Adaptive Diffusions for Scalable Learning over Graphs

Dimitris Berberidis[1], Athanasios N. Nikolakopoulos[2] and Georgios B. Giannakis[1,2]
Dept. of ECE[1] and Digital Tech. Center[2], University of Minnesota
Minneapolis, MN 55455, USA
E-mails: {bermp001,anikolak,georgios}@umn.edu

arXiv:1804.02081v2 [stat.ML] 5 Sep 2018

***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. Furthermore, a robust version of the classifier facilitates learning even in noisy environments. 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.**

***Index Terms*—Semi-supervised Classification, Random Walks, Diffusions.**

I. 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 [10]. Graph-based SSL has been pursued by various intertwined methods, including iterative label propagation [6], [42], [28], [24], kernels on graphs [30], manifold regularization [5], graph partitioning [39], [18], transductive learning [38], competitive infection models [35], and bootstrapped label propagation [9]. SSL based on graph filters was discussed in [36], and further developed in [11] for bridge monitoring. Recently, approaches based on node-embeddings [33], [17], [41], as well as deep-learning architectures [20], [2] 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 [22], [3], [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 resultant 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. Furthermore, an algorithm is developed to identify (and potentially remove) outlying or anomalous samples jointly with adapting the diffusions. 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 II introduces random-walk based diffusions. Our novel approach along with relevant analytical results are the subjects of Section III. Section IV presents a robust version of our algorithm, and Section V places our work in the context of related methods. Finally, Section VI presents experiments, while Section VII 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.

Work was supported by NSF 171141, 1514056 and 1500713

II. 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. [10], [24], [40]. 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 defined over the nodes, that is, the state space is equivalent to $\mathcal{V}$. The transition probabilities are

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

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).

A. Personalized PageRank Classifier

Inspired by its celebrated network centrality metric [8], 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 $\boldsymbol{\theta}_{\text{PPR}} = (1-\alpha)\begin{bmatrix}\alpha & \alpha^2 & \cdots & \alpha^K\end{bmatrix}^\mathsf{T}$, 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))

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

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.

B. Heat Kernel Classifier

The heat kernel (HK) is another popular diffusion that has recently been employed for SSL [30] and community detection on graphs [21]. HK is also a special case of (4) with $\boldsymbol{\theta}_{\text{HK}} = e^{-t}\begin{bmatrix} t & \frac{t^2}{2} & \cdots & \frac{t^K}{K!}\end{bmatrix}^\mathsf{T}$, yielding class distributions (cf. (4))

$$\mathbf{f}_c(\boldsymbol{\theta}_{\text{HK}}) = e^{-t} \sum_{k=1}^K \frac{t^k}{k!} \mathbf{p}_c^{(k)}. \tag{7}$$

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 [14].

III. 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. Figure 1 gives a high-level illustration of the proposed adaptive diffusion framework, where two classes (red and green) are first diffused over the graph (cf. (2)), followed by adaptation of class-specific diffusion coefficients; diffusions are then built (cf. (4)) and used for class prediction (cf. (5)).

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 bring target variables (entries of $\bar{\mathbf{y}}_{\mathcal{L}_c}$) and entries of $\mathbf{f}$ to the same scale, 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

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

where $\mathbf{L} := \mathbf{D} - \mathbf{W}$ is the Laplacian matrix of the graph. 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.

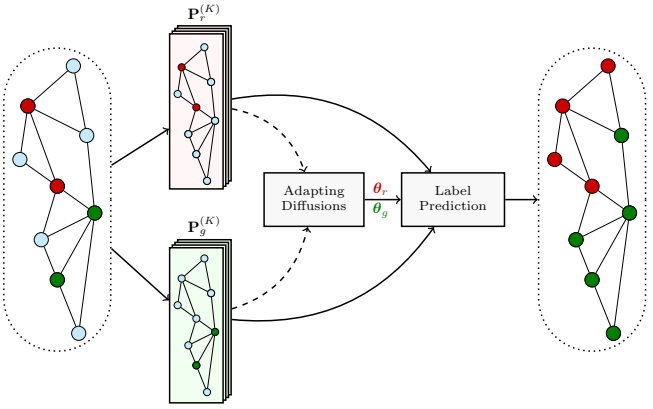

Fig. 1. High-level illustration of adaptive diffusions. The nodes belong to two classes (red and green). The per-class diffusions are learned by exploiting the landing probability spaces produced by random walks rooted at the sample nodes (second layer: up for red; down for green).

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

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

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

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

$$\begin{aligned} \mathbf{A}_c &= (\mathbf{P}_c^{(K)})^\mathsf{T}\mathbf{D}_{\mathcal{L}}^\dagger\mathbf{P}_c^{(K)} + \lambda(\mathbf{P}_c^{(K)})^\mathsf{T}\mathbf{D}^{-1}\mathbf{L}\mathbf{D}^{-1}\mathbf{P}_c^{(K)} \qquad (14) \\ &= (\mathbf{P}_c^{(K)})^\mathsf{T}\left[\left(\mathbf{D}_{\mathcal{L}}^\dagger + \lambda\mathbf{D}^{-1}\right)\mathbf{P}_c^{(K)} - \lambda\mathbf{D}^{-1}\mathbf{H}\mathbf{P}_c^{(K)}\right] \\ &= (\mathbf{P}_c^{(K)})^\mathsf{T}\left(\mathbf{D}_{\mathcal{L}}^\dagger\mathbf{P}_c^{(K)} + \lambda\mathbf{D}^{-1}\tilde{\mathbf{P}}_c^{(K)}\right) \qquad (15) \end{aligned}$$

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 $\mathcal{O}(K^2N)$ for computing the first summand, and $\mathcal{O}(|\mathcal{E}|K)$ 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 $\mathcal{O}(|\mathcal{E}|K)$ is saved.

In a nutshell, the solver in (12)-(15) that we term adaptive-diffusion (AdaDIF), incurs complexity of order $\mathcal{O}(K^2N)$.

Remark 1. The problem in (12) is a *quadratic program (QP)* of dimension equal to the walk length (or the size of dictionary when in dictionary mode). In general, solving a QP with K variables to a given precision requires a $(O)(K^3)$ worst-case complexity. Specifically, K in our setting is $\mathcal{O}(10)$ and thus negligibly small compared to the quantities that depend on

the graph dimensions. For instance, the graphs that we tested have $\mathcal{O}(10^4)$ nodes (N) and $\mathcal{O}(10^5)$ edges ($|\mathcal{E}|$). Therefore, since $K \ll N$ and $K \ll |\mathcal{E}|$ by many orders of magnitude, the $(O)(K^3)$ complexity for QP is actually dominated by the $\mathcal{O}(|\mathcal{E}|K)$ (which is the same as PPR and HK) performing the random walks and $\mathcal{O}(NK^2)$ for computing $\mathbf{A}_c$.

A. 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.

Proposition 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*

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

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 $\mathcal{O}(K^2N)$ per class, if $K < |\mathcal{E}|/N$, this becomes $\mathcal{O}(|\mathcal{E}|K)$; and for $|\mathcal{Y}|$ classes, the overall complexity of AdaDIF is $\mathcal{O}(|\mathcal{Y}||\mathcal{E}|K)$, which is in the same order as that of non-adaptive diffusions such as PPR and HK. For larger K however, an additional $\mathcal{O}(K^2N)$ is required per class, mainly to obtain $\mathbf{A}_c$ in (15).

Overall, if $\mathcal{O}(KN)$ 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.

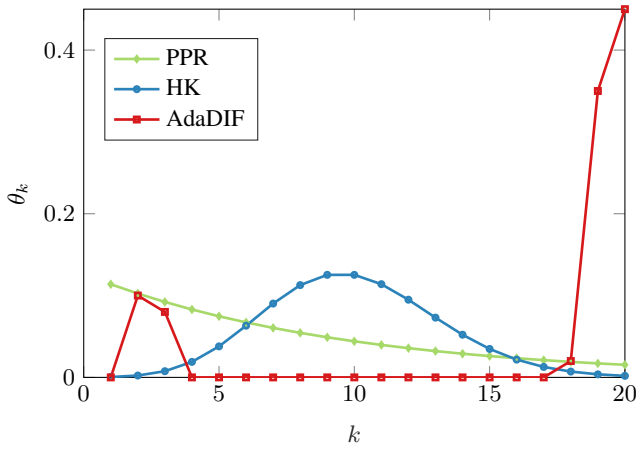

Fig. 2. Illustration of $K = 20$ landing probability coefficients for PPR with $\alpha = 0.9$, HK with $t = 10$, and AdaDIF ($\lambda = 15$).

B. 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 1 (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* $\boldsymbol{\gamma}$*-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 the Appendix B for a proof.

Theorem 1. *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

$$d_{\min +} := \min_{i \in \mathcal{L}_+} d_i, \quad d_{\min -} := \min_{j \in \mathcal{L}_-} d_j, \quad d_{\max} := \max_{i \in \mathcal{V}} d_i$$

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 1 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. In Appendix C we give a tighter threshold for PPR.

Conveniently, our experiments suggest that $K \in [10, 20]$ is usually sufficient to achieve high performance for most real graphs. See, for example, Fig. 3 for an experimental evaluation of K_γ for different values of γ-distinguishability threshold, and proportions of sampled nodes on the `BlogCatalog` graph. 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.

Remark 2. Note also that, while PPR and HK in theory rely on infinitely long random walks, the coefficients decay rapidly ($\theta_k = \alpha^k$ for PPR and $\theta_k = t^k/k!$ for HK). This means that not only $\theta_k \to 0$ as $k \to \infty$ in both cases, but the convergence rate is also very fast (especially for HK). This agrees with our intuition on random walks, as well as our result in Theorem 1 suggesting that, to guarantee a level of distinguishability (which is necessary for accuracy) between classes, classifiers should rely on relatively short-length random walks. Moreover, when operating in an adaptive framework such as the one proposed here, using finite-step (preferably short-length) landing probabilities is much more practical, since it restricts the number of free variables (θ_k's).

