbf{p}_c^{(2)} \quad \cdots \quad \mathbf{H}\mathbf{p}_c^{(K)}\right] \\ &= \left[\mathbf{p}_c^{(2)} \quad \mathbf{p}_c^{(3)} \quad \cdots \quad \mathbf{p}_c^{(K+1)}\right] \end{aligned}$$

is a "shifted" version of $\mathbf{P}_c^{(K)}$, where each $\mathbf{p}_c^{(k)}$ is advanced by one step, and

$$\tilde{\mathbf{P}}_c^{(K)} := \left[\tilde{\mathbf{p}}_c^{(1)} \quad \tilde{\mathbf{p}}_c^{(2)} \quad \cdots \quad \tilde{\mathbf{p}}_c^{(K)}\right]$$

with $\tilde{\mathbf{p}}_c^{(i)} := \mathbf{p}_c^{(i)} - \mathbf{p}_c^{(i+1)}$ containing the "differential" landing probabilities. The complexity of "naively" finding the $K\times K$ matrix $\mathbf{A}_c$ (and thus also $\mathbf{b}_c$) is $O(K^2N)$ for computing the first summand, and $O(|\mathcal{E}|K)$ for the second summand in (14), after leveraging the sparsity of $\mathbf{L}$, which means $|\mathcal{E}| \ll N^2$. But since columns of $\tilde{\mathbf{P}}_c^{(K)}$ are obtained as differences of consecutive columns of $\mathbf{P}_c^{(K)}$, this load of $O(|\mathcal{E}|K)$ is saved. In a nutshell, the solver in (12)-(15) that we term adaptive-diffusion (AdaDIF), incurs complexity of order $O(K^2N)$.

3.1 Limiting behavior and computational complexity

In this section, we offer further insights on the model (4), along with complexity analysis of the parametric solution in (12). To start, the next proposition establishes the limiting behavior of AdaDIF as the regularization parameter grows; for the proof see [7].

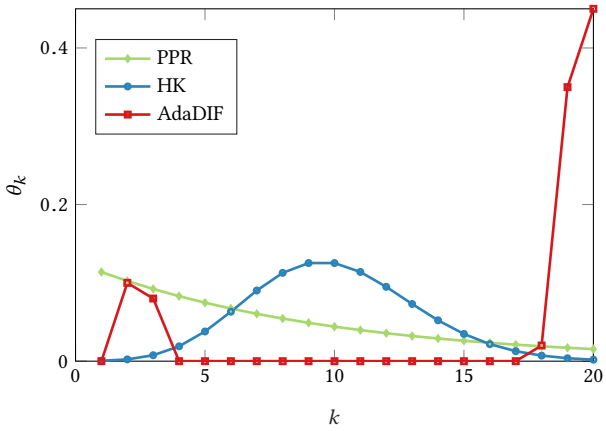

Figure 2: Illustration of $K = 20$ landing probability coefficients for PPR with $\alpha = 0.9$, HK with $t = 10$, and AdaDIF.

Proposition 3.1. *If the second largest eigenvalue of* $\mathbf{H}$ *has multiplicity 1, then for* K *sufficiently large but finite, the solution to* (12) *as* $\lambda \to \infty$ *satisfies*

$$\hat{\boldsymbol{\theta}}_c = \mathbf{e}_K, \ \forall \ \mathcal{L}_c \subseteq \mathcal{V}. \tag{16}$$

Our experience with solving (12) reveal that the sufficiently large K required for (16) to hold, can be as small as 10^2. As $\lambda \to \infty$, the effect of the loss in (10) vanishes. According to Proposition 1, this causes AdaDIF to boost smoothness by concentrating the simplex weights (entries of $\hat{\boldsymbol{\theta}}_c$) on landing probabilities of the late steps (k close to K). If on the other extreme, smoothness-over-the-graph is not promoted (cf. $\lambda = 0$), the sole objective of AdaDIF is to construct diffusions that best fit the available labeled data. Since short-length random walks from a given node typically lead to nodes of the same class, while longer walks to other classes, AdaDIF with $\lambda = 0$ tends to leverage only a few landing probabilities of early steps (k close to 1). For moderate values of λ, AdaDIF effectively adapts per-class diffusions by balancing the emphasis on initial versus final landing probabilities.

Fig. 2 depicts an example of how AdaDIF places weights $\{\theta_k\}_{k=1}^K$ on landing probabilities after a maximum of $K = 20$ steps, generated from few samples belonging to one of 7 classes of the `Cora` citation network. Note that the learnt coefficients may follow radically different patterns than those dictated by standard *non-adaptive* diffusions such as PPR or HK. It is worth noting that the simplex constraint induces sparsity of the solution in (12), thus 'pushing' $\{\theta_k\}$ entries to zero.

The computational core of the proposed method is to build the landing probability matrix $\mathbf{P}_c^{(K)}$, whose columns are computed fast using power iterations leveraging the sparsity of $\mathbf{H}$ (cf. (2)). This endows AdaDIF with high computational efficiency, especially for small K. Specifically, since for solving (12) adaDIF incurs complexity $O(K^2N)$ per class, if $K < |\mathcal{E}|/N$, this becomes $O(|\mathcal{E}|K)$; and for $|\mathcal{Y}|$ classes, the overall complexity of AdaDIF is $O(|\mathcal{Y}||\mathcal{E}|K)$, which is in the same order as that of non-adaptive diffusions such as PPR and HK. For larger K however, an additional $O(K^2N)$ is required per class, mainly to obtain $\mathbf{A}_c$ in (15).

