Active and Adaptive Techniques for Learning over Large-Scale Graphs

A DISSERTATION
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Dimitrios Bermperidis

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Prof. Georgios B. Giannakis, Advisor

May, 2019

Acknowledgments

Foremost, I would like to express my sincere gratitude to my advisor Prof. Georgios B. Giannakis for giving me the opportunity to be a part of the SPiNCOM research group, as well as the ECE/CS graduate program of University of Minnesota. With his continued guidance and support, he has greatly helped me in taking my first steps into the world of research and engineering, leading to the completion of this thesis. In addition, he has and continues to aid me in developing clear scientific thought and expression.

In a special way I would like to extend my appreciation to Prof. George Karypis, Prof. Zhili Zhang, and Prof. Mingyi Hong who agreed to serve on my thesis committee, and provided with much appreciated feedback that helped improve this work. Thanks go to a number of other professors in the departments of Electrical Engineering and Computer Science whose graduate level courses provided me with the necessary background to embark on this area of research.

Special thanks also go to Prof. Vassilis Kekatos for his help and guidance through the first stages of my PhD. I thank my good friends, companions and collaborators Dr. Fateme Sheikholeslami, Dr. Donghoon Lee, Vassilis Ioannidis, Panos Traganitis, and Charilao Kanatsoulis. Special thanks to my friend and collaborator Dr. Athanasios Nikolakopoulos for helping me expand my research/technical skills and for donating some of his unlimited enthusiasm to me. I thank all my fellow lab-mates and good friends in SPINCOM for their support and all the fruitful discussions we had over the years.

Last but not the least, I would like to express by deepest thanks to my family: my parents Kostas and Maria, my grand-parents Dimitris, Fevronia, and Xrisanthi, and my dear brother Theodoros for their continued love and support throughout my life.

Dimitris Berberidis, Minneapolis, May 31, 2019.

Dedication

This dissertation is dedicated to my family.

Abstract

Behind every complex system be it physical, social, biological, or man-made, there is an intricate network encoding the interactions between its components. Learning over large-scale networks is a challenging field, and practical methods must combine *scalability* in computations to cope with millions of nodes associated possibly with large amounts of meta-information; along with sufficient *versatility* to capture the elaborate structure and dynamics of the complex phenomena under study. There is also a need for modeling *expressiveness* to ensure accurate learning, along with transparency and *interpretability* that will shed light on the overall system understanding, and will provide valuable insights about its function. Approaches to learning over networks must also *defend against adversarial behavior*, thus remaining operational even under severely adverse circumstances.

The contribution of the present thesis lies at addressing the aforementioned challenges by developing simple yet versatile algorithmic solutions focused on core graph-learning tasks. To tackle active sampling for semi-supervised node classification, a novel framework is proposed in order to guide the sampling of informative nodes. The proposed framework is tailored to Gaussian-Markov random fields, and relies on the notion of *maximum expected-change* to select the most informative node to be labeled. Interestingly, several existing methods for active learning are subsumed by the proposed approach. Focusing on the node classification task, a generalized yet highly scalable *diffusion-based* classifier is developed, where each class diffusion is adaptive to the graph structure and the underlying label distribution. Adaptability is further leveraged for the node embedding task. As node embedding is naturally viewed as a low-rank factorization of a node-to-node similarity matrix, a versatile approach is introduced to learn the similarity matrix of a given graph with minimal computational overhead, and in a fully unsupervised manner. Extensive experimentation using both synthetic graphs as well as numerous real networks demonstrates the effectiveness, interpretability and scalability of the proposed methods. More importantly, the process of design and experimentation sheds light on the behavior of different methods and the peculiarities of real-world data, while at the same time generates new ideas and directions to be explored.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Learning over Graphs

In many shapes and forms, networks play a fundamental role in our life. The seamless regulations of genes and metabolites within cells are responsible for our biological existence. The coherent interactions between billions of neurons in our brains enables us to comprehend the world. The social bonds and ties between individuals comprise the fabric of our society. Power networks supply us with energy. Trade networks maintain our ability to exchange goods and services. Communication networks enable and empower the most revolutionary technologies of the 21st century. Behind every complex system there is an intricate network capturing the interactions among its components. Reasoning about these networks is one of the major intellectual and scientific challenges of the 21st century.

Statistical learning over networks comprises a suite of computational tools to address core tasks relevant to a host of applications arising in diverse disciplines. For example, node classification in protein-protein interaction networks aims to associate proteins with specific biological functions, thereby facilitating the understanding of pathogenic and physiological mechanisms. Link prediction in user-product purchase networks can enable personalized recommendations, as well as targeted content delivery and advertising. Community detection in brain networks can help identify the fundamental structures that control and mediate its information pathways, that also unveil its higher-order connectivity patterns and organization.

Let us begin by shortly introducing the graph, an abstract mathematical concept, and at the same time one of the most intuitive and familiar ways of representing relations between

interconnected entities. A graph is typically denoted by $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, where $\mathcal{V}$ is the set of N nodes, and $\mathcal{E}$ contains the edges. There are several different ways to represent a graph. The *adjacency-list* and *edge-list* formats are most frequently employed for low-level algorithmic design and software implementations since they naturally leverage sparsity, and are more amenable to simple operations such as node exploration (e.g., breadth-first-search using an adjacency list). Nevertheless, for problems that involve higher level abstractions (e.g., spectral clustering) the *adjacency matrix* format provides with convenient representation that can exploit the full arsenal of linear algebra tools. Thus, in the machine learning context, graphs are most frequently represented by the $N \times N$ (possibly) weighted adjacency matrix $\mathbf{A}$ whose $(i, j)-$th entry denotes the weight of the edge that connects node v_i with node v_j.

In many applications, graphs emerge naturally, with $\mathcal{V}$ and $\mathcal{E}$ readily collected and stored. In other cases, a graph can be inferred from a set of N nodal (a.k.a. feature) vectors $\{\mathbf{x}_i\}_{i=1}^N$ representing data points, such that each node of the graph corresponds to a data point. Matrix $\mathbf{A}$ can be obtained from the feature vectors $\{\mathbf{x}_i\}_{i=1}^N$ using different similarity measures. For example, one may use the radial basis function $w_{i,j} = \exp\left(-\|\mathbf{x}_i - \mathbf{x}_j\|_2^2/\sigma^2\right)$ that assigns large edge weights to pairs of points that are neighbors in Euclidean space, or the Pearson correlation coefficients $w_{i,j} = \langle \mathbf{x}_i, \mathbf{x}_j \rangle / \left(\|\mathbf{x}_i\|_2 \|\mathbf{x}_j\|_2\right)$ .. If $w_{i,j} \neq 0 \; \forall i, j$, the resulting graph will be fully connected, but one may obtain a more structured graph by thresholding small weights to 0, or by connecting every point only with its $k-$nearest neighbors.

The unique properties and characteristics of graphs have led to a learning approach that is distinct from the "traditional" one that deals with vectors. Thus, graph-based learning, learning-over-graphs, or graph-mining are often interchangable terms that are frequently used to describe one or more of the following tasks.

- **Classification.** In many cases, each node v_i is associated with one (or more) discrete label(s) $y_i \in \mathcal{Y}$ drawn from a finite alphabet. In this context, the goal of semi-supervised classification amounts to propagating an observed subset of labels $\{y_i\}_{i \in \mathcal{L}}$, where $\mathcal{L} \subseteq \mathcal{V}$, to the rest of the network, in order to infer the labels $\{y_i\}_{i \in \mathcal{U}}$ of the set of unlabeled nodes $\mathcal{U} := \mathcal{V}/\mathcal{L}$.

- **Community detection/Clustering.** Another typical graph learning task is that of categorizing the nodes of the graph to K overlapping (soft-clustering) or non-overlapping clusters, also known as *communities*. The communities $\{\mathcal{C}_k\}_{k=1}^K$, where $\mathcal{C}_k \subseteq \mathcal{V}$, are

typically inferred in a fully *unsupervised* manner, that relies only on the graph structure. Given a subset of nodes $\mathcal{S}_k \subset \mathcal{C}_k$ that are *known* to belong to the k-th community however, semi-supervised community detection aims at inferring the remaining hidden (latent) members of $\mathcal{C}_k$ – a task that has also been posed and gathered significant research interest.

- **Link Prediction/ Node Recommendation.** Many real graphs are dynamic in nature, with new links appearing (or disappearing) with time. Link prediction is the task of identifying the patterns according to which new links are formed, and accurately inferring them beforehand. One way to pose the link prediction task is that, given the edges $\mathcal{E}_t$ at time t, one would like to determine $\Pr\{(v_i, v_j) \in \mathcal{E}_{t+1}\}\ \forall (v_i, v_j) \in \mathcal{V} \times \mathcal{V} \setminus \mathcal{E}_t$; that is, the likelihood that a new potential edge (v_i, v_j) will be present in the next instance $\mathcal{E}_{t+1}$ of the edge set. Similarly, *node recommendation* is the task of predicting the links $\{(v_i, v_j)\}_{j \in \mathcal{V}_i}$ that a given node v_i is expected (or would "prefer") to form in a future instance of the graph; and $\mathcal{V}_i$ can thus be recommended to v_i as a set of possible connections.

- **Node Embedding.** Many graph learning tasks can be addressed by first extracting features over the nodes of the graph, effectively "embedding" them in a Eucledian space; the extracted features may then be used by downstream machine learning methods to perform classification, link prediction, clustering, and other tasks. Thus, node embedding boils down to determining $f(\cdot) : \mathcal{V} \to \mathbb{R}^d$, where $d \ll N$. In other works, a function is sought to map every node of $\mathcal{G}$ to a vector in the $d-$dimensional Euclidean space. Typically, the embedding is low dimensional with d much smaller than the number of nodes. Given $f(\cdot)$, the low-dimensional vector representation of each node v_i is $\mathbf{e}_i = f(v_i)\ \ \forall v_i \in \mathcal{V}$. Since the number of nodes is finite, instead of finding a general $f(\cdot)$ inductively, one may pose the embedding task in its most general form as a the following minimization problem over the embedded vectors

$$\{\mathbf{e}_i^*\}_{i=1}^N = \arg \min_{\{\mathbf{e}_i\}_{i=1}^N} \sum_{v_i, v_j \in \mathcal{V}} \ell\left(s_{\mathcal{G}}(v_i, v_j), s_{\mathcal{E}}(\mathbf{e}_i, \mathbf{e}_j)\right) \tag{1.1}$$

where $\ell(\cdot, \cdot) : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ is a loss function; $s_{\mathcal{G}}(\cdot, \cdot) : \mathcal{V} \times \mathcal{V} \to \mathbb{R}$ is a similarity metric over pairs of graph *nodes*; and $s_{\mathcal{E}}(\cdot, \cdot) : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ a similarity metric over pairs of *vectors* in the $d-$dimensional Euclidean space. In par with (4.1), node embedding can be viewed as the design of nodal vectors $\{\mathbf{e}_i\}_{i=1}^N$ that successfully "encode" a certain notion

of pairwise similarities among graph nodes.

1.2 Motivation and Context

Learning-over-graphs is well motivated by a large number of applications. This has led to ever-increasing research efforts in this area during the last two decades. Nevertheless, many real-world issues and practical aspects of the problem face formidable challenges to this day. The main focus of the present work is to identify and address such challenges using simple, robust, and interepretable mathematical and algorithmic tools.

1.2.1 Applications of graph-based learning

In this subsection, we discuss some important and contemporary applications of graph-based learning. The goal here is to demonstrate how graphs naturally emerge in diverse domains, and how learning/mining over such graphs can be instrumental in addressing real-world problems.

Social networks

Social networks is a broad term used to describe the class of naturally-occurring complex networks that arise upon registering the interactions (edges) between a set of social entities or actors (nodes). The nodes in social networks may correspond to individuals, groups of individuals, organizations, products, websites, and many other possible types of interacting entities. Social networks are ubiquitous in the physical world, and even more so in cyberspace. Specifically, the proliferation of Internet connectivity since the start of the 21-st century has given birth to a huge variety of online social networking platforms [1], connecting individuals in many contexts (professional, social, dating, and others). Ease of Internet access has led to several of the social networking platforms (e.g., Facebook, Twitter, Youtube) growing to unprecedented proportions, scaling to billions of nodes, and encompassing a sizeable portion of the human population. The analysis of such networks is of interest both for the increase in profitability, as well as the advancements on our understanding of socials structures that it can offer. A common task is to group individuals to overlapping categories. This may help in identifying underlying characteristics and trends in a given populations; see, for instance, Fig. 1.1 for a depiction of the 2004 political US blogosphere graph during the presidential elections, where liberal and

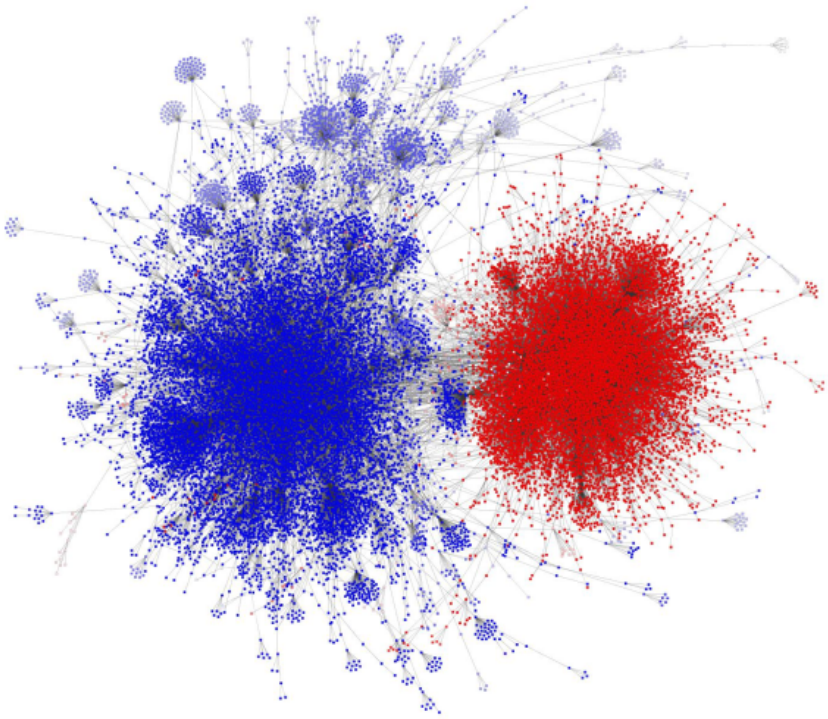

Figure 1.1: Visualization of the 2004 US presidential elections blogosphere graph. Blue nodes correspond to democratic political blogs, while red denotes republican ones. Links correspond to blogs that contain references to each other. The clustering of the graph to a blue and red group indicates the high degree of polarization that characterizes the US political landscape.

conservative blogs are clearly separated into two clusters. Furthermore, ascertaining social network nodes as e.g., male or female and conservative or liberal, can in general be treated as clustering in lack of supervision, or classification in the presence of some additional information on a subset of the individuals. For example, given a friendship network, the political preferences of a few individuals, and using the "homophily" assumption, an individual's political opinion will most likely be aligned with those of close friends, the political opinions of unobserved individuals can be inferred by "association." Finally, social networking platforms rely heavily on link prediction and node recommendation to expand their networks, and keep their users interested by suggesting the formation of new friendships.

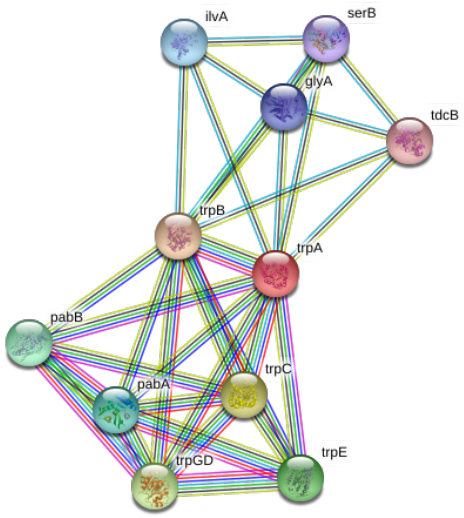

Figure 1.2: Subgraph of the Homo Sapiens protein-protein interaction (PPI) network extracted from the *Sting Consortium* repository (link).

Biological networks

Protein–protein interactions (PPI) networks link proteins according to physical contacts of high specificity established between two or more molecules as a result of biochemical events; see e.g., Fig. 1.2 for a subraph extracted from the Homo Sapiens PPI network. Given a set of functionally uncharacterized genes or proteins from a Genome-Wide Association Study, or differential expression analysis, experimental biologists often have little a priori information available to guide the design of hypothesis-based experiments to determine molecular functions. For example, what is the expected phenotype if a particular gene is removed? It would greatly improve hypothesis formation if biologists had prior insight from predicted functions of interesting genes or proteins in databases. Computational annotation of genes or proteins with unknown functions is thus a fundamental research area in computational biology. From a graph-learning perspective, the task of protein function prediction falls under the category of semi-supervised classification. In this context, diffusion-based network models are widely used for protein function prediction using protein network data, and have been shown to outperform neighborhood-based and module-based methods [63].

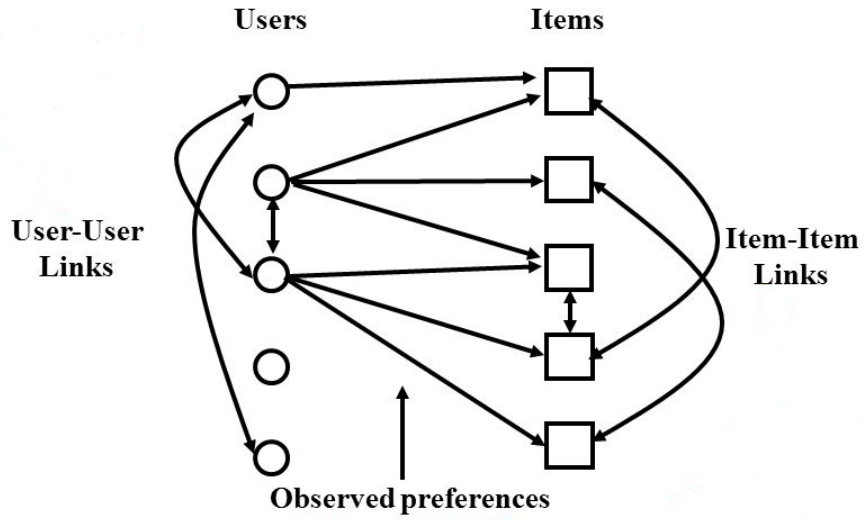

Figure 1.3: Graph-based representation of the recommendation setting. Users (round nodes) and items (square nodes) form a heterogeneous network. User-item links are formed from observed user preferences and/or implicit feedback, while (optional) intra-user links may be available from friendships networks, and intra-item links may be inferred from data.

.

Recommender systems

Top-n recommendation algorithms provide ranked lists of items tailored to the particular user-specific preferences, as depicted by their past interactions. Over the past years, they have become an indispensable component of most e-commerce applications as well as content delivery platforms. Top-n recommendation often relies on graph-mining tools, and is intimately intertwined with the task of link prediction. This becomes apparent upon considering that the recommendation problem can be readily represented as a bipartite graph that connects a set of users to a set of items. The links connecting user-nodes to item-nodes may represent observed user-interactions (implicit feedback) or explicit item preferences as expressed by the users. This past information that is encoded in the bipartite graph structure can be leveraged to predict future or "meaningful" links, which can be directly translated to recommended items. Furthermore, information on user-user (e.g., via social networking) and/or item-item relations may be available or inferred in order to construct more general graph models; see Fig. 1.3 for an illustration. In fact, methods relying on item-item relations are among the most popular approaches for

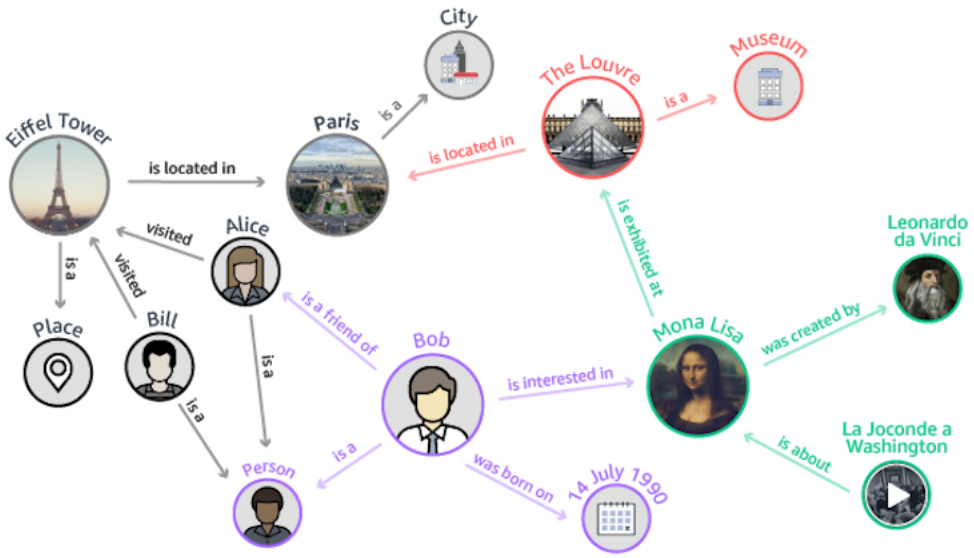

Figure 1.4: Example of a knowledge graph constructed from `entity/relation/entity` triples.

top-n recommendation. Such methods work by building a model that captures the relations between the items, which is then used to recommend new items that are "close" to the ones each user has consumed in the past. Item-based models have been shown to achieve high top-n recommendation accuracy [111, 97], while being scalable and easy to interpret [35].

Knowledge graphs

A knowledge graph (KG) is a multi-relational graph composed of entities (nodes) and relations (different types of edges). Each edge is represented as a triplet of the form (head entity, relation, tail entity), also called a fact, indicating that two entities are connected by a specific relation; see, for example, the triplet `(EiffelTower, IsLocatedIn, Paris)`, where entities `EiffelTower` and `Paris` are connected via the predicate (relation) `IsLocatedIn`. Such triplets often follow the Resource Description Framework (RDF) semantic web specifications and are frequently extracted from raw text data using various natural language processing tools. Given a collection of RDF triplets, constructing a meaningful KG involves many challenging tasks such as resolving entity ambiguities, identifying and resolving inconsistencies, and dealing with

incomplete data. KGs can then be used to enhance the quality of semantic queries. Nevertheless, although effective in representing structured data, the underlying symbolic nature of such triples can render KGs hard to manipulate. To tackle this issue, a new research direction that is known as KG embedding has been introduced and quickly gained massive popularity [126]. The key idea is to embed components of a KG including entities and relations into continuous vector spaces, so as to simplify the manipulation while preserving the inherent structure of the KG. Those entity and relation embeddings can further be used to benefit all kinds of tasks, such as KG completion, relation extraction, entity classification, and entity resolution.

1.2.2 Prior work in context

Learning over networks has been extensively pursued over the past two decades, with a wide range of tools employed towards classifying nodes [30], predicting links [83] and discovering communities [38] present in real and man-made networks. We will briefly summarize important prior work on the tasks that the present thesis addresses, namely node classification, active learning, and node embedding.

Semi-supervised node classification methods can be divided into three general categories with regards to modeling and algorithmic complexity. The first category includes non-parametric approaches leveraging homophily – a frequently observed property of real networks – that similar to the more general smoothness-over-the-manifold assumption, is adopted by most semi-supervised learning (SSL) methods [30]. This category also encompasses approaches employing kernels on graphs [91], manifold regularization [18], transductive learning [118], iterative label propagation [19, 140, 85, 77], graph partitioning [123, 64], competitive infection models [107], and bootstrapped label propagation [25]. A notable subset of approaches relies on random walks, which diffuse probabilistically the available information through the network. Celebrated representatives include the Personalized PageRank [24] and the Heat Kernel [32] that were found to perform remarkably well in node classification [84] and community detection [70] tasks, and have also been theoretically linked to particular network models [14, 71, 73]. The upshot of diffusion-based approaches is their ability to leverage sparsity that facilitates computations and scalability. With their computational efficiency granted, the effectiveness of diffusion-based methods—as well as most other non-parametric approaches—can vary considerably depending on whether the chosen model conforms with the latent characteristics of the target application

and the underlying network topology.

The second category comprises recently popular approaches to learning over networks using node-embeddings [26, 46]. Embedding-based learning is a two stage process. First, an embedding method is employed to map nodes to vectors in a low-dimensional Euclidean space. Second, the extracted vectors that correspond to training nodes are used as an input to a parametric supervised learning algorithm (e.g., logistic regression or SVM). The trained model is then applied to predict vector representations of the remaining nodes. From a high-level vantage point, node embedding methods form vector representations that preserve network properties while adhering to structural and relational nodal characteristics. Thus, early embedding efforts naturally relied on spectral or singular value decompositions [44] of the adjacency or the Laplacian matrix of the network; see e.g., [130]. Recently, novel node-embedding methods have emerged that are based on random walks, also borrowing ideas and heuristics from advances in natural language processing; see e.g. [101, 47, 134]. The main advantage of the embedding-based approaches is that they provide the means for any traditional feature-based learning algorithm to be applied on networks, thus greatly increasing the range of available options. Their performance however, is influenced by the extent to which their defining heuristic is aligned with the properties of the learning task at hand, which will be generally unknown. Furthermore, embedding all nodes of a network typically entails large computational complexity and memory requirements that may prohibit their application to large real-world networks.

The third category of prior works includes convolutional neural network (CNN) architectures that have been adapted to account for the network structure [13, 69, 115]. Such graph (G)CNNs have recently gained popularity, and can be interpreted as jointly performing node-embedding and learning. GCNNs have been reported to yield state-of-the-art performance in certain benchmark networks. However, GCNNs heavily rely on additional information provided by feature vectors that accompany some real networks (e.g. keywords and Abstracts in citation networks), and may perform poorly in network-only setups. Generally, in the absence of further meta-information, the excessive number of parameters may render GCNNs vulnerable to overfitting and challenging to train. This is especially true when the amount of training data is limited. Moreover, while the use of 'shallow' architectures proposed recently [69] can afford GCNNs with reasonable computational complexity, to effectively capture the complex dynamics of real networks, deeper architectures may be necessary, which can very easily lead to prohibitive computational overhead. Finally, the intrinsically opaque nature of GCNNs renders their outputs challenging to interpret -

what is desirable in most applications.