C. 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))

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

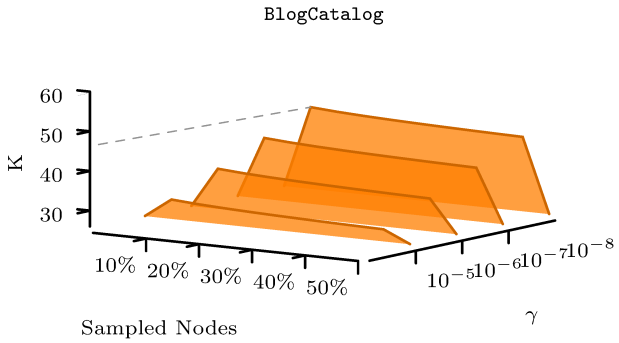

Fig. 3. Experimental evaluation K_γ for different values of γ-distinguishability threshold, and proportions of sampled nodes on `BlogCatalog` graph.

where $a_k(\boldsymbol{\theta}) := \sum_{d=1}^{D} \theta_d C_{kd}$, and $\mathbf{C} := \begin{bmatrix}\mathbf{c}_1 \cdots \mathbf{c}_D\end{bmatrix} \in \mathbb{R}^{K \times D}$ 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 2 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 $\mathbf{C} = \begin{bmatrix}\boldsymbol{\theta}_{\text{PPR}} & \boldsymbol{\theta}_{\text{HK}} & \cdots\end{bmatrix}$. 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 $|\mathcal{E}|$.

D. 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

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 $\in \{\text{True}, \text{False}\}$, Unconstrained $\in \{\text{True}, \text{False}\}$
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**
    $\mathbf{F}_c^{\Delta} = \text{DICTIONARY}\,(\mathbf{W}, \mathcal{L}_c, K, \mathbf{C})$
    Obtain $\mathbf{b}_c$ and $\mathbf{A}_c$ as in (17) and (18)
    $\mathbf{F}_c = \mathbf{F}_c^{\Delta}$
  else
    $\{\mathbf{P}_c^{(K)}, \tilde{\mathbf{P}}_c^{(K)}\} = \text{LANDPROB}(\mathbf{W}, \mathcal{L}_c, K)$
    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 (19) and (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 $\hat{y}_i = \arg\max_{c\in\mathcal{Y}} \left[\mathbf{f}_c(\hat{\boldsymbol{\theta}}_c)\right]_i, \quad \forall i \in \mathcal{U}$

Algorithm 2 LANDPROB

Input: $\mathbf{W}, \mathcal{L}_c, K$
Output: $\mathbf{P}_c^{(K)}, \quad \tilde{\mathbf{P}}_c^{(K)}$
$\mathbf{H} = \mathbf{W}\mathbf{D}^{-1}$
$\mathbf{p}_c^{(0)} = \mathbf{v}_c$
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

this non-negativity constraint, (12) can afford a closed-form solution as

$$\hat{\boldsymbol{\theta}}_c = \mathbf{A}_c^{-1}(\mathbf{b}_c - \lambda^*\mathbf{1}) \tag{19}$$

$$\lambda^* = \frac{\mathbf{1}^\mathsf{T}\mathbf{A}_c^{-1}\mathbf{b}_c - 1}{\mathbf{b}^\mathsf{T}\mathbf{A}_c^{-1}\mathbf{b}_c}. \tag{20}$$

Retaining the hyperplane constraint $\mathbf{1}^\mathsf{T}\boldsymbol{\theta} = 1$ 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. Towards the goal of developing more robust methods to design diffusions, the ensuing section presents our proposed approach that relies on minimizing the leave-one-out loss of the resulting classifier.

IV. ADAPTIVE DIFFUSIONS ROBUST TO ANOMALIES

Although the loss function in (10) is simple and easy to implement, it may lack robustness against nodes with labels that do not comply with a diffusion-based information propagation model. In real-world graphs, such 'difficult' nodes may arise due to model limitations, observation noise, or even deliberate mislabeling by adversaries. For such setups, this section introduces a novel adaptive diffusion classifier with: i) robustness in finding $\boldsymbol{\theta}$ by ignoring errors that arise due to outlying/anomalous nodes; as well as, ii) capability to identify and remove such 'difficult' nodes.

Let us begin by defining the following per-class $c \in \mathcal{Y}$ loss

$$\ell_{\text{rob}}^c(\mathbf{y}_{\mathcal{L}_c}, \boldsymbol{\theta}) := \sum_{i\in\mathcal{L}} \frac{1}{d_i}\left([\bar{\mathbf{y}}_{\mathcal{L}_c}]_i - [\mathbf{f}_c(\boldsymbol{\theta}; \mathcal{L}\setminus i)]_i\right)^2 \tag{21}$$

where $\mathbf{f}_c(\boldsymbol{\theta}; \mathcal{L}\setminus i)$ is the class-c diffusion after removing the i^{th} node from the set of all labels. Intuitively, (21) evaluates the ability of a propagation mechanism effected by $\boldsymbol{\theta}$ to predict the presence of class c label on each node $i \in \mathcal{L}$, using the remaining labeled nodes $\mathcal{L}\setminus i$. Since each class-specific distribution $\mathbf{f}_c(\boldsymbol{\theta})$ is constructed by random walks that are rooted in $\mathcal{L}_c \subseteq \mathcal{L}$, it follows that

$$\mathbf{f}_c(\boldsymbol{\theta}; \mathcal{L}\setminus i) = \begin{cases} \mathbf{f}_c(\boldsymbol{\theta}), & i \notin \mathcal{L}_c \\ \mathbf{f}_c(\boldsymbol{\theta}; \mathcal{L}_c \setminus i), & i \in \mathcal{L}_c \end{cases} \tag{22}$$

since $\mathbf{f}_c(\boldsymbol{\theta})$ is not directly affected by the removal of a label that belongs to other classes, and it is not used as a class-c seed. The class-c diffusion upon removing the i^{th} node from the seeds $\mathcal{L}_c$ is given as (cf. (4))

$$\mathbf{f}_c(\boldsymbol{\theta}; \mathcal{L}_c\setminus i) = \sum_{k=1}^{K}\theta_k\mathbf{p}_{\mathcal{L}_c\setminus i}^{(k)}$$

where $\mathbf{p}_{\mathcal{L}_c\setminus i}^{(k)} := \mathbf{H}^k\mathbf{v}_{\mathcal{L}_c\setminus i}$, and

$$[\mathbf{v}_{\mathcal{L}_c\setminus i}]_j = \begin{cases} 1/|\mathcal{L}_c\setminus i|, & j \in \mathcal{L}_c\setminus i \\ 0, & \text{else} \end{cases}. \tag{23}$$

The robust loss in (21) can be expressed more compactly as

$$\ell_{\text{rob}}^c(\mathbf{y}_{\mathcal{L}_c}, \boldsymbol{\theta}) := \|\mathbf{D}_{\mathcal{L}}^{-\frac{1}{2}}\left(\bar{\mathbf{y}}_{\mathcal{L}_c} - \mathbf{R}_c^{(K)}\boldsymbol{\theta}\right)\|_2^2 \tag{24}$$

where $\mathbf{D}_{\mathcal{L}}^{-\frac{1}{2}} := \left(\mathbf{D}_{\mathcal{L}}^{\dagger}\right)^{-\frac{1}{2}}$, and

$$\left[\mathbf{R}_c^{(K)}\right]_{ik} := \begin{cases} \left[\mathbf{p}_{\mathcal{L}_c\setminus i}^{(k)}\right]_i, & i \in \mathcal{L}_c \\ \left[\mathbf{p}_c^{(k)}\right]_i, & \text{else} \end{cases}. \tag{25}$$

Since $\mathbf{p}_c^{(k)} = |\mathcal{L}_c|^{-1}\sum_{i\in\mathcal{L}_c}\mathbf{p}_{\mathcal{L}_c\setminus i}^{(k)}$, evaluating (24) only requires the rows of $\mathbf{R}_c^{(K)}$ and entries of $\mathbf{y}_{\mathcal{L}_c}$ that correspond to $\mathcal{L}$, since the rest of the diagonal entries of $\mathbf{D}_{\mathcal{L}}^{\dagger}$ are 0. Having defined $\ell_{\rm rob}^c(\cdot)$, per-class diffusion coefficients $\hat{\boldsymbol{\theta}}_c$ can be obtained by solving

$$\hat{\boldsymbol{\theta}}_c = \arg\min_{\boldsymbol{\theta}\in\mathcal{S}^K} \ell_{\rm rob}^c(\mathbf{y}_{\mathcal{L}_c},\boldsymbol{\theta}) + \lambda_\theta\|\boldsymbol{\theta}\|_2^2 \tag{26}$$

where ℓ_2 regularization with parameter λ_θ is introduced in order to prevent overfitting and numerical instabilities. Note that, smoothness regularization in (11) is less appropriate in context of robustness, since it promotes "spreading" of the random walks (cf. Prop. 1), making class-diffusions more similar and increasing the difficulty of detecting outliers. Similar to (12), quadratic programming can be adopted to solve (26).