Overall, if $O(KN)$ memory requirements are met, the runtime of AdaDIF scales *linearly* with $|\mathcal{E}|$, provided that K remains small. Thankfully, small values of K are usually sufficient to achieve high learning performance. As will be shown in the next section, this observation is in par with the analytical properties of diffusion based classifiers, where it turns out that K large does not improve classification accuracy.

3.2 On the choice of K

Here we elaborate on how the selection of K influences the classification task at hand. As expected, the effect of K is intimately linked to the topology of the underlying graph, the labeled nodes, and their properties. For simplicity, we will focus on binary classification with the two classes denoted by " + " and " − ." Central to our subsequent analysis is a concrete measure of the effect an extra landing probability vector $\mathbf{p}_c^{(k)}$ can have on the outcome of a diffusion-based classifier. Intuitively, this effect is diminishing as the number of steps K grows, as both random walks eventually converge to the same stationary distribution. Motivated by this, we introduce next the γ-distinguishability threshold.

Definition 3.2 (γ-distinguishability threshold). Let $\mathbf{p}_+$ and $\mathbf{p}_-$ denote respectively the seed vectors for nodes of class " + " and "−," initializing the landing probability vectors in matrices $\mathbf{X}_c := \mathbf{P}_c^{(K)}$, and $\check{\mathbf{X}}_c := \left[\mathbf{p}_c^{(1)} \cdots \mathbf{p}_c^{(K-1)} \mathbf{p}_c^{(K+1)}\right]$, where $c \in \{+,-\}$. With $\mathbf{y} := \mathbf{X}_+\boldsymbol{\theta} - \mathbf{X}_-\boldsymbol{\theta}$ and $\check{\mathbf{y}} := \check{\mathbf{X}}_+\boldsymbol{\theta} - \check{\mathbf{X}}_-\boldsymbol{\theta}$, the γ-distinguishability threshold of the diffusion-based classifier is the smallest integer K_γ satisfying

$$\|\mathbf{y} - \check{\mathbf{y}}\| \leq \gamma \ .$$

The following theorem establishes an upper bound on K_γ expressed in terms of fundamental quantities of the graph, as well as basic properties of the labeled nodes per class; see [7] for a proof.

Theorem 3.3. *For any diffusion-based classifier with coefficients* $\boldsymbol{\theta}$ *constrained to a probability simplex of appropriate dimensions, the* γ*-distinguishability threshold is upper-bounded as*

$$K_\gamma \leq \frac{1}{\mu'} \log\left[\frac{2\sqrt{d_{\max}}}{\gamma}\left(\sqrt{\frac{1}{d_{\min-}|\mathcal{L}_-|}} + \sqrt{\frac{1}{d_{\min+}|\mathcal{L}_+|}}\right)\right]$$

where $d_{\min+} := \min_{i\in\mathcal{L}_+} d_i$, $d_{\min-} := \min_{j\in\mathcal{L}_-} d_j$, $d_{\max} := \max_{i\in\mathcal{V}} d_i$ *and* $\mu' := \min\{\mu_2, 2-\mu_N\}$ *where* $\{\mu_n\}_{n=1}^N$ *denote the eigenvalues of the normalized graph Laplacian in ascending order.*

The γ-distinguishability threshold can guide the choice of the dimension K of the landing probability space. Indeed, using class-specific landing probability steps $K \geq K_\gamma$, does not help distinguishing between the corresponding classes, in the sense that the classifier output is not perturbed by more than γ. Intuitively, the information contained in the landing probabilities $K_\gamma + 1, K_\gamma + 2, \ldots$ is essentially the same for both classes and thus, using them as features unnecessarily increases the overall complexity of the classifier, and also "opens the door" to curse of dimensionality related concerns.

Theorem 3.3 makes no assumptions on the diffusion coefficients, so long they belong to a probability simplex. Of course, specifying the diffusion function can specialize and further tighten the corresponding γ-distinguishability threshold. Conveniently, our experiments suggest that $K \in [10, 20]$ is usually sufficient to achieve

high performance for most real graphs. Nevertheless, longer random walks may be necessary in e.g., graphs with small μ', especially when the number of labeled nodes is scarce. To deal with such challenges, the ensuing modification of AdaDIF that scales linearly with K is nicely motivated.

3.3 Dictionary of diffusions

The present section deals with a modified version of AdaDIF, where the number of parameters (dimension of $\boldsymbol{\theta}$) is restricted to $D < K$, meaning the "degrees of freedom" of each class-specific distribution are fewer than the number of landing probabilities. Specifically, consider (cf. (4))

$$\mathbf{f}_c(\boldsymbol{\theta}) = \sum_{k=1}^{K} a_k(\boldsymbol{\theta})\mathbf{p}_c^{(k)} = \mathbf{P}_c^{(K)}\mathbf{a}(\boldsymbol{\theta})$$

where $a_k(\boldsymbol{\theta}) := \sum_{d=1}^{D} \theta_d C_{kd}$, and $\mathbf{C} := \left[\mathbf{c}_1 \cdots \mathbf{c}_D\right] \in \mathbb{R}^{K\times D}$ is a *dictionary* of D coefficient vectors, the i^{th} forming the column $\mathbf{c}_i \in \mathcal{S}^K$. Since $\mathbf{a}(\boldsymbol{\theta}) = \mathbf{C}\boldsymbol{\theta}$, it follows that

$$\mathbf{f}_c(\boldsymbol{\theta}) = \mathbf{P}_c^{(K)}\mathbf{C}\boldsymbol{\theta} = \sum_{d=1}^{D} \theta_d \mathbf{f}_c^{(d)}$$

where $\mathbf{f}_c^{(d)} := \sum_{k=1}^{K} C_{kd}\mathbf{p}_c^{(k)}$ is the d^{th} diffusion.