Active learning is the task of actively querying objects for information in order to maximize a certain learning utility. In the present context, it refers to selecting which nodes to label in order to maximize classification accuracy (or minimize teh generalization error). Prior art in graph-based active learning can be divided in two categories. The first includes the *non-adaptive* design-of-experiments-type methods, where sampling strategies are designed *offline* depending only on the graph structure, based on ensemble optimality criteria. The non-adaptive category also includes the variance minimization sampling [62], as well as the error upper bound minimization in [49], and the data non-adaptive Σ-optimality approach in [89]. The second category includes methods that select samples adaptively and jointly with the classification process, taking into account both graph structure as well as previously obtained labels. Such *data-adaptive* methods give rise to sampling schemes that are not optimal on average, but adapt to a given realization of labels on the graph. Adaptive methods include the Bayesian risk minimization [141], the information gain maximization [86], as well as the manifold preserving method of [139]; see also [65, 40, 29]. Finally, related works deal with selective sampling of nodes that arrive sequentially in a gradually augmented graph [48, 39, 117], as well as active sampling to infer the graph structure [102, 53].

Node Embedding. Early embedding works mostly focused on a structure-preserving dimensionality reduction of feature vectors (instead of nodes); see for instance [59, 16, 108, 56, 52]. In this context, graphs are constructed from pairwise feature vector relations and are treated as representations of the manifold that data lie on; embedded vectors are then generated so that they preserve the corresponding pair-wise proximities on the manifold. More recently, nodal vector embedding of a graph has attracted considerable attention in different fields, and is often posed as the factorization of a properly defined node similarity matrix [10, 131, 74, 99, 104, 28, 114, 138]. Efforts in this direction mostly focus on designing meaningful similarity metrics to factorize. While some methods (e.g. [10, 99]) maintain scalability by factorizing similarity matrices in an implicit manner (without explicitly forming them), others such as [104, 28] form and/or factorize dense similarity matrices that scale poorly to large graphs. Another line of work opts to gradually fit pairs of embedded vectors to existing edges using stochastic optimization tools [120, 119]. Such approaches are naturally scalable and entail a high degree of locality. Recently, stochastic

edge-fitting has been generalized to implicitly accommodate long-range node similarities [122]. Meanwhile, other works have approached node embeddings using random-walk-based tools and concepts originating from natural language processing [101, 47, 109]; see also related works on embedding of knowledge graphs [23, 129]. Methods that rely on graph convolutional neural networks and autoencoders have also been proposed for node embedding [125, 121, 27]. Moreover, a gamut of related embedding tasks are gaining traction, such as embedding based on structural roles of nodes [106, 36], supervised embeddings for classification [134], and inductive embedding methods that utilize multiple graphs [51]

1.2.3 Challenges

The present subsection identifies and outlines some of the significant challenges of practical learning-over-graphs tasks.

C1. Versatility and Adaptability of Learning Frameworks. As seen in Section 1.2.1, real-world graphs may arise from vastly different domains, ranging from knowledge databases to protein interactions. Naturally, such graphs are expected to have different properties and unique characteristics. It then becomes apparent that, for any given task, there may not be a "one-size-fits-all" approach.

C2. Effective and interpretable learning over networks under scarce training data. In several applications, learning over networks must rely on a limited number of training data. Node classification in biological networks for instance, seeks the unknown function of all proteins based on a small subset of them. In link prediction for top-n recommendations, relevant items are sought in the user-item bipartite network based on implicit user feedback regarding a tiny fraction of them. Under such training setups, the challenge is to effectively capture the latent patterns, while avoiding the pitfall of overfitting the scarce available information. At the same time, being able to explain the outcomes of a learning algorithm is becoming increasingly important. Currently, available methods are either too simplistic to attain desirable learning performance, or, they offer over-parametrized 'black box' approaches without quantifiable generalization ability, and with limited transparency to produce interpretable outcomes.

C3. Dealing with unreliable data. The tacit assumption behind standard statistical learning schemes is that the available training data reliably reflect the function to be learned. In various applications however, this is not the case. For example, learning over social networks in our era of 'misinformation' and 'fake news,' renders the majority of learning schemes brittle to the presence of malicious or heavily biased data. *Is there a way to attain robust learning-over-networks even from unreliable data?* Means of addressing this issue has potentially transformative consequences of both theoretical and practical interest.

C4. Massive networks that change over time. Real networks often comprise hundreds of millions of nodes, and learning tasks on them must be performed in a timely fashion. Dynamic networks, such as those corresponding to social media or the Web, must be analyzed frequently for their predictions to be valid and useful. Likewise, detection of suspicious activity in transaction networks needs to be performed daily, if not on-the-fly. Oftentimes the sheer volume of such networks proves to be simply too-much for state-of-the-art approaches to handle. This is why time-and-again in practice one typically resorts to simpler and efficient schemes based on e.g., the PageRank [100] or simple Random-Walks [50] that have documented reliable and scalable performance. *Can we combine the efficiency of diffusion-based approaches with the due flexibility to learn complex interactions?* This crucial question has not been sufficiently addressed.

1.3 Thesis Outline and Contributions

Chapter 2 deals with *active sampling* of graph nodes representing training data for binary classification. The graph may be given or constructed using similarity measures among nodal features. Leveraging the graph for classification builds on the premise that labels across neighboring nodes are correlated according to a categorical Markov random field (MRF). This model is further relaxed to a Gaussian (G)MRF with labels taking continuous values - an approximation that not only mitigates the combinatorial complexity of the categorical model, but also offers optimal unbiased soft predictors of the unlabeled nodes. The proposed sampling strategy is based on querying the node whose label disclosure is expected to inflict the largest change on the GMRF, and in this sense it is the most informative on average. Connections are established with other sampling methods including uncertainty sampling, variance minimization, and sampling based on the $\Sigma-$optimality criterion. A simple yet effective heuristic is also introduced for

increasing the exploration capabilities of the sampler, and reducing bias of the resultant classifier, by adjusting the confidence on the model label predictions. The novel sampling strategies are based on quantities that are readily available without the need for model retraining, rendering them computationally efficient and scalable to large-scale graphs. Numerical tests using synthetic and real data demonstrate that the proposed methods achieve accuracy that is comparable or superior to the state-of-the-art even at reduced runtime.

Chapter 3 aims at improving the classifier itself, focusing specifically on *diffusion-based classifiers*. The effectiveness of the latter 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 contribution of this chapter is to alleviate these shortcomings and markedly improve the performance of random-walk-based classifiers by *adapting the diffusion functions of every class* 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 they can achieve results competitive to state-of-the-art methods, while also being considerably faster [20].

Finally, Chapter 4 identifies and addresses several major challenges faced by *node embedding*. Practical embedding methods have to deal with real-world graphs that arise from different domains, with inherently diverse underlying processes as well as similarity structures and metrics. On the other hand, similar to principal component analysis in feature vector spaces, node embedding is an inherently *unsupervised* task. Lacking metadata for validation, practical schemes motivate standardization and limited use of tunable hyperparameters. Lastly, node embedding methods must be scalable in order to cope with large-scale real-world graphs of networks with ever-increasing size. This last chapter puts forth an adaptive node embedding framework that adjusts the embedding process to a given underlying graph, in a fully unsupervised manner. This is achieved by leveraging the notion of a tunable node similarity matrix that assigns weights on multihop paths. The design of multihop similarities ensures that the resultant

embeddings also inherit interpretable spectral properties. The proposed model is thoroughly investigated, interpreted, and numerically evaluated using stochastic block models. Moreover, an unsupervised algorithm is developed for training the model parameters effieciently. Extensive node classification, link prediction, and clustering experiments are carried out on many real-world graphs from various domains, along with comparisons with state-of-the-art scalable and unsupervised node embedding alternatives. The proposed method enjoys superior performance in many cases, while also yielding interpretable information on the underlying graph structure.

The thesis is summarized, and interesting open problems are outlined in Chapter 5.

1.4 Notational Conventions

The following notation will be used throughout the subsequent chapters. Lower- (upper-) case boldface fonts denote vectors (matrices). Calligraphic letters are reserved for sets, e.g., $\mathcal{S}$. Symbol $\mathcal{T}$ ($\mathcal{H}$) as superscript stands for matrix and vector transposition (conjugate transposition). For vectors, $\|\cdot\|_2$ or $\|\cdot\|$ represents the Euclidean norm, while $\|\cdot\|_0$ denotes the ℓ_0 pseudo-norm counting the number of nonzero entries. The floor (ceiling) operation $\lfloor c \rfloor$ ($\lceil c \rceil$) denotes the largest integer no greater (the smallest integer but no smaller) than the given number $c > 0$; and $|\mathcal{S}|$ counts the number of entries in $\mathcal{S}$. Let $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ denote the vector Gaussian distribution with mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$; and $\mathrm{erf}(x) := (1/\sqrt{\pi}) \int_{-x}^{x} e^{-\tilde{x}^2} d\tilde{x}$ the Gauss error function. For any integer $m > 0$, symbol $[m]$ denotes the set $\{1, 2, \ldots, m\}$. Finally, $\succeq$ represents positive semi-definiteness of matrices, while the ordered eigenvalues of matrix $\boldsymbol{X} \in \mathbb{R}^{n \times n}$ are given as $\lambda_1(\boldsymbol{X}) \geq \lambda_2(\boldsymbol{X}) \geq \cdots \geq \lambda_n(\boldsymbol{X})$.

Chapter 2

Active Learning over Graphs with Maximum Expected-Change

Consider a connected undirected graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, where $\mathcal{V}$ is the set of N nodes, and $\mathcal{E}$ contains the edges that are also represented by the $N \times N$ weighted adjacency matrix $\mathbf{A}$. Let us further suppose that a binary label $y_i \in \{-1, 1\}$ is associated with each node v_i. The weighted binary labeled graph can either be given, or, it can be inferred from a set of N data points $\{\mathbf{x}_i, y_i\}_{i=1}^N$ such that each node of the graph corresponds to a data point.

In this context, semi-supervised learning amounts to propagating an observed subset of labels to the rest of the network. Thus, upon observing $\{y_i\}_{i\in\mathcal{L}}$ where $\mathcal{L} \subseteq \mathcal{V}$, henceforth collected in the $|\mathcal{L}| \times 1$ vector $\mathbf{y}_\mathcal{L}$, the goal is to infer the labels of the unlabeled nodes $\{y_i\}_{i\in\mathcal{U}}$ concatenated in the vector $\mathbf{y}_\mathcal{U}$, where $\mathcal{U} := \mathcal{V}/\mathcal{L}$. Let us consider labels as random variables that follow an unknown joint distribution $(y_1, y_2, \ldots, y_N) \sim p(y_1, y_2, \ldots, y_N)$, or $\mathbf{y} \sim p(\mathbf{y})$ for brevity.

For the purpose of inferring unobserved from observed labels, it would suffice if the joint posterior distribution $p\left(\mathbf{y}_\mathcal{U}|\mathbf{y}_\mathcal{L}\right)$ were available; then, $\mathbf{y}_\mathcal{U}$ could be obtained as a combination of labels that maximizes $p\left(\mathbf{y}_\mathcal{U}|\mathbf{y}_\mathcal{L}\right)$. Moreover, obtaining the marginal posterior distributions $p\left(y_i|\mathbf{y}_\mathcal{L}\right)$ of each unlabeled node i is often of interest, especially in the present greedy active sampling approach. To this end, it is well documented that MRFs are suitable for modeling probability mass functions over undirected graphs using the generic form, see e.g., [141]

$$p(\mathbf{y}) := \frac{1}{Z_\beta} \exp\left(-\frac{\beta}{2}\Phi(\mathbf{y})\right) \tag{2.1a}$$

where the "partition function" Z_β ensures that (2.1a) integrates to 1, β is a scalar that controls the smoothness of $p(\mathbf{y})$, and $\Phi(\mathbf{y})$ is the so termed "energy" of a realization $\mathbf{y}$, given by

$$\Phi(\mathbf{y}) := \frac{1}{2} \sum_{i,j \in \mathcal{V}} w_{i,j} \left(y_i - y_j\right)^2 = \mathbf{y}^T \mathbf{L} \mathbf{y} \tag{2.1b}$$

that captures the graph-induced label dependencies through the graph Laplacian matrix $\mathbf{L} := \mathbf{D} - \mathbf{A}$ with $\mathbf{D} := \mathrm{diag}(\mathbf{A}\mathbf{1})$. This categorical MRF model in (2.1a) naturally incorporates the known graph structure (through $\mathbf{L}$) in the label distribution by assuming label configurations where nearby labels (large edge weights) are similar, and have lower energy as well as higher likelihood. Still, finding the joint and marginal posteriors using (2.1a) and (2.1b) incurs *exponential complexity* since $\mathbf{y}_\mathcal{U} \in \{-1, 1\}^{|\mathcal{U}|}$. To deal with this challenge, less complex continuous-valued models are well motivated for a scalable approximation of the marginal posteriors. This prompts our next step to allow for continuous-valued label configurations $\boldsymbol{\psi}_\mathcal{U} \in \mathbb{R}^{|\mathcal{U}|}$ that are modeled by a GMRF.

2.0.1 GMRF relaxation

Consider approximating the binary field $\mathbf{y} \in \{-1, 1\}^{|\mathcal{U}|}$ that is distributed according to (2.1a) with the continuous-valued $\boldsymbol{\psi} \sim \mathcal{N}(\mathbf{0}, \mathbf{C})$, where the covariance matrix satisfies $\mathbf{C}^{-1} = \mathbf{L}$. Label propagation under this relaxed GMRF model becomes readily available in closed form. Indeed, $\boldsymbol{\psi}_{\mathcal{U}|\mathcal{L}}$ of unlabeled nodes conditioned on the labeled ones obeys

$$\boldsymbol{\psi}_{\mathcal{U}|\mathcal{L}} \sim \mathcal{N}(\boldsymbol{\mu}_{\mathcal{U}|\mathcal{L}}, \mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}) \tag{2.2}$$

where $\mathbf{L}_{\mathcal{U}\mathcal{U}}$ is the part of the graph Laplacian that corresponds to unlabeled nodes in the partitioning

$$\mathbf{L} = \begin{bmatrix} \mathbf{L}_{\mathcal{U}\mathcal{U}} & \mathbf{L}_{\mathcal{U}\mathcal{L}} \\ \mathbf{L}_{\mathcal{L}\mathcal{U}} & \mathbf{L}_{\mathcal{L}\mathcal{L}} \end{bmatrix}. \tag{2.3}$$

Given the observed $\boldsymbol{\psi}_\mathcal{L}$, the minimum mean-square error (MMSE) estimator of $\boldsymbol{\psi}_\mathcal{U}$ is given by the conditional expectation

$$\begin{aligned} \boldsymbol{\mu}_{\mathcal{U}|\mathcal{L}} &= \mathbf{C}_{\mathcal{U}\mathcal{L}} \mathbf{C}_{\mathcal{L}\mathcal{L}}^{-1} \boldsymbol{\psi}_\mathcal{L} \\ &= -\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1} \mathbf{L}_{\mathcal{U}\mathcal{L}} \boldsymbol{\psi}_\mathcal{L} \end{aligned} \tag{2.4}$$

where the first equality holds because for jointly Gaussian zero-mean vectors the MMSE estimator coincides with the linear (L)MMSE one (see e.g., [67, p. 382]), while the second equality is derived in Appendix A1. When binary labels $\mathbf{y}_{\mathcal{L}}$ are obtained, they can be treated as measurements of the continuous field ($\boldsymbol{\psi}_{\mathcal{L}} := \mathbf{y}_{\mathcal{L}}$), and (2.4) reduces to

$$\boldsymbol{\mu}_{\mathcal{U}|\mathcal{L}} = -\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}\mathbf{L}_{\mathcal{U}\mathcal{L}}\mathbf{y}_{\mathcal{L}}. \tag{2.5}$$

Interestingly, the conditional mean of the GMRF in (2.5) may serve as an approximation of the marginal posteriors of the unknown labels. Specifically, for the $i-$th node, we adopt the approximation

$$\begin{aligned} p\left(y_i = 1|\mathbf{y}_{\mathcal{L}}\right) &= \frac{1}{2}\left(\mathbb{E}\left[\left[\mathbf{y}_{\mathcal{U}|\mathcal{L}}\right]_i\right] + 1\right) \\ &\approx \frac{1}{2}\left(\mathbb{E}\left[\left[\boldsymbol{\psi}_{\mathcal{U}|\mathcal{L}}\right]_i\right] + 1\right) \\ &= \frac{1}{2}\left(\left[\boldsymbol{\mu}_{\mathcal{U}|\mathcal{L}}\right]_i + 1\right) \end{aligned} \tag{2.6}$$

where the first equality follows from the fact that the expectation of a Bernouli random variable equals its probability. Given the approximation of $p\left(y_i|\mathbf{y}_{\mathcal{L}}\right)$ in (2.6), and the uninformative prior $p(y_i = 1) = 0.5\ \forall i \in \mathcal{V}$, the maximum a posteriori (MAP) estimate of y_i, which in the Gaussian case here reduces to the minimum distance decision rule, is given as

$$\hat{y}_i = \begin{cases} 1 & \left[\boldsymbol{\mu}_{\mathcal{U}|\mathcal{L}}\right]_i > 0 \\ -1 & \text{else} \end{cases}, \quad \forall i \in \mathcal{U} \tag{2.7}$$

thus completing the propagation of the observed $\mathbf{y}_{\mathcal{L}}$ to the unlabeled nodes of the graph.

It is worth stressing at this point, that as the set of labeled samples changes, so does the dimensionality of the conditional mean in (2.5), along with the "auto-" and "cross-" Laplacian sub-matrices that enable soft label propagation via (2.5), and hard label propagation through (2.7).

Two remarks are now in order. First, it is well known that the Laplacian of a graph is not invertible, since $\mathbf{L}\mathbf{1} = \mathbf{0}$; see, e.g. [72]. To deal with this issue, we instead use $\mathbf{L} + \delta\mathbf{I}$, where $\delta \ll 1$ is selected arbitrarily small but large enough to guarantee the numerical stability of e.g., $\mathbf{L}_{\mathcal{U}\mathcal{U}}$ in (2.5). A closer look at the energy function $\Phi(\mathbf{y}) := \sum_{i,j\in\mathcal{V}} a_{i,j}\left(y_i - y_j\right)^2 + \delta\sum_{i\in\mathcal{V}} y_i^2$ reveals that this simple modification amounts to adding a "self-loop" of weight δ to each node of

Algorithm 1 Active Graph Sampling Algorithm

Input: Adjacency matrix $\mathbf{A}$, $\delta \ll 1$
Initialize: $\mathcal{U}^0 = \mathcal{V}$, $\mathcal{L}^0 = \emptyset$, $\boldsymbol{\mu} = \mathbf{0}$, $\mathbf{G}_0 = (\mathbf{L} + \delta \mathbf{I})^{-1}$
First query is chosen at random
for $t = 1 : T$ **do**
  Scan $\mathcal{U}^{t-1}$ to find best query node v_{k_t} as in (2.8)
  Obtain label y_{k_t} of v_{k_t}
  Update the GMRF mean as in (2.9)
  Update $\mathbf{G}_t$ as in (2.10)
  $\mathcal{U}^t = \mathcal{U}^{t-1}/\{k_t\}$, $\mathcal{L}^t = \mathcal{L}^{t-1} \cup \{k_t\}$
end for
Predict remaining unlabeled nodes as in (2.7)

the graph. Alternatively, δ can be viewed as a regularizer that "pushes" the entries of the Gaussian field $\boldsymbol{\psi}_{\mathcal{U}}$ closer to 0, which also causes the (approximated) marginal posteriors $p(y_i|\mathbf{y}_{\mathcal{L}})$ to be closer to 0.5 (cf. eq. (6)). In that sense, δ enforces the priors $p(y_i = 1) = p(y_i = -1) = 0.5$. Second, the method introduced here for label propagation (cf. (2.5)) is algorithmically similar to the one reported in [141]. The main differences are: i) we perform soft label propagation by minimizing the mean-square prediction error of unlabeled from labeled samples; and ii) our model approximates $\{-1, 1\}$ labels with a *zero-mean* Gaussian field, while the model in [141] approximates $\{0, 1\}$ labels also with a zero-mean Gaussian field (instead of one centered at 0.5). Apparently, [141] treats the two classes differently since it exhibits a bias towards class 0; thus, simply denoting class 0 as class 1 yields different marginal posteriors and classification results. In contrast, our model is bias-free and treats the two classes equally.

2.0.2 Active sampling with GMRFs

In passive learning, $\mathcal{L}$ is either chosen at random, or, it is determined a priori. In our pool based active learning setup, the learner can examine a set of instances (nodes in the context of graph-cognizant classification), and can choose which instances to label. Given its cardinality $|\mathcal{L}|$, one way to approximate the exponentially complex task of selecting $\mathcal{L}$ is to *greedily* sample one node per iteration t with index

$$k_t = \arg \max_{i \in \mathcal{U}^{t-1}} U(v_i, \mathcal{L}^{t-1}) \tag{2.8}$$

where $\mathcal{U}^{t-1}$ is the unlabeled set at time $t-1$, and $U(v, \mathcal{L}^{t-1})$ is a utility function that evaluates how informative node v is while taking into account information already available in $\mathcal{L}^{t-1}$. Upon disclosing label y_{k_t}, it can be shown that instead of re-solving (2.5), the GMRF mean can be updated recursively using the "dongle node" trick in [141] as

$$\boldsymbol{\mu}^{+y_{k_t}}_{\mathcal{U}^{t-1}|\mathcal{L}^{t-1}} = \boldsymbol{\mu}_{\mathcal{U}^{t-1}|\mathcal{L}^{t-1}} + \frac{1}{g_{k_t k_t}}(y_{k_t} - [\boldsymbol{\mu}_{\mathcal{U}^{t-1}|\mathcal{L}^{t-1}}]_{k_t})\mathbf{g}_{k_t} \tag{2.9}$$

where $\boldsymbol{\mu}^{+y_{k_t}}_{\mathcal{U}^{t-1}|\mathcal{L}^{t-1}}$ is the conditional mean of the unlabeled nodes when node v_{k_t} is assigned label y_{k_t} (thus "gravitating" the GMRF mean $[\boldsymbol{\mu}_{\mathcal{U}^{t-1}|\mathcal{L}^{t-1}}]_{k_t}$ toward its replacement y_{k_t}); vector $\mathbf{g}_{k_t} := [\mathbf{L}^{-1}_{\mathcal{U}^{t-1}\mathcal{U}^{t-1}}]_{:k_t}$ and scalar $g_{k_t k_t} := [\mathbf{L}^{-1}_{\mathcal{U}^{t-1}\mathcal{U}^{t-1}}]_{k_t k_t}$ are the k_t-th column and diagonal entry of the Laplacian inverse, respectively. Subsequently, the new conditional mean vector $\mu_{\mathcal{U}^t|\mathcal{L}^t}$ defined over $\mathcal{U}^t$ is given by removing the $i-$th entry of $\mu^{+y_i}_{\mathcal{U}^{t-1}|\mathcal{L}^{t-1}}$. Using Shur's lemma it can be shown that the inverse Laplacian $\mathbf{G}_t^{-k_t}$ when the k_t-th node is removed from the unlabeled sub-graph can be efficiently updated from $\mathbf{G}_t := \mathbf{L}^{-1}_{\mathcal{U}^t\mathcal{U}^t}$ as [89]

$$\begin{bmatrix} \mathbf{G}_t^{-k_t} & \mathbf{0} \\ \mathbf{0}^T & 0 \end{bmatrix} = \mathbf{G}_t - \frac{1}{g_{k_t k_t}}\mathbf{g}_{k_t}\mathbf{g}_{k_t}^T \tag{2.10}$$

which requires only $\mathcal{O}(|\mathcal{U}|^2)$ computations instead of $\mathcal{O}(|\mathcal{U}|^3)$ for matrix inversion. Alternatively, one may obtain $\mathbf{G}_t^{-k_t}$ by applying the matrix inversion lemma employed by the RLS-like solver in [141]. The resultant greedy active sampling scheme for graphs is summarized in Algorithm 1. Note that, existing data-adaptive sampling schemes, e.g., [141], [65], [86], often require *model-retraining* by examining candidate labels per unlabeled node (cf. (2.8)). Thus, even when retraining is efficient, it still needs to be performed $|\mathcal{U}| \times \#\mathrm{Classes}$ times per iteration of Algorithm 1, which in practice significantly increases runtime, especially for larger graphs.

In summary, different sampling strategies emerge by selecting distinct utilities $U(v, \mathcal{L}^{t-1})$ in (2.8). In this context, the goal of the present work is to develop novel active learning schemes within a maximum-expected change framework that achieve high accuracy with a small number of samples. A further desirable attribute of the sought approach is to bypass the need for GMRF retraining.

2.1 Expected model change

Judiciously selecting the utility function is of paramount importance in designing an efficient active sampling algorithm. In the present work, we introduce and investigate the relative merits of different choices under the prism of expected change (EC) measures that we advocate as information-revealing utility functions. From a high-level vantage point, the idea is to identify and sample nodes of the graph that are expected to have the greatest impact on the available GMRF model of the unknown labels. Thus, contrary to the expected error reduction and entropy minimization approaches that actively sample with the goal of increasing the "confidence" on the model, our focus is on effecting maximum perturbation of the model with each node sampled. The intuition behind our approach is that by sampling nodes with large impact, one may take faster steps towards an increasingly accurate model.