Towards mitigating the effects of outliers, and inspired by the robust estimators introduced in [19], we further enhance $\ell_{\rm rob}^c(\cdot)$ by explicitly modeling the effect of outliers with a sparse vector $\mathbf{o}\in\mathbb{R}^N$, leading to the modified cost

$$\ell_{\rm rob}^c(\mathbf{y}_{\mathcal{L}_c},\mathbf{o},\boldsymbol{\theta}) := \|\mathbf{D}_{\mathcal{L}}^{-\frac{1}{2}}\left(\mathbf{o}+\bar{\mathbf{y}}_{\mathcal{L}_c}-\mathbf{R}_c^{(K)}\boldsymbol{\theta}\right)\|_2^2. \tag{27}$$

The non-zero entries of $\mathbf{o}$ can capture large residuals (prediction errors $|[\bar{\mathbf{y}}_{\mathcal{L}_c}]_i - [\mathbf{f}_c(\boldsymbol{\theta};\mathcal{L}\setminus i)]_i|$) which may be the result of outlying, anomalous or mislabeled nodes. Thus, when operating in the presence of anomalies, the robust classifier aims at identifying both diffusion parameters $\{\hat{\boldsymbol{\theta}}_c\}_{c\in\mathcal{Y}}$ as well as per class outlier vectors $\{\hat{\mathbf{o}}_c\}_{c\in\mathcal{Y}}$. The two tasks can be performed jointly by solving the following optimization problem

$$\{\hat{\boldsymbol{\theta}}_c,\hat{\mathbf{o}}_c\}_{c\in\mathcal{Y}} = \arg\min_{\substack{\boldsymbol{\theta}_c\in\mathcal{S}^K\\ \mathbf{o}_c\in\mathbb{R}^N}} \sum_{c\in\mathcal{Y}}\left[\ell_{\rm rob}^c(\mathbf{y}_{\mathcal{L}_c},\mathbf{o}_c,\boldsymbol{\theta}_c)+\lambda_\theta\|\boldsymbol{\theta}_c\|_2^2\right] + \lambda_o\|\mathbf{D}_{\mathcal{L}}^{-\frac{1}{2}}\mathbf{O}\|_{2,1} \tag{28}$$

where $\mathbf{O} := \begin{bmatrix}\mathbf{o}_1 & \cdots & \mathbf{o}_{|\mathcal{Y}|}\end{bmatrix}$ concatenates the outlier vectors $\mathbf{o}_c$, and $\|\mathbf{X}\|_{2,1} := \sum_{i=1}^I\sqrt{\sum_{j=1}^J X_{i,j}^2}$ for any $\mathbf{X}\in\mathbb{R}^{I\times J}$. The term $\lambda_o\|\mathbf{D}_{\mathcal{L}}^{-\frac{1}{2}}\mathbf{O}\|_{2,1}$ in (28) acts as a regularizer that promotes sparsity over the rows of $\mathbf{O}$; it can also be interpreted as an ℓ_1-norm regularizer over a vector that contains the ℓ_2 norms of the rows of $\mathbf{O}$. The reason for using such block-sparse regularization is to force outlier vectors $\mathbf{o}_c$ of different classes to have the same support (pattern of non-zero entries). In other words, the $|\mathcal{Y}|$ different diffusion/outlier detectors are *forced* to consent on which nodes are outliers.

Since (28) is non-convex, convergence of gradient-descent-type methods to the global optimum is not guaranteed. Nevertheless, since (28) is biconvex (i.e., convex with respect to each variable) the following alternating minimization scheme

$$\hat{\mathbf{O}}^{(t)} = \arg\min_{\mathbf{O}} \sum_{c\in\mathcal{Y}}\left[\ell_{\rm rob}^c(\mathbf{y}_{\mathcal{L}_c},\mathbf{o}_c,\hat{\boldsymbol{\theta}}_c^{(t-1)})+\lambda_\theta\|\hat{\boldsymbol{\theta}}_c^{(t-1)}\|_2^2\right] + \lambda_o\|\mathbf{D}_{\mathcal{L}}^{-\frac{1}{2}}\mathbf{O}\|_{2,1} \tag{29}$$

$$\hat{\boldsymbol{\theta}}_c^{(t)} = \arg\min_{\boldsymbol{\theta}\in\mathcal{S}^K} \ell_{\rm rob}^c(\mathbf{y}_{\mathcal{L}_c},\hat{\mathbf{o}}_c^{(t)},\boldsymbol{\theta})+\lambda_\theta\|\boldsymbol{\theta}\|_2^2+\lambda_o\|\mathbf{D}_{\mathcal{L}}^{-\frac{1}{2}}\hat{\mathbf{O}}^{(t)}\|_{2,1} \tag{30}$$

with $\hat{\mathbf{O}}^{(0)} := [\mathbf{0}\ldots\mathbf{0}]$ converges to a partial optimum [16].

By further simplifying (30) and solving (29) in closed form, we obtain

$$\hat{\boldsymbol{\theta}}_c^{(t)} = \arg\min_{\boldsymbol{\theta}\in\mathcal{S}^K} \ell_{\rm rob}^c(\bar{\mathbf{y}}_{\mathcal{L}_c}+\hat{\mathbf{o}}_c^{(t-1)},\boldsymbol{\theta})+\lambda_\theta\|\boldsymbol{\theta}\|_2^2 \tag{31}$$

$$\hat{\mathbf{O}}^{(t)} = \mathrm{SoftThres}_{\lambda_o}\left(\tilde{\mathbf{Y}}^{(t)}\right) \tag{32}$$

where

$$\tilde{\mathbf{Y}}^{(t)} := \left[\tilde{\mathbf{y}}_1{}^{(t)},\ldots,\mathbf{y}_{|\mathcal{Y}|}^{(t)}\right]$$

is the matrix that concatenates the per class residual vectors $\tilde{\mathbf{y}}_c^{(t)} := \bar{\mathbf{y}}_{\mathcal{L}_c} - \mathbf{R}_c^{(K)}\hat{\boldsymbol{\theta}}_c^{(t)}$, and $\mathbf{Z} = \mathrm{SoftThres}_{\lambda_o}(\mathbf{X})$ is a row-wise soft-thresholding operator such that

$$\mathbf{z}_i = \|\mathbf{x}_i\|_2[1-\lambda_o/(2\|\mathbf{x}_i\|_2)]_+$$

where $\mathbf{z}_i$ and $\mathbf{x}_i$ are the i^{th} rows of $\mathbf{Z}$ and $\mathbf{X}$ respectively, see e.g. [34]. Intuitively, the soft-thresholding operation in (32) extracts the outliers by scaling down residuals and "trimming" them wherever their across-classes ℓ_2 norm is below a certain threshold.

The alternating minimization between (31) and (32) terminates when $\|\hat{\boldsymbol{\theta}}_c^{(t)}-\hat{\boldsymbol{\theta}}_c^{(t-1)}\|_\infty \le \epsilon,\ \forall c\in\mathcal{Y}$, where $\epsilon\ge 0$ is a prescribed tolerance. Having obtained the tuples $\{\hat{\boldsymbol{\theta}}_c,\hat{\mathbf{o}}_c\}_{c\in\mathcal{Y}}$, one may remove the anomalous samples that correspond to non-zero rows of $\hat{\mathbf{O}}$ and perform the diffusion with the remaining samples. The robust (r) AdaDIF is summarized as Algorithm 4, and has $\mathcal{O}(K|\mathcal{L}||\mathcal{E}|)$ computational complexity.

V. Contributions in Context of Prior Works

Following the seminal contribution in [8] 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 [25], [15]. Most formal approaches to generalize PageRank focus either on the *teleportation* component (see e.g. [31], [32] as well as [7] for an application to semi-supervised classification), or, on the so-termed *damping* mechanism [12], [4]. 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

Algorithm 4 ROBUST ADAPTIVE DIFFUSIONS

Input: Adjacency matrix: $\mathbf{W}$, Labeled nodes: $\{y_i\}_{i\in\mathcal{L}}$
***parameters*:** Regularization parameters: $\lambda_\theta, \lambda_o$, # of landing probabilities: K
Output: Predictions: $\{\hat{y}_i\}_{i\in\mathcal{U}}$
Outliers: $\underset{c\in\mathcal{Y}}{\cup}\mathcal{L}_c^o$
Extract $\mathcal{Y} = \{$ Set of unique labels in: $\{y_i\}_{i\in\mathcal{L}}\}$
for $c \in \mathcal{Y}$ **do**
  $\mathcal{L}_c = \{i \ \in \mathcal{L} : y_i = c\}$
  for $i \in \mathcal{L}_c$ **do**
    $\{\mathbf{p}^{(k)}_{\mathcal{L}_c\setminus i}\}_{k=1}^K = \text{LANDPROB}(\mathbf{W}, \mathcal{L}_c \setminus i, K)$
  end for
  Obtain $\mathbf{R}_c^{(K)}$ as in (25)
end for
$\hat{\mathbf{O}}^{(0)} = [\mathbf{0}, \ldots, \mathbf{0}], t = 0$
while $\|\hat{\boldsymbol{\theta}}_c^{(t)} - \hat{\boldsymbol{\theta}}_c^{(t-1)}\|_\infty \leq \epsilon$ **do**
  $t \leftarrow t + 1$
  Obtain $\{\hat{\boldsymbol{\theta}}_c^{(t)}\}_{c\in\mathcal{Y}}$ as in (31)
  Obtain $\hat{\mathbf{O}}^{(t)}$ as in (32)
end while
Set of outliers: $\mathcal{S} := \{i \in \mathcal{L} : \|[\hat{\mathbf{O}}]_{i,:}\|_2 > 0\}$
for $c \in \mathcal{Y}$ **do**
  $\mathcal{L}_c^o = \mathcal{L}_c \cap \mathcal{S}$
  $\mathcal{L}_c \leftarrow \mathcal{L}_c \setminus \mathcal{L}_c^o$
end for
Obtain $\hat{y}_i = \arg\max_{c\in\mathcal{Y}} \left[\mathbf{f}_c(\hat{\boldsymbol{\theta}}_c)\right]_i, \quad \forall i \in \mathcal{U}$

functional rankings, tailored to the underlying semi-supervised classification task. A related method termed AptRank was recently proposed in [45] specifically for protein function prediction. Differently from AdaDIF, AptRank splits the data into training and validation sets of predetermined proportions and adopt as cross-validation approach for obtaining diffusion coefficients. There are two additional differences with AptRank: a) AptRank trains a single diffusion for all classes whereas AdaDIF identifies different diffusions, and b) the proposed robust leave-one-out method (r-AdaDIF) gathers residuals from all leave-one-out splits into one cost function (cf. (21)) and then optimizes the (per class) diffusion.