To find the optimal $\boldsymbol{\theta}$, the optimization problem in (12) is solved with

$$\mathbf{b}_c = -\frac{2}{|\mathcal{L}|}(\mathbf{F}_c^{\Delta})^{\mathsf{T}}\mathbf{D}_{\mathcal{L}}^{\dagger}\mathbf{y}_{\mathcal{L}^c} \tag{17}$$

$$\mathbf{A}_c = (\mathbf{F}_c^{\Delta})^{\mathsf{T}}\mathbf{D}_{\mathcal{L}}^{\dagger}\mathbf{F}_c^{\Delta} + \lambda(\mathbf{F}_c^{\Delta})^{\mathsf{T}}\mathbf{D}^{-1}\mathbf{L}\mathbf{D}^{-1}\mathbf{F}_c^{\Delta} \tag{18}$$

where $\mathbf{F}_c^{\Delta} := [\mathbf{f}_c^{(1)} \cdots \mathbf{f}_c^{(D)}]$ effectively replaces $\mathbf{P}_c^{(K)}$ as the basis of the space on which each $\mathbf{f}_c$ is constructed. The description of AdaDIF in *dictionary mode* is given as a special case of Algorithm 1, together with the subroutine in Algorithm 3 for memory-efficient generation of $\mathbf{F}_c^{\Delta}$.

The motivation behind this dictionary-based variant of AdaDIF is two-fold. First, it leverages the properties of a judiciously selected basis of known diffusions, e.g. by constructing $\mathbf{C} = \left[\boldsymbol{\theta}_{\text{PPR}}, \boldsymbol{\theta}_{\text{HK}}, \ldots\right]$. In that sense, our approach is related to multi-kernel methods, e.g. [1], although significantly more scalable than the latter. Second, the complexity of AdaDIF in dictionary mode is $\mathcal{O}(|\mathcal{E}|(K + D))$, where D can be arbitrarily smaller than K, leading to complexity that is *linear* with respect to both K and $|\mathcal{E}|$.

3.4 Unconstrained diffusions

Thus far, the diffusion coefficients $\boldsymbol{\theta}$ have been constrained on the $K-$dimensional probability simplex $\mathcal{S}^K$, resulting in sparse solutions $\hat{\boldsymbol{\theta}}_c$, as well as $\mathbf{f}_c(\hat{\boldsymbol{\theta}}_c) \in \mathcal{S}^N$. The latter also allows each $\mathbf{f}_c(\boldsymbol{\theta})$ to be interpreted as a pmf over $\mathcal{V}$. Nevertheless, the simplex constraint imposes a limitation to the model: landing probabilities may only have *non-negative* contribution on the resulting class distribution. Upon relaxing this non-negativity constraint, (12) simplifies to

$$\hat{\boldsymbol{\theta}}_c = \arg\min_{\substack{\boldsymbol{\theta}\in\mathbb{R}^K \\ \mathbf{1}^{\mathsf{T}}\boldsymbol{\theta}=1}} \boldsymbol{\theta}^{\mathsf{T}}\mathbf{A}_c\boldsymbol{\theta} + \boldsymbol{\theta}^{\mathsf{T}}\mathbf{b}_c \tag{19}$$

Algorithm 1 Adaptive Diffusions

Input: Adjacency matrix: $\mathbf{W}$, Labeled nodes: $\{y_i\}_{i\in\mathcal{L}}$
***parameters*:** Regularization parameter: λ, # of landing probabilities: K, Dictionary mode $\in$ {True, False}, Unconstrained $\in$ {True, False}
Output: Predictions: $\{\hat{y}_i\}_{i\in\mathcal{U}}$
Extract $\mathcal{Y}$ = { Set of unique labels in: $\{y_i\}_{i\in\mathcal{L}}$}
for $c \in \mathcal{Y}$ **do**
  $\mathcal{L}_c = \{i \in \mathcal{L} : y_i = c\}$
  if Dictionary mode **then**
    $\mathbf{F}_c^{\Delta}$ = Dictionary $(\mathbf{W}, \mathcal{L}_c, K, \mathbf{C})$
    Obtain $\mathbf{b}_c$ and $\mathbf{A}_c$ as in (17) and (18)
    $\mathbf{F}_c = \mathbf{F}_c^{\Delta}$
  else
    $\{\mathbf{P}_c^{(K)}, \tilde{\mathbf{P}}_c^{(K)}\}$ = LandProb$(\mathbf{W}, \mathcal{L}_c, K)$
    Obtain $\mathbf{b}_c$ and $\mathbf{A}_c$ as in (13) and (15)
    $\mathbf{F}_c = \mathbf{P}_c^{(K)}$
  end if
  if Unconstrained **then**
    Obtain $\hat{\boldsymbol{\theta}}_c$ as in (20)
  else
    Obtain $\hat{\boldsymbol{\theta}}_c$ by solving (12)
  end if
  $\mathbf{f}_c(\hat{\boldsymbol{\theta}}_c) = \mathbf{F}_c\hat{\boldsymbol{\theta}}_c$
end for
Obtain $\hat{y}_i = \arg\max_{c\in\mathcal{Y}} \left[\mathbf{f}_c(\hat{\boldsymbol{\theta}}_c)\right]_i$, $\forall i \in \mathcal{U}$