2.1.1 EC of model predictions

An intuitive measure of expected model change for a given node v_i is the expected number of unlabeled nodes whose label prediction will change after sampling the label of v_i. To start, consider per node i the measure

$$F(y_i, \boldsymbol{\mu}_{\mathcal{U}|\mathcal{L}}) := \sum_{j \in \mathcal{U}-\{i\}} \mathbb{1}_{\{\hat{y}_j^{+y_i} \neq \hat{y}_j\}} \tag{2.11}$$

where $\hat{y}_j^{+y_i}$ is the predicted label for the $j-$th node after the label of the $i-$th node is revealed, denoting the number of such "flips" in the predicted labels of (2.7). For notational brevity, we henceforth let $\mu_i = [\boldsymbol{\mu}_{\mathcal{U}|\mathcal{L}}]_i$. The corresponding utility function is

$$\begin{aligned} U_{FL}(v_i, \mathcal{L}) &:= \mathbb{E}_{y_i|\mathbf{y}_\mathcal{L}}\left[F(y_i, \boldsymbol{\mu}_{\mathcal{U}|\mathcal{L}})\right] \\ &= p(y_i = 1|\mathbf{y}_\mathcal{L}) F(y_i = 1, \boldsymbol{\mu}_{\mathcal{U}|\mathcal{L}}) \\ &+ p(y_i = -1|\mathbf{y}_\mathcal{L}) F(y_i = -1, \boldsymbol{\mu}_{\mathcal{U}|\mathcal{L}}) \\ &\approx \frac{1}{2}(\mu_i + 1) F(y_i = 1, \boldsymbol{\mu}_{\mathcal{U}|\mathcal{L}}) \\ &+ \left(1 - \frac{1}{2}(\mu_i + 1)\right) F(y_i = -1, \boldsymbol{\mu}_{\mathcal{U}|\mathcal{L}}) \end{aligned} \tag{2.12}$$

where the approximation is because (2.6) was used in place of $p(y_i = 1|\mathbf{y}_{\mathcal{L}})$. Note that model retraining using (2.9) is required to be performed twice (in general, as many as the number of classes) for each node in $\mathcal{U}$ in order to obtain the labels $\{\hat{y}_j\}^{+y_i}$ in (2.11).

2.1.2 EC using KL divergence

The utility function in (2.12) depends on the hard label decisions of (2.7), but does not account for perturbations that do not lead to changes in label predictions. To obtain utility functions that are more sensitive to the soft GMRF model change, it is prudent to measure how much the continuous distribution of the unknown labels changes after sampling. Towards this goal, we consider first the KL divergence between two pdfs $p(\mathbf{x})$ and $q(\mathbf{x})$, which is defined as

$$\mathcal{D}_{KL}(p||q) := \int p(\mathbf{x}) \ln \frac{p(\mathbf{x})}{q(\mathbf{x})} d\mathbf{x} = \mathbb{E}_p \left[\ln \frac{p(\mathbf{x})}{q(\mathbf{x})} \right].$$

For the special case where $p(\mathbf{x})$ and $q(\mathbf{x})$ are Gaussian with identical covariance matrix $\mathbf{C}$ and corresponding means $\mathbf{m}_p$ and $\mathbf{m}_q$, their KL divergence is expressible in closed form as

$$\mathcal{D}_{KL}(p||q) = \frac{1}{2}(\mathbf{m}_p - \mathbf{m}_q)^T \mathbf{C}^{-1} (\mathbf{m}_p - \mathbf{m}_q) \tag{2.13}$$

Upon recalling that $\boldsymbol{\psi}_{\mathcal{U}}$ defined over the unlabeled nodes is Gaussian [cf. (2.2)], and since the Gaussian field obtained after node $v_i \in \mathcal{U}$ is designated label y_i is also Gaussian, we have

$$\boldsymbol{\psi}_{\mathcal{U}}^{+y_i} \sim \mathcal{N}(\boldsymbol{\mu}_{\mathcal{U}|\mathcal{L}}^{+y_i}, \mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}). \tag{2.14}$$

It thus follows that the KL divergence induced on the GMRF after sampling y_i is (cf. (2.13))

$$\begin{aligned} \mathcal{D}_{KL}(\boldsymbol{\psi}_{\mathcal{U}}^{+y_i}||\boldsymbol{\psi}_{\mathcal{U}}) &= \frac{1}{2}\left[(\boldsymbol{\mu}_{\mathcal{U}|\mathcal{L}}^{+y_i} - \boldsymbol{\mu}_{\mathcal{U}|\mathcal{L}})^T \mathbf{L}_{\mathcal{U}\mathcal{U}} (\boldsymbol{\mu}_{\mathcal{U}|\mathcal{L}}^{+y_i} - \boldsymbol{\mu}_{\mathcal{U}|\mathcal{L}}) \right] \\ &= \frac{1}{2g_{ii}^2}(y_i - \mu_i)^2 \mathbf{g}_i^T \mathbf{L}_{\mathcal{U}\mathcal{U}} \mathbf{g}_i = \frac{1}{2g_{ii}}(y_i - \mu_i)^2 \end{aligned} \tag{2.15}$$

where the second equality relied on (2.9), and the last equality used the definition of g_{ii}. The divergence in (2.15) can be also interpreted as the normalized innovation of observation y_i. Averaging (2.15) over the candidate values of y_i yields the expected KL divergence of the GMRF

utility as

$$\begin{aligned} U_{KLG}(v_i, \mathcal{L}) &:= \mathbb{E}_{y_i|\mathbf{y}_\mathcal{L}} \left[\mathcal{D}_{KL}(\boldsymbol{\psi}_\mathcal{U}^{+y_i} || \boldsymbol{\psi}_\mathcal{U}) \right] \\ &= p(y_i = 1|\mathbf{y}_\mathcal{L}) \mathcal{D}_{KL}(\boldsymbol{\psi}_\mathcal{U}^{+y_i=1} || \boldsymbol{\psi}_\mathcal{U}) \\ &+ p(y_i = -1|\mathbf{y}_\mathcal{L}) \mathcal{D}_{KL}(\boldsymbol{\psi}_\mathcal{U}^{+y_i=-1} || \boldsymbol{\psi}_\mathcal{U}) \\ &\approx \frac{1}{2} \Bigg[\frac{1}{2g_{ii}} (\mu_i + 1)(1 - \mu_i)^2 \\ &+ \left(1 - \frac{1}{2}(\mu_i + 1) \right) \frac{1}{g_{ii}} (-1 - \mu_i)^2 \Bigg] \\ &= \frac{1}{2g_{ii}} (1 - \mu_i^2). \end{aligned} \tag{2.16}$$

Interestingly, the utility in (2.16) leads to a form of uncertainty sampling, since $(1 - \mu_i^2)$ is a measure of uncertainty of the model prediction for node v_i, further normalized by g_{ii}, which is the variance of the Gaussian field (cf. [62]). Note also that the expected KL divergence in (2.16) also relates to the information gain between $\{\psi_j\}_{j \in \mathcal{U}/\{i\}}$ and y_i.

Albeit easy to compute since model retraining is not required, U_{KLG} quantifies the impact of disclosing y_i on the GMRF, but not the labels $\{y_j\}_{j \in \mathcal{U}/\{i\}}$ themselves. To account for the labels themselves, an alternative KL-based utility function could treat $\{y_j\}_{j \in \mathcal{U}-\{i\}}$ as Bernouli variables [c.f. (2.6)]; that is

$$y_j \sim \text{Ber}((\mu_j + 1)/2). \tag{2.17}$$

In that case, one would ideally be interested in obtaining the expected KL divergence between the true posteriors, that is

$$\mathbb{E}_{y_i|\mathbf{y}_\mathcal{L}} \left[\mathcal{D}_{KL} \left(p(\mathbf{y}_\mathcal{U}|\mathbf{y}_\mathcal{L}, y_i) || p(\mathbf{y}_\mathcal{U}|\mathbf{y}_\mathcal{L}) \right) \right]. \tag{2.18}$$

Nevertheless, the joint pdfs of the labels are not available by the GMRF model; in fact, any attempt at evaluating the joint posteriors incurs exponential complexity. One way to bypass this limitation is by approximating the joint posterior $p(\mathbf{y}_\mathcal{U}|\mathbf{y}_\mathcal{L})$ with the product of marginal posteriors $\prod_{j \in \mathcal{U}} p(\mathbf{y}_j|\mathbf{y}_\mathcal{L})$, since the later are readily given by the GMRF. Using this independence assumption causes the joint KL divergence in (4.23) to break down to the sum of marginal

per-unlabeled-node KL divergences. The resulting utility function can be expressed as

$$U_{KL}(v_i, \mathcal{L}) := \sum_{j \in \mathcal{U}/\{i\}} I(y_j, y_i) \tag{2.19}$$

where

$$\begin{aligned} I(y_j, y_i) &:= \mathbb{E}_{y_i|\mathbf{y}_\mathcal{L}} \left[\mathcal{D}_{KL} \left(p(y_j|\mathbf{y}_\mathcal{L}, y_i) || p(y_j|\mathbf{y}_\mathcal{L}) \right) \right] \\ &\approx \frac{1}{2}(\mu_i + 1) \mathcal{D}_{KL}(y_j^{+y_i=1} || y_j) \\ &+ \left(1 - \frac{1}{2}(\mu_i + 1) \right) \mathcal{D}_{KL}(y_j^{+y_i=-1} || y_j) \end{aligned} \tag{2.20}$$

since for univariate distributions the expected KL divergence between the prior and posterior is equivalent to the mutual information between the observed random variable and its unknown label. Note also that the KL divergence between univariate distributions is simply

$$\mathcal{D}_{KL}(y_j^{+y_i} || y_j) = H(y_j^{+y_i}, y_j) - H(y_j^{+y_i}) \tag{2.21}$$

where $H(y_j^{+y_i}, y_j)$ denotes the cross-entropy, which for Bernouli variables is

$$\begin{aligned} H(y_j^{+y_i}, y_j) = &-\frac{1}{2}(\mu_j^{+y_i} + 1) \log \frac{1}{2}(\mu_j + 1) \\ &- \left[1 - \frac{1}{2}(\mu_j^{+y_i} + 1) \right] \log \left[1 - \frac{1}{2}(\mu_j + 1) \right]. \end{aligned} \tag{2.22}$$

Combining (2.19)-(2.22) yields U_{KL}. Intuitively, this utility promotes the sampling of nodes that are expected to induce large change on the model (cross-entropy between old and new distributions), while at the same time increasing the "confidence" on the model (negative entropy of updated distribution). Furthermore, the mutual-information-based expressions (19) and (20) establish a connection to the information-based metrics in [86] and [75], giving an expected-model-change interpretation of the entropy reduction method.

2.1.3 EC without model retraining

In this section, we introduce two measures of model change that do not require model retraining (cf. Remark 3), and hence are attractive in their simplicity. Specifically, retraining (i.e., computing

$\boldsymbol{\mu}^{+y_i}_{\mathcal{U}^{t-1}|\mathcal{L}^{t-1}},\ \forall i \in \mathcal{U}$ and $\forall y_i \in \mathcal{Y}$) is not required if per-node utility $U(v, \mathcal{L}^{t-1})$ can be given in *closed-form* as a function of $\mathbf{G}_{t-1}$ and $\boldsymbol{\mu}_{\mathcal{U}^{t-1}|\mathcal{L}^{t-1}}$. Two such measures are explored here: one based on the sum of total variations that a new sample inflicts on the (approximate) marginal distributions of the unknown labels, and one based on the mean-square deviation that a new sample is expected to inflict on the GMRF.

The *total variation* between two probability distributions $p(x)$ and $q(x)$ over a finite alphabet $\mathcal{X}$ is

$$\delta(p, q) := \frac{1}{2} \sum_{x \in \mathcal{X}} |p(x) - q(x)|.$$

Using the approximation in (2.6), the total variation between the distribution of an unknown label y_j and the same label $y_j^{+y_i}$ after y_i becomes available is

$$\begin{aligned} \delta(y_j^{+y_i}, y_j) &= \frac{1}{2} \left(|\mu_j^{+y_i} - \mu_j| + |1 - \mu_j^{+y_i} - (1 - \mu_j)| \right) \\ &= |\mu_j^{+y_i} - \mu_j|. \end{aligned} \tag{2.23}$$

Consequently, the sum of total variations over all the unlabeled nodes $\{v_j\}_{j \in \mathcal{U}/[\{i\}]}$ is

$$\begin{aligned} \Delta(\mathbf{y}_{\mathcal{U}}^{+y_i}, \mathbf{y}_{\mathcal{U}}) &:= \sum_{j \in \mathcal{U}} \delta(y_j^{+y_i}, y_j) = \|\boldsymbol{\mu}_{\mathcal{U}|\mathcal{L}}^{+y_i} - \boldsymbol{\mu}_{\mathcal{U}|\mathcal{L}}\|_1 \\ &= \frac{1}{g_{ii}} |y_i - \mu_i| \|\mathbf{g}_i\|_1 \end{aligned}$$

where the second equality follows by concatenating all total variations (cf. (2.23)) in vector form, and the last one follows by the GMRF update rule in (2.9). Finally, the expected sum of total variations utility score-function is defined as

$$\begin{aligned} U_{TV}(v_i, \mathcal{L}) &:= \mathbb{E}_{y_i|\mathbf{y}_{\mathcal{L}}} \left[\Delta(\mathbf{y}_{\mathcal{U}}^{+y_i}, \mathbf{y}_{\mathcal{U}}) \right] \\ &= \mathbb{E}_{y_i|\mathbf{y}_{\mathcal{L}}} \left[|y_i - \mu_i| \right] \frac{1}{g_{ii}} \|\mathbf{g}_i\|_1 \end{aligned}$$

and since

$$\mathbb{E}_{y_i|\mathbf{y}_{\mathcal{L}}} \left[|y_i - \mu_i| \right] = p(y_i = 1|\mathbf{y}_{\mathcal{L}})|1 - \mu_i| + p(y_i = -1|\mathbf{y}_{\mathcal{L}})| - 1 - \mu_i| \approx 2(1 - \mu_i^2)$$

it follows that the utility function based on total variation can be expressed as

$$U_{TV}(v_i, \mathcal{L}) = \frac{2}{g_{ii}}(1 - \mu_i^2)\|\mathbf{g}_i\|_1. \tag{2.24}$$

The second measure is based on the *mean-square deviation* (MSD) between two RV's X_1 and X_2

$$\begin{aligned}\mathrm{MSD}(X_1, X_2) &:= \int (X_1 - X_2)^2 f(X_1, X_2) dX_1 dX_2 \\ &= \mathbb{E}\left[(X_1 - X_2)^2\right].\end{aligned}$$

Our next proposed utility score is the expected MSD between the Gaussian fields $\boldsymbol{\psi}_{\mathcal{U}}$ and $\boldsymbol{\psi}_{\mathcal{U}}^{+y_i}$ before and after obtaining y_i; that is,

$$\begin{aligned}U_{MSD}(v_i, \mathcal{L}) &= \mathbb{E}_{y_i|\mathbf{y}_{\mathcal{L}}}\left[\mathrm{MSD}(\boldsymbol{\psi}_{\mathcal{U}}^{+y_i}, \boldsymbol{\psi}_{\mathcal{U}})\right] \\ &\approx \frac{1}{2}(\mu_i + 1)\mathrm{MSD}(\boldsymbol{\psi}_{\mathcal{U}}^{+y_i=1}, \boldsymbol{\psi}_{\mathcal{U}}) \\ &+ \left(1 - \frac{1}{2}(\mu_i + 1)\right)\mathrm{MSD}(\boldsymbol{\psi}_{\mathcal{U}}^{+y_i=-1}, \boldsymbol{\psi}_{\mathcal{U}})\end{aligned} \tag{2.25}$$

where

$$\begin{aligned}\mathrm{MSD}(\boldsymbol{\psi}_{\mathcal{U}}^{+y_i}, \boldsymbol{\psi}_{\mathcal{U}}) &:= \mathbb{E}\left[\|\boldsymbol{\psi}_{\mathcal{U}}^{+y_i} - \boldsymbol{\psi}_{\mathcal{U}}\|^2\right] \\ &= 2\mathrm{tr}(\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}) + \|\boldsymbol{\mu}_{\mathcal{U}|\mathcal{L}}^{+y_i} - \boldsymbol{\mu}_{\mathcal{U}|\mathcal{L}}\|_2^2 \\ &\propto \frac{1}{g_{ii}^2}(y_i - \mu_i)^2\|\mathbf{g}_i\|_2^2.\end{aligned} \tag{2.26}$$

The second equality in (2.26) is derived in Appendix A2 under the assumption that $\boldsymbol{\psi}_{\mathcal{U}}$ and $\boldsymbol{\psi}_{\mathcal{U}}^{+y_i}$ are independent random vectors. Furthermore, the term $2\mathrm{tr}(\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1})$ is ignored since it does not depend on y_i, and the final expression of (2.26) is obtained using (2.9). Finally, substituting (2.26) into (2.25) yields the following closed-form expression of the MSD-based utility score function

$$U_{MSD}(v_i, \mathcal{L}) \propto (1 - \mu_i^2)\frac{\|\mathbf{g}_i\|_2^2}{g_{ii}^2}. \tag{2.27}$$

Note that U_{TV} and U_{MSD} are proportional to the expected KL divergence of the Gaussian field U_{KLG} in the previous section since

$$U_{TV}(v_i, \mathcal{L}) \propto U_{KLG}(v_i, \mathcal{L}) \|\mathbf{g}_i\|_1 \tag{2.28}$$

and

$$U_{MSD}(v_i, \mathcal{L}) \propto U_{KLG}(v_i, \mathcal{L}) \|\mathbf{g}_i\|_2 \tag{2.29}$$

with the norms $\|\mathbf{g}_i\|_1$ and $\|\mathbf{g}_i\|_2$ quantifying the average influence of the $i-$th node over the rest of the unlabeled nodes.

It is worth mentioning that our TV- and MSD-based methods relate to the $\Sigma-$optimality-based active learning [89] and the variance minimization [62] correspondingly. This becomes apparent upon recalling that $\Sigma-$optimality and variance-minimization utility score functions are respectively given by

$$U_{\Sigma-opt}(v_i) = \frac{\|\mathbf{g}_i\|_1^2}{g_{ii}}$$

and

$$U_{VM}(v_i) := \frac{\|\mathbf{g}_i\|_2^2}{g_{ii}}.$$

Then, further inspection reveals that the metrics are related by

$$U_{TV}(v_i) \propto \frac{1}{g_{ii}}(1 - \mu_i^2) U_{\Sigma-opt}(v_i) \tag{2.30}$$

and correspondingly

$$U_{MSD}(v_i) \propto \frac{1}{g_{ii}}(1 - \mu_i^2) U_{VM}(v_i). \tag{2.31}$$

In fact, U_{TV} and U_{MSD} may be interpreted as *data-driven* versions of $U_{\Sigma-opt}$ and U_{VM} that are enhanced with the uncertainty term $g_{ii}^{-1}(1 - \mu_i^2)$. On the one hand, $U_{\Sigma-opt}$ and U_{VM} are design-of-experiments-type methods that rely on ensemble criteria and offer *offline* sampling schemes more suitable for applications where the set $\mathcal{L}$ of nodes may *only* be labeled as a *batch*. On the other hand, U_{TV} and U_{MSD} are data-adaptive sampling schemes that adjust to the *specific realization* of labels, and are expected to outperform their batch counterparts in general. This connection is established due to $U_{VM}(v_i)$ and $U_{\Sigma-opt}(v_i)$ being l_2 and l_1 *ensemble* loss metrics on the GMRF (see equations 2.3 and 2.5 in [12]); similarly, MSD (mean square deviation) and

Table 2.1: Summary of EC methods based on different metrics of change

Method	Change metric	Retraining	Utility function
FL	# of flipped labels	Yes	Eq. (11) and (12)
KLG	KL divergence of GMRF	No	$\propto \frac{1}{g_{ii}}(1-\mu_i^2)$
KL	KL divergence of (Bernouli) discrete labels	Yes	Eq. (19) – (22)
MSD	Mean-square deviation of GMRF	No	$\propto (1-\mu_i^2)\frac{\|\mathbf{g}_i\|_2^2}{g_{ii}^2}$
TV	Total variation of discrete labels	No	$\propto (1-\mu_i^2)\frac{\|\mathbf{g}_i\|_1}{g_{ii}}$

TV (total variation) are also l_2 (on the GMRF distribution) and l_1 (on the binary labels pmf) metrics of *change*.

Note that, while the proposed methods were developed for binary classification, they can easily be modified to cope with multiple classes using the one-vs-the-rest trick. Specifically, for any set $\mathcal{C}$ of possible classes, it suffices to solve $|\mathcal{C}|$ binary problems, each one focused on detecting the presence or absence of a class. Consequently, the maximum among the GMRF means $\mu_i^{(c)}\ \forall c \in \mathcal{C}$ reveals which class is the most likely for the $i-$th node. In addition, the marginal posteriors are readily given by normalizing the binary posteriors in (2.6), that is

$$p(y_i = c) = \bar{\mu}_i^{(c)} = \frac{\mu_i^{(c)} + 1}{\sum_{c\in\mathcal{C}}(\mu_i^{(c)} + 1)}.$$

Using this approximation, the TV-based scheme can be generalized to

$$U_{TV}(v_i, \mathcal{L}) \propto \sum_{c\in\mathcal{C}} \left[1 - (\bar{\mu}_i^{(c)})^2\right] \frac{\|\mathbf{g}_i\|_1}{g_{ii}} \tag{2.32}$$

and similarly for the MSD-based scheme.

A summary of the five different methods that were considered in the context of the proposed EC-based active learning framework is given in Table I.

2.1.4 Computational Complexity analysis

The present section analyzes the computational complexity of implementing the proposed adaptive sampling methods, as well as that of other common adaptive and non-adaptive active learning approaches on graphs. Complexity here refers to float-point multiplications and is given

Table 2.2: Computational and memory complexity of various methods

	Offline	Sampling	Update	Memory
Random	$\mathcal{O}(\lvert\mathcal{E}\rvert\lvert\mathcal{C}\rvert)$	$*$	$*$	$\mathcal{O}(\lvert\mathcal{E}\rvert + N\lvert\mathcal{C}\rvert)$
VM [62], Σ-opt [89]	$\mathcal{O}(\lvert\mathcal{L}\rvert N^2)$	$*$	$*$	$\mathcal{O}(N^2)$
Uncertainty (min. margin)	$*$	$\mathcal{O}(\log \lvert\mathcal{C}\rvert N)$	$\mathcal{O}(\lvert\mathcal{E}\rvert\lvert\mathcal{C}\rvert)$	$\mathcal{O}(\lvert\mathcal{E}\rvert + N\lvert\mathcal{C}\rvert)$
EER [140], TSA [65]	$\mathcal{O}(\lvert\mathcal{E}\rvert N)$	$\mathcal{O}(\lvert\mathcal{C}\rvert^2 N^2)$	$\mathcal{O}(N^2)$	$\mathcal{O}(N^2)$
FL	$\mathcal{O}(\lvert\mathcal{E}\rvert N)$	$\mathcal{O}(\lvert\mathcal{C}\rvert N^2)$	$\mathcal{O}(N^2)$	$\mathcal{O}(N^2)$
TV, MSD	$\mathcal{O}(\lvert\mathcal{E}\rvert N)$	$\mathcal{O}(\lvert\mathcal{C}\rvert N)$	$\mathcal{O}(N^2)$	$\mathcal{O}(N^2)$

in $\mathcal{O}(\cdot)$ notation as function of the number of nodes N, number of edges $|\mathcal{E}|$ and number of classes $|\mathcal{C}|$. Three types of computational tasks are considered separately: computations that can be performed offline (e.g., initialization), computations required to update model after a new node is observed (only for adaptive methods), and the complexity of selecting a new node to sample (cf. eq. (2.8)).

Let as begin with the "plain-vanilla" label propagation scenario where nodes are randomly (or passively) sampled. In that case, the online framework described in Algorithm 1 and Section II.B is not necessary and the nodes can be classified offline after collecting $|\mathcal{L}|$ samples and obtaining (2.5) for each class in $\mathcal{C}$. Exploiting the sparsity of the $\mathbf{L}$, (2.5) can be approximated via a Power-like iteration (see, e.g., [128]) with $\mathcal{O}(|\mathcal{E}||\mathcal{C}|)$ complexity. Similarly to passive sampling, non-adaptive approaches such as the variance-minimization (VM) in [62] and Σ-opt design in [89] can also be implemented offline. However, unlike passive sampling, the non-adaptive sampling methods require computation of $\mathbf{G}_0 = (\mathbf{L} + \delta\mathbf{I})^{-1}$, which can be approximated with $\mathcal{O}(|\mathcal{E}|N)$ multiplications via the Jacobi method. The offline complexity of VM and Σ-opt is dominated by the complexity required to design the label set $\mathcal{L}$ which is equivalent to $|\mathcal{L}|$ iterations of Algorithm 1 using $U_{VM}(v_i)$ and $U_{\Sigma-opt}(v_i)$ correspondingly. Thus, the total offline complexity of VM and Σ-opt is $\mathcal{O}(|\mathcal{L}|N^2)$, while $\mathcal{O}(N^2)$ memory is required to store and process $\mathbf{G}_0$.

In the context of adaptive methods, computational efficiency largely depends on whether matrix $\mathbf{G}$ is used for sampling and updating. Simple methods such as uncertainty sampling based on minimum margin do not require $\mathbf{G}$ and have soft labels updated after each new sample using

iterative label-propagation (see, e.g., [86]) with $\mathcal{O}(|\mathcal{E}||\mathcal{C}|)$ complexity. Uncertainty-sampling-based criteria are also typically very lightweight requiring for instance sorting class-wise the soft labels of each node ($\mathcal{O}(\log|\mathcal{C}|N)$ per sample). While uncertainty-based methods are faster and more scalable, their accuracy is typically significantly lower than that of more sophisticated methods that use $\mathbf{G}$. Methods that use $\mathbf{G}$ such as the proposed EC algorithms in Section III, the expected-error minimization (EER) in [141], and the two-step approximation (TSA) algorithm in [65] all require $\mathcal{O}(N^2)$ to perform the update in (2.10). However, TSA and EER use retraining (cf. Remark 3) that incurs complexity $\mathcal{O}(|\mathcal{C}|^2N^2)$ to perform one sample selection; computing the "expected error" requires fictitiously labeling every unlabeled node, and re-computing the metric by treating the fictitious label as the true label. More specifically, consider the normalization of the binary posteriors that is required (similar to the one discussed in Remark 4) in order to define a posterior pmf over multiple classes ($|\mathcal{C}| > 2$). Normalization entails $|\mathcal{C}|$ divisions, and happens $|\mathcal{C}|$ times (once for every possible label of an unlabeled node). This gives rise to a nested loop where the outer loop repeats $|\mathcal{C}|$ times and the inner loop requires $|\mathcal{C}|N$ computations, yielding a total complexity $\mathcal{O}(|\mathcal{C}|^2N)$ for computing the expected error score for one node. Since these scores have to be computed over all unlabeled nodes (in order to select the best one), the overall complexity to obtain a sample according to EER or TSA is $\mathcal{O}(|\mathcal{C}|^2N^2)$. In contrast, the proposed MSD and TV methods (cf. (2.24), (2.27)) only require $\mathcal{O}(|\mathcal{C}|N)$ for sampling. Note that the performance gap between EER and TSA on the one hand, and TV and MSD on the other grows as the number of classes $|\mathcal{C}|$ increases.