Recently, community detection (CD) methods were proposed in [46] and [44], that search the Krylov subspace of landing probabilities of a given community's seeds, to identify a diffusion that satisfies *locality* of non-zero entries over the graph. In CD, the problem definition is: "given certain members of a community, identify the remaining (latent) members." There is a subtle but important distinction between CD and semi-supervised classification (SSC): CD focuses on the retrieval of *communities* (i.e., nodes of a given class), whereas SSC focuses on the predicting the labels/attributes of every *node*. While CD treats the detection of various overlapping communities of the graph as independent tasks, SSC classifies nodes by taking all information from labeled nodes into account. More specifically, the proposed AdaDIF trains the diffusion of each class by actively avoiding the assignment of large diffusion values to nodes that are known (they have been labeled) to belong to a different class. Another important difference of AdaDIF with [46] and [44]—which again arises from the different contexts—is the length of the walk compared to the size of the graph. Since [46] and [44] aim at identifying small and local communities, they perform local walks of length smaller than the diameter of the graph. In contrast, SSC typically demands a certain degree of globality in information exchange, achieved by longer random walks that surpass the graph diameter.

AdaDIF also shares links with SSL methods based on graph signal processing proposed in [36], and further pursued in [11] for bridge monitoring; see also [37] and [13] 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, including a novel approach for training the model in an anomaly/outlier-resilient manner. Furthermore, while [36] focuses on binary classification and [11] 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 [36] the weighting parameters remain unconstrained and in [11] 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 I
NETWORK CHARACTERISTICS

Graph	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{E}\rvert$	$\lvert\mathcal{Y}\rvert$	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

VI. 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 I. 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

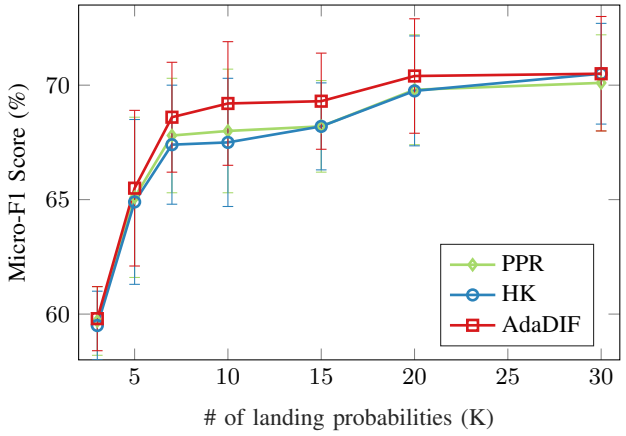

Fig. 4. Micro-F1 score for AdaDIF and non-adaptive diffusions on 5% labeled Cora graph as a function of the length of underline random walks.

in Section II; ii) Label propagation (LP) [42]; ii) Node2vec [17]; iii) Deepwalk [33]; iv) Planetoid-G [41]; 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 $t \in [1.0, 5.0, 10.0, 15.0]$. 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 for 50 steps for convergence to be in effect; see Fig 4; LP was also run for 50 steps. For Node2vec, we fixed most parameters to the values suggested in [17], 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 [17], [33], 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 (cf. Fig. 4); 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 $t \in [5, 8, 12, 15, 20]$, and the other five polynomial coefficients $c_i = k^\beta$ with $\beta \in [2, 4, 6, 8, 10]$.

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, class-balanced sampling, and predicted the labels of the remaining nodes. Thus, instead of sampling nodes over the graph uniformly at random, we randomly sample a given number of nodes per class. For each graph, we performed 20 experiments, each time sampling $5, 10$, and 20 nodes per class. For each experiment, classification accuracy was measured on the unlabeled nodes in terms of Micro-F1 and Macro-F1 scores; see e.g., [29]. The results were averaged over 20 experiments, with mean and standard deviation reported in Table II. 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 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 is reflected on the runtime comparisons listed in Fig. 7. All experiments were run on a machine with i5 @3.50 Mhz CPU, and 16GB of RAM. We used the Python implementations provided by the authors of the compared algorithms. The Python implementation of AdaDIF, uses only tools provided by scipy, numpy, and CVXOPT libraries. We also developped an efficient implementation that exploits parallelism, which is straightforward since each class can be treated separately. Note that, while AdaDIF has (as expected) a relatively small computational overhead over fixed diffusions, it is faster than GCNs that use Tensorflow, and orders of magnitude faster than embedding-based approaches.

Finally, Table III 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 [17] and [33], 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. It is worth noting that, for mutlilabel graphs with many classes, the performance boost over fixed diffusions can probably be largely attributed to the fact that AdaDIF is free to treat each class differently. To demonstrate that different classes are indeed diffused in a markedly different manner, Fig. 6 plots all 50 diffusion coefficient vectors $\{\boldsymbol{\theta}_c\}_{c\in\mathcal{C}}$ yielded by AdaDIF on the PPI graph with 30% of nodes labeled. Each line in the plot corresponds to the values of $\boldsymbol{\theta}_c$ for a different c; evidently there is large diversity among classes.

A. Analysis/interpretation of results

Here we will follow an experimental approach that is aimed at understanding and interpreting our results. We now solely focus on diffusion-based classiefiers, additionaly using a simple benchmark for diffusion-based classification: the $k-$step landing probabilities. Specifically, we compare the classification accuracy on the three multiclass datasets of AdaDIF, PPR, and HK, with the accuracy of the classifier that uses only the $k-$th landing probability vectors $\{\mathbf{p}_c^{(k)}\}_{c\in\mathcal{Y},k\in[1,K]}$. The setting was similar to the one in the previous section, and with class-balanced sampling of 20 nodes per class, while the $k-$step classifiers were examined for a wide range of steps $k \in [1, 100]$. The $k-$step classifier reveals the predictive power of individual landing probabilities, resulting in curves (see Fig. 5) that appear

TABLE II
MICRO F1 AND MACRO F1 SCORES ON MULTICLASS NETWORKS (CLASS-BALANCED SAMPLING)

	Graph	Cora			Citeseer			PubMed		
	$\lvert\mathcal{L}_c\rvert$	5	10	20	5	10	20	5	10	20
Micro-F1	AdaDIF	$\mathbf{67.5 \pm 2.2}$	$\mathbf{71.0 \pm 2.0}$	$\mathbf{73.2 \pm 1.2}$	$\mathbf{42.3 \pm 4.4}$	$\mathbf{49.5 \pm 3.0}$	$\mathbf{53.5 \pm 1.2}$	62.0 ± 6.0	68.5 ± 4.5	$\mathbf{74.1 \pm 1.7}$
	PPR	67.1 ± 2.3	70.2 ± 2.1	72.8 ± 1.5	41.1 ± 5.2	48.7 ± 2.5	52.5 ± 0.9	$\mathbf{63.1 \pm 1.1}$	$\mathbf{69.5 \pm 3.8}$	$\mathbf{74.1 \pm 1.8}$
	HK	67.0 ± 2.5	70.5 ± 2.5	72.9 ± 1.2	40.0 ± 5.6	48.0 ± 2.4	51.8 ± 1.1	62.0 ± 0.6	68.3 ± 4.7	$\mathbf{74.0 \pm 1.8}$
	LP	61.8 ± 3.5	66.3 ± 4.2	71.0 ± 2.7	40.7 ± 2.5	48.0 ± 3.7	51.9 ± 1.3	56.2 ± 11.0	68.0 ± 6.1	69.3 ± 2.4
	Node2vec	68.9 ± 1.9	70.2 ± 1.6	72.4 ± 1.2	39.2 ± 3.7	46.5 ± 2.4	51.0 ± 1.4	61.7 ± 13.0	66.4 ± 4.6	71.1 ± 2.4
	Deepwalk	68.4 ± 2.0	70.0 ± 1.6	72.0 ± 1.4	38.4 ± 3.9	45.5 ± 2.0	50.4 ± 1.5	61.5 ± 1.3	65.8 ± 5.0	70.5 ± 2.2
	Planetoid-G	63.5 ± 4.7	65.6 ± 2.7	69.0 ± 1.5	37.8 ± 4.0	44.9 ± 3.3	49.8 ± 1.4	60.7 ± 2.0	63.4 ± 2.3	68.0 ± 1.5
	GCN	60.1 ± 3.7	65.5 ± 2.5	68.6 ± 1.9	38.3 ± 3.2	44.2 ± 2.2	48.0 ± 1.8	60.0 ± 1.9	63.6 ± 2.5	70.5 ± 1.5
Macro-F1	AdaDIF	$\mathbf{65.5 \pm 2.5}$	$\mathbf{70.6 \pm 2.2}$	$\mathbf{72.0 \pm 1.1}$	$\mathbf{36.1 \pm 3.9}$	$\mathbf{44.0 \pm 2.8}$	$\mathbf{48.1 \pm 1.2}$	60.4 ± 0.6	67.0 ± 4.4	$\mathbf{72.6 \pm 1.8}$
	PPR	65.0 ± 2.3	70.0 ± 2.3	71.9 ± 1.5	34.7 ± 5.0	43.5 ± 2.3	47.6 ± 0.6	$\mathbf{61.7 \pm 0.6}$	$\mathbf{68.1 \pm 3.6}$	$\mathbf{72.6 \pm 1.8}$
	HK	65.0 ± 2.5	70.0 ± 2.6	$\mathbf{72.0 \pm 1.1}$	33.9 ± 5.4	42.8 ± 2.2	47.0 ± 0.6	60.5 ± 0.6	66.8 ± 4.7	$\mathbf{72.7 \pm 1.8}$
	LP	60.1 ± 3.2	66.5 ± 4.1	70.6 ± 2.3	34.8 ± 4.6	41.8 ± 3.9	51.5 ± 1.2	51.5 ± 12.3	66.2 ± 6.6	67.8 ± 2.0
	Node2vec	62.4 ± 2.0	64.7 ± 1.7	69.2 ± 1.2	34.6 ± 2.7	41.6 ± 1.9	45.3 ± 1.5	59.5 ± 1.2	64.0 ± 3.8	72.3 ± 1.4
	Deepwalk	61.8 ± 2.2	64.5 ± 2.0	68.5 ± 1.4	34.0 ± 2.5	41.0 ± 2.0	44.7 ± 1.8	59.3 ± 1.2	63.8 ± 4.0	72.1 ± 1.3
	Planetoid-G	59.9 ± 4.5	63.0 ± 3.0	68.7 ± 1.9	33.3 ± 2.5	40.2 ± 2.2	43.6 ± 2.0	57.7 ± 1.5	61.9 ± 3.5	66.1 ± 1.8
	GCN	53.8 ± 6.6	61.9 ± 2.6	63.8 ± 1.5	32.8 ± 2.0	39.1 ± 1.8	43.0 ± 1.7	54.4 ± 4.1	57.2 ± 5.2	60.5 ± 2.4