Algorithm 2 LandProb

Input: $\mathbf{W}, \mathcal{L}_c, K$
Output: $\mathbf{P}_c^{(K)}$, $\tilde{\mathbf{P}}_c^{(K)}$
$\mathbf{H} = \mathbf{W}\mathbf{D}^{-1}$
$\mathbf{p}_c^{(0)} = \mathbf{v}_c$
for $k = 1 : K + 1$ **do**
  $\mathbf{p}_c^{(k)} = \mathbf{H}\mathbf{p}_c^{(k-1)}$
  $\tilde{\mathbf{p}}_c^{(k)} = \mathbf{p}_c^{(k-1)} - \mathbf{p}_c^{(k)}$
end for

Algorithm 3 Dictionary

Input: $\mathbf{W}, \mathcal{L}_c, K, \mathbf{C}$
Output: $\mathbf{F}_c^{\Delta}$
$\mathbf{H} = \mathbf{W}\mathbf{D}^{-1}$
$\mathbf{p}_c^{(0)} = \mathbf{v}_c$
$\{\mathbf{f}_c^{(d)}\}_{d=1}^{D} = \mathbf{0}$
for $k = 1 : K$ **do**
  $\mathbf{p}_c^{(k)} = \mathbf{H}\mathbf{p}_c^{(k-1)}$
  for $d = 1 : D$ **do**
    $\mathbf{f}_c^{(d)} = \mathbf{f}_c^{(d)} + C_{kd}\mathbf{p}_c^{(k)}$
  end for
end for

which can afford a closed-form solution as

$$\hat{\boldsymbol{\theta}}_c = \mathbf{A}_c^{-1}(\mathbf{b}_c - \lambda^*\mathbf{1}), \qquad \lambda^* = \frac{\mathbf{1}^{\mathsf{T}}\mathbf{A}_c^{-1}\mathbf{b}_c - 1}{\mathbf{b}^{\mathsf{T}}\mathbf{A}_c^{-1}\mathbf{b}_c}. \tag{20}$$

Retaining the hyperplane constraint $\mathbf{1}^\top\boldsymbol{\theta} = 1$ prevents the trivial solution $\boldsymbol{\theta} = \mathbf{0}$, and forces at least one entry of $\boldsymbol{\theta}$ to be positive. Note that for $K > |\mathcal{L}|$, (19) may become ill conditioned, and yield inaccurate solutions. This can be mitigated by imposing ℓ_2-norm regularization on $\boldsymbol{\theta}$, which is equivalent to adding $\epsilon\mathbf{I}$ to $\mathbf{A}_c$, with $\epsilon > 0$ sufficiently large to stabilize the linear system.

A step-by-step description of the proposed AdaDIF approach is given by Algorithm 1, along with the subroutine in Algorithm 2. Determining the limiting behavior of unconstrained AdaDIF, as well as exploring the effectiveness of different regularizers (e.g., sparsity inducing ℓ_1-norm) is part of our ongoing research.

4 RELATION TO PRIOR WORKS

Following the seminal contribution in [9] that introduced PageRank as a network centrality measure, there has been a vast body of works studying its theoretical properties, computational aspects, as well as applications beyond Web ranking [16, 25]. Most formal approaches to generalize PageRank focus either on the *teleportation* component (see e.g. [30, 31] as well as [8] for an application to semi-supervised classification), or, on the so-termed *damping* mechanism [4, 13]. Perhaps the most general setting can be found in [4], where a family of functional rankings was introduced by the choice of a parametric damping function that assigns weights to successive steps of a walk initialized according to the teleportation distribution. The per class distributions produced by AdaDIF are in fact members of this family of functional rankings. However, instead of choosing a fixed damping function as in the aforementioned approaches, AdaDIF learns a class-specific and graph-aware damping mechanism. In this sense, AdaDIF undertakes statistical learning in the space of functional rankings, tailored to the underlying semi-supervised classification task.

AdaDIF also shares links with SSL methods based on graph signal processing proposed in [35], and further pursued in [12] for bridge monitoring; see also [36] and [14] for recent advances on graph filters. Similar to our approach, these graph filter based techniques are parametrized via assigning different weights to a number of consecutive powers of a matrix related to the structure of the graph. Our contribution however, introduces different loss and regularization functions for adapting the diffusions. Furthermore, while [35] focuses on binary classification and [12] identifies a single model for all classes, our approach allows for different classes to have different propagation mechanisms. This feature can accommodate differences in the label distribution of each class over the nodes, while also making AdaDIF readily applicable to multi-label graphs. Moreover, while in [35] the weighting parameters remain unconstrained and in [12] belong to a hyperplane, AdaDIF constrains the diffusion parameters on the probability simplex, which allows the random-walk-based diffusion vectors to denote valid probability mass functions over the nodes of the network. This certainly enhances interpretability of the method, improves the numerical stability of the involved computations, and also reduces the search-space of the model is beneficial under data scarcity. Finally, imposing the simplex constraint makes the model amenable to a rigorous analysis that relates the dimensionality of the feature space to basic graph properties, as well as to a disciplined exploration of its limiting behavior.