The complexity analysis is summarized in Table II and indicates that the proposed *retraining-free* adaptive methods exhibit lower overall complexity than EER and TSA. An important modification is proposed in the ensuing section in order to deal with the challenge of *bias* that is inherent to all data-adaptive sampling schemes.

2.2 Promoting exploration by adjusting model confidence

It has been observed that active learning schemes may become "myopic" [113], meaning that they become overly focus on *exploiting* (focusing on) a small region of the sample space, and neglect *exploration*. Uncertainty sampling in particular can be prone to such behavior, due to the fact that it is more "myopic," in the sense that it does not take into account the effect of a potential sample on the generalization capabilities of the classifier. Since the TV- and MSD-based

utility score functions in (2.24) and (2.26) are influenced by the uncertainty factor $(1-\mu_i^2)$, it is important to mitigate this effect before testing the performance of the proposed approaches.

Let us begin by observing that most active learning methods, including those based on EC we introduced here, are based on utility score functions that take the general form

$$U(v_i, \mathcal{L}) = \mathbb{E}_{y_i|\mathbf{y}_\mathcal{L}}\left[C(y_i, \mathcal{L})\right] \tag{2.33}$$

where $C(y_i, \mathcal{L})$ is any metric that evaluates the effect of node v_i on the model, *given that* its label is y_i. Using the existing probability model to predict how the model itself will change, induces "myopic" behavior especially in the early stages of the sampling process when the inferred model is most likely far from the true distribution.

One possible means of reducing bias is by complementing greedy active learning strategies with random sampling. Thats is, instead of selecting the index k_t of the node to be sampled at the $t-$th iteration according to (2.8), one can opt for a two-branch hybrid rule

$$k_t = \begin{cases} \arg\max_{i\in\mathcal{U}^{t-1}} U(v_i, \mathcal{L}^{t-1}), & \text{w.p.} \quad (1-\pi^t) \\ \text{Unif}\{1, \ldots, |\mathcal{L}^{t-1}|\}, & \text{w.p.} \quad \pi^t \end{cases}. \tag{2.34}$$

where π^t is the probability that at iteration t the sampling strategy switches to uniform random sampling over the unlabeled nodes. Naturally, one should generally select a sequence $\{\pi^t\}$ such that $\pi^t \to 0$ as t increases the model becomes more accurate. Upon testing the simple heuristic in (2.34) we observed that it can significantly improve the performance of the more "myopic" active sampling strategies. Specifically, uncertainty sampling which relies purely on exploitation can be greatly enhanced by completing it with the exploration queries introduced by (2.34).

Another option is to sample nodes that maximize the minimum over all possible labels change. That is, instead of (2.33) one can adopt utility scores of the general form

$$U(v_i, \mathcal{L}) = \min_{y_i\in\{-1,1\}} C(y_i, \mathcal{L}). \tag{2.35}$$

Albeit intuitive, (2.34) is not as appropriate for promoting exploration in more sophisticated strategies such as the ones presented in this work, because it does not account for the graph structure, and it is somewhat aggressive in assuming that with probability π^t the model is completely uninformative. For similar reasons, (2.35) also does not produce satisfactory results.

In the present section, we introduce a "softer" heuristic that is better tailored to the sampling strategies at hand. The main idea is to implement $U(v_i, \mathcal{L})$ in (2.33) using a different set of probabilities than the ones provided by the model (cf. (2.6)). Specifically, we suggest to use label predictions that are closer to a "non-informative" prior early on, and gradually converge to the ones provided by the trained model as our confidence on the latter increases. Thus, instead of taking the expectation in (2.33) over $p\left(y_i|\mathbf{y}_{\mathcal{L}}\right)$, one may instead use a convex combination of the latter and a node prior $\pi(y_i)$, that is

$$\check{p}\left(y_i|\mathbf{y}_{\mathcal{L}}; \alpha_t\right) = \alpha_t \pi(y_i) + (1 - \alpha_t) p\left(y_i|\mathbf{y}_{\mathcal{L}}\right) \tag{2.36}$$

where $0 \leq \alpha_t \leq 1$ is a constant that quantifies the confidence on the current estimate of the posterior. If no prior is available, one may simply use $\pi(y_i = 1) = \pi(y_i = -1) = 1/2$. Intuitively pleasing results were obtained when combining (2.36) with EC methods. For instance, combining (2.36) with our proposed TV method yields the following modified MSD utility score function

$$U_{MSD}(v_i, \mathcal{L}, a_t) \propto \left[a_t + (1 - a_t)(1 - \mu_i^2)\right] \frac{\|\mathbf{g}_i\|_2^2}{g_{ii}^2} \tag{2.37}$$

where a_t tunes the sensitivity of the sampling process to the uncertainty metric $(1 - \mu_i^2)$. As more samples become available, the confidence that the current estimate of the posterior is close to the true distribution may increase. Thus, instead of using a constant α throughout the sampling process, one may use a sequence $\{\alpha_t\}_{t=1}^{T}$, where t is the iteration index, T the total number of samples, and a_t is *inversely proportional* to t. Finally, note that by setting $\alpha_t = 1 \forall t$ the uncertainty terms vanish with MSD and TV becoming non-adaptive.

2.3 Experimental Results

The present section includes numerical experiments carried to assess the performance of the proposed methods in terms of prediction accuracy. Specifically, the ensuing subsections present plots of accuracy

$$\text{Accuracy} = \frac{1}{|\mathcal{U}|} \sum_{i \in \mathcal{U}} \mathbf{1}_{\{\hat{y}_i = y_i\}}$$

as a function of the number of nodes sampled by the GMRF-based active learning algorithms (cf. Algorithm 1). We compare the proposed methods (number of flips (FL), KL divergence, MSD, sum of TVs) with the variance minimization (VM) [62], $\Sigma-$optimality [89], expected error minimization (EER) [141], and two-step approximation method (TSA) [65]. Furthermore, we compare with the minimum-margin uncertainty sampling (UNC) scheme that samples the node with smallest difference between the largest soft labels, which is equivalent to using the utility function $U_{UNC}(v_i, \mathcal{L}) := -|\mu_i^{(c_1)} - \mu_i^{(c_2)}|$, where c_1 and c_2 is the most-probable and second-most probable class for node v_i correspondingly. Finally, all methods are compared to the predictions that are given by the GMRF method used here (cf. (2.2)-(2.7)) with nodes sampled randomly (passive learning). For all graph tested the prediction accuracy remained high for a large range of $\delta \in [0.1, 0.001]$ with the exact value tuned for every graph in order to maximize accuracy for passive (random) sampling.

2.3.1 Synthetic graphs

Following [65], we first considered a 10×10 rectangular grid similar to the one in Fig. 1, where each node is connected to four neighboring nodes. Red dots correspond to nodes belonging to class 1, and uncolored intersections correspond to nodes belonging to class -1. To make the classification task more challenging, the class 1 region was separated into two 3×3 squares (upper left and lower right) and additional class 1 nodes were added w.p. 0.5 along the dividing lines. Plotted in Fig. 3 is the accuracy-vs-number of samples performance averaged over 50 Monte Carlo runs. As expected, most algorithms outperform random sampling. In addition, one observes that purely exploratory non-adaptive methods (VM and $\Sigma-$optimality) enjoy relatively high accuracy for a small number of samples, but are eventually surpassed by adaptive methods. It can also be observed that the novel TV method with $a_t = t^{-1/2}$ performs equally well to the state-of-the-art TSA method. Interestingly, it does so while using a much simpler criterion that avoids model retraining, and therefore requires significantly shorter runtime. Note finally that the performance of ERR is poor because the sampler easily becomes "trapped" in one of the two class 1 regions, and does not explore the graph.

The purpose of the experiment in Fig. 1 was to simulate problems where a complex label distribution appears on a simple uniform graph (e.g., image segmentation). To simulate more structured graphs, we generated a 1000-node network using the Lancichinetti–Fortunato–Radicchi (LFR) method [78]. The LFR algorithm is widely used to generate benchmark graphs

that resemble real world networks by exhibiting community structure and degree distributions that follow the power law. Figure 2 reveals the sparsity pattern of the adjacency matrix of the LFR graph that was used, while the 3 clearly visible clusters correspond to groups of nodes in the same class, that is

$$y_i = \begin{cases} 1, & i \in [1, 250] \\ 2, & i \in [251, 600] \\ 3, & i \in [601, 1000] \end{cases}$$

Note that, unlike the one in Fig. 1, the graph used here is characterized by a community structure that matches the nodes labels. This is a highly favorable scenario for the non-adaptive VM and Σ-opt approaches that rely solely on the graph structure. Indeed, as seen in Fig. 4, VM and Σ-opt quickly reach 90% accuracy by selecting 5 most influential samples. Nevertheless, between 5 and 10 samples our proposed MSD and TV adaptive methods enjoy superior accuracy before converging to 100% accuracy.

2.3.2 Similarity graphs from real datasets

Real binary classification datasets taken from the UC Irvine Machine Learning Repository [4] and the LibSVM webpage [6] were used for further testing of the proposed methods. First, each entry of the feature vectors was normalized to lie between -1 and 1. Then, a graph was constructed using the Pearson correlations among pairs of normalized feature vectors as weights of the adjacency matrix $\mathbf{W}$; thresholding was also applied to negative and small weights leading to sparse adjacency matrices. It was observed that sparsification generally improves the prediction accuracy, while also reducing the computational burden. In the presented experiments, thresholds were tuned until one of the methods achieved the highest possible classification accuracy.

Having constructed the graphs, the proposed expected model change sampling schemes were compared with UNC, TSA, EER, VM and $\Sigma-$optimality on seven real datasets listed in Table III; in the latter, "baseline accuracy" refers to the proportion of the largest class in each dataset, and thus the highest accuracy that can be achieved by naively assuming that all labels belong to the majority class. Plotted in Figs. 5 to 10 are the results of the numerical tests, where it is seen that the performance of the proposed low-complexity TV- and MSD-based scheme is comparable or superior to that of competing alternatives. The confidence parameter was set to $a_t = 1/\sqrt{t}$ for the smaller datasets, where only few data were sampled, and the model was expected to be less accurate, whereas for the larger ones it was set to $a_t = 0$.

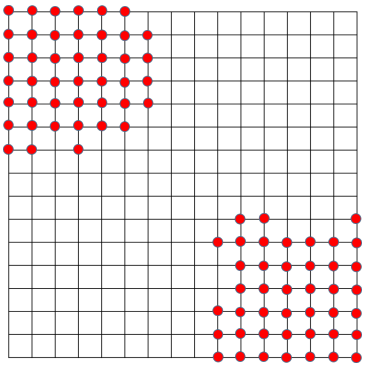

Figure 2.1: Rectangular grid synthetic graph with two separate class 1 regions.

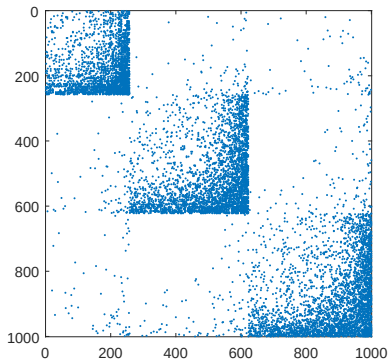

Figure 2.2: Adjacency matrix of LFR graph with 1,000 nodes and 3 classes.

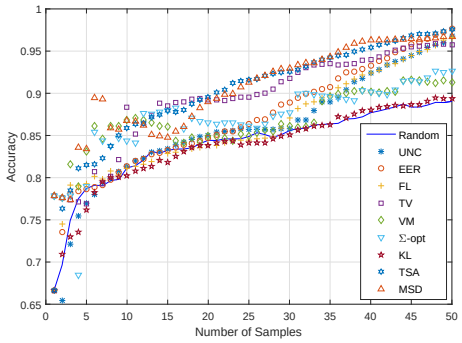

Figure 2.3: Test results for synthetic grid in Fig. 1 .

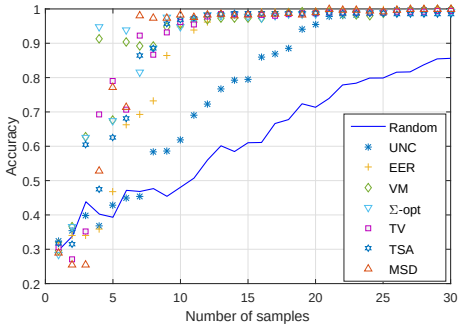

Figure 2.4: Test results for synthetic LFR graph in Fig. 2.

2.3.3 Real graphs

Experiments were also performed on real labeled graphs. Specifically, the CORA and CITESEER [5] citation networks with 2708 and 3312 nodes correspondingly were used; similarly to [89], we isolated the largest connected components. In citation networks, each node corresponds to a scientific publication and is linked only with cited or citing papers. Nodal labels correspond to the scientific field that each paper belongs to (6 classes for CITESEER and 7 for CORA). Lately, CORA and CITESEER have been used as benchmarks for graph convolutional neural networks (GCNs) [94], as well as for classification based on node embeddings (Planetoid-G) [133]. For this reason, together with the GMRF-based passive (random) sampling benchmark, we also use GCNs and Planetoid-G as passive benchmarks. Note that the latter two methods

Table 2.3: Dataset list

Dataset	# of nodes	Baseline Accuracy
Coloncancer	62	0.64
Ionosphere	351	0.64
Leukemia	70	0.65
Australian	690	0.55
Parkinsons	191	0.75
Ecoli	326	0.57

require validation samples for early stopping during training, while GMRF does not. In order to proceed with comparisons, a set of validation samples was given to GCN and Planetoid-G equal in size to the training set. The benchmark political-blog network [7] with 1490 nodes and two classes was also used. The confidence sequence $\alpha_t = t^{-1/2}$ was used for all graphs, with $\delta = 0.005$ similarly to [11]. The results of the experiments are depicted in Figs. 11-13 and demonstrate the effectiveness of the proposed MSD and TV algorithms on these social graphs. For the CORA network, TV achieves state of the art performance equal to EER, TSA and Σ-opt, while for the CITESEER network its accuracy slightly surpasses that of competing methods. Note that for both citation networks and given randomly selected samples, the performance of GCNs and Planetoid-G barely reaches that of the simple GMRF classifier. This is mostly attributed to the latter being more suitable for the *graph-only* setting that we are dealing with here, whereas the former mostly exploit node features that are available for citation networks (bag-of-words description of abstracts). For the political-blogs network, non-adaptive TV and Σ-opt methods perform poorly, while the proposed MSD method performs at least as good as the significantly more complex TSA. The bar plot in Fig. 14 depicts the relative runtimes of different adaptive methods. Observe that MSD and TV are two orders of magnitude faster than EER and TSA for the larger multilabel citation graphs, and one order of magnitude faster for the smallest binary-labeled political blogs network.

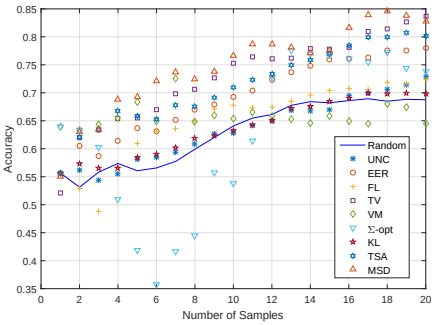

Figure 2.5: Coloncancer dataset.

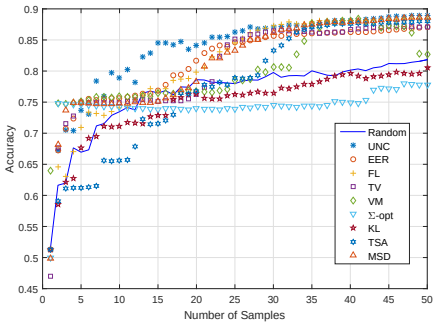

Figure 2.6: Ionosphere dataset.

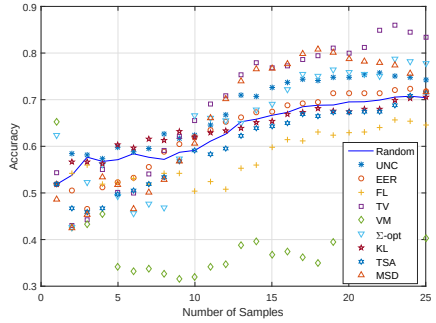

Figure 2.7: Leukemia dataset.

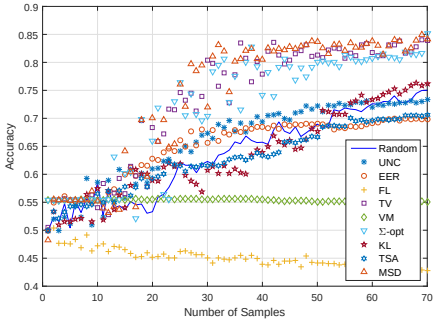

Figure 2.8: Australian dataset.

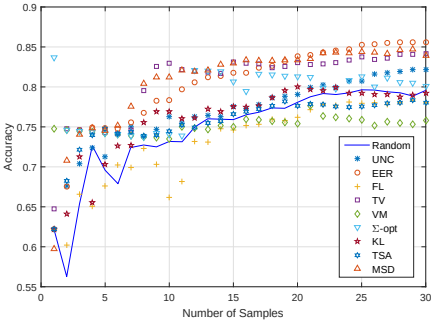

Figure 2.9: Parkinsons dataset.

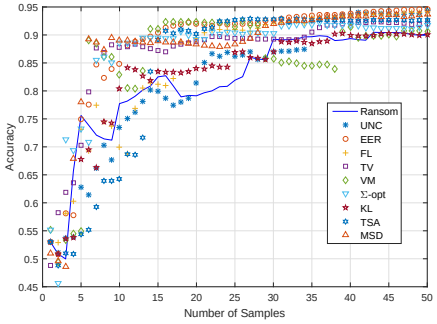

Figure 2.10: Ecoli dataset.

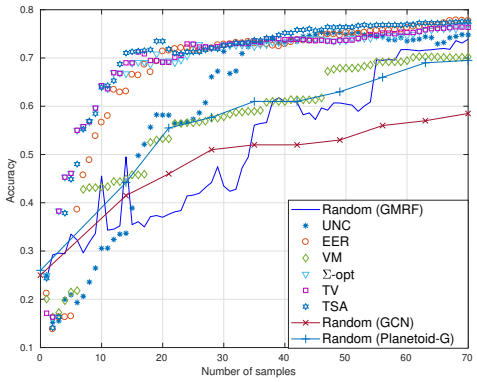

Figure 2.11: CORA citation network.

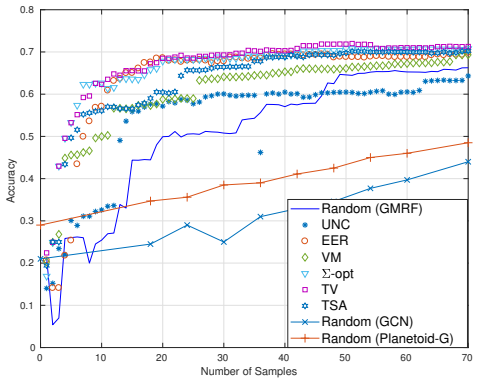

Figure 2.12: CITESEER citation network.

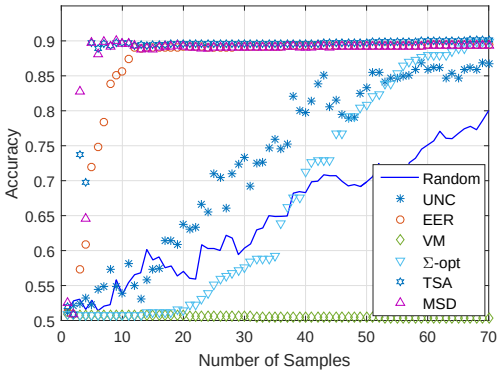

Figure 2.13: Political blogs network.

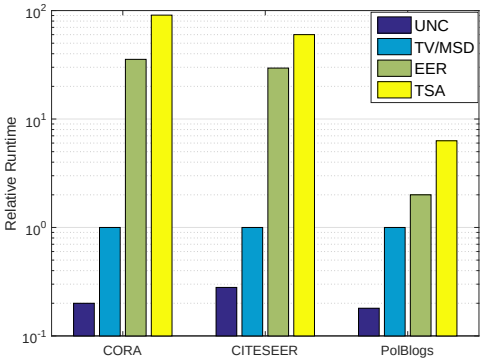

Figure 2.14: Relative runtime of different adaptive methods for experiments on real social graphs.

Chapter 3

Scalable Classification over Graphs with Adaptive Diffusions

The present Section further studies the node classification task, and develops a flexible yet scalable framework based on diffusions and random-walks.

In general, node classifiers rely on a certain measure of node-to-node influence. Then, 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. [30, 77, 127]. 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, meaning with state space $\mathcal{V}$. The transition probabilities are

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

where $X_k \in \mathcal{V}$ denotes the position of the random walker (state) at the k^{th} step; $d_j := \sum_{k\in\mathcal{N}_j} A_{kj}$ is the degree of node j; and, $\mathcal{N}_j$ its neighborhood. Since we consider undirected graphs the limiting distribution of $\{X_k\}$ always exists and it is unique if it is connected and non-bipartite. It is given by the dominant right eigenvector of the column-stochastic transition probability matrix $\mathbf{H} := \mathbf{A}\mathbf{D}^{-1}$, where $\mathbf{D} := \operatorname{diag}(d_1, d_2, \ldots, d_N)$ [81]. 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

$$p_i^{(k)} := \sum_{j\in\mathcal{V}} \Pr\{X_k = i | X_0 = j\} \Pr\{X_0 = j\} \tag{3.1}$$

that in vector form $\mathbf{p}^{(k)} := [p_1^{(k)} \ \dots \ p_N^{(k)}]^\mathsf{T}$ 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. (3.1))

$$\mathbf{p}_c^{(k)} = \mathbf{H}^k \mathbf{v}_c \tag{3.2}$$

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

$$[\mathbf{v}_c]_i = \begin{cases} 1/|\mathcal{L}_c|, & i \in \mathcal{L}_c \\ 0, & \text{else} \end{cases} \tag{3.3}$$

acts as initial distribution. Using (3.2), we model diffusions per class c over the graph driven by $\{\mathbf{p}_c^{(k)}\}_{k=0}^K$ as

$$\mathbf{f}_c(\boldsymbol{\theta}) = \sum_{k=0}^{K} \theta_k \mathbf{p}_c^{(k)} \tag{3.4}$$

where θ_k denotes the importance assigned to the k^{th} hop neighborhood. By setting $\theta_0 = 0$ (since it is not useful for classification purposes) and 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, $\mathbf{f}_c(\boldsymbol{\theta})$ can be compactly

expressed as

$$\mathbf{f}_c(\boldsymbol{\theta}) = \sum_{k=1}^{K} \theta_k \mathbf{p}_c^{(k)} = \mathbf{P}_c^{(K)} \boldsymbol{\theta} \tag{3.5}$$

where $\mathbf{P}_c^{(K)} := \begin{bmatrix} \mathbf{p}_c^{(1)} & \cdots & \mathbf{p}_c^{(K)} \end{bmatrix}$. Note that $\mathbf{f}_c(\boldsymbol{\theta})$ denotes 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

$$\hat{y}_i(\boldsymbol{\theta}) := \arg\max_{c\in\mathcal{Y}} [\mathbf{f}_c(\boldsymbol{\theta})]_i \tag{3.6}$$

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

The upshot of (3.4) is a *unifying form* of superimposed diffusions allowing tunable simplex weights, taking up to K steps per class to come up with an influence metric for the semi-supervised classifier (3.6) over graphs. Next, we outline two notable members of the family of diffusion-based classifiers that can be viewed as special cases of (3.4).

3.0.1 Personalized PageRank Classifier

Inspired by its celebrated network centrality metric [24], the Personalized PageRank (PPR) algorithm has well-documented merits for label propagation; see, e.g. [84]. PPR is a special case of (3.4) corresponding to $\boldsymbol{\theta}_{\mathrm{PPR}} = (1-\alpha)\begin{bmatrix} \alpha^0 & \alpha^1 & \cdots & \alpha^K \end{bmatrix}^{\mathsf{T}}$, 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. (4.2))

$$\mathbf{f}_c(\boldsymbol{\theta}_{\mathrm{PPR}}) = (1-\alpha)\sum_{k=0}^{K} \alpha^k \mathbf{p}_c^{(k)} \tag{3.7}$$

satisfying asymptotically in the number of random walk steps

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

which implies that $\mathbf{f}_c(\boldsymbol{\theta}_{\mathrm{PPR}})$ approximates the solution of a linear system. Indeed, as shown in [14], PPR amounts to solving a weighted regularized least-squares problem over $\mathcal{V}$; see also [71]

for a PPR interpretation as an approximate geometric discriminant function defined in the space of landing probabilities.

3.0.2 Heat Kernel Classifier

The heat kernel (HK) is another popular diffusion that has recently been employed for SSL [91] and community detection on graphs [70]. HK is also a special case of (3.4) with $\boldsymbol{\theta}_{\rm HK} = e^{-t}\left[1 \quad t \quad \frac{t^2}{2} \quad \cdots \quad \frac{t^K}{K!}\right]^{\mathsf{T}}$, yielding class distributions (cf. (3.4))

$$\mathbf{f}_c(\boldsymbol{\theta}_{\rm HK}) = e^{-t}\sum_{k=0}^{K}\frac{t^k}{k!}\mathbf{p}_c^{(k)}. \tag{3.8}$$

Furthermore, it can be readily shown that

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

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 $\mathbf{f}_c(\boldsymbol{\theta}_{\rm HK})$ 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 (depending on the value of t) [32].