TABLE III
MICRO F1 AND MACRO F1 SCORES OF VARIOUS ALGORITHMS ON MULTILABEL NETWORKS

	Graph	PPI			BlogCatalog			Wikipedia		
	$\lvert\mathcal{L}\rvert/\lvert\mathcal{V}\rvert$	10%	20%	30%	10%	20%	30%	10%	20%	30%
Micro-F1	AdaDIF	15.4 ± 0.5	17.9 ± 0.7	$\mathbf{19.2 \pm 0.6}$	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	$\mathbf{16.5 \pm 0.6}$	$\mathbf{18.2 \pm 0.3}$	19.1 ± 0.3	$\mathbf{35.0 \pm 0.3}$	$\mathbf{36.3 \pm 0.3}$	$\mathbf{37.2 \pm 0.2}$	$\mathbf{42.3 \pm 0.9}$	$\mathbf{44.0 \pm 0.6}$	$\mathbf{45.1 \pm 0.4}$
	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	$\mathbf{13.4 \pm 0.6}$	$\mathbf{15.4 \pm 0.7}$	$\mathbf{16.5 \pm 0.7}$	$\mathbf{23.0 \pm 0.6}$	$\mathbf{25.3 \pm 0.4}$	$\mathbf{27.0 \pm 0.4}$	$\mathbf{7.7 \pm 0.3}$	$\mathbf{8.3 \pm 0.3}$	$\mathbf{9.0 \pm 0.2}$
	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	$\mathbf{13.4 \pm 0.6}$	$\mathbf{15.4 \pm 0.5}$	$\mathbf{16.5 \pm 0.4}$	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

to be different for each network, characterizing the graph-label distribution relationship of the latter. For `Cora` graph (left two plots), performance of the $k-$step classifier improves sharply after the first few steps, peaks for $k \approx 20$, and then quickly degrades, suggesting that using the landing probabilities of $k > 40$ or 50 would most likely degrade the performance of a diffusion-based classifier. Interestingly, AdaDIF relying on combinations of the first 15 steps, and PPR and HK of the first 50, all achieve higher accuracy than that of the best single step. On the other hand, `Citeseer` graph (middle two plots) behaves in a significantly different manner, with the $k-$step classifier requiring longer walks to reach high accuracy that was retained for much longer. Furthermore, accumulating landing probabilities the way PPR or HK does yields lower Micro-F1 accuracy than that of the single best step. On the other hand, by smartly combining the first 15 steps that are of lower quality, AdaDIF surpasses the Micro-F1 scores of the longer walks. Interestingly, the Macro-F1 metric for `Citeseer` behaves differently than the Micro-F1, and quickly decreases after ~ 25 steps. The disagreement between the two metrics can be explained as the diffusions of one or more of the larger classes gradually "overwhelms" those of one or more smaller classes, thus lowering the Macro-F1 score, since the latter is a metric that averages *per-class*. In contrast, the Micro-F1 metric averages *per-node* and takes much less of an impact if a few nodes from the smaller classes are mislabeled. Finally, for `PubMed` graph (right two plots), steps in the range $[20, 40]$ yield *consistently* high accuracy both in terms of Micro- as well as Macro-averaged F1-score. Since HK and mostly PPR largely accumulate steps in that range, it seems reasonable that both fixed diffusions are fairly accurate in `PubMed` graph.

B. Tests on simulated label-corruption setup

Here we outline experimental results performed to evaluate the performance of different diffusion-based classifiers in the presence of anomalous nodes. The main goal is to evaluate whether r-AdaDIF (Algorithm 4) yields improved performance over AdaDIF, HK and PPR, as well as the ability of r-AdaDIF to detect anomalous nodes. We also tested a different type of rounding from class-diffusions to class labels that was shown in [43] to be robust in the presence of erroneous labels on a graph constructed by images of handwritten digits. The idea is to first normalize diffusions with node degrees, sort each diffusion vector, and assign to each node the class for which the corresponding rank is higher. We applied this type of rounding on PPR diffusions (denoted as PPR w. ranking). Since a ground truth set of anomalous nodes is not available in real graphs, we chose to infuse the true labels with artificial anomalies generated by the following simulated label corruption

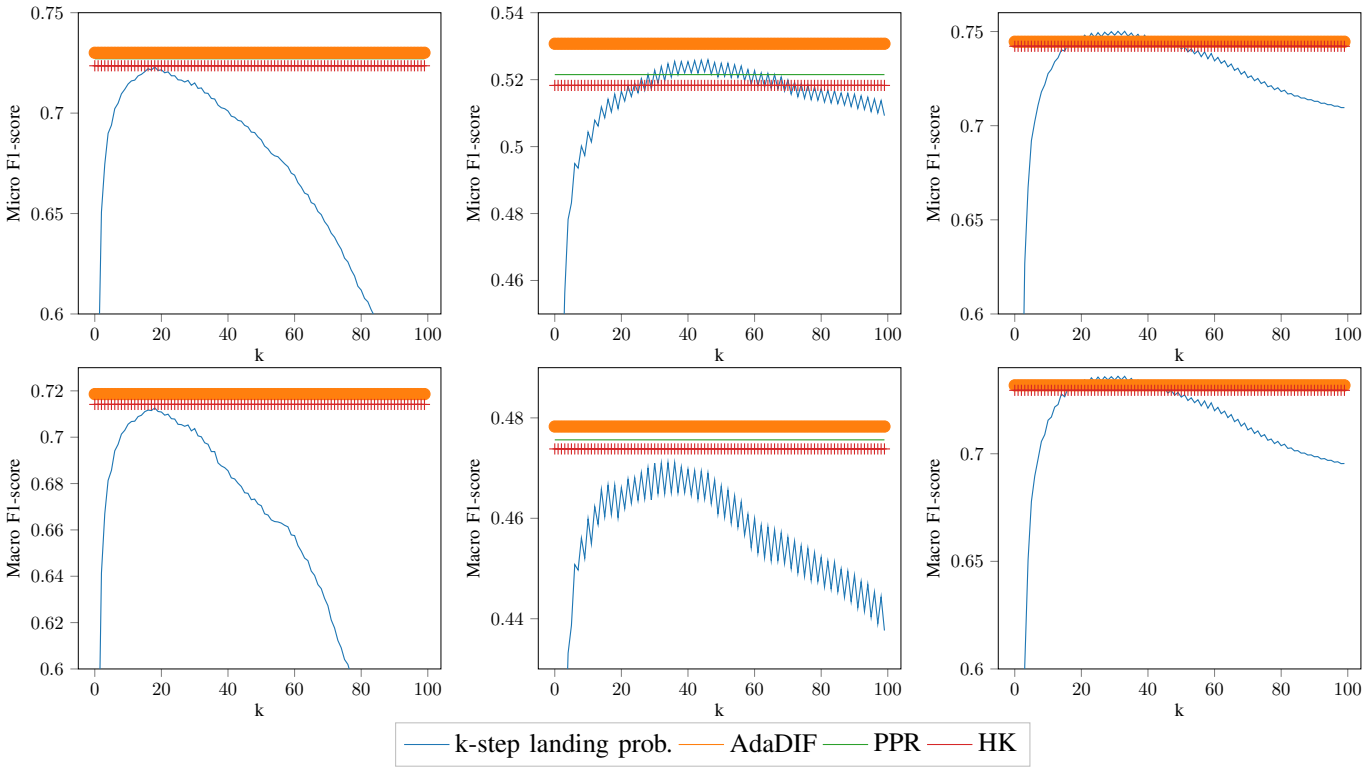

Fig. 5. Classification accuracy of AdaDIF, PPR, and HK compared to the accuracy of $k-$step landing probability classifier. **Top Left)** `Cora` Micro-F1 score; **Bottom Left)** `Cora` Macro-F1 score; **Top Middle)** `Citeseer` Micro-F1 score; **Bottom Middle)** `Citeseer` Macro-F1 score; **Top Right)** `PubMed` Micro-F1 score; **Bottom Right)** `PubMed` Macro-F1 score