Table 1: Network Characteristics

Network	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{E}\rvert$	$\lvert\mathcal{Y}\rvert$	Multilabel
`Citeseer`	3,233	9,464	6	No
`Cora`	2,708	10,858	7	No
`PubMed`	19,717	88,676	3	No
`PPI (H. Sapiens)`	3,890	76,584	50	Yes
`Wikipedia`	4,733	184,182	40	Yes
`BlogCatalog`	10,312	333,983	39	Yes

5 EXPERIMENTAL EVALUATION

Our experiments compare the classification accuracy of the novel AdaDIF approach with state-of-the-art alternatives. For the comparisons, we use 6 benchmark labeled graphs whose dimensions and basic attributes are summarized in Table 1. All 6 graphs have nodes that belong to multiple classes, while the last 3 are *multilabeled* (each node has *one or more* labels). We evaluate performance of AdaDIF and the following: i) PPR and HK, which are special cases of AdaDIF as discussed in Section 2; ii) Node2vec [18]; iii) Deepwalk [32]; iv) Planetoid-G [40]; and, v) graph convolutional networks (GCNs) [20].

We performed 10-fold cross-validation to select parameters needed by i) - v). For HK, we performed grid search over $t \in [1, 5, 10, 15]$. For PPR, we fixed $\alpha = 0.98$ since it is well documented that α close to 1 yields reliable performance; see e.g., [27]. Both HK and PPR were run up to $K = 30$ for convergence to be in effect. For Node2vec, we fixed most parameters to the values suggested in [18], and performed grid search for $p, q \in [0.25, 1.0, 2.0, 4.0]$. Since Deepwalk can be seen as Node2vec with $p = q = 1.0$, we used the Node2vec Python implementation for both. As in [18, 32], we used the embeded node-features to train a supervised logistic regression classifier with ℓ_2 regularization. For AdaDIF, we fixed $\lambda = 15.0$, while $K = 15$ was sufficient to attain desirable accuracy; only the values of Boolean variables *Unconstained* and *Dictionary Mode* (see Algorithm 1) were tuned by validation. For the multilabel graphs, we found $\lambda = 5.0$ and even shorter walks of $K = 10$ to perform well. For the dictionary mode of AdaDIF, we preselected $D = 10$, with the first five collumns of $\mathbf{C}$ being HK coefficients with parameters $t \in [5.0, 20.0]$, and the other five polynomial coefficients $c_i = k^\beta$ with $\beta \in [2.0, 10]$.

For multiclass experiments, we evaluated the performance of all algorithms on the three benchmark citation networks, namely `Cora`, `Citeseer`, and `PubMed`. We obtained the labels of an increasing number of nodes via uniform random sampling, and predicted the labels of the remaining nodes. For each experiment, classification accuracy was measured on the unlabeled nodes in terms of Micro-F1 and Macro-F1 scores; see e.g., [28]. The results were averaged over 20 experiments, with mean and standard deviation reported in Table 2. Evidently, AdaDIF achieves state of the art performance for all graphs. For `Cora` and `PubMed`, AdaDIF was switched to dictionary mode, while for `Citeseer`, where the gain in accuracy is more significant, unconstrained diffusions were employed. In the multiclass setting, diffusion-based classifiers (AdaDIF, PPR, and HK) outperformed the embedding-based methods by a small margin, and GCNs by a larger margin. It should be noted

Table 2: Micro F1 and Macro F1 Scores of Various Algorithms on Multiclass Networks

	Network	Cora			Citeseer			PubMed		
	$\|\mathcal{L}\|/\|\mathcal{V}\|$	2.5%	5%	10%	2.5%	5%	10%	0.25%	0.5%	1.0%
Micro-F1	AdaDIF	**70.5 ± 2.4**	**73.7 ± 1.7**	**77.0 ± 1.0**	**51.9 ± 0.9**	**55.1 ± 1.0**	**58.6 ± 0.7**	**72.8 ± 2.4**	**76.1 ± 0.8**	76.5 ± 0.5
	PPR	69.8 ± 2.5	73.3 ± 1.4	**77.0 ± 1.0**	49.7 ± 2.2	53.0 ± 1.5	57.5 ± 0.8	71.4 ± 2.6	74.4 ± 1.1	76.0 ± 0.8
	HK	70.0 ± 2.4	73.5 ± 1.8	76.7 ± 1.2	50.0 ± 2.1	53.5 ± 1.5	57.3 ± 0.9	**72.8 ± 2.6**	75.1 ± 1.0	**76.8 ± 0.7**
	Node2vec	69.5 ± 1.8	73.0 ± 1.6	75.5 ± 1.4	46.0 ± 2.7	49.7 ± 1.7	52.1 ± 1.4	**72.8 ± 2.8**	74.8 ± 1.6	75.1 ± 1.4
	Deepwalk	68.2 ± 2.5	72.1 ± 1.8	74.9 ± 1.2	45.0 ± 2.4	48.5 ± 1.7	51.2 ± 1.2	72.4 ± 2.6	73.8 ± 1.3	74.5 ± 1.2
	Planetoid-G	62.5 ± 5.1	67.3 ± 4.3	75.8 ± 1.1	43.0 ± 1.8	46.8 ± 1.9	55.2 ± 1.3	63.4 ± 3.7	65.2 ± 2.0	67.8 ± 1.5
	GCN	58.3 ± 4.0	66.5 ± 2.1	71.3 ± 1.7	38.9 ± 2.7	44.5 ± 2.0	50.3 ± 1.6	57.7 ± 3.4	64.5 ± 2.7	70.0 ± 1.5
Macro-F1	AdaDIF	**69.0 ± 2.3**	**72.3 ± 1.8**	**75.7 ± 1.2**	**46.6 ± 1.1**	**49.6 ± 1.6**	**53.9 ± 1.0**	**71.5 ± 2.5**	**74.2 ± 0.7**	75.2 ± 0.8
	PPR	66.7 ± 4.2	71.8 ± 1.6	75.3 ± 1.1	44.1 ± 2.0	48.4 ± 1.5	53.5 ± 0.8	69.5 ± 2.6	72.8 ± 1.1	74.7 ± 0.8
	HK	67.1 ± 4.2	72.1 ± 1.9	75.5 ± 1.4	44.8 ± 2.0	48.9 ± 1.5	53.7 ± 1.0	71.0 ± 2.6	73.5 ± 1.1	**75.6 ± 0.8**
	Node2vec	67.1 ± 2.6	71.6 ± 1.8	74.0 ± 1.3	42.6 ± 2.5	46.6 ± 1.7	48.7 ± 1.3	70.3 ± 3.2	73.0 ± 1.8	73.5 ± 1.4
	Deepwalk	66.1 ± 3.2	70.5 ± 2.1	73.8 ± 1.4	41.6 ± 2.4	45.5 ± 1.5	48.5 ± 1.2	70.0 ± 3.2	72.0 ± 1.7	73.1 ± 1.3
	Planetoid-G	58.0 ± 5.1	64.3 ± 4.3	74.3 ± 1.6	37.4 ± 2.1	41.6 ± 2.2	52.0 ± 2.4	61.0 ± 3.9	63.7 ± 3.0	65.2 ± 2.0
	GCN	52.0 ± 6.8	61.9 ± 2.6	64.8 ± 1.9	33.0 ± 3.0	39.2 ± 1.7	43.3 ± 1.6	52.1 ± 4.4	60.2 ± 3.9	65.3 ± 2.2