3.1 Adaptive Diffusions

Besides the unifying view of (3.4), the main contribution here is on efficiently designing $\mathbf{f}_c(\boldsymbol{\theta}_c)$ in (4.2), 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 3.1 gives a high-level illustration of the proposed adaptive diffusion framework, where two classes (red and green) are to be diffused over the graph (cf. (3.2)), with class-specific diffusion coefficients adapted as will be described next. Diffusions are then built (cf. (4.2)), and employed for class prediction (cf. (3.6)).

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

$$\hat{\mathbf{f}}_c = \arg\min_{\mathbf{f}\in\mathbb{R}^N}\ell(\mathbf{y}_{\mathcal{L}_c},\mathbf{f}) + \lambda R(\mathbf{f}) \tag{3.9}$$

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 (4.2), the $N-$dimensional optimization problem (3.9) reduces to solving for the $K-$dimensional vector ($K \ll N$)

$$\hat{\boldsymbol{\theta}}_c = \arg\min_{\boldsymbol{\theta} \in \mathcal{S}^K} \ell(\mathbf{y}_{\mathcal{L}_c}, \mathbf{f}_c(\boldsymbol{\theta})) + \lambda R(\mathbf{f}_c(\boldsymbol{\theta})). \tag{3.10}$$

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

$$\begin{aligned} \ell(\mathbf{y}_{\mathcal{L}_c}, \mathbf{f}) &:= \sum_{i \in \mathcal{L}} \frac{1}{d_i} ([\bar{\mathbf{y}}_{\mathcal{L}_c}]_i - f_i)^2 \\ &= (\bar{\mathbf{y}}_{\mathcal{L}_c} - \mathbf{f})^{\mathsf{T}} \mathbf{D}_{\mathcal{L}}^{\dagger} (\bar{\mathbf{y}}_{\mathcal{L}_c} - \mathbf{f}) \end{aligned} \tag{3.11}$$

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 $\mathbf{D}_{\mathcal{L}}^{\dagger} = \mathrm{diag}(\mathbf{d}_{\mathcal{L}}^{(-1)})$ 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

$$\begin{aligned} R(\mathbf{f}) &= \frac{1}{2} \sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{N}_i} \left(\frac{f_i}{d_i} - \frac{f_j}{d_j} \right)^2 \\ &= \mathbf{f}^{\mathsf{T}} \mathbf{D}^{-1} \mathbf{L} \mathbf{D}^{-1} \mathbf{f}. \end{aligned} \tag{3.12}$$

where $\mathbf{L} := \mathbf{D} - \mathbf{W}$ is the Laplacian matrix of the graph. Intuitively speaking, (3.11) 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 (3.12) promotes similarity of the entries of $\mathbf{f}$ that correspond to neighboring nodes. In (3.11) and (3.12), each

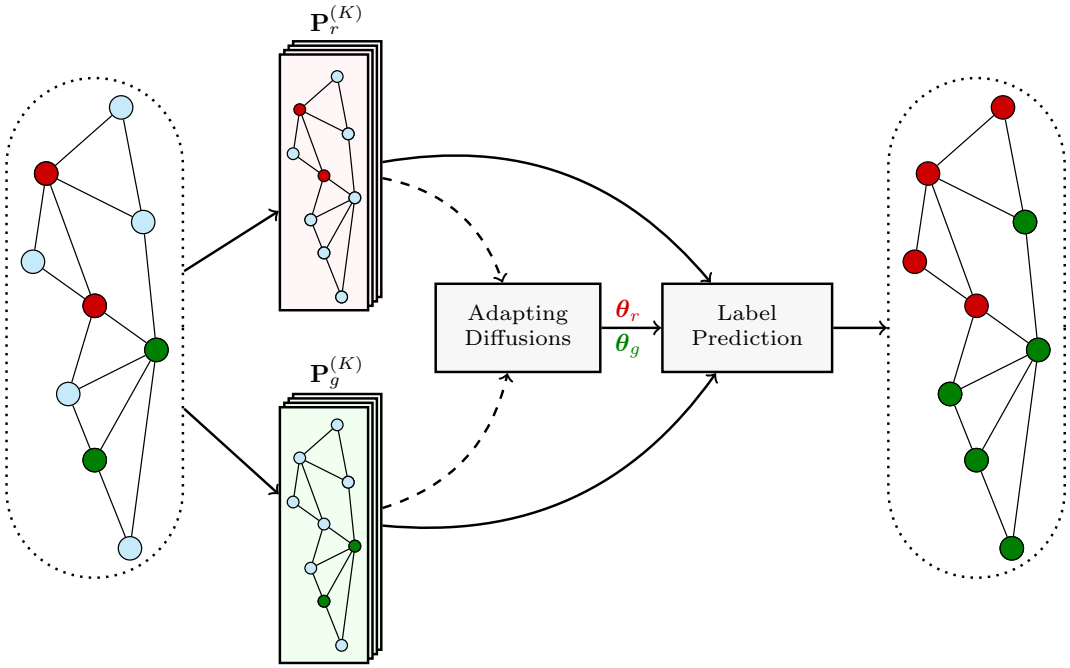

Figure 3.1: High-level illustration of adaptive diffusions. The nodes belong to two classes (red and green). The per-class diffusions are learned by exploiting the landing probability spaces produced by random walks rooted at the sample nodes (second layer: up for red; down for green).

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 (3.11) and (3.12) into (3.10), and recalling from (4.2) that $\mathbf{f}_c(\boldsymbol{\theta}) = \mathbf{P}_c^{(K)}\boldsymbol{\theta}$, yields the convex quadratic program

$$\hat{\boldsymbol{\theta}}_c = \arg\min_{\boldsymbol{\theta}\in\mathcal{S}^K} \boldsymbol{\theta}^\mathsf{T}\mathbf{A}_c\boldsymbol{\theta} + \boldsymbol{\theta}^\mathsf{T}\mathbf{b}_c \tag{3.13}$$

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

$$\mathbf{b}_c = -\frac{2}{|\mathcal{L}|}(\mathbf{P}_c^{(K)})^\mathsf{T}\mathbf{D}_{\mathcal{L}}^{\dagger}\mathbf{y}_{\mathcal{L}_c} \tag{3.14}$$

$$\mathbf{A}_c = (\mathbf{P}_c^{(K)})^\mathsf{T}\mathbf{D}_{\mathcal{L}}^{\dagger}\mathbf{P}_c^{(K)} + \lambda(\mathbf{P}_c^{(K)})^\mathsf{T}\mathbf{D}^{-1}\mathbf{L}\mathbf{D}^{-1}\mathbf{P}_c^{(K)} \tag{3.15}$$

$$= (\mathbf{P}_c^{(K)})^\mathsf{T} \left[\left(\mathbf{D}_\mathcal{L}^\dagger + \lambda \mathbf{D}^{-1} \right) \mathbf{P}_c^{(K)} - \lambda \mathbf{D}^{-1} \mathbf{H} \mathbf{P}_c^{(K)} \right]$$
$$= (\mathbf{P}_c^{(K)})^\mathsf{T} \left(\mathbf{D}_\mathcal{L}^\dagger \mathbf{P}_c^{(K)} + \lambda \mathbf{D}^{-1} \tilde{\mathbf{P}}_c^{(K)} \right) \tag{3.16}$$

where

$$\mathbf{H}\mathbf{P}_c^{(K)} = \begin{bmatrix} \mathbf{H}\mathbf{p}_c^{(1)} & \mathbf{H}\mathbf{p}_c^{(2)} & \cdots & \mathbf{H}\mathbf{p}_c^{(K)} \end{bmatrix}$$
$$= \begin{bmatrix} \mathbf{p}_c^{(2)} & \mathbf{p}_c^{(3)} & \cdots & \mathbf{p}_c^{(K+1)} \end{bmatrix}$$

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

$$\tilde{\mathbf{P}}_c^{(K)} := \begin{bmatrix} \tilde{\mathbf{p}}_c^{(1)} & \tilde{\mathbf{p}}_c^{(2)} & \cdots & \tilde{\mathbf{p}}_c^{(K)} \end{bmatrix}$$

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 $\mathcal{O}(K^2 N)$ for computing the first summand, and $\mathcal{O}(|\mathcal{E}|K)$ for the second summand in (3.15), 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 (3.13)-(3.16) that we term adaptive-diffusion (AdaDIF), incurs complexity of order $\mathcal{O}(K^2 N)$.

Note that, the problem in (3.13) is a *quadratic program (QP)* of dimension K (or the dictionary size D to be discussed in Section III-C when in dictionary mode). In general, solving a QP with K variables to a given precision requires a $\mathcal{O}(K^3)$ worst-case complexity. Although this may appear heavy, K in our setting is 10 – 30 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 complexity for QP is dominated by the $\mathcal{O}(|\mathcal{E}|K)$ (same as PPR and HK) performing the random walks and $\mathcal{O}(NK^2)$ for computing $\mathbf{A}_c$.

3.1.1 Limiting behavior and computational complexity

In this section, we offer further insights on the model (4.2), along with complexity analysis of the parametric solution in (3.13). 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* (3.13) *as* $\lambda \to \infty$ *satisfies*

$$\hat{\boldsymbol{\theta}}_c = \mathbf{e}_K, \quad \forall\, \mathcal{L}_c \subseteq \mathcal{V}. \tag{3.17}$$

Our experience with solving (3.13) reveal that the sufficiently large K required for (3.17) to hold, can be as small as 10^2.

As $\lambda \to \infty$, the effect of the loss in (3.10) 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. 3.2 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 (3.13), 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. (3.2)). This endows AdaDIF with high computational efficiency, especially for small K. Specifically, since for solving (3.13) 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 (3.16).

Overall, if $\mathcal{O}(KN)$ memory requirements are met, the runtime of AdaDIF scales *linearly*

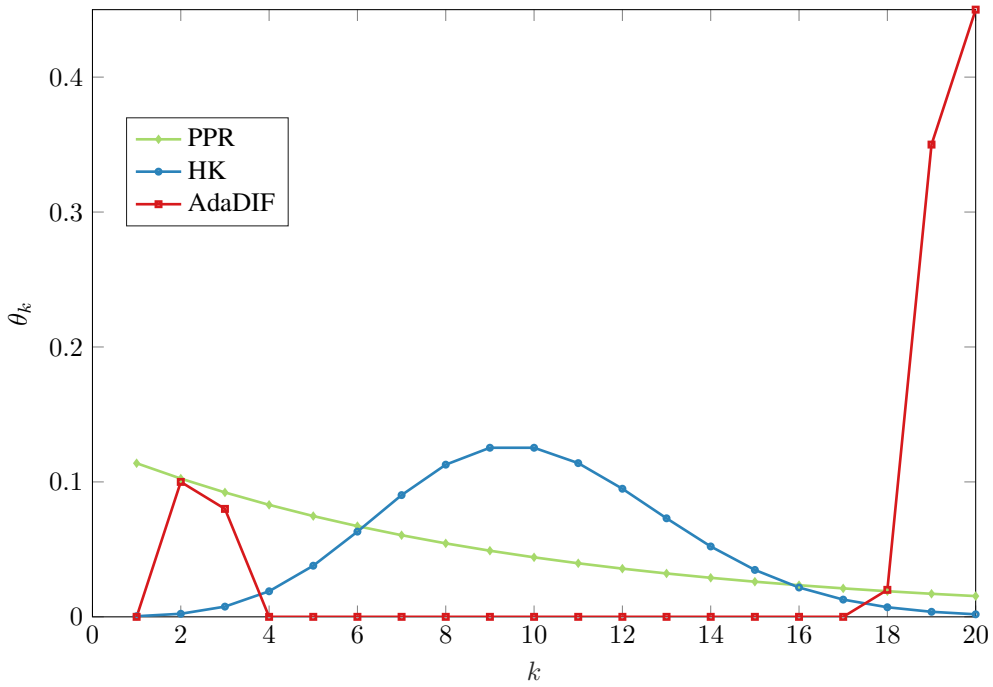

Figure 3.2: Illustration of $K = 20$ landing probability coefficients for PPR with $\alpha = 0.9$, HK with $t = 10$, and AdaDIF ($\lambda = 15$).

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.

3.1.2 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 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. Note also that in settings where one can freely choose the nodes to sample, this result could be used to guide such choice in a disciplined way.

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 B.2.1 we give a tighter threshold for PPR.

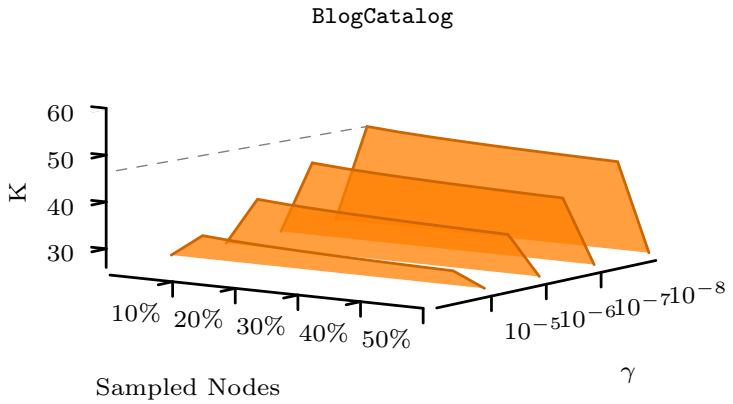

Figure 3.3: Experimental evaluation K_γ for different values of γ-distinguishability threshold, and proportions of sampled nodes on `BlogCatalog` graph.

Conveniently, our experiments suggest that $K \in [10, 20]$ is usually sufficient to achieve high performance for most real graphs ; see also Fig. 3.3 where K_γ is found numerically 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 1. 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) which mitigates overfitting and results in optimization problems that scale well with the network size.

3.1.3 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. (4.2))

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

where $a_k(\boldsymbol{\theta}) := \sum_{d=1}^{D} \theta_d C_{kd}$, and $\mathbf{C} := \begin{bmatrix}\mathbf{c}_1 \cdots \mathbf{c}_D\end{bmatrix} \in \mathbb{R}^{K\times D}$ 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 (3.13) is solved with

$$\mathbf{b}_c = -\frac{2}{|\mathcal{L}|}(\mathbf{F}_c^{\Delta})^{\mathsf{T}}\mathbf{D}_{\mathcal{L}}^{\dagger}\mathbf{y}_{\mathcal{L}^c} \tag{3.18}$$

$$\mathbf{A}_c = (\mathbf{F}_c^{\Delta})^{\mathsf{T}}\mathbf{D}_{\mathcal{L}}^{\dagger}\mathbf{F}_c^{\Delta} + \lambda(\mathbf{F}_c^{\Delta})^{\mathsf{T}}\mathbf{D}^{-1}\mathbf{L}\mathbf{D}^{-1}\mathbf{F}_c^{\Delta} \tag{3.19}$$

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 $\mathbf{C} = \begin{bmatrix}\boldsymbol{\theta}_{\mathrm{PPR}} & \boldsymbol{\theta}_{\mathrm{HK}} & \cdots\end{bmatrix}$. In that sense, our approach is related to multi-kernel methods, e.g. [11], 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 $|\mathcal{E}|$.

Algorithm 2 ADAPTIVE DIFFUSIONS

Input: Adjacency matrix: $\mathbf{A}$, 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**
    $\mathbf{F}_c^{\Delta} = \text{DICTIONARY}\ (\mathbf{A}, \mathcal{L}_c, K, \mathbf{C})$
    Obtain $\mathbf{b}_c$ and $\mathbf{A}_c$ as in (3.18) and (3.19)
    $\mathbf{F}_c = \mathbf{F}_c^{\Delta}$
  else
    $\{\mathbf{P}_c^{(K)}, \tilde{\mathbf{P}}_c^{(K)}\} = \text{LANDPROB}(\mathbf{A}, \mathcal{L}_c, K)$
    Obtain $\mathbf{b}_c$ and $\mathbf{A}_c$ as in (3.14) and (3.16)
    $\mathbf{F}_c = \mathbf{P}_c^{(K)}$
  end if
  if Unconstrained **then**
    Obtain $\hat{\boldsymbol{\theta}}_c$ as in (3.20) and (3.21)
  else
    Obtain $\hat{\boldsymbol{\theta}}_c$ by solving (3.13)
  end if
  $\mathbf{f}_c(\hat{\boldsymbol{\theta}}_c) = \mathbf{F}_c\hat{\boldsymbol{\theta}}_c$
end for
Obtain $\hat{y}_i = \arg\max_{c\in\mathcal{Y}} \left[\mathbf{f}_c(\hat{\boldsymbol{\theta}}_c)\right]_i, \ \ \forall i \in \mathcal{U}$

3.1.4 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, (3.13) can afford a closed-form solution as

$$\hat{\boldsymbol{\theta}}_c = \mathbf{A}_c^{-1}(\mathbf{b}_c - \lambda^*\mathbf{1}) \tag{3.20}$$

$$\lambda^* = \frac{\mathbf{1}^\mathsf{T}\mathbf{A}_c^{-1}\mathbf{b}_c - 1}{\mathbf{b}^\mathsf{T}\mathbf{A}_c^{-1}\mathbf{b}_c}. \tag{3.21}$$

Algorithm 3 LANDPROB

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

Input: $\mathbf{A}, \mathcal{L}_c, K, \mathbf{C}$
Output: $\mathbf{F}_c^{\Delta}$
$\mathbf{H} = \mathbf{A}\mathbf{D}^{-1}$
$\mathbf{p}_c^{(0)} = \mathbf{v}_c$
$\{\mathbf{f}_c^{(d)}\}_{d=1}^{D} = \mathbf{0}$
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

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 $K > |\mathcal{L}|$, (3.20) 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.

3.2 Adaptive Diffusions Robust to Anomalies

Although the loss function in (3.11) 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

$$\ell^c_{\text{rob}}(\mathbf{y}_{\mathcal{L}_c}, \boldsymbol{\theta}) := \sum_{i \in \mathcal{L}} \frac{1}{d_i} \left([\bar{\mathbf{y}}_{\mathcal{L}_c}]_i - [\mathbf{f}_c(\boldsymbol{\theta}; \mathcal{L} \setminus i)]_i \right)^2 \tag{3.22}$$

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, (3.22) 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

$$\mathbf{f}_c(\boldsymbol{\theta}; \mathcal{L} \setminus i) = \begin{cases} \mathbf{f}_c(\boldsymbol{\theta}), & i \notin \mathcal{L}_c \\ \mathbf{f}_c(\boldsymbol{\theta}; \mathcal{L}_c \setminus i), & i \in \mathcal{L}_c \end{cases} \tag{3.23}$$

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. (4.2))

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

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

$$[\mathbf{v}_{\mathcal{L}_c \setminus i}]_j = \begin{cases} 1/|\mathcal{L}_c \setminus i|, & j \in \mathcal{L}_c \setminus i \\ 0, & \text{else} \end{cases} . \tag{3.24}$$

The robust loss in (3.22) can be expressed more compactly as

$$\ell_{\text{rob}}^{c}(\mathbf{y}_{\mathcal{L}_c}, \boldsymbol{\theta}) := \|\mathbf{D}_{\mathcal{L}}^{-\frac{1}{2}} \left(\bar{\mathbf{y}}_{\mathcal{L}_c} - \mathbf{R}_c^{(K)}\boldsymbol{\theta}\right)\|_2^2 \tag{3.25}$$

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

$$\left[\mathbf{R}_c^{(K)}\right]_{ik} := \begin{cases} \left[\mathbf{p}_{\mathcal{L}_c\setminus i}^{(k)}\right]_i, & i \in \mathcal{L}_c \\ \left[\mathbf{p}_c^{(k)}\right]_i, & \text{else} \end{cases}. \tag{3.26}$$

Since

$$\mathbf{p}_c^{(k)} = |\mathcal{L}_c|^{-1} \sum_{i\in\mathcal{L}_c} \mathbf{p}_{\mathcal{L}_c\setminus i}^{(k)},$$

evaluating (3.25) 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 $\ell_{\text{rob}}^{c}(\cdot)$, per-class diffusion coefficients $\hat{\boldsymbol{\theta}}_c$ can be obtained by solving

$$\hat{\boldsymbol{\theta}}_c = \arg\min_{\boldsymbol{\theta}\in\mathcal{S}^K} \ell_{\text{rob}}^{c}(\mathbf{y}_{\mathcal{L}_c}, \boldsymbol{\theta}) + \lambda_\theta \|\boldsymbol{\theta}\|_2^2 \tag{3.27}$$

where ℓ_2 regularization with parameter λ_θ is introduced in order to prevent overfitting and numerical instabilities. Note that smoothness regularization in (3.12) is less appropriate in the context of robustness, since it promotes "spreading" of the random walks (cf. Prop. 1), thus making class-diffusions more similar and increasing the difficulty of detecting outliers. Similar to (3.13), quadratic programming can be adopted to solve (3.27).

Towards mitigating the effects of outliers, and inspired by the robust estimators introduced in [68], we further enhance $\ell_{\text{rob}}^{c}(\cdot)$ by explicitly modeling the effect of outliers with a sparse vector $\mathbf{o} \in \mathbb{R}^N$, leading to the modified cost

$$\ell_{\text{rob}}^{c}(\mathbf{y}_{\mathcal{L}_c}, \mathbf{o}, \boldsymbol{\theta}) := \|\mathbf{D}_{\mathcal{L}}^{-\frac{1}{2}} \left(\mathbf{o} + \bar{\mathbf{y}}_{\mathcal{L}_c} - \mathbf{R}_c^{(K)}\boldsymbol{\theta}\right)\|_2^2. \tag{3.28}$$

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

$$\{\hat{\boldsymbol{\theta}}_c, \hat{\mathbf{o}}_c\}_{c\in\mathcal{Y}} = \arg\min_{\substack{\boldsymbol{\theta}_c\in\mathcal{S}^K \\ \mathbf{o}_c\in\mathbb{R}^N}} \sum_{c\in\mathcal{Y}} \left[\ell^c_{\text{rob}}(\mathbf{y}_{\mathcal{L}_c}, \mathbf{o}_c, \boldsymbol{\theta}_c) + \lambda_\theta\|\boldsymbol{\theta}_c\|_2^2\right] + \lambda_o\|\mathbf{D}_{\mathcal{L}}^{-\frac{1}{2}}\mathbf{O}\|_{2,1} \tag{3.29}$$

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 (3.29) 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 (3.29) is non-convex, convergence of gradient-descent-type methods to the global optimum is not guaranteed. Nevertheless, since (3.29) is biconvex (i.e., convex with respect to each variable) the following alternating minimization scheme

$$\hat{\mathbf{O}}^{(t)} = \arg\min_{\mathbf{O}} \sum_{c\in\mathcal{Y}} \left[\ell^c_{\text{rob}}(\mathbf{y}_{\mathcal{L}_c}, \mathbf{o}_c, \hat{\boldsymbol{\theta}}_c^{(t-1)}) + \lambda_\theta\|\hat{\boldsymbol{\theta}}_c^{(t-1)}\|_2^2\right] + \lambda_o\|\mathbf{D}_{\mathcal{L}}^{-\frac{1}{2}}\mathbf{O}\|_{2,1} \tag{3.30}$$

$$\hat{\boldsymbol{\theta}}_c^{(t)} = \arg\min_{\boldsymbol{\theta}\in\mathcal{S}^K} \ell^c_{\text{rob}}(\mathbf{y}_{\mathcal{L}_c}, \hat{\mathbf{o}}_c^{(t)}, \boldsymbol{\theta}) + \lambda_\theta\|\boldsymbol{\theta}\|_2^2 + \lambda_o\|\mathbf{D}_{\mathcal{L}}^{-\frac{1}{2}}\hat{\mathbf{O}}^{(t)}\|_{2,1} \tag{3.31}$$

with $\hat{\mathbf{O}}^{(0)} := [\mathbf{0}\ldots\mathbf{0}]$ converges to a partial optimum [45].

By further simplifying (3.31) and solving (3.30) in closed form, we obtain

$$\hat{\boldsymbol{\theta}}_c^{(t)} = \arg\min_{\boldsymbol{\theta}\in\mathcal{S}^K} \ell^c_{\text{rob}}(\bar{\mathbf{y}}_{\mathcal{L}_c} + \hat{\mathbf{o}}_c^{(t-1)}, \boldsymbol{\theta}) + \lambda_\theta\|\boldsymbol{\theta}\|_2^2 \tag{3.32}$$

$$\hat{\mathbf{O}}^{(t)} = \text{SoftThres}_{\lambda_o}\left(\tilde{\mathbf{Y}}^{(t)}\right) \tag{3.33}$$

where

$$\tilde{\mathbf{Y}}^{(t)} := \left[\tilde{\mathbf{y}_{\mathbf{1}}}^{(t)}, \ldots, \mathbf{y}_{|\mathcal{Y}|}^{(t)}\right]$$

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. [103]. Intuitively, the soft-thresholding operation in (3.33) 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 (3.32) and (3.33) terminates when

$$\|\hat{\boldsymbol{\theta}}_c^{(t)} - \hat{\boldsymbol{\theta}}_c^{(t-1)}\|_\infty \leq \epsilon, \ \forall c \in \mathcal{Y}$$

where $\epsilon \geq 0$ 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 5, and has $\mathcal{O}(K|\mathcal{L}||\mathcal{E}|)$ computational complexity.

3.3 Contributions in Context of Prior Works

Following the seminal contribution in [24] 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 [79, 41]. Most formal approaches to generalize PageRank focus either on the *teleportation* component (see e.g. [95, 96] as well as [21] for an application to semi-supervised classification), or, on the so-termed *damping* mechanism [34, 15]. Perhaps the most general setting can be found in [15], where a family of

Algorithm 5 ROBUST ADAPTIVE DIFFUSIONS

Input: Adjacency matrix: $\mathbf{A}$, 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**
    $\{\mathbf{p}^{(k)}_{\mathcal{L}_c\setminus i}\}_{k=1}^K = \text{LANDPROB}(\mathbf{A}, \mathcal{L}_c \setminus i, K)$
  end for
  Obtain $\mathbf{R}_c^{(K)}$ as in (3.26)
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 (3.32)
  Obtain $\hat{\mathbf{O}}^{(t)}$ as in (3.33)
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 $\hat{y}_i = \arg\max_{c\in\mathcal{Y}} \left[\mathbf{f}_c(\hat{\boldsymbol{\theta}}_c)\right]_i, \ \ \forall i \in \mathcal{U}$

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. A related method termed AptRank was recently proposed in [63] specifically for protein function prediction. Differently from AdaDIF the method exploits meta-information regarding the hierarchical organization of functional roles of proteins and it performs random walks on the heterogeneous protein-function network. AptRank splits the data into training and validation sets of predetermined proportions and adopt

as cross-validation approach for obtaining diffusion coefficients. Furthermore a1) AptRank trains a single diffusion for all classes whereas AdaDIF identifies different diffusions, and a2) the proposed robust leave-one-out method (r-AdaDIF) gathers residuals from all leave-one-out splits into one cost function (cf. (3.22)) and then optimizes the (per class) diffusion.