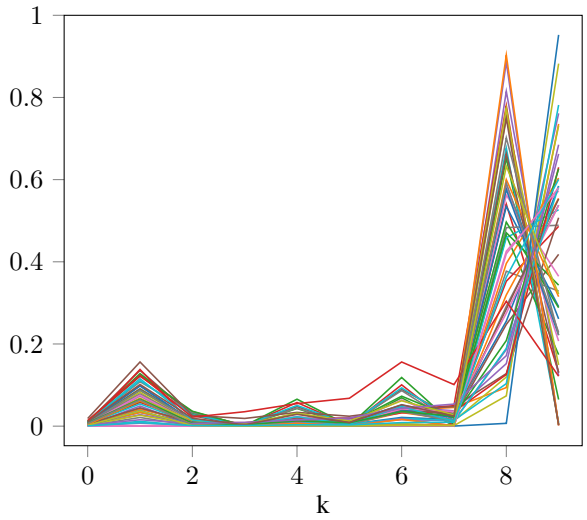

Fig. 6. AdaDIF diffusion coefficients for the 50 different classes of PPI graph (30% sampled). Each line corresponds to a different $\boldsymbol{\theta}_c$. Diffusion is characterized by high diversity among classes.

process: Go through $\mathbf{y}_{\mathcal{L}}$ and for each entry $[\mathbf{y}_{\mathcal{L}}]_i = c$ draw with probability p_{cor} a label $c' \sim \text{Unif}\{\mathcal{Y} \setminus c\}$; and replace $[\mathbf{y}_{\mathcal{L}}]_i \leftarrow c'$. In other words, anomalies are created by corrupting some of the true labels by randomly and uniformly "flipping" them to a different label. Increasing the corruption probability p_{cor} of the training labels $\mathbf{y}_{\mathcal{L}}$ is expected to have increasingly negative impact on classification accuracy over $\mathbf{y}_{\mathcal{U}}$. Indeed, as depicted in Fig. 8, the accuracy of all diffusion-based classifiers on `Cora`

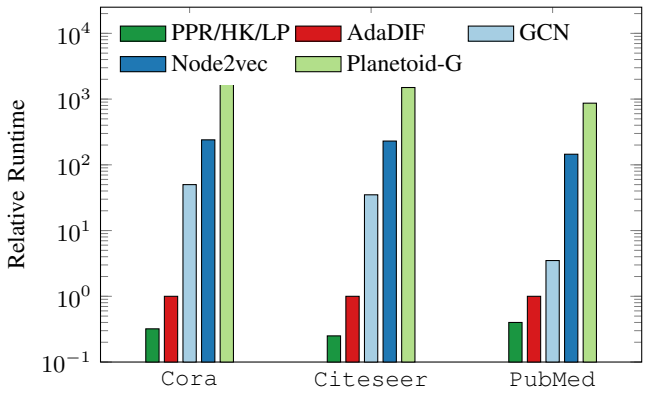

Fig. 7. Relative runtime comparisons for multiclass graphs.

graph degrades as p_{cor} increases. All diffusions were run for $K = 50$, while for r-AdaDIF we found $\lambda_o = 14.6 \times 10^{-3}$ and $\lambda_\theta = 67.5 \times 10^{-5}$ to perform well for moderate values of p_{cor}. Results were averaged over 50 Monte Carlo experiments, and for each experiment 5% of the nodes were sampled uniformly at random. While tuning λ_o for a specific p_{cor} generally yields improved results, we use the same λ_o across the range of p_{cor} values, since the true value of the latter is generally not available in practice. In this setup, r-AdaDIF demonstrates higher accuracy compared to non-robust classifiers. Moreover, the performance gap increases as more labels become corrupted, until it reaches a "break point" at $p_{\text{cor}} \approx 0.35$. Interestingly, r-AdaDIF performs worse in the absence of anomalies ($p_{\text{cor}} = 0$) that can be attributed to the fact that it only removes useful

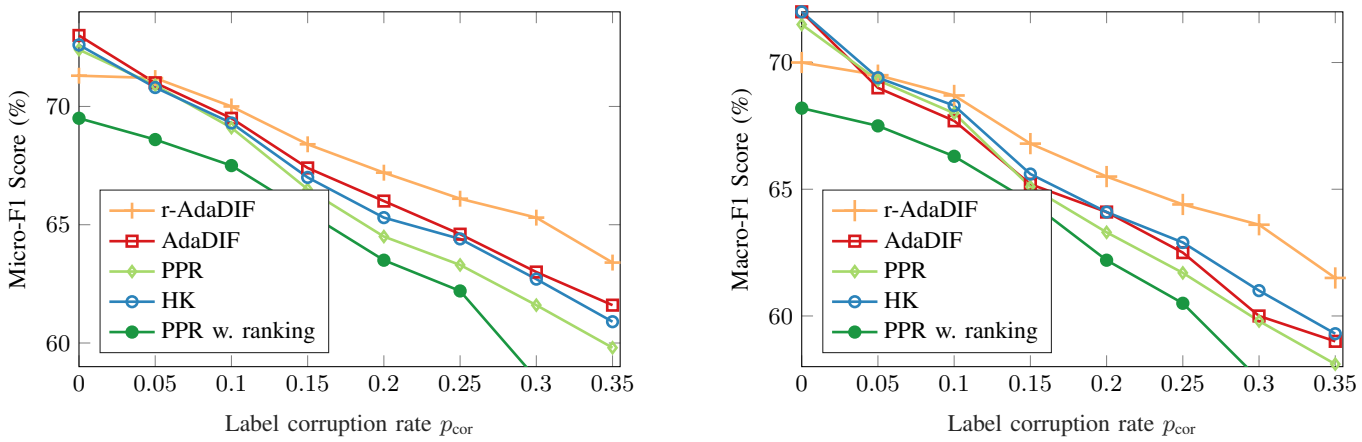

Fig. 8. Classification accuracy of various diffusion-based classifiers on Cora, as a function of the probability of label corruption.

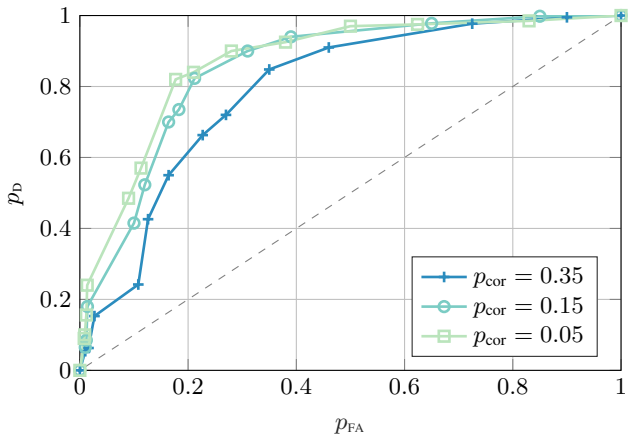

Fig. 9. Anomaly detection performance of r-AdaDIF for different label corruption probabilities. The horizontal axis corresponds to the frequency with which r-AdaDIF returns a true positive (probability of detection) and the vertical axis corresponds to the frequency of false positives (probability of false alarm).

samples and thus reduces the training set. We also observed that, although PPR w. ranking displays relative robustness as p_{cor} increases, overall it performs worse than PPR with value based rounding, at least on `Cora` graph.

As mentioned earlier, the performance of r-AdaDIF in terms of outlier detection depends on parameter λ_o. Specifically, for $\lambda_o \to 0$ the regularizer in (28) is effectively removed and all samples are characterized as outliers. On the other hand, for $\lambda_o \gg 1$ (28) yields $\hat{\mathbf{O}} = [\mathbf{0}, \ldots, \mathbf{0}]$, meaning that no outliers are unveiled. For intermediate values of λ_o, r-AdaDIF trades off falsely identifying nominal samples as outliers (false alarm) with correctly identifying anomalies (correct detection). This tradeoff of r-AdaDIF's anomaly detection behavior was experimentally evaluated over 50 Monte Carlo runs by sweeping over a large range of values of λ_o, and for different values of p_{cor}; see the probability of detection (p_{D}) versus probability of false alarms (p_{FA}) depicted in Fig. 9. Evidently, r-AdaDIF performs much better than a random guess ("coin toss") detector whose curve is given by the grey dotted line, while the detection performance improves as the corruption rate decreases.