Table 3: Micro F1 and Macro F1 Scores of Various Algorithms on Multilabel Networks

	Network	PPI			BlogCatalog			Wikipedia		
	$\|\mathcal{L}\|/\|\mathcal{V}\|$	10%	20%	30%	10%	20%	30%	10%	20%	30%
Micro-F1	AdaDIF	15.4 ± 0.5	17.9 ± 0.7	**19.2 ± 0.6**	31.5 ± 0.6	34.4 ± 0.5	36.3 ± 0.4	28.2 ± 0.9	30.0 ± 0.5	31.2 ± 0.7
	PPR	13.8 ± 0.5	15.8 ± 0.6	17.0 ± 0.4	21.1 ± 0.8	23.6 ± 0.6	25.2 ± 0.6	10.5 ± 1.5	8.1 ± 0.7	7.2 ± 0.5
	HK	14.5 ± 0.5	16.7 ± 0.6	18.1 ± 0.5	22.2 ± 1.0	24.7 ± 0.7	26.6 ± 0.7	9.3 ± 1.4	7.3 ± 0.7	6.0 ± 0.7
	Node2vec	**16.5 ± 0.6**	**18.2 ± 0.3**	19.1 ± 0.3	**35.0 ± 0.3**	**36.3 ± 0.3**	**37.2 ± 0.2**	**42.3 ± 0.9**	**44.0 ± 0.6**	**45.1 ± 0.4**
	Deepwalk	16.0 ± 0.6	17.9 ± 0.5	18.8 ± 0.4	34.2 ± 0.4	35.7 ± 0.3	36.4 ± 0.4	41.0 ± 0.8	43.5 ± 0.5	44.1 ± 0.5
Macro-F1	AdaDIF	**13.4 ± 0.6**	**15.4 ± 0.7**	**16.5 ± 0.7**	**23.0 ± 0.6**	**25.3 ± 0.4**	**27.0 ± 0.4**	**7.7 ± 0.3**	**8.3 ± 0.3**	**9.0 ± 0.2**
	PPR	12.9 ± 0.4	14.7 ± 0.5	15.8 ± 0.4	17.3 ± 0.5	19.5 ± 0.4	20.8 ± 0.3	4.4 ± 0.3	3.8 ± 0.6	3.6 ± 0.2
	HK	**13.4 ± 0.6**	**15.4 ± 0.5**	**16.5 ± 0.4**	18.4 ± 0.6	20.7 ± 0.4	22.3 ± 0.4	4.2 ± 0.4	3.7 ± 0.5	3.5 ± 0.2
	Node2vec	13.1 ± 0.6	15.2 ± 0.5	16.0 ± 0.5	16.8 ± 0.5	19.0 ± 0.3	20.1 ± 0.4	7.6 ± 0.3	8.2 ± 0.3	8.5 ± 0.3
	Deepwalk	12.7 ± 0.7	15.1 ± 0.6	16.0 ± 0.5	16.6 ± 0.5	18.7 ± 0.5	19.6 ± 0.4	7.3 ± 0.3	8.1 ± 0.2	8.2 ± 0.2

however that GCNs were mainly designed to combine the graph with node features. In our "featureless" setting, we used the identity matrix columns as input features, as suggested in [20, Appendix]. The scalabilty of AdaDIF[1] is reflected on the runtime comparisons listed in Fig. 3. All experiments were run on a machine with i5 @3.50 Mhz CPU, and 16GB of RAM. For the compared algorithms we used the implementations provided by the authors.