Recently, community detection (CD) methods were proposed in [55] and [54], 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 nodes of 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* (that is 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 [55] and [54]—which again arises from the different contexts—is the length of the walk compared to the size of the graph. Since [55] and [54] 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 [110], and further pursued in [31] for bridge monitoring; see also [112] and [88] 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 [110] focuses on binary classification and [31] 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 [110] the weighting parameters remain unconstrained and in [31] belong to a hyperplane, AdaDIF constrains the diffusion parameters on the probability

Table 3.1: 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

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.

3.4 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 4.1. 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 3.01; ii) Label propagation (LP) [140]; iii) Node2vec [47]; iv) Deepwalk [101]; v) Planetoid-G [134]; and, vi) graph convolutional networks (GCNs) [69]. We note here that AptRank [63] was not considered in our experiments since it relies on meta-information that is not available for the benchmark datasets used here.

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., [84]. Both HK and PPR were run for 50 steps for convergence to be in effect; see Fig 3.4; LP was also run for 50 steps. For Node2vec, we fixed most parameters to the values suggested in [47], and performed grid search for $p, q \in [0.25, 1.0, 2.0, 4.0]$. Since Deepwalk can be seen as Node2vec with $p = q = 1.0$,

Table 3.2: 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	67.5 ± 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	$\mathbf{68.9 \pm 1.9}$	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 3.3: 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

we used the Node2vec Python implementation for both. As in [47, 101], 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. 3.4); 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

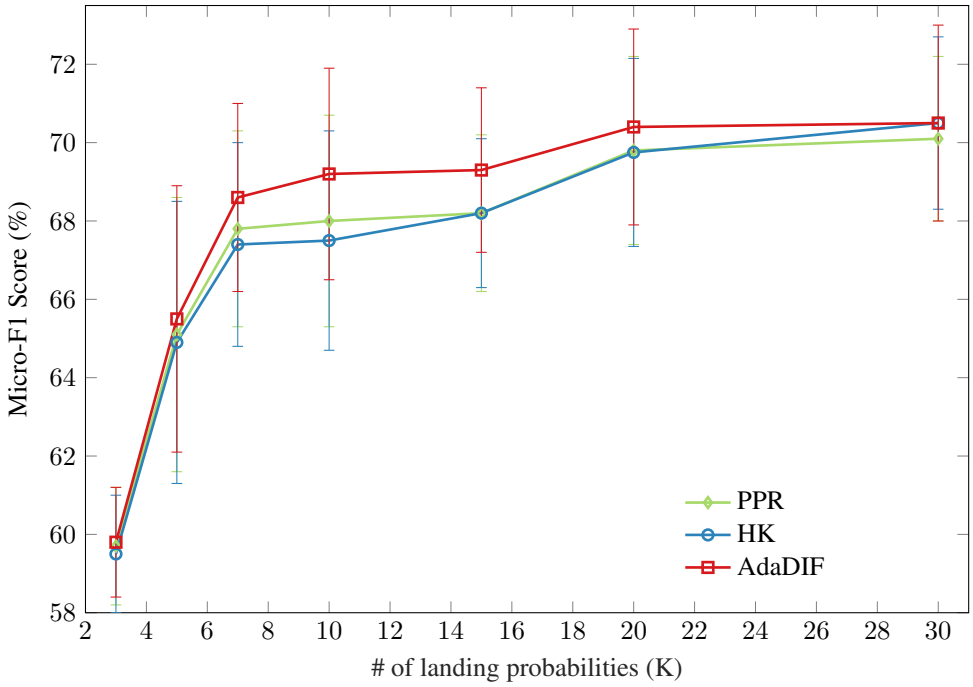

Figure 3.4: Micro-F1 score for AdaDIF and non-adaptive diffusions on 5% labeled `Cora` graph as a function of the length of underline random walks.

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., [90]. The results were averaged over 20 experiments, with mean and standard deviation reported in Table 3.2. 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 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 [69, Appendix].

The scalabilty of AdaDIF is reflected on the runtime comparisons listed in Fig. 3.7. All

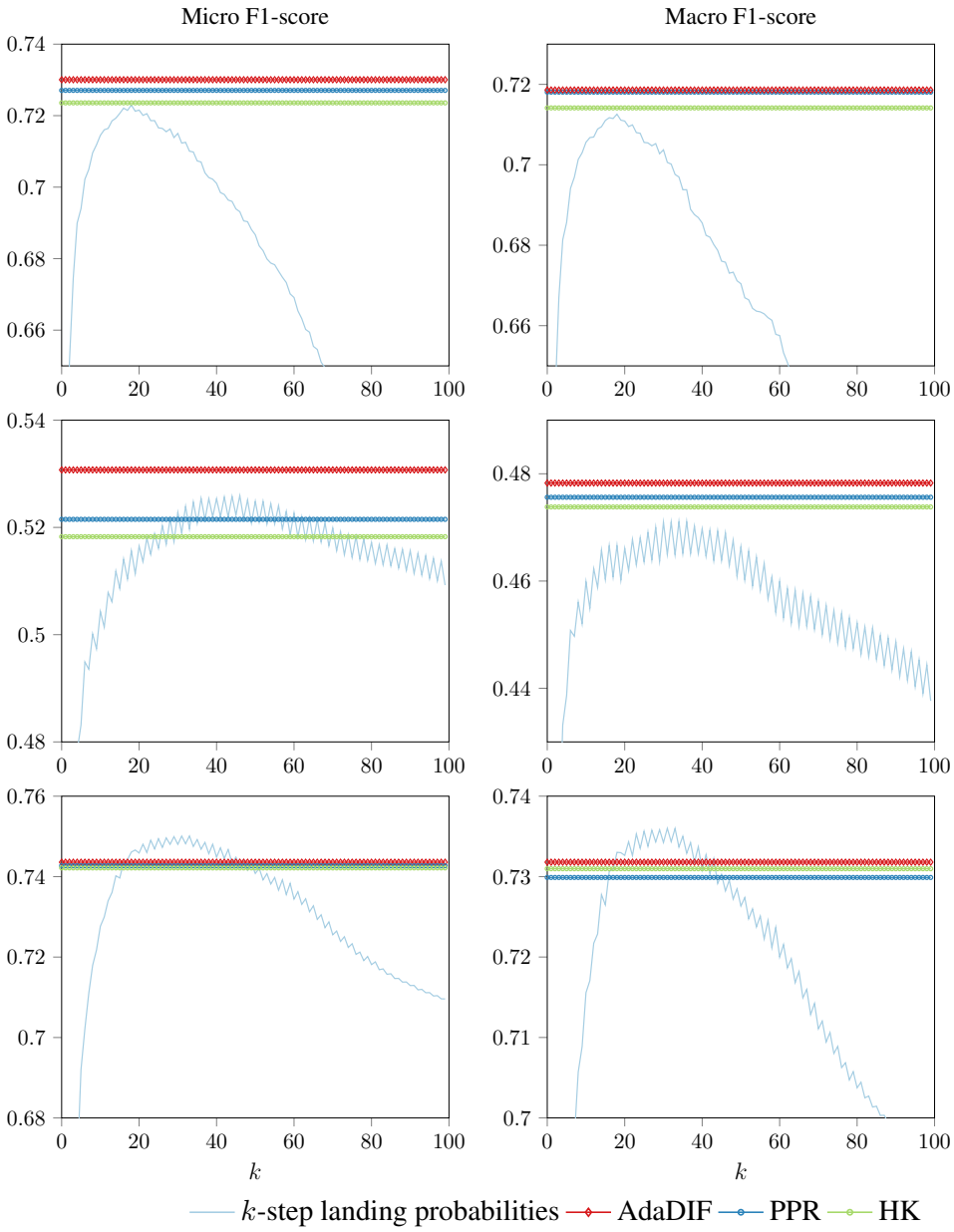

Figure 3.5: Classification accuracy of AdaDIF, PPR, and HK compared to the accuracy of $k-$step landing probability classifier. First line is `Cora` graph, second is `Citeseer`, and third is `PubMed`. Left column is Micro F1 and right column is Macro F1 score.

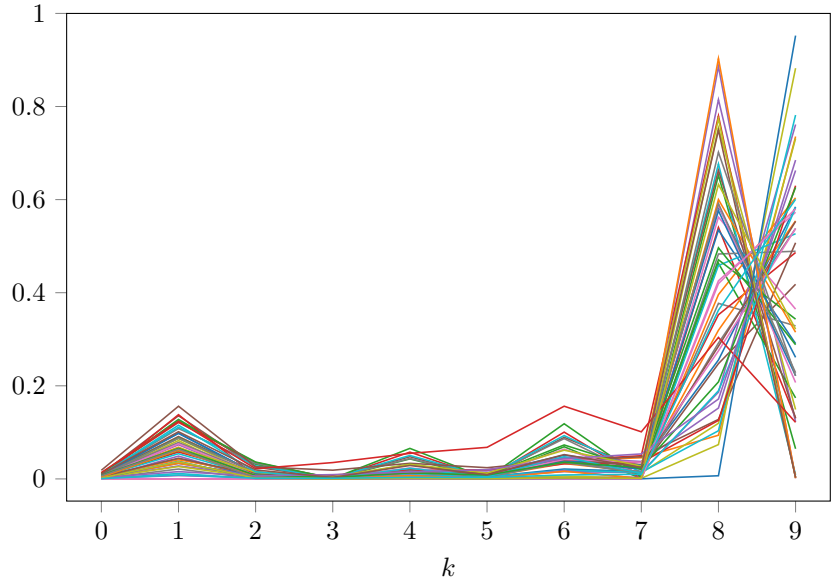

Figure 3.6: AdaDIF diffusion coefficients for the 50 different classes of PPI graph (30% sampled). Each line corresponds to a different $\boldsymbol{\theta}_c$. Diffusion is characterized by high diversity among classes.

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 CVX-OPT libraries. We also developped an efficient implementation that exploits parallelism, which is straightforward since each class can be treated separately. While AdaDIF incurs (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 3.3 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 [47] and [101], 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

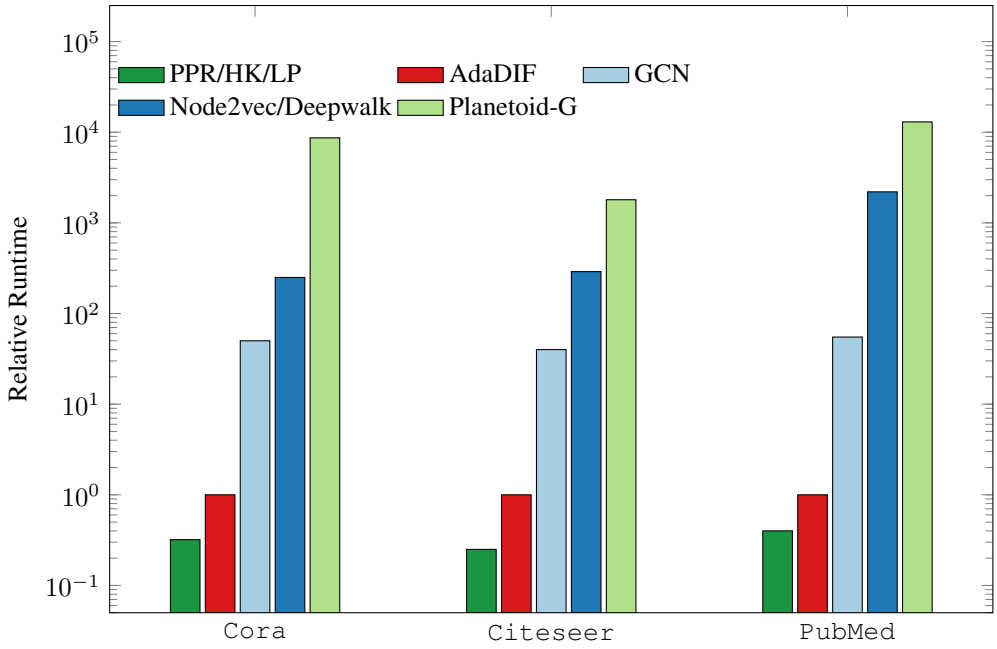

Figure 3.7: Relative runtime comparisons for multiclass graphs.

Macro-F1 score. It is worth noting that for multilabel graphs with many classes, the performance boost over fixed diffusions can be largely attributed to AdaDif's flexibility to treat each class differently. To demonstrate that different classes are indeed diffused in a markedly different manner, Fig. 3.6 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, while the overall "form" of the corresponding diffusion coefficients adheres to the general pattern observed in Fig.3.2 there is indeed large diversity among classes.

3.4.1 Analysis/interpretation of results

Here we will follow an experimental approach that is aimed at understanding and interpreting our results. We will focus on diffusion-based classifiers, along with 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 is 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. 3.5) that appear to be different for each network, characterizing the graph-label distribution relationship of the latter. For the `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, the `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 the `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 the `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 the `PubMed` graph.

3.4.2 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 5) 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 [42] 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 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. 3.8, the accuracy of all diffusion-based classifiers on `Cora` 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 samples and thus reduces the training set. Although PPR w. ranking displays relative robustness as p_{cor} increases, overall it performs worse than PPR with value based rounding, at least on the `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 (3.29) is effectively removed and all samples are characterized as outliers. On the other hand, for $\lambda_o \gg 1$ (3.29) 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. 3.9. 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.

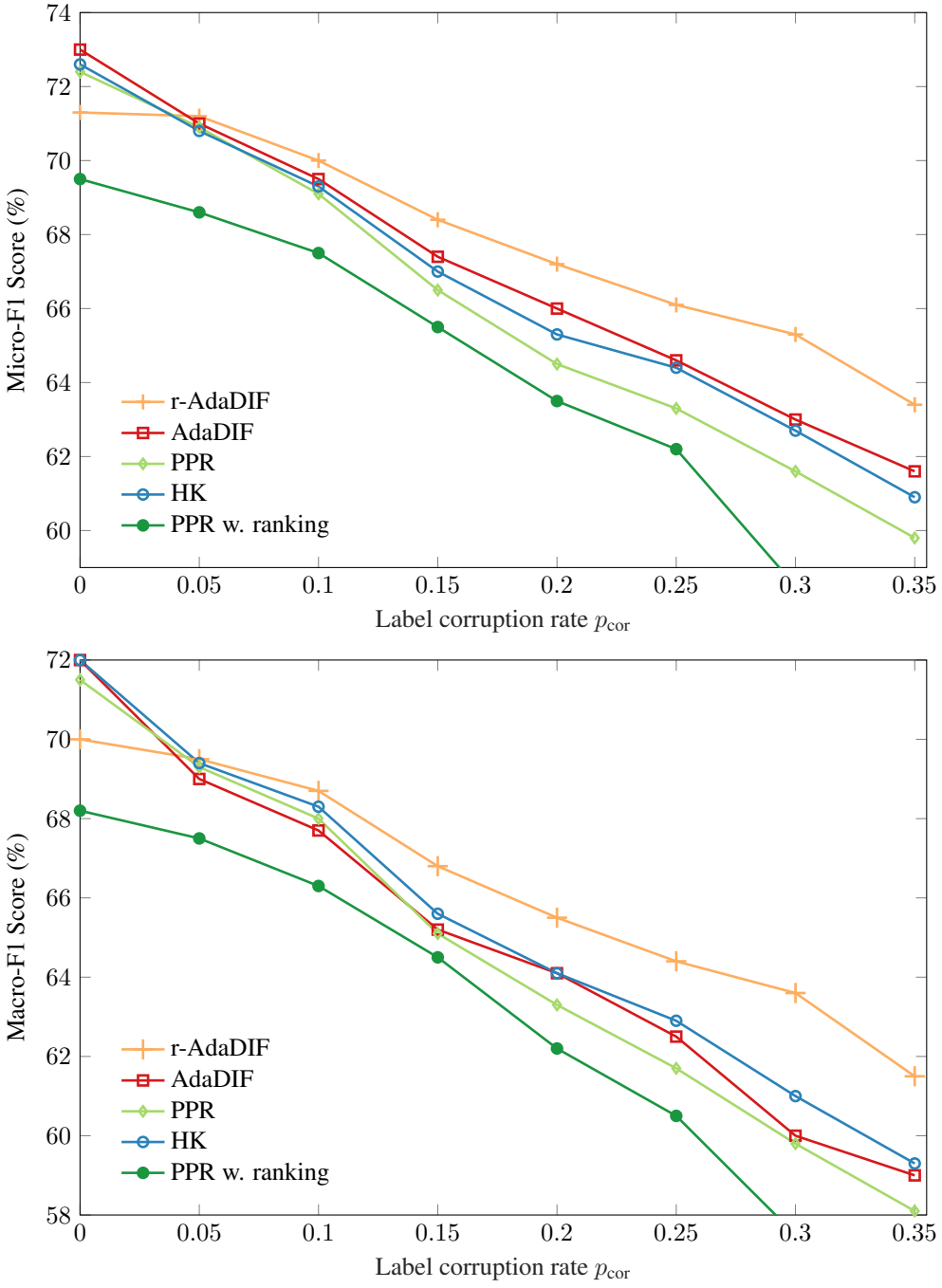

Figure 3.8: Classification accuracy of various diffusion-based classifiers on Cora, as a function of the probability of label corruption.

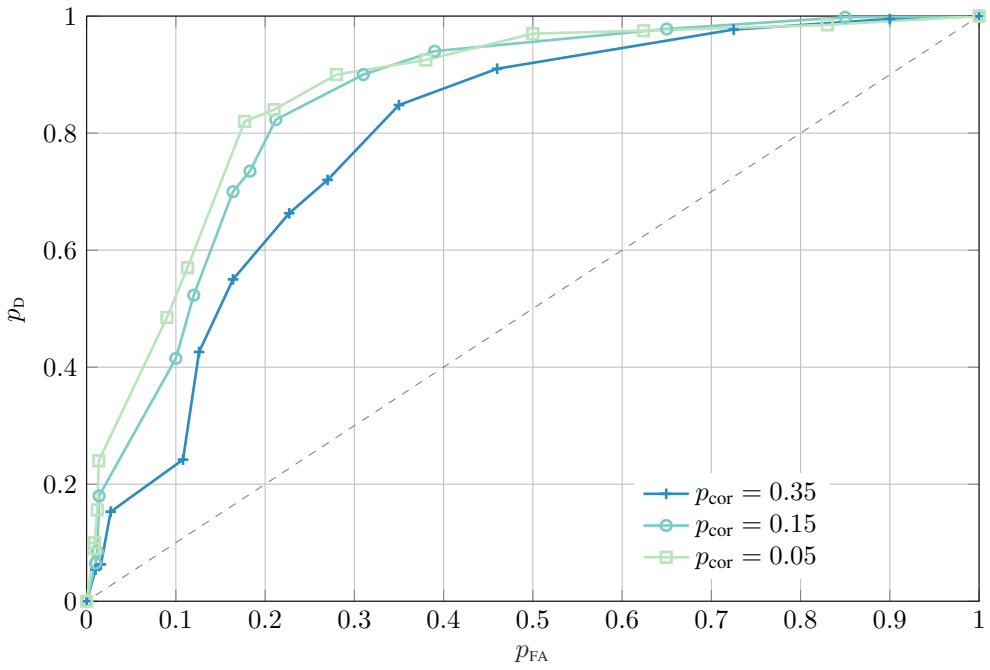

Figure 3.9: Anomaly detection performance of r-AdaDIF for different label corruption probabilities. The horizontal axis corresponds to the frequency with which r-AdaDIF returns a true positive (probability of detection) and the vertical axis corresponds to the frequency of false positives (probability of false alarm).

Chapter 4

Unsupervised Node Embedding with Adaptive Similarities

The promising results of adaptive diffusions on semi-supervised classification prompted the further application of the multi-length walk model to the unsupervised task of node embedding.

Recalling Section 1.1, node embedding boils down to determining $f(\cdot) : \mathcal{V} \to \mathbb{R}^d$, where $d \ll N$. In other works, a function is sought to map every node of $\mathcal{G}$ to a vector in the $d-$dimensional Euclidean space. Typically, the embedding is low dimensional with d much smaller than the number of nodes. Given $f(\cdot)$, the low-dimensional vector representation of each node v_i is

$$\mathbf{e}_i = f(v_i) \;\; \forall v_i \in \mathcal{V} \; .$$

Since the number of nodes is finite, instead of finding a general $f(\cdot)$ (induction), one may pose the embedding task in its most general form as a the following minimization problem over the embedded vectors

$$\{\mathbf{e}_i^*\}_{i=1}^N = \arg \min_{\{\mathbf{e}_i\}_{i=1}^N} \sum_{v_i, v_j \in \mathcal{V}} \ell \left(s_{\mathcal{G}}(v_i, v_j), s_{\mathcal{E}}(\mathbf{e}_i, \mathbf{e}_j) \right) \tag{4.1}$$

where $\ell(\cdot, \cdot) : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ is a loss function; $s_{\mathcal{G}}(\cdot, \cdot) : \mathcal{V} \times \mathcal{V} \to \mathbb{R}$ is a similarity metric over pairs of graph *nodes*; and $s_{\mathcal{E}}(\cdot, \cdot) : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ a similarity metric over pairs of *vectors* in the $d-$dimensional Euclidean space.

In par with (4.1), node embedding can be viewed as the design of nodal vectors $\{\mathbf{e}_i\}_{i=1}^N$ that

successfully "encode" a certain notion of pairwise similarities among graph nodes.

4.0.1 Embedding as matrix factorization

Starting from the generalized framework in (4.1), one may arrive at concrete approaches by specifying choices of $s_{\mathcal{G}}(\cdot,\cdot)$, $s_{\mathcal{E}}(\cdot,\cdot)$, and $\ell(\cdot,\cdot)$. To start, suppose that the node similarity metric is symmetric; that is, $s_{\mathcal{G}}(v_i,v_j)=s_{\mathcal{G}}(v_j,v_i)\ \forall v_i,v_j\in\mathcal{V}$. Furthermore, let the loss function be quadratic

$$\ell(x,x')=\left(x-x'\right)^2$$

and the nodal vector similarity be the inner product

$$s_{\mathcal{E}}(\mathbf{e}_i,\mathbf{e}_j)=\mathbf{e}_i^\top\mathbf{e}_j.$$

Using these specifications, (4.1) reduces to the following symmetric matrix factorization problem

$$\mathbf{E}^*=\arg\min_{\mathbf{E}\in\mathbb{R}^{N\times d}}\|\mathbf{S}_{\mathcal{G}}-\mathbf{E}\mathbf{E}^\top\|_F^2 \tag{4.2}$$

where $\mathbf{S}_{\mathcal{G}}\in\mathbb{R}^{N\times N}$ is the symmetric similarity matrix with $[\mathbf{S}_{\mathcal{G}}]_{i,j}=[\mathbf{S}_{\mathcal{G}}]_{j,i}=s_{\mathcal{G}}(v_i,v_j)$, and matrix $\mathbf{E}:=[\mathbf{e}_1\dots\mathbf{e}_N]^\top$ concatenates all node embeddings as rows. A well-known analytical solution to (4.2) relies on the singular value decomposition (SVD) of the similarity matrix, that is $\mathbf{S}_{\mathcal{G}}=\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$, where $\mathbf{U}$ and $\mathbf{V}$ are the $N\times N$ unitary matrices formed by the left and right singular vectors, and $\mathbf{\Sigma}$ is diagonal with non-negative singular values sorted in decreasing order; in our case, $\mathbf{U}=\mathbf{V}$ since $\mathbf{S}_{\mathcal{G}}$ is symmetric. Given the SVD of $\mathbf{S}_{\mathcal{G}}$, the low-rank ($d\ll N$) solver in (4.2) is $\mathbf{E}^*=\mathbf{U}_d\mathbf{\Sigma}_d^{1/2}$, where $\mathbf{\Sigma}_d$ contains the d largest singular values, and $\mathbf{U}_d$ the corresponding singular vectors [43]. Matrices $\mathbf{U}_d$ and $\mathbf{\Sigma}_d$ can be obtained directly using the reduced-complexity scheme known as *truncated* SVD.

If in addition $\mathbf{S}_{\mathcal{G}}$ is *sparse*, (4.2) can be solved even more efficiently, with complexity that scales with the number of edges. One such example with sparse similarities is when $\mathbf{S}_{\mathcal{G}}=\mathbf{A}$, where $\mathbf{A}$ is the graph adjacency matrix. Embeddings generally gain scalability by avoiding the explicit construction of a *dense* $\mathbf{S}_{\mathcal{G}}$. In fact, simply storing $\mathbf{S}_{\mathcal{G}}$ in the working memory becomes prohibitive even for graphs of moderate sizes (say $N>10^5$).

In the ensuing section, we will design a family of dense similarity matrices that (among other properties) can be decomposed implicitly, at the cost of input sparsity.

4.0.2 Multihop graph node similarities

Having reduced the node embedding problem to the one in (4.2), it remains to specify the graph similarity metric that gives rise to $\mathbf{S}_{\mathcal{G}}$. Towards this end, and in order to maintain expressibility, we will design a parametric model for $\mathbf{S}_{\mathcal{G}}$, with each pairwise node similarity metric expressed as

$$s_{\mathcal{G}}(v_i, v_j; \boldsymbol{\theta}) = \sum_{k=1}^{K} \theta_k s_k(v_i, v_j), \quad \text{s.t.} \quad \boldsymbol{\theta} \in \mathcal{S}^K \tag{4.3}$$

where $\mathcal{S}^K := \{\boldsymbol{\theta} \in \mathbb{R}^K : \boldsymbol{\theta} \geq \mathbf{0}, \boldsymbol{\theta}^\top \mathbf{1} = 1\}$ is the K-dimensional probability simplex, and $s_k(v_i, v_j)$ is a similarity metric that depends on all k-hop paths of possibly repeated nodes that start from v_i and end at v_j (or vice-versa). Thus, $s_{\mathcal{G}}(\cdot, \cdot; \boldsymbol{\theta})$ contains all k-hop interactions between two nodes, each weighted by a non-negative importance score θ_k with $k = 1, \ldots, K$.