VII. 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 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.

Appendix

A. Proof of Proposition 1

For $\lambda \to \infty$, the effect of $\ell(\cdot)$ in (9) vanishes, and the optimization problem becomes equivalent to solving

$$\min_{\boldsymbol{\theta} \in \mathcal{S}^K} \boldsymbol{\theta}^{\mathsf{T}} \mathbf{A} \boldsymbol{\theta} \tag{33}$$

where $\mathbf{A} := (\mathbf{P}_c^{(K)})^{\mathsf{T}} \mathbf{D}^{-1} \mathbf{L} \mathbf{D}^{-1} \mathbf{P}_c^{(K)}$ has (i,j) entry given by $A_{ij} = (\mathbf{p}_c^{(i)})^{\mathsf{T}} \mathbf{D}^{-1} \mathbf{L} \mathbf{D}^{-1} \mathbf{p}_c^{(j)}$; and $\mathbf{p}_c^{(K)}$ is the vector of K-step landing probabilities with initial distribution $\mathbf{v}_c$ and transition matrix $\mathbf{H} = \sum_{n=1}^{N} \lambda_n \mathbf{u}_n \mathbf{v}_n^{\mathsf{T}}$, where $\lambda_1 > \lambda_2 > \cdots > \lambda_N$ are its eigenvalues. Since $\mathbf{H}$ is a column-stochastic transition probability matrix, it holds that $\lambda_1 = 1$, $\mathbf{v} = \mathbf{1}$, and $\mathbf{u}_1 = \boldsymbol{\pi}$, where $\boldsymbol{\pi} = \lim_{k \to \infty} \mathbf{p}_c^{(k)}$ is the steady-state

distribution that can be also expressed as $\boldsymbol{\pi} = \mathbf{d}/(2|\mathcal{E}|)$ [26]. The landing probability vector for class c is thus

$$\begin{aligned}\mathbf{p}_c^{(K)} &= \mathbf{H}^K\mathbf{v}_c = \left[\frac{1}{2|\mathcal{E}|}\mathbf{d}\mathbf{1}^\mathsf{T} + \sum_{n=2}^{N}\lambda_n^K\mathbf{u}_n\mathbf{v}_n^\mathsf{T}\right]\mathbf{v}_c \\ &= \frac{1}{2|\mathcal{E}|}\mathbf{d} + \sum_{n=2}^{N}\lambda_n^K\mathbf{u}_n\gamma_n \approx \frac{1}{2|\mathcal{E}|}\mathbf{d} + \lambda_2^K\mathbf{u}_2\gamma_2 \qquad (34)\end{aligned}$$

where $\gamma_n := \mathbf{v}_n^\mathsf{T}\mathbf{v}_c$, and the approximation in (34) holds because $\lambda_2^K \gg \lambda_n^K$, for $n \in [3, N]$, and K large enough but finite. Using (34), A_{ij} can be rewritten as

$$\begin{aligned}A_{ij} =& \left[\frac{1}{2|\mathcal{E}|}\mathbf{d}^\mathsf{T} + \lambda_2^i\mathbf{u}_2^\mathsf{T}\gamma_2\right]\mathbf{D}^{-1}\mathbf{L}\mathbf{D}^{-1}\left[\frac{1}{2|\mathcal{E}|}\mathbf{d} + \lambda_2^j\mathbf{u}_2\gamma_2\right] \\ =& \left[\frac{1}{2|\mathcal{E}|}\mathbf{1}^\mathsf{T} + \lambda_2^i\mathbf{u}_2^\mathsf{T}\mathbf{D}^{-1}\gamma_2\right]\mathbf{L}\left[\frac{1}{2|\mathcal{E}|}\mathbf{1} + \lambda_2^j\mathbf{D}^{-1}\mathbf{u}_2\gamma_2\right] \\ =& \frac{1}{4|\mathcal{E}|^2}\mathbf{1}^\mathsf{T}\mathbf{L}\mathbf{1} + \frac{\lambda_2^i\gamma_2}{2|\mathcal{E}|}\mathbf{u}_2^\mathsf{T}\mathbf{D}^{-1}\mathbf{L}\mathbf{1} + \frac{\lambda_2^j\gamma_2}{2|\mathcal{E}|}\mathbf{1}^\mathsf{T}\mathbf{L}\mathbf{D}^{-1}\mathbf{u}_2 \\ &+ \gamma_2^2\lambda_2^{i+j}\mathbf{u}_2^\mathsf{T}\mathbf{D}^{-1}\mathbf{L}\mathbf{D}^{-1}\mathbf{u}_2 \\ =& C\lambda_2^{i+j} \qquad (35)\end{aligned}$$

where $C := \gamma_2^2\mathbf{u}_2^\mathsf{T}\mathbf{D}^{-1}\mathbf{L}\mathbf{D}^{-1}\mathbf{u}_2$, the second equality uses $\mathbf{D}^{-1}\mathbf{d} = \mathbf{1}$, and the last equality follows because $\mathbf{L}\mathbf{1} = \mathbf{0}$. Using (35), one obtains $\mathbf{A} = C\boldsymbol{\lambda}_2\boldsymbol{\lambda}_2^\mathsf{T}$, where $\boldsymbol{\lambda}_2 := \begin{bmatrix}\lambda_2 & \lambda_2^2 & \cdots & \lambda_2^K\end{bmatrix}^\mathsf{T}$, while (33) reduces to

$$\min_{\boldsymbol{\theta}\in\mathcal{S}^K}\left(\boldsymbol{\lambda}_2^\mathsf{T}\boldsymbol{\theta}\right)^2. \qquad (36)$$

Since $\boldsymbol{\lambda}_2^\mathsf{T}\boldsymbol{\theta} > 0 \;\; \forall\boldsymbol{\theta} \in \mathcal{S}^K$, it can be shown that the KKT optimality conditions for (36) are identical to those of

$$\min_{\boldsymbol{\theta}\in\mathcal{S}^K}\boldsymbol{\lambda}_2^\mathsf{T}\boldsymbol{\theta}. \qquad (37)$$

Therefore, (36) admits minimizer(s) identical to (37). Finally, we will show that the minimizer of (37) is $\mathbf{e}_K$. Since the problem is convex, it suffices to show that $\nabla_{\boldsymbol{\theta}}^\mathsf{T}(\boldsymbol{\lambda}_2^\mathsf{T}\boldsymbol{\theta})_{\boldsymbol{\theta}=\mathbf{e}_K}(\boldsymbol{\theta} - \mathbf{e}_K) \geq 0 \;\; \forall\boldsymbol{\theta} \in \mathcal{S}^K$, or, equivalently

$$\begin{aligned}\boldsymbol{\lambda}_2^\mathsf{T}(\boldsymbol{\theta} - \mathbf{e}_K) \geq 0 &\Leftrightarrow \sum_{k=1}^{K}\theta_k\lambda_2^k - \lambda_2^K \geq 0 \\ &\Leftrightarrow \sum_{k=1}^{K}\theta_k\lambda_2^{k-K} \geq 1 \\ &\Leftrightarrow \sum_{k=1}^{K}\theta_k\lambda_2^{k-K} \geq \sum_{k=1}^{K}\theta_k \\ &\Leftrightarrow \sum_{k=1}^{K}\theta_k\left(\lambda_2^{k-K} - 1\right) \geq 0\end{aligned}$$

which holds since $\boldsymbol{\theta} \geq \mathbf{0}$ and $\lambda_2^{k-K} \geq 1 \;\; \forall k \in [1, K]$, and completes the proof of the proposition.

B. Proof of Theorem 1

We need to find the smallest integer K such that $\max_{\boldsymbol{\theta}\in\mathcal{S}^K}\|\mathbf{y} - \check{\mathbf{y}}\| \leq \gamma$. We have

$$\begin{aligned}\|\mathbf{y} - \check{\mathbf{y}}\| &= \|\mathbf{X}_+\boldsymbol{\theta} - \mathbf{X}_-\boldsymbol{\theta} - \check{\mathbf{X}}_+\boldsymbol{\theta} + \check{\mathbf{X}}_-\boldsymbol{\theta}\| \leq \\ &\leq \|\theta_K\mathbf{p}_+^{(K)} - \theta_K\mathbf{p}_-^{(K)}\| + \|\theta_K\mathbf{p}_+^{(K+1)} - \theta_K\mathbf{p}_-^{(K+1)}\| \\ &\leq \|\mathbf{H}^K\mathbf{p}_+ - \mathbf{H}^K\mathbf{p}_-\| + \|\mathbf{H}^{K+1}\mathbf{p}_+ - \mathbf{H}^{K+1}\mathbf{p}_-\| \qquad (38)\end{aligned}$$

since $\boldsymbol{\theta} \in \mathcal{S}^K$. Therefore, to determine an upper bound for the γ-distinguishability threshold it suffices to find the smallest integer K for which (38) is upper bounded by γ.

Let $\mathbf{q}_1, \ldots, \mathbf{q}_N$ be the eigenvectors corresponding to the eigenvalues $0 = \mu_1 < \mu_2 \leq \cdots \leq \mu_N < 2$ of the normalized Laplacian $\tilde{\mathbf{L}}$. The transition probability matrix is then

$$\mathbf{H} = \mathbf{D}^{\frac{1}{2}}(\mathbf{I} - \tilde{\mathbf{L}})\mathbf{D}^{-\frac{1}{2}}. \qquad (39)$$

For the first term of the RHS of (38), we have

$$\begin{aligned}\|\mathbf{H}^K\mathbf{p}_+ - \mathbf{H}^K\mathbf{p}_-\| &\leq \|\mathbf{H}^K\mathbf{p}_+ - \boldsymbol{\pi}\| + \|\mathbf{H}^K\mathbf{p}_- - \boldsymbol{\pi}\| \\ &= \|\mathbf{D}^{\frac{1}{2}}(\mathbf{I} - \tilde{\mathbf{L}})^K\mathbf{D}^{-\frac{1}{2}}\mathbf{p}_+ - \frac{\mathbf{D}\mathbf{1}}{2|\mathcal{E}|}\| \\ &+ \|\mathbf{D}^{\frac{1}{2}}(\mathbf{I} - \tilde{\mathbf{L}})^K\mathbf{D}^{-\frac{1}{2}}\mathbf{p}_- - \frac{\mathbf{D}\mathbf{1}}{2|\mathcal{E}|}\|. \qquad (40)\end{aligned}$$