Finally, Table 3 presents the results on multilabel graphs, where we compare with Deepwalk and Node2vec, since the rest of the methods are designed for multiclass problems. Since these graphs entail a large number of classes, we increased the number of training samples. Similar to [18] and [32], during evaluation of accuracy the number of labels per sampled node is known, and check how many of them are in the top predictions. First, we observe that AdaDIF markedly outperforms PPR and HK across graphs and metrics. Furthermore, for the PPI and BlogCatalog graphs the Micro-F1 score of AdaDIF comes close to that of the much heavier state-of-the-art Node2vec. Finally, AdaDIF outperforms the competing alternatives in terms of Macro-F1 score.

[1]Open source implementation available at: https://github.com/DimBer/SSL_lib

6 CONCLUSIONS

The present work, introduces a principled, data-efficient approach to learning class-specific diffusion functions tailored for the underlying network topology. Experiments on real networks confirm that adapting the diffusion function to the given graph and observed labels, significantly improves the performance over fixed diffusions; reaching—and many times surpassing—the classification accuracy of computationally heavier state-of-the-art competing methods.

Emerging from this work are many exciting directions to explore. First, one can investigate different cost functions with respect to which the diffusions are adapted, e.g., by taking into account robustness of the resulting classifier in the presence of adversarial data. Furthermore, it is worth investigating the space of nonlinear

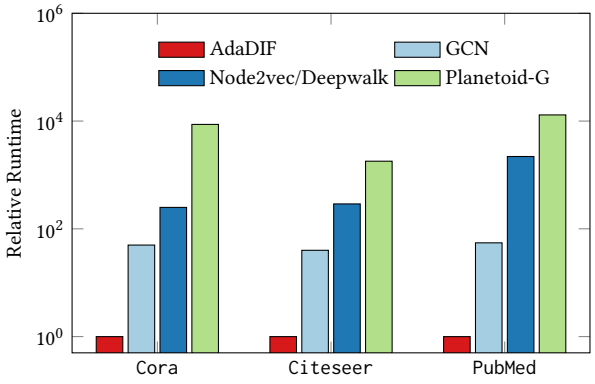

Figure 3: Relative runtime for multiclass networks.

functions of the landing probabilities to determine the degree to which accuracy can be boosted further. Last but not least, it will be interesting to develop adaptive diffusion methods, where learning and adaptation are performed *on-the-fly*, without any memory and computational overhead.

ACKNOWLEDGMENTS

This work was supported by NSF 171141, 1514056 and 1500713.

REFERENCES

[1] A. Argyriou, M. Herbster, and M. Pontil, "Combining graph laplacians for semi–supervised learning," in *Proc. Advances in Neural Information Processing Systems*, Vancouver, Can., 2006, pp. 67–74.

[2] J. Atwood and D. Towsley, "Diffusion-convolutional neural networks," in *Proc. Advances in Neural Information Processing Systems*, Barcelona, Spain, 2016, pp. 1993–2001.

[3] K. Avrachenkov, A. Mishenin, P. Gonçalves, and M. Sokol, "Generalized optimization framework for graph-based semi-supervised learning," *Proc. SIAM Intl. Conf. on Data Mining*, Anaheim, CA, 2012, pp. 966–974.

[4] R. Baeza-Yates, P. Boldi, and C. Castillo, "Generic damping functions for propagating importance in link-based ranking," *Internet Math.*, vol. 3, no. 4, pp. 445–478, 2006.

[5] M. Belkin, P. Niyogi, and V. Sindhwani, "Manifold regularization: A geometric framework for learning from labeled and unlabeled examples," *J. Mach. Learn. Res.*, no. 7, Nov, 2006, pp. 2399–2434.

[6] Y. Bengio, O. Delalleau, and N. Le Roux, "Label propagation and quadratic criterion," in *Semi-Supervised Learning*. Cambridge, MA, USA: MIT Press, 2006.

[7] D. Berberidis, A. N. Nikolakopoulos, and G. B. Giannakis, "Adaptive Diffusions for Scalable Learning over Graphs," *arXiv preprint arXiv:1804.02081*.

[8] D. Berberidis, A. N. Nikolakopoulos, and G. B. Giannakis, "Random walks with restarts for graph-based classification: Teleportation tuning and sampling design," in *Proc. of IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, Calgary, Can., April 2018.

[9] S. Brin and L. Page, "Reprint of: The anatomy of a large-scale hypertextual web search engine," *Comput. Netw.*, vol. 56, no. 18, pp. 3825–3833, 2012.

[10] E. Buchnik and E. Cohen, "Bootstrapped graph diffusions: Exposing the power of nonlinearity," *arXiv preprint arXiv:1703.02618*, 2017.

[11] O. Chapelle, B. Schölkopf, and A. Zien, *Semi-Supervised Learning*. Cambridge, MA, USA: MIT Press, 2006.

[12] S. Chen, F. Cerda, P. Rizzo, J. Bielak, J. H. Garrett, and J. Kovacevic, "Semi-supervised multiresolution classification using adaptive graph filtering with application to indirect bridge structural health monitoring," *IEEE Trans. Signal Process.*, vol. 62, no. 11, pp. 2879–2893, June 2014.

[13] P. G. Constantine and D. F. Gleich, "Random alpha pagerank," *Internet Math.*, vol. 6, no. 2, pp. 189–236, 2009.

[14] M. Contino, E. Isufi and G. Leus,"Distributed edge-variant graph filters," *Proc. Intl. Work. on Computational Advances in Multi-Sensor Adaptive Processing*, Curacao, Dutch Antilles, Dec. 2017, pp. 1-5.