Let $\mathbf{S}$ be any similarity matrix that is characterized by the same sparsity pattern as the adjacency matrix, that is

$$S_{i,j} = \begin{cases} s_{i,j}, & (i,j) \in \mathcal{E} \\ 0, & (i,j) \notin \mathcal{E} \end{cases}, \tag{4.4}$$

where $\{s_{i,j}\}$s denote the generic non-negative values of entries that correspond to edges of $\mathcal{G}$. Maintaining the same sparsity pattern as $\mathbf{A}$ allows for the (i,j) entry of $\mathbf{S}^k$ to be interpreted as a measure of influence between v_i and v_j that depends on all k-hop paths that connect them; that is, $\left[\mathbf{S}^k\right]_{i,j} = s_k(v_i, v_j)$. For instance, selecting $\mathbf{S} = \mathbf{A}$ is equivalent to using the k-step similarity $s_k(v_i, v_j) = |\{k - \text{length paths connecting } v_i \text{ to } v_j\}|$ [136]. Likewise, if $\mathbf{S} = \mathbf{A}\mathbf{D}^{-1}$ where $\mathbf{D} = \text{diag}(\mathbf{1}^T\mathbf{A})$, then $s_k(v_i, v_j)$ can be interpreted as the probability that a random walk starting from v_j lands on v_i after exactly k steps, e.g., [28]. Thus, for a properly selected $\mathbf{S}$ with entries as in (4.4), tunable multihop similarity metrics in (4.3) can be collected as entries of the power series matrix

$$\mathbf{S}_{\mathcal{G}}(\boldsymbol{\theta}) = \sum_{k=1}^{K} \theta_k \mathbf{S}^k, \quad \text{s.t.} \quad \boldsymbol{\theta} \in \mathcal{S}^K \ . \tag{4.5}$$

Upon substituting (4.5) into (4.2) yields the tunable embeddings $\mathbf{E}^*(\boldsymbol{\theta})$ that depend on the choice of parameters $\boldsymbol{\theta}$. From the eigen-decomposition $\mathbf{S} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{U}^\top$, and given that $\mathbf{U}^\top\mathbf{U} = \mathbf{I}$, we readily arrive at

$$\mathbf{S}^k = \mathbf{U}\boldsymbol{\Sigma}^k\mathbf{U}^\top \tag{4.6}$$

and after plugging (4.6) into (4.5), we obtain

$$\mathbf{S}_{\mathcal{G}}(\boldsymbol{\theta}) = \mathbf{U}\left(\sum_{k=1}^{K}\theta_k\boldsymbol{\Sigma}^k\right)\mathbf{U}^\top, \quad \text{s.t.} \quad \boldsymbol{\theta}\in\mathcal{S}^K\,. \tag{4.7}$$

Furthermore, the truncated singular pairs of $\mathbf{S}_{\mathcal{G}}(\boldsymbol{\theta})$ conveniently follow from those of $\mathbf{S}$, and they have to be computed once. Specifically, the truncated singular vectors and singular values are $\mathbf{U}_d(\boldsymbol{\theta}) = \mathbf{U}_d$ and $\boldsymbol{\Sigma}_d(\boldsymbol{\theta}) = \sum_{k=1}^{K}\theta_k\boldsymbol{\Sigma}_d^k$, respectively. Thus, if $\mathbf{S}\in\mathrm{Sym}_N$ the solution to (4.2) with $\mathbf{S}_{\mathcal{G}}$ parametrized by $\boldsymbol{\theta}$ is simply given as

$$\mathbf{E}^*(\boldsymbol{\theta}) = \mathbf{U}_d\sqrt{\boldsymbol{\Sigma}_d(\boldsymbol{\theta})}\,. \tag{4.8}$$

Note that this holds only for non-negative parameters $\theta_k \geq 0\ \forall\ k$. If $\theta_k < 0$ for at least one $k\in\{1,\ldots,K\}$, then the diagonal entries of $\boldsymbol{\Sigma}_d(\boldsymbol{\theta})$ cannot be guaranteed to be non-negative and sorted in decreasing order, which would cause $(\mathbf{U}_d(\boldsymbol{\theta}), \boldsymbol{\Sigma}_d(\boldsymbol{\theta}))$ to *not* be a valid SVD pair.

Having narrowed down $\mathbf{S}_{\mathcal{G}}$ to belong to the parametrized family in (4.5), we proceed to select an appropriate sparsity-preserving $\mathbf{S}$ in order to obtain a solid model.

4.0.3 Spectral multihop embeddings

While any symmetric $\mathbf{S}$ that obeys (4.4) can be used for constructing multihop similarities (cf. (4.5)), judicious designs of $\mathbf{S}$ can effect certain desirable properties. Bearing this in mind, consider the following identity

$$\mathbf{S}\in\mathcal{P}_N^+ \iff \mathbf{S} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{U}^\top = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^\top \tag{4.9}$$

where $\mathcal{P}_N^+$ denotes the space of $N\times N$ symmetric positive definite (SPD) matrices, and $\boldsymbol{\Lambda}$ is the diagonal matrix that contains the eigenvalues of $\mathbf{S}$ sorted in decreasing order. For SPD matrices as in (B.4), the SVD is identical to the eigenvalue decomposition (EVD). Thus, if $\mathbf{S}\in\mathcal{P}_N^+$, the solution to (4.2) is also given as (cf. (4.8))

$$\mathbf{E}^*(\boldsymbol{\theta}) = \mathbf{U}_d\sqrt{\boldsymbol{\Lambda}_d(\boldsymbol{\theta})} \tag{4.10}$$

where $\mathbf{U}_d$ are also the first d *eigenvectors* of $\mathbf{S}$, and

$$\mathbf{\Lambda}_d(\boldsymbol{\theta}) = \sum_{k=1}^{K} \theta_k \mathbf{\Lambda}_d^k$$

is the Kth order polynomial of its eigenvalues defined by $\boldsymbol{\theta}$.

Consider now specifying $\mathbf{S}$ as

$$\mathbf{S} = \frac{1}{2}\left(\mathbf{I} + \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}\right). \tag{4.11}$$

Recalling that

$$\lambda_i\left(\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}\right) \in [-1, 1] \; \forall \; i,$$

and after using the identity shifting and scaling, we deduce that $\lambda_i(\mathbf{S}) \in [0, 1] \; \forall \; i$; hence, matrix $\mathbf{S}$ in (4.11) is SPD. It can also be readily verified that the first d eigenvectors of $\mathbf{S}$ coincide with the eigenvectors corresponding to the d smallest eigenvalues of the symmetric normalized Laplacian matrix

$$\mathbf{L}_{\text{sym}} := \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}. \tag{4.12}$$

These smallest eigenvalues are known to contain useful information on cluster structures of different resolution levels, a key property that has been successfully employed by spectral clustering [124]. Intuitively, assigning weight $\boldsymbol{\theta}_k$ to k-hop paths in the node similarity of (4.5), is equivalent to shrinking the d-dimensional spectral node embeddings (rows of $\mathbf{U}_d$) coordinates according to $\mathbf{\Lambda}_d(\boldsymbol{\theta})$. Interestingly, assigning large weights to longer paths ($K \gg 1$) is equivalent to fast shrinking the coordinates that correspond to small eigenvalues and capture the fine-grained structures and local relations, what leads to a coarse, high-level cluster description of the graph.

4.0.4 Relation to random walks

Apart from the spectral embedding interpretation discussed in the last subsection, using powers of (4.11) to capture multihop similarities also admits an interesting random walk interpretation. We begin by expressing the kth power of $\mathbf{S}$ as

$$\mathbf{S}^k = \frac{1}{2^k}\left(\mathbf{I} + \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}\right)^k$$

$$= \sum_{\tau=0}^{k} \alpha_\tau(k) \left(\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}\right)^\tau \tag{4.13}$$

where the sequence

$$\alpha_\tau(k) := \begin{cases} \frac{1}{2^k}\binom{k}{\tau}, & 0 \le \tau \le k \\ 0, & \text{else} \end{cases} \tag{4.14}$$

can be interpreted as nonzero weights that $\mathbf{S}^k$ assigns to all paths with the number of hops *up to* k (see Fig. 4.2).

Using (4.13) and (4.14), the multihop similarity in (4.5) becomes

$$\begin{aligned} \mathbf{S}_\mathcal{G}(\boldsymbol{\theta}) &= \sum_{\tau=0}^{K} c_\tau(\boldsymbol{\theta}) \left(\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}\right)^\tau \\ &= \mathbf{D}^{-1/2} \left(\sum_{\tau=0}^{K} c_\tau(\boldsymbol{\theta})\mathbf{P}^\tau\right) \mathbf{D}^{1/2} \end{aligned} \tag{4.15}$$

where

$$c_\tau(\boldsymbol{\theta}) := \sum_{k=1}^{K} \theta_k \alpha_\tau(k) \tag{4.16}$$

and $\mathbf{P} = \mathbf{A}\mathbf{D}^{-1}$ is the probability transition matrix of a simple random walk defined over $\mathcal{G}$; that is, $P_{i,j}$ is the probabiity that a random walker positioned on node (state) j transitions to node i in one step. Thus, the k-hop similarity function defined in (4.3) is expressed as

$$s_\mathcal{G}(v_i, v_j, \boldsymbol{\theta}) = \sqrt{\frac{d_j}{d_i}} \sum_{\tau=0}^{K} c_\tau(\boldsymbol{\theta}) \Pr\{X_\tau = v_i | X_0 = v_j\} \tag{4.17}$$

where

$$\Pr\{X_\tau = v_i | X_0 = v_j\} := [\mathbf{P}^\tau]_{ij}$$

is the probability that a random walk starting from v_j lands on v_i after τ steps.

Interestingly, $\mathbf{S}_\mathcal{G}(\boldsymbol{\theta})$ does not weigh landing probabilities of different lengths independently. Instead, it accumulates the latter as weighted combinations (cf. (4.16)) in a basis of "wavelet"-type functions of different resolution (see Fig. 1).

Having established links to spectral clustering and random walks, our novel $\mathbf{S}_\mathcal{G}(\boldsymbol{\theta})$ is well motivated as a family of node similarity matrices. Nevertheless, before devising an algorithm for

learning $\boldsymbol{\theta}$ and testing it on real graphs, we will evaluate how well the basis $\{\mathbf{S}^k\}_{k=1}^K$, on which $\mathbf{S}_{\mathcal{G}}(\boldsymbol{\theta})$ is built, can capture underlying node similarities.

4.1 Model expressiveness

This section introduces a performance metric that quantifies how well a node similarity matrix derived from the graph itself matches the "true" underlying similarity structure between nodes. The discussion is followed by numerical evaluation of the performance of different similarity matrices (including the one in (4.13)) on graphs that are generated according to the stochastic block model [137].

To begin, suppose that for a given set of nodes, an adjacency matrix $\mathbf{A}$ is generated as

$$\mathbf{A} \sim f_A(\mathbf{A})$$

where $f_A(\mathbf{A})$ is a probability density function defined over the space of all possible adjacency matrices. Let the "true" underlying similarity between nodes v_i and v_j be

$$s^*(v_i, v_j) := \Pr\{(i,j) \in \mathcal{E}\} = \mathbb{E}_{f_A}\left[A_{i,j}\right]$$

which is the probability that the two nodes are connected. The "true" similarity matrix is thus given as the expected adjacency matrix

$$\mathbf{S}^* := \mathbb{E}_{f_A}\left[\mathbf{A}\right].$$

We define the quality-of-match (QoM) between the underlying $\mathbf{S}^*$ and any similarity $\hat{\mathbf{S}} = F(\mathbf{A})$ estimated from the adjacency matrix as

$$\mathrm{QoM} := \mathbb{E}_{f_A}\left[\mathrm{PC}\left(\mathbf{S}^*, F(\mathbf{A})\right)\right] \tag{4.18}$$

where

$$\mathrm{PC}\left(\mathbf{X}_1, \mathbf{X}_2\right) := \frac{\left(\mathrm{vec}\left(\mathbf{X}_1\right)\right)^\top \mathrm{vec}\left(\mathbf{X}_2\right)}{\|\mathbf{X}_1\|_F \|\mathbf{X}_2\|_F} \tag{4.19}$$

is the Pearson correlation between two matrices $\mathbf{X}_1$ and $\mathbf{X}_2$, with $\mathrm{vec}\left(\mathbf{X}\right)$ denoting matrix vectorization. The latter is used for appropriate rescaling of the "true" similarity matrix in order

for the comparison with $\mathbf{S}_{\mathcal{G}}$ to be meaningful. Intuitively, (4.18) measures how well the estimated node similarities in $\hat{\mathbf{S}}$ are expected to match the pattern of true underlying similarities in $\mathbf{S}^*$, when edges are generated according to the known $f_A(\cdot)$.

4.1.1 Numerical experiments and observations

We numerically evaluate the QoM achieved by different similarity matrices, on a set of N nodes whose interconnections are generated according to a stochastic block model (SBM). For this set of experiments, we divided the nodes into three clusters of equal size

$$\mathcal{C}_l = \{i : (l-1)N/3 \leq i \leq lN/3\},\ l \in \{1, 2, 3\}$$

with inter- and intra-connection probabilities

$$\Pr\{(i,j) \in \mathcal{E}\} = \begin{cases} p & , (i,j) \text{ in the same } \mathcal{C}_l \\ cq & , i \in \mathcal{C}_1 \text{ and } j \in \mathcal{C}_3 \\ q & \text{else} \end{cases} \tag{4.20}$$

where p is the probability of connection when two nodes belong to the same cluster, and $c < 1$ introduces asymmetry and a hierarchical clustering organization (see Fig. 2-top left), by making two of the clusters less likely to connect; we have related Python scripts available.[1] The SBM probability matrix [137] is given as

$$\mathbf{W}_{\text{sbm}} = \begin{bmatrix} p & q & cq \\ q & p & q \\ cq & q & p \end{bmatrix} \tag{4.21}$$

and the underlying similarity can be expressed as

$$\mathbf{S}^* = \mathbb{E}\left[\mathbf{A}\right] = \mathbf{W}_{\text{sbm}} \otimes \left(\mathbf{1}_{N/3}\mathbf{1}_{N/3}^T\right) - \text{diag}(p\mathbf{1}_N) \tag{4.22}$$

where $\otimes$ denotes the Kronecker product.

For each experiment, we set $N = 150$ and generated a graph according to (4.20). We then compared the QoM between (4.22) and the kth power of the proposed (4.11), the kth power of

[1] https://github.com/DimBer/ASE-project/tree/master/sim_tests

the adjacency ($\mathbf{A}^k$), as well as each of the following well known similarity metrics:

1. $\hat{\mathbf{S}}_{PPR} := (1-\alpha)(\mathbf{I}-\alpha\mathbf{A}\mathbf{D}^{-1})^{-1}$: the steady state probability that a random walk restarting at v_j with probability $1-\alpha$ at every step is located at v_i. Essentially a personalized PageRank (PPR) computed for every node of the graph, inheriting the properties of the celebrated centrality measure [24, 41, 71].

2. $\hat{\mathbf{S}}_{KATZ} := (1-\beta)(\mathbf{I}-\beta\mathbf{A})^{-1}\mathbf{A}$: the Katz index [136], an exponentially weighted summation over paths of all possible hops between two nodes.

3. $\hat{\mathbf{S}}_{NEIGH} := \mathbf{A}^2$: the number of common neighbors that every pair of nodes shares.

4. $\hat{\mathbf{S}}_{AA} := \mathbf{A}\mathbf{D}^{-1}\mathbf{A}$: Adamic-Adar [8] is a variant of common neighbors where each set of neighbors is weighted inversely proportional to its cardinality.

The resulting QoM was averaged over 200 experiments. Parameters α in $\hat{\mathbf{S}}_{PPR}$ and β in $\hat{\mathbf{S}}_{KATZ}$ were tuned to maximize the performance of the metrics. Figure 3 depicts QoM as a function of k, for three different scenarios.

In the first scenario (Fig. 3-a), with graphs being dense and clustered ($p = 0.3$, $q = 0.1$), the proposed $\mathbf{S}^k$ improves sharply in the first few steps, reaching maximum QoM after 4 or 5 steps, and gradually decreases as k continues to increase. The kth order proximities that are given as entries of $\mathbf{A}^k$ follow a similar trend, however their QoM peaks shortly after 2 or 3 steps and declines fast for larger k. The matrix plots of a randomly selected experiment depicted in Fig. 2 can aid in understanding the underlying mechanism that gives rise to this highly step-dependent behavior. Specifically, $\mathbf{S}^1$ (bottom left) that has the same sparsity pattern as the adjacency is a poor match to the dense block-structure of $\mathbf{S}^*$. On the other side of the spectrum, $\mathbf{S}^{15}$ (bottom right) is too "flat" and also a poor similarity metric. Meanwhile, taking $k = 6$ promotes enough mixing without "dissipating." As a result, $\mathbf{S}^6$ (bottom center) visibly matches the structure of $\mathbf{S}^*$. Interestingly, for $k \in [4, 10]$ the proposed $\mathbf{S}^k$ surpasses in QoM all other similarity metrics that were tested. Nevertheless, the simple $2-$hop Adamic-adar, common-neighbors similarities perform reasonably well by exploiting the relatively dense structure of the graphs.

Results were markedly different in the second scenario shown in Fig. 3-b. Here, graphs were generated with the same clustering structure but significantly sparser, with edge probability parameters $p = 0.15$ and $q = 0.05$. For sparser graphs, $\mathbf{A}^k$ and $\mathbf{S}^k$ require more steps to reach peak QoM (4 and 9 respectively). Similarly, PPR which relies on long paths performs much

better than the short-reaching Adamic-Adar. This behavior is intuitively reasonable because the sparser a graph is, the longer become the paths that need to be explored around each node, in order for the latter to "gauge" its position on the graph.

Finally, a third scenario (Fig. 3-c) was examined, where each graph was generated without a clustering structure ($p = q = 0.1$ and $c = 1$); essentially an Erdos-Renyi graph. For this degenerate case that is of no real practical interest, all pairs of nodes are equally similar; this type of similarity requires infinitely long paths to be described.

In a nutshell, the presented numerical study hints at the two following facts. First, $\mathbf{S}^k$ can successfully model similarities that are based on grouping nodes in arbitrary and multilevel sets with variable degrees of homophily and heterophily. The second fact, is that the performance of $\mathbf{S}^k$ varies significantly with k. Moreover, the way that k affects performance may also vary from graph to graph, depending on the underlying properties – what suggests viewing this way as a graph "signature" that is also validated by the real graphs in Section 6. Thus, a principled means of specifying $\mathbf{S}_{\mathcal{G}}(\boldsymbol{\theta})$ by learning the parameters that match this graph "signature" in an unsupervised mode, is highly motivated.

4.2 Unsupervised similarity learning

We have arrived at the point where for a given graph, it is prudent to select a specific $\boldsymbol{\theta} \in \mathcal{S}^K$ without supervision. Following the discussion in Section 3, it would be ideal to fit $\mathbf{S}_{\mathcal{G}}(\boldsymbol{\theta})$ to a true $\mathbf{S}^*$ by minimizing an expected cost

$$\boldsymbol{\theta}^* = \arg \min_{\boldsymbol{\theta} \in \mathcal{S}^K} \mathbb{E}_{f_A} \left[\ell \left(\mathbf{S}^*, \mathbf{S}_{\mathcal{G}}(\mathbf{A}; \boldsymbol{\theta}) \right) \right] . \tag{4.23}$$

Unfortunately, we only have one realization $\mathbf{A}$ of $f_A(\cdot)$, which means that without prior knowledge, the best approximation of $\mathbf{S}^*$ that we can obtain is the adjacency matrix itself, that is $\mathbf{S}^* \approx \mathbf{A}$. Using this approximation yields

$$\min_{\boldsymbol{\theta} \in \mathcal{S}^K} \ell \left(\mathbf{A}, \mathbf{S}_{\mathcal{G}}(\mathbf{A}; \boldsymbol{\theta}) \right) . \tag{4.24}$$

While straightforward, (4.24) yields embeddings with limited generalization capability. Simply put, regardless of the choice of $\ell(\cdot)$, solving (4.24) amounts to predicting a set of edges by tuning a similarity metric that is generated by the *same* set of edges.

To mitigate overfitting but also promote generalization of the similarity metric and of the resulting embeddings, we explore the following idea. Suppose we are given a pair $\mathbf{A}_1, \mathbf{A}_2$ of adjacency matrices both drawn independently from $f_A(\cdot)$. In this case, we would be able to use one as approximation of $\mathbf{S}^* \approx \mathbf{A}_1$, and the other to form the multihop similarity matrix $\mathbf{S}_{\mathcal{G}}(\mathbf{A}_2; \boldsymbol{\theta})$; parameters $\boldsymbol{\theta}$ can then be learned by solving

$$\min_{\boldsymbol{\theta} \in \mathcal{S}^K} \ell \left(\mathbf{A}_1, \mathbf{S}_{\mathcal{G}}(\mathbf{A}_2; \boldsymbol{\theta}) \right). \tag{4.25}$$

Since separate samples are not available, we approximate the aforementioned process by randomly extracting part of $\mathbf{A}$ and approaching (4.25) as

$$\min_{\boldsymbol{\theta} \in \mathcal{S}^K} \ell_{\mathcal{S}} \left(\mathbf{A}, \mathbf{S}_{\mathcal{G}}(\mathbf{A} * \mathbf{S}^c; \boldsymbol{\theta}) \right) \tag{4.26}$$

where $\mathcal{S} \in \{1, \ldots, N\}^2$ is a subset of all possible pairs of nodes with $|\mathcal{S}| = N_s$, and $\mathbf{S}^c$ is an $N \times N$ binary section matrix with $S^c_{i,j} = 0$, if $\{i, j\} \in \mathcal{S}$, and $S^c_{i,j} = 1$, otherwise. Furthermore, $\ell_{\mathcal{S}}(\cdot, \cdot)$ in (4.26) denotes cost $\ell(\cdot, \cdot)$ applied selectively only to entries of the matrix variables that belong to $\mathcal{S}$. Here, such that $\mathcal{S} = \mathcal{S}^+ \cup \mathcal{S}^-$, with $\mathcal{S}^+ \in \mathcal{E}$ being as subset of the edges and $\mathcal{S}^- \in \{1, \ldots, N\}^2 \setminus \mathcal{E}$ a subset of node index tuples that are not connected (non-edges). To balance the influence of existing and non-existing edges, we use subsets of equal cardinality, that is $|\mathcal{S}^+| = |\mathcal{S}^-| = N_s/2$.

To arrive from the unsupervised similarity learning framework (4.26) to a practical method, it remains to specify two modular sub-systems: one responsible for sampling edges, and one specifying $\ell(\cdot, \cdot)$ to find $\boldsymbol{\theta}^*$ by solving (4.26).

4.2.1 Edge sampling

The choice of the sampling scheme for $\mathcal{S}$ plays an important role in the overall performance of the proposed adaptive embedding framework. Ideally, edge sampling should take into account the following criteria.

1. Sample $\mathcal{S}^+$ should be representative of the graph;
2. Edge removal should inflict minimal perturbation;
3. Edge removal should avoid isolating nodes; and

4. Sampling scheme should be simple and scalable.

Aiming at a 'sweet spot' of these objectives, we populate $\mathcal{S}^+$ by sampling edges according to the following procedure: first, a node v_1 is sampled uniformly at random from $\mathcal{V}$; then, a second node v_2 is sampled uniformly from the neighborhood set $\mathcal{N}_{\mathcal{G}}(v_1)$ of v_1. The selected edge is removed only if both adjacent nodes have degree greater than one. Non-edges $\mathcal{S}^-$ are obtained by uniform sampling without replacement over $\{1,\dots,N\}^2 \setminus \mathcal{E}$. The overall procedure is summarized in Algorithm 7. For $N_s \ll N$, sampling probabilities remain approximately unchanged despite the removals, since the probability of selecting the same node is relatively small. Thus, one may approximate $\Pr\{e_t = (i,j)\} \approx \Pr\{e_0 = (i,j)\}$, and assuming for simplicity that $d_i > 1 \forall i$, it follows that

$$\begin{aligned}\Pr\{e_0 = (i,j)\} &= \Pr\{v_1 = i, v_2 = j\} + \Pr\{v_1 = j, v_2 = i\} \\ &= \Pr\{v_2 = i | v_1 = j\} \Pr\{v_1 = j\} + \Pr\{v_2 = j | v_1 = i\} \Pr\{v_1 = i\} \\ &= \frac{1}{d_j}\frac{1}{N} + \frac{1}{d_i}\frac{1}{N} \propto \frac{d_i + d_j}{d_i d_j}, \end{aligned} \tag{4.27}$$

meaning that edge $e = (i,j)$ is removed with probability that is proportional to the harmonic mean of the degrees of the nodes that it connects. As shown in [92], the perturbation that the removal of edge $e = (i,j)$ inflicts on the spectrum of an undirected graph is proportional to $d_i d_j$; that is, removing edges that connect high-degree nodes leads to higher perturbation. Thus, Algorithm 7 tends to inflict minimal perturbation by sampling with probability that is inversely proportional to $d_i d_j$ for $d_i,\ d_j \gg 1$; this is because the denominator of (4.27) dominates its numerator for large degrees. On the other hand, for smaller d_i and d_j, the numerator ensures relatively high probabilities for moderate-degree nodes. The combination of the two effects yields edge samples that are fairly representative of the graph, while inflicting low perturbation when removed.