Since $\mathbf{q}_1 = \frac{\mathbf{D}^{\frac{1}{2}}\mathbf{1}}{\sqrt{2|\mathcal{E}|}}$ [26], we have for $c \in \{+, -\}$ that

$$\begin{aligned}\mathbf{D}^{\frac{1}{2}}\mathbf{q}_1\langle\mathbf{q}_1, \mathbf{D}^{-\frac{1}{2}}\mathbf{p}_c\rangle &= \mathbf{D}^{\frac{1}{2}}\frac{\mathbf{D}^{\frac{1}{2}}\mathbf{1}}{\sqrt{2|\mathcal{E}|}}\left\langle\frac{\mathbf{D}^{\frac{1}{2}}\mathbf{1}}{\sqrt{2|\mathcal{E}|}}, \mathbf{D}^{-\frac{1}{2}}\mathbf{p}_c\right\rangle \\ &= \frac{\mathbf{D}\mathbf{1}}{\sqrt{2|\mathcal{E}|}}\frac{\langle\mathbf{1}, \mathbf{p}_c\rangle}{\sqrt{2|\mathcal{E}|}} = \frac{\mathbf{D}\mathbf{1}}{2|\mathcal{E}|}\,. \qquad (41)\end{aligned}$$

Upon defining $\mathbf{M} := (\mathbf{I} - \tilde{\mathbf{L}})^K - \mathbf{q}_1\mathbf{q}_1^\mathsf{T}$, and taking into account (41), inequality (40) can be written as

$$\begin{aligned}&\|\mathbf{H}^K\mathbf{p}_+ - \mathbf{H}^K\mathbf{p}_-\| \\ &\quad\leq \|\mathbf{D}^{\frac{1}{2}}\|\|\mathbf{M}\|\left(\|\mathbf{D}^{-\frac{1}{2}}\mathbf{p}_+\| + \|\mathbf{D}^{-\frac{1}{2}}\mathbf{p}_-\|\right). \qquad (42)\end{aligned}$$

The factors in (42) can be bounded as

$$\begin{aligned}\|\mathbf{D}^{-\frac{1}{2}}\mathbf{p}_+\| &= \sqrt{\sum_{i\in\mathcal{L}_+}\left(\frac{1}{|\mathcal{L}_+|}d_i^{-\frac{1}{2}}\right)^2} \\ &= \sqrt{\sum_{i\in\mathcal{L}_+}\frac{1}{|\mathcal{L}_+|^2}d_i^{-1}} \leq \frac{1}{\sqrt{d_{\min_+}|\mathcal{L}_+|}}, \qquad (43) \\ \|\mathbf{D}^{-\frac{1}{2}}\mathbf{p}_-\| &= \sqrt{\sum_{i\in\mathcal{L}_-}\frac{1}{|\mathcal{L}_-|^2}d_i^{-1}} \leq \frac{1}{\sqrt{d_{\min_-}|\mathcal{L}_-|}}, \qquad (44) \\ \|\mathbf{M}\| &= \sup_{\mathbf{v}}\frac{\langle\mathbf{M}\mathbf{v}, \mathbf{v}\rangle}{\mathbf{M}\mathbf{v}} = \max_{i\neq 1}|1 - \mu_i|^K, \qquad (45) \\ \|\mathbf{D}^{\frac{1}{2}}\| &= \sqrt{d_{\max}} \qquad (46)\end{aligned}$$

where (45) follows from the properties of the normalized Laplacian. Therefore, (42) becomes

$$\begin{aligned}\|\mathbf{H}^K\mathbf{p}_+ - \mathbf{H}^K\mathbf{p}_-\| \leq &\left(\frac{1}{\sqrt{d_{\min_-}|\mathcal{L}_-|}} + \frac{1}{\sqrt{d_{\min_+}|\mathcal{L}_+|}}\right) \\ &\cdot \max_{i\neq 1}|1 - \mu_i|^K \cdot \sqrt{d_{\max}}. \qquad (47)\end{aligned}$$

Letting $\mu' := \min\{\mu_2, 2-\mu_N\}$, and using the fact that

$$(1-\mu')^K \leq e^{-K\mu'} \tag{48}$$

we obtain

$$\|\mathbf{H}^K\mathbf{p}_+ - \mathbf{H}^K\mathbf{p}_-\| \leq \left(\sqrt{\frac{d_{\max}}{d_{\min_-}|\mathcal{L}_-|}} + \sqrt{\frac{d_{\max}}{d_{\min_+}|\mathcal{L}_+|}}\right) e^{-K\mu'}. \tag{49}$$

Likewise, we can bound the second term in (38) as

$$\|\mathbf{H}^{K+1}\mathbf{p}_+ - \mathbf{H}^{K+1}\mathbf{p}_-\| \leq \left(\sqrt{\frac{d_{\max}}{d_{\min_-}|\mathcal{L}_-|}} + \sqrt{\frac{d_{\max}}{d_{\min_+}|\mathcal{L}_+|}}\right) e^{-(K+1)\mu'}. \tag{50}$$

In addition, we note that for all $\mu' > 0, K \in \mathbb{Z}$ it holds that

$$e^{-K\mu'} + e^{-(K+1)\mu'} < 2e^{-K\mu'}. \tag{51}$$

Upon substituting (49) and (50) into (38), and also using (51), we arrive at

$$\|\mathbf{y} - \check{\mathbf{y}}\| \leq 2\left(\sqrt{\frac{d_{\max}}{d_{\min_-}|\mathcal{L}_-|}} + \sqrt{\frac{d_{\max}}{d_{\min_+}|\mathcal{L}_+|}}\right) e^{-K\mu'}. \tag{52}$$

To determine an upper bound on the γ-distinguishability threshold, it suffices to find the smallest integer K for which (52) becomes less than γ; that is,

$$2\left(\sqrt{\frac{d_{\max}}{d_{\min_-}|\mathcal{L}_-|}} + \sqrt{\frac{d_{\max}}{d_{\min_+}|\mathcal{L}_+|}}\right) e^{-K\mu'} \leq \gamma. \tag{53}$$

Multiplying both sides of (53) by the positive number $e^{K\mu'}/\gamma$, and taking logarithms yields

$$\log\left[\tfrac{2\sqrt{d_{\max}}}{\gamma}\left(\sqrt{\tfrac{1}{d_{\min_-}|\mathcal{L}_-|}} + \sqrt{\tfrac{1}{d_{\min_+}|\mathcal{L}_+|}}\right)\right] \leq K\mu'.$$

Therefore, using as landing probabilities

$$\left\lceil \frac{1}{\mu'} \log\left[\tfrac{2\sqrt{d_{\max}}}{\gamma}\left(\sqrt{\tfrac{1}{d_{\min_-}|\mathcal{L}_-|}} + \sqrt{\tfrac{1}{d_{\min_+}|\mathcal{L}_+|}}\right)\right]\right\rceil$$

the ℓ_2 distance between any two diffusion-based classifiers will be at most γ; and the proof is complete.

C. Bound for PageRank

Substituting PageRank's diffusion coefficients in the proof of Theorem 1, inequality (53) becomes

$$2(1-\alpha)\alpha^K\left(\sqrt{\frac{d_{\max}}{d_{\min_-}|\mathcal{L}_-|}} + \sqrt{\frac{d_{\max}}{d_{\min_+}|\mathcal{L}_+|}}\right) e^{-K\mu'} \leq \gamma.$$

Multiplying both sides by the positive number $e^{K\mu'}\alpha^{-K}/\gamma$ and taking logarithms yields

$$\log\left[\tfrac{2\sqrt{d_{\max}}}{\gamma/(1-\alpha)}\left(\sqrt{\tfrac{1}{d_{\min_-}|\mathcal{L}_-|}} + \sqrt{\tfrac{1}{d_{\min_+}|\mathcal{L}_+|}}\right)\right] \leq K(\mu' - \log\alpha)$$

which results in the γ-distinguishability threshold bound

$$K_\gamma^{\mathrm{PR}} \leq \tfrac{1}{\mu' - \log\alpha} \log\left[\tfrac{2\sqrt{d_{\max}}}{\gamma/(1-\alpha)}\left(\sqrt{\tfrac{1}{d_{\min_-}|\mathcal{L}_-|}} + \sqrt{\tfrac{1}{d_{\min_+}|\mathcal{L}_+|}}\right)\right].$$

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, "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.

[8] 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.

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

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

[11] 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.

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

[13] 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.

[14] 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.

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

[16] 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.

[17] 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.

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

[19] V. Kekatos and G. B. Giannakis, "From sparse signals to sparse residuals for robust sensing," *IEEE Trans. Signal Process.*, vol. 59, no. 7, pp. 3355–3368, July 2011.

[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] W. Liu, J. Wang, and S.-F. Chang, "Robust and scalable graph-based semisupervised learning," *Proc. of the IEEE*, vol. 100, no. 9, pp. 2624–2638, 2012.

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

[30] 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.

[31] 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.

[32] 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.

[33] 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.

[34] 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.

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

[36] 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.

[37] 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.

[38] 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.

[39] 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.

[40] 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.

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

[42] 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.

[43] D. F. Gleich, and M. W. Mahoney, "Using Local Spectral Methods to Robustify Graph-Based Learning Algorithms," in *Proc. of the Intl Conf. on Knowl. Disc. and Data Mining*, Sidney Australia, Aug. 2015.

[44] K. He, P. Shi, J. E. Hopcroft, and D. Bindel, "Local Spectral Diffusion for Robust Community Detection," in *Proc. of the SIGKDD workshop*, San Francisco CA, Aug. 2016.

[45] B. Jiang, K. Kloster, D. F. Gleich, and M. Gribskov, "AptRank: an adaptive PageRank model for protein function prediction on bi-relational graphs," in *Bioinformatics*, vol. 33, no. 12, pp. 1829–1836, Aug. 2017.

[46] K. He, Y. Sun, D. Bindel, J. E. Hopcroft, and Y. Li, "Detecting overlapping communities from local spectral subspaces," in *Proc. of the Intl Conf. on Data Mining* , Atlantic City NJ, Aug. 2015, pp. 769–774