[15] F. Chung, "The heat kernel as the pagerank of a graph," *Proc. Natl. Acad. Sci.*, vol. 104, no. 50, pp. 19 735–19 740, 2007.

[16] D. F. Gleich, "Pagerank beyond the web," *SIAM Rev.*, vol. 57, no. 3, pp. 321–363, 2015.

[17] J. Gorski, F. Pfeuffer, and K. Klamroth, "Biconvex sets and optimization with biconvex functions: a survey and extensions," *Math. Methods of Oper. Res.*, vol. 66, no. 3, pp. 373–407, Dec. 2007.

[18] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proc. of ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, San Francisco, CA, 2016, pp. 855–864.

[19] T. Joachims, "Transductive learning via spectral graph partitioning," *Proc. of Intl. Conf. on Machine Learn.*, Washington DC, 2003, pp. 290–297.

[20] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[21] K. Kloster and D. F. Gleich, "Heat kernel based community detection," in *Proc. of ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, New York, NY, 2014, pp. 1386–1395.

[22] I. M. Kloumann, J. Ugander, and J. Kleinberg, "Block models and personalized pagerank," *Proc. Natl. Acad. Sci.*, vol. 114, no. 1, pp. 33–38, 2017.

[23] R. I. Kondor and J. Lafferty, "Diffusion kernels on graphs and other discrete input spaces," in *Proc. of Int. Conf. on Machine Learning*, Syndey, Australia, 2002, pp. 315–322.

[24] B. Kveton, M. Valko, A. Rahimi, and L. Huang, "Semi-supervised learning with max-margin graph cuts," in *Proc. of. Int. Conf. on Artificial Intelligence and Statistics*, Sardinia, Italy, 2010, pp. 421–428.

[25] A. N. Langville and C. D. Meyer, "Deeper inside pagerank," *Internet Math.*, vol. 1, no. 3, pp. 335–380, 2004.

[26] D. A. Levin and Y. Peres, *Markov Chains and Mixing Times*. New York, NY, USA: Amer. Math. Soc., 2017.

[27] F. Lin and W. W. Cohen, "Semi-supervised classification of network data using very few labels," in *Proc. of Int. Conf. on Advances in Social Network Analysis and Mining*, Odense, Denmark, 2010, pp. 192–199.

[28] C. D. Manning, P. Raghavan, and H. Schutze, *Introduction to Information Retrieval*. Cambridge, MA: Cambridge University Press, 2008.

[29] E. Merkurjev, A. L. Bertozzi, and F. Chung, "A semi-supervised heat kernel pagerank MBO algorithm for data classification," Univ. of California Los Angeles, Los Angeles, US, Tech. Rep., 2016.

[30] A. N. Nikolakopoulos and J. D. Garofalakis, "Ncdawarerank: A novel ranking method that exploits the decomposable structure of the web," *Proc. ACM Intl. Conf. on Web Search and Data Mining*, Rome, Italy, 2013, pp. 143–152.

[31] A. N. Nikolakopoulos, A. Korba, and J. D. Garofalakis, "Random surfing on multipartite graphs," in *Proc. of IEEE Int. Conf. on Big Data*, Washington DC, Dec. 2016, pp. 736–745.

[32] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," *Proc. ACM SIGKDD Intl. Conf. on Knowl. Disc. and Data Mining*, New York, NY, 2014, pp. 701–710.

[33] A. T. Puig, A. Wiesel, G. Fleury, and A. O. Hero, "Multidimensional shrinkage-thresholding operator and group lasso penalties," *IEEE Signal Process. Lett.*, vol. 18, no. 6, pp. 363–366, 2011.

[34] N. Rosenfeld and A. Globerson, "Semi-supervised learning with competitive infection models," *arXiv preprint arXiv:1703.06426*, 2017.

[35] A. Sandryhaila and J. M. F. Moura, "Discrete signal processing on graphs," *IEEE Trans. Signal Process.*, vol. 61, no. 7, pp. 1644–1656, April 2013.

[36] S. Segarra, A. Marques, and A. Ribeiro,"Optimal graph-filter design and applications to distributed linear network operators," *IEEE Trans. on Signal Process.*, vol. 65, no. 15, pp. 4117–4131, August 2017.

[37] P. P. Talukdar and K. Crammer, "New regularized algorithms for transductive learning," in *Proc. of Joint Eur. Conf. on Machine Learning and Knowledge Discovery in Databases*, 2009, pp. 442–457.

[38] J. Ugander and L. Backstrom, "Balanced label propagation for partitioning massive graphs," in *Proc. of ACM Int. Conf. on Web Search and Data Mining*, Rome, Italy, 2013, pp. 507–516.

[39] X.-M. Wu, Z. Li, A. M. So, J. Wright, and S.-F. Chang, "Learning with partially absorbing random walks," *Proc. Adv. in Neural Inform. Proc. Systems*, Lake Tahoe, CA, Dec. 2012, pp. 3077–3085.

[40] Z. Yang, W. W. Cohen, and R. Salakhutdinov, "Revisiting semi-supervised learning with graph embeddings," *arXiv preprint arXiv:1603.08861*, 2016.

[41] X. Zhu, Z. Ghahramani, and J. Lafferty, "Semi-supervised learning using Gaussian fields and harmonic functions," in *Proc. of Int. Conf. on Machine Learning*, Washington DC, Aug. 2003.