4.2.2 Parameter training

Subsequently, for a given sample $\mathcal{S}$, we can obtain the corresponding optimal parameters as (cf. (4.26))

$$\boldsymbol{\theta}^*_{\mathcal{S}} = \arg\min_{\boldsymbol{\theta} \in \mathcal{S}^K} \sum_{i,j \in \mathcal{S}} \ell\left(A_{i,j}, s_{\mathcal{G}^-}(v_i, v_j; \boldsymbol{\theta})\right) \tag{4.28}$$

where $\mathcal{G}^- := (\mathcal{V}, \mathcal{E} \setminus \mathcal{S}^+)$ is the original graph with the randomly sampled subset $\mathcal{S}^+$ of edges removed.

Interestingly, one way that (4.28) could be solved is by explicitly computing the entries of $\mathbf{S}_\mathcal{G}(\boldsymbol{\theta})$ that are in $\mathcal{S}$. This would require performing K sparse matrix-vector products to obtain every column of $\mathbf{S}^k$ for $k \in \{1, \dots, K\}$, for all the columns that contain sampled entries. In the worst case, if all nodes in the tuples of $\mathcal{S}$ correspond to different columns of $\mathbf{S}_\mathcal{G}(\boldsymbol{\theta})$, two random walks are required for every tuple, for a total of $2N_s$ random walks. This requires $\mathcal{O}\left(N_s K |\mathcal{E}|\right)$ computations, and $\mathcal{O}\left(N_s N\right)$ memory if they are to be performed concurrently or in matrix form. Since K will typically be in the order of tens, these requirements will be affordable, if N_s is relatively small. Nevertheless, they quickly become cumbersome for $N_s \gg K$, which may be necessary to estimate the K-dimensional $\boldsymbol{\theta}$.

Algorithm 6 ADAPTIVE SIMILARITY EMBEDDING

Input: $\mathcal{G}$ **Output:** $\mathbf{E}$

// Training phase
$\boldsymbol{\Theta} = \emptyset$
while $|\boldsymbol{\Theta}| < T_s$ **do**
  $\mathcal{G}^-, \mathcal{S}^+, \mathcal{S}^- =$ SAMPLE EDGES($\mathcal{G}$)
  $\boldsymbol{\theta}^*_\mathcal{S} =$ TRAIN PARAMETERS($\mathcal{G}^-, \mathcal{S}^+, \mathcal{S}^-$)
  $\boldsymbol{\Theta} = \boldsymbol{\Theta} \cup \boldsymbol{\theta}^*_\mathcal{S}$
end while
$\boldsymbol{\theta}^* = T_s^{-1} \sum_{\boldsymbol{\theta} \in \boldsymbol{\Theta}} \boldsymbol{\theta}$

// Embedding phase
$\mathbf{S} = \frac{1}{2}\left(\mathbf{I} + \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}\right)$
$\mathbf{S} = \mathbf{U}_d \boldsymbol{\Sigma}_d \mathbf{U}_d^T$
$\boldsymbol{\Sigma}_d(\boldsymbol{\theta}^*) = \sum_{k=1}^K \theta_k^* \boldsymbol{\Sigma}_d^k$

return $\mathbf{E} = \mathbf{U}_d \sqrt{\boldsymbol{\Sigma}_d(\boldsymbol{\theta}^*)}$

Instead, we will rely on the fact that the proposed embeddings are smooth and differentiable wrt to $\boldsymbol{\theta}$ (cf. (4.10)), to develop a solution that allows for selecting arbitrarily large N_s, using the approximation

$$s_{\mathcal{G}^-}(v_i, v_j; \boldsymbol{\theta}) \approx s_\mathcal{E}(\mathbf{e}_i^-(\boldsymbol{\theta}, \mathbf{e}_j^-(\boldsymbol{\theta})) = \left(\mathbf{e}_i^-(\boldsymbol{\theta})\right)^\top \mathbf{e}_j^-(\boldsymbol{\theta})$$

Algorithm 7 SAMPLE EDGES

Input: $\mathcal{G}$ **Output:** $\mathcal{G}^-, \mathcal{S}^+, \mathcal{S}^-$

// Sample edges
$\mathcal{S}^+ = \emptyset, \mathcal{G}^- = \mathcal{G}$
while $|\mathcal{S}^+| < N_s/2$ **do**
  Sample $v_1 \sim \text{Unif}\,(\mathcal{V})$
  if $|\mathcal{N}_{\mathcal{G}^-}(v_1)| > 1$ **then**
    Sample $v_2 \sim \text{Unif}\,(\mathcal{N}_{\mathcal{G}^-}(v_1))$
    if $|\mathcal{N}_{\mathcal{G}^-}(v_2)| > 1$ **then**
      $\mathcal{S}^+ = \mathcal{S}^+ \cup (v_1, v_2)$
      $\mathcal{G}^- = \mathcal{G}^- \setminus (v_1, v_2)$
    end if
  end if
end while

// Sample non-edges
$\mathcal{S}^- = \emptyset$
while $|\mathcal{S}^-| < N_s/2$ **do**
  Sample $(v_1, v_2) \sim \text{Unif}\,(\mathcal{V} \times \mathcal{V})$
  if $(v_1, v_2) \notin \mathcal{E}$ **then**
    $\mathcal{S}^- = \mathcal{S}^- \cup (v_1, v_2)$
  end if
end while

return $\mathcal{G}^-, \mathcal{S}^+, \mathcal{S}^-$

Algorithm 8 TRAIN PARAMETERS

Input: $\mathcal{G}, \mathcal{S}^+, \mathcal{S}^-$ **Output:** $\boldsymbol{\theta}^*_{\mathcal{S}}$
$\mathbf{S} = \frac{1}{2}\left(\mathbf{I} + \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}\right)$
$\mathbf{S} = \mathbf{U}_d \boldsymbol{\Sigma}_d \mathbf{U}_d^T$
$\mathcal{S} = \mathcal{S}^+ \cup \mathcal{S}^-$
Form $\mathcal{X}_{\mathcal{S}} = \{\mathbf{x}_{(i,j)}\}_{(i,j)\in\mathcal{S}}$ as in (4.30)

return $\boldsymbol{\theta}^*_{\mathcal{S}} =$ SIMPLEXSVM($\mathcal{X}_{\mathcal{S}}, \mathcal{S}^+, \mathcal{S}^-$)

$$= \left(\sqrt{\boldsymbol{\Sigma}_d^-(\boldsymbol{\theta})}\,\mathbf{u}_i^-\right)^\top \sqrt{\boldsymbol{\Sigma}_d^-(\boldsymbol{\theta})}\,\mathbf{u}_j^- = \left(\mathbf{u}_i^-\right)^\top \boldsymbol{\Sigma}_d^-(\boldsymbol{\theta})\mathbf{u}_j^-$$
$$= \mathbf{x}_{i,j}^\top\,\boldsymbol{\theta} \tag{4.29}$$

Algorithm 9 SIMPLEXSVM

Input: $\mathcal{X}, \mathcal{S}^+, \mathcal{S}^-$ **Output:** $\boldsymbol{\theta}^*$
$\boldsymbol{\theta}_0 = \frac{1}{K}\mathbf{1},\ t = 1$
while $\|\boldsymbol{\theta}_t - \boldsymbol{\theta}_{t-1}\|_\infty \geq \text{tol}$ **do**
  $t = t + 1, \eta_t = a/\sqrt{t}$
  $\mathcal{S}_a^+ = \{e \in \mathcal{S}^+ |\ \mathbf{x}_e^T \boldsymbol{\theta}_{t-1} \leq \epsilon\}$
  $\mathcal{S}_a^- = \{e \in \mathcal{S}^- |\ \mathbf{x}_e^T \boldsymbol{\theta}_{t-1} \geq -\epsilon\}$
  $\mathbf{g}_t = \sum_{e \in \mathcal{S}_a^-} \mathbf{x}_e - \sum_{e \in \mathcal{S}_a^+} \mathbf{x}_e$
  $\mathbf{z}_t = (1 - 2\eta_t\lambda)\boldsymbol{\theta}_{t-1} - \frac{\eta_t}{N_s}\mathbf{g}_t$
  $\boldsymbol{\theta}_t =$SIMPLEXPROJ($\mathbf{z}_t$)
end while

return $\boldsymbol{\theta}_t$

where

$$\mathbf{x}_{i,j} = \left(\mathbf{u}_i^- * \mathbf{u}_j^-\right)^\top \boldsymbol{\Sigma}_d^K, \tag{4.30}$$

and

$$\boldsymbol{\Sigma}_d^K = \begin{bmatrix} \sigma_1 & \sigma_1^2 & \cdots & \sigma_1^K \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{d-1} & \sigma_{d-1}^2 & \cdots & \sigma_{d-1}^K \\ \sigma_d & \sigma_d^2 & \cdots & \sigma_d^K \end{bmatrix}.$$

Conveniently, $\{\mathbf{x}_{i,j}\}$s act as features over every possible pair of nodes, which when linearly combined with weights $\boldsymbol{\theta}$ to produce similarities, allow us to approach (4.28) using well-understood learning and optimization tools. For instance, let $\ell(\cdot)$ be the Hinge loss

$$\ell(y, f) := \max(0, \epsilon - yf) \tag{4.31}$$

and define targets $y_{i,j} = 2A_{i,j} - 1$ such that $y_{i,j} \in \{-1, 1\}$. We can then equivalently express (4.28) as

$$\boldsymbol{\theta}_{\mathcal{S}}^* = \arg\min_{\boldsymbol{\theta} \in \mathcal{S}^K} \sum_{i,j \in \mathcal{S}} \max(0, \epsilon - y_{i,j}\mathbf{x}_{i,j}^\top\ \boldsymbol{\theta}) + \lambda\|\boldsymbol{\theta}\|_2^2 \tag{4.32}$$

where $\lambda \geq 0$ is the regularization parameter of the ℓ_2 regularization typically used to improve the generalization of SVMs [116]. To solve our variant of simplex-constrained SVMs (cf.

(4.32)), we employ the projected-gradient descent approach [22] that we describe in Algorithm 4, where SIMPLEXPROJ($\cdot$) is a subroutine that implements projections onto $\mathcal{S}^K$; the latter can be performed with $\mathcal{O}(K \log K)$ complexity as described in [33]. The overall parameter learning procedure for a given sample is summarized in Algorithm 3 [2].

In general, if runtime or computational resources allow, the sampling and training process described in the last two sections can be repeated T_s times to obtain different $\{\boldsymbol{\theta}^*_{\mathcal{S}}\}$s, which can then be averaged in order to reduce their variance. In practice, this may not be necessary if N_s is large enough, which will yield a near-deterministic $\boldsymbol{\theta}$. The overall proposed adaptive-similarity embedding (ASE) framework is summarized in Algorithm 1.

4.2.3 Complexity

The computational complexity of ASE is dominated by the cost of performing the truncated SVD of $\mathbf{S}$ in the training as well as testing phases of Algorithm 1. Relying on the sparsity ($|\mathcal{E}| \ll N^2$) and symmetry of $\mathbf{S}$, the Lanczos algorithm followed by EVD of a tridiagonal matrix yield the truncated SVD in a very efficient manner. Provided that $d \ll N$, the decomposition can be achieved in $\mathcal{O}(|\mathcal{E}|d)$ time and using $\mathcal{O}(Nd)$ memory. Therefore, for the $T_s \geq 1$ training rounds and a single embedding round of Algorithm 1, the overall complexity is $\mathcal{O}((T_s + 1)|\mathcal{E}|d)$.

4.3 Related work

Two recent embedding methods also pursue similarity matrices that combine walks of different lengths [136, 109]. Most relevant to the proposed ASE is the "Arbitrary-Order Proximity Preserved Network Embedding" [136] approach, where a method is proposed for obtaining the SVD of a polynomial of the adjacency matrix without having to recompute the singular vectors.

Compared to [136], we put forth the following contributions. First, we introduce a family of multihop similarities whose decomposition leads to embeddings that inherit the rich information contained in the spectral embeddings (cf. Section 2.3). An equally important contribution in terms of modeling is that our embeddings can be differentiated with respect to (wrt) weights $\boldsymbol{\theta}$ (cf. (4.29)-(4.32)), whereas the embeddings in [136] are non-differentiable wrt the weights. Hence,

[2] The SVM-based parameter learning presented here is by no means the only viable option. In our implementation, we also provide learning mechanisms based on least-squares, logistic regression, as well as finding the best single k. However here, due to space constrains, we present and report results of the SVM-based approach.

[136] can only proceed in a "forward" fashion given some order proximity weights $\boldsymbol{\theta}$, whereas our approach allows for "navigating" the space of possible similarity functions $s(v_i, v_j; \boldsymbol{\theta})$ in a smooth fashion, meaning that $\boldsymbol{\theta}$ can be learned with simple optimization on well-defined fitting models such as logistic regression or SVMs (cf. (4.32)). This leads to the third main contribution, which is a means of learning "personalized" $\boldsymbol{\theta}$ (cf. Section 4) in an unsupervised fashion, meaning without downstream information such as node or edge labels/attributes that can guide cross-validation in high-dimensional discretized parameter grids.

The second related embedding method presented in [109] builds on the concept of graph attention mechanisms to place weights on lengths of truncated random walks. These mechanisms are used to build a similarity matrix containing co-occurrence probabilities. The matrix is jointly decomposed by maximizing a graph-likelihood function. The model in [109] is a generalization of the ones implicitly adopted by [101] and [47], building on similar tools and concepts that emerge from natural language processing. Different from [101, 47] and the proposed ASE, [109] explicitly constructs and factorizes a dense $N \times N$ similarity matrix. The detailed procedure incurs complexity that is *cubic* wrt N, and becomes at best *quadratic* after model approximations, meaning that [109] scales rather poorly beyond small graphs.

4.4 Experimental Evaluation

The present section reports extensive experimental results on a variety of real-world networks. The aim of the presented tests is twofold. First, to determine and quantify the quality of the proposed ASE embeddings for different downstream learning tasks. Second, to analyze and interpret the resulting embedding parameters for different networks.

Datasets. In our experiments, we used the following real-world networks (see also Table 1).

1. **`ca-AstroPh`**. The Astro Physics collaboration network is from the e-print arXiv and covers scientific collaborations between co-authored papers submitted to Astro Physics category [2]. If an author i co-authored a paper with author j, the graph contains a undirected edge from i to j. If the paper is co-authored by k authors, this generates a completely connected (sub)graph on k nodes.

2. **`ca-CondMat`**. Condense Matter Physics collaboration network from ArXiv [2].

3. **CoCit**. A co-citation network of papers citing other papers extracted by [122]; labels represent conferences in which papers were published.

4. **com-DBLP**. Computer science research bibliography collaboration network [2].

5. **com-Amazon**. Network collected by crawling Amazon website [2]. It is based on "Customers Who Bought This Item Also Bought" feature of the Amazon website. If a product i is frequently co-purchased with product j, the graph contains an undirected edge from i to j.

6. **vk2016-17**. VK is a Russian all-encompassing social network. In [122], two snapshots of the network were extracted in November 2016 and May 2017, to obtain information about link appearance.

7. **email-Enron**. Enron email communication network covering all the email communication within a dataset of around half a million emails [2].

8. **PPI (H.Sapiens)**. Subgraph of the protein-protein interaction network for Homo Sapiens. The subgraph corresponds to the graph induced by nodes for which labels (representing biological states) were obtained from the hallmark gene sets [47].

9. **Wikipedia**. This is a co-occurrence network of words appearing in the first million bytes of the Wikipedia dump. The labels represent the Part-of-Speech (POS) tags inferred using the Stanford POS-Tagger [47].

10. **BlogCatalog**. A network of social relationships of the bloggers listed on the BlogCatalog website. The labels represent blogger interests inferred through the meta-data provided by the bloggers.

Methods. Experiments were run using the following *unsupervised* and *scalable* embedding methods.

1. **ASE**. Our proposed adaptive similarity embedding. Based on observations made in Sections 3, and to retain optimization stability, we set the maximum number of steps to $K = 10$. We also use the default SVM regularizer ($\lambda = 1$). To have a single learning round

Table 4.1: Network Characteristics

Graph	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{E}\rvert$	$\lvert\mathcal{Y}\rvert$	Density
PPI (H. Sapiens)	3,890	76,584	50	10^{-2}
Wikipedia	4,733	184,182	40	1.6×10^{-2}
BlogCatalog	10,312	333,983	39	6.2×10^{-3}
ca-CondMat	23,133	93,497	-	3.5×10^{-4}
ca-AstroPh	18,772	198,110	-	1.1×10^{-3}
email-Enron	36,692	183,831	-	2.7×10^{-4}
CoCit	44,312	195,362	15	2×10^{-4}
vk2016-17	78,593	2,680,542	-	8.7×10^{-4}
com-Amazon	334,863	925,872	-	1.7×10^{-5}
com-DBLP	317,080	1,049,866	-	2.1×10^{-5}

Table 4.2: Inferred parameters and interpretation

Graph	θ_1	θ_2	θ_3	θ_4	θ_5	θ_6	θ_7	θ_8	θ_9	θ_{10}	range	strength
PPI (H. Sapiens)	0.00	**0.14**	**0.31**	**0.29**	**0.21**	**0.04**	0.00	0.00	0.00	0.00	medium	medium
Wikipedia	0.00	0.00	0.00	0.00	0.00	0.00	0.00	**0.01**	**0.37**	**0.62**	long	strong
BlogCatalog	**1.00**	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	short	very strong
ca-CondMat	**0.55**	**0.33**	**0.12**	0.00	0.00	0.00	0.00	0.00	0.00	0.00	short	strong
ca-AstroPh	**0.76**	**0.24**	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	short	strong
email-Enron	**0.24**	**0.25**	**0.18**	**0.14**	**0.1**	**0.06**	**0.02**	0.00	0.00	0.00	medium	weak
CoCit	**0.61**	**0.33**	**0.06**	0.00	0.00	0.00	0.00	0.00	0.00	0.00	short	strong
vk2016-17	**0.71**	**0.29**	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	short	strong
com-Amazon	**0.10**	**0.10**	**0.10**	**0.10**	**0.09**	**0.09**	**0.09**	**0.09**	**0.09**	**0.09**	short	very weak
com-DBLP	**0.11**	**0.10**	**0.10**	**0.09**	**0.09**	**0.09**	**0.09**	**0.09**	**0.09**	**0.08**	short	very weak

with learned parameters having small enough variance, we sampled with $N_s/2 = 1,000$. We made our implementation of ASE freely available [3].

2. **VERSE** [122]. This is a scalable framework for generating node embeddings according to a similarity function by minimizing a KL-divergence-objective via stochastic optimization. We used the default version with similarity (PPR with $\alpha = 0.85$), as suggested and implemented by the authors.[4]

3. **Deepwalk** [101]. This approach learns an embedding by sampling random walks from each node, and applying word2vec-based learning on those walks. We use the default parameters proposed in [101], i.e., walk length $t = 80$, number of walks per node $\gamma = 80$, window size $w = 10$, and the scalable C++ implementation[5] provided in [122].

[3] https://github.com/DimBer/ASE-project
[4] https://github.com/xgfs/verse
[5] https://github.com/xgfs/deepwalk-c

4. **HOPE** [99]. This SVD-based approach approximates high-order proximities and leverages directed edges. We report the results obtained with the default parameters, i.e, Katz similarity as the similarity measure with β inversely proportional to the spectral radius.

5. **LINE** [120]. This approach learns a d-dimensional embedding in two steps, both using adjacency similarity. First, it learns $d/2$ dimensions using first-order proximity; then, it learns another $d/2$ features using second-order proximity. Last, the two halves are normalized and concatenated. We obtained a copy of the code[6], and run experiments with $T = 10^{10}$ samples (although $T = 10^9$ yielded the same accuracy for smaller graphs), and $s = 5$ negative samples, as described in the paper.

6. **Spectral**. This approach relies on the first d eigenvectors of $\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$. The baseline was developed for clustering [124], and has also been run as a benchmark for node embeddings [47]. In our case, spectral embedding is of particular interest since it can be obtained by column-wise normalization of the embeddings generated by the proposed method.

We excluded comparisons with Node2vec [47] and AROPE [136] because they use cross-validation for hyper-parameter selection. Thus comparing Node2vec and AROPE to methods such as LINE, Deepwalk, HOPE, VERSE, and EMB that all operate with *fixed* hyperparameters in a fully *unsupervised* manner would be unfair. We also excluded comparisons with GraRep [28] and M-NMF [104] due to their limited scalability ($\mathcal{O}(N^2 d)$ computational and $\mathcal{O}(N^2)$ memory complexity).

Evaluation methodology. Our experiment setting follows the one in [122]. All methods are set to embed nodes to dimension $d = 100$. Using the resulting embeddings as feature vectors, we evaluated their performance in terms of node classification and link prediction accuracy, and clustering quality. All experiments were repeated 10 times and reported are the averaged results.

Interpretation of results. One interesting aspect of the proposed ASE method, is that the inferred parameters $\boldsymbol{\theta}^*$ from the first phase of Algorithm 1 can be used to characterise the underlying similarity structure of the graph, and the way nodes "interact" over different path

[6] https://github.com/tangjianpku/LINE

lengths (short, medium, and long range). The "strength" of interactions is inferred by how uniform the coefficients of $\boldsymbol{\theta}^*$ are, and depend on the value of λ. Since the default value was $\lambda = 1$ for all graphs, the results can be interpreted as relative interaction strengths between them. The resulting $\{\boldsymbol{\theta}^*\}$s for all graphs are listed in Table 2.

It can be immediately observed that the type of node interactions varies significantly across different graphs, with similar behavior for graphs that belong to the same domain. Specifically, `ca-CondMat, ca-AstroPh`, and `CoCit` that belong to the citation/co-authorship domain all show relatively strong interactions of short range. `BlogCatalog` shows very strong short-range similarities of only one-hop neighborhood interactions among bloggers. On the other hand, the `Wikipedia` word co-occurrence network shows a strong tendency for long-range interactions; while other graphs, such as the `PPI` protein interaction network stay on the medium range.

Node classification. Graphs with labeled nodes are frequently used to measure the ability of embedding methods to produce features suitable for classification. For each experiment, nodes were randomly split to a training set and a test set. Similar to other works, and to cope with multi-label targets, we fed the training features and labels into the one-vs-the-rest configuration of logistic regression classifier provided by the `sklearn` Python library. In the testing phase, we sorted the predicted class probabilities for each node in decreasing order, and extracted the top-k_i ranking labels, were k_i is the true number of labels of node v_i. We then computed the Micro- and Macro-averaged F_1 scores [90] of the predicted labels.

Apart from comparisons with alternative embedding methods, node classification can reveal whether available node labels (metadata) are distributed in a manner that matches the node relations/interactions that are inferred by ASE. To reveal this information, we obtain embeddings for every $k \in \{1, \ldots, 10\}$ by ignoring the training phase and "forcing" $\boldsymbol{\theta}^* = \mathbf{e}_k$ in Algorithm 6, and then using each embedding for classification with 10% labeling rate. Figure 4.4 plots Micro and Macro F_1 for all labeled graphs as a function of k, while red shade is placed on the hops where the *unsupervised* ASE parameters $\boldsymbol{\theta}^*$ are non-zero (cf. Table 1). As seen in Fig. 4.4, the accuracy on the four labeled graphs evolves with k in a markedly different manner. Nevertheless, ASE identifies the trends and tends to assign non-zero weights to hops that yield a desirable trade-off between Micro and Macro F_1. Bearing in mind that ASE does *not* use labels for training or validation, this is rather remarkable considering the fact that $\boldsymbol{\theta}^*$ depends only on

the graph.

We also compared the classification accuracy of ASE embeddings with those of the alternative embedding approaches, with results plotted in Fig. 4.5. The plots for some method-graph pairs are not discernible when values are too low. While the relative performance of any given method varies from graph to graph, ASE adapts to each graph and yields consistently reliable embeddings, with accuracy that in most cases reaches or surpasses that of state-of-the-art methods, especially in terms of Macro F_1. The two exceptions are the Macro F_1 in `CoCit`, and Micro F_1 in `Wikipedia`, where VERSE and HOPE are correspondingly more accurate. Interestingly, HOPE achieving high Micro F_1 and low Macro F_1 in `Wikipedia` is in agreement with the findings in Fig. 4.4, combined with the fact that HOPE focuses on longer paths.

Link prediction. Link prediction is the task of estimating the probability that a link between two unconnected nodes will appear in the future. We repeat the experiment performed in [122] on the `vk2016-17` social network. For every possible edge, we build a feature vector as the Hadamard product between the embedded vectors of its two adjacent nodes. Using the two time instances of `vk2016-17`, we predict whether a new friendship link appears between November 2016 and May 2017, using 50% of the new links for training and 50% for testing. To train the binary logistic regression classifier, we also randomly sample non-existing edges as negative examples. The link prediction accuracy for different embeddings is reported in Table 3. While for this experiment ASE does not reach the accuracy of VERSE, it provides the second most accurate link prediction, far surpassing the also SVD-based HOPE and spectral embeddings.

Table 4.3: Link Prediction Accuracy on `vk2016-17`

	VERSE	ASE	LINE	Deepwalk	HOPE	Spectral
Acc.	0.79	0.75	0.74	0.69	0.62	0.60

Node clustering. Finally, the embedded vectors were used to cluster the nodes into different communities, using the `sklearn` library K-means with the default K-means++ initialization [12]. We evaluate the quality of node clustering with conductance, a well-known metric for measuring the goodness of a community [80]; conductance is minimized for large, well connected communities that are also well separated from the rest of the graph. Each plot in Fig. 4.6 gives the

average conductance across communities, as a function of the total number of clusters. Results indicate that the proposed ASE as well as the spectral clustering benchmark yield much lower conductance compared to other embeddings. Apparently, since ASE builds on the same basis of eigenvectors used by normalized spectral clustering, it inherits the property of the latter to approximately minimize the normalized-cut metric [124], which is very similar to conductance. A closer look at the resulting clusters, reveals that clustering beased on VERSE, Deepwalk, LINE, and HOPE splits graphs into very large communities of roughly equal size, cutting a large number of edges in the process. This is an indication that these methods are subject to a *resolution limit*, which is the inability to detect well-separated communities that are below a certain size [37]. On the other hand, Spectral and the proposed ASE separate the graph into a large-core component, and many smaller well-separated communities, a structure that many large-scale information networks have been observed to have [80]. Indeed, the conductance gap is smaller for `BlogCatalog`, which is relatively small and with less pronounced communities.

Parameter sensitivity. We also present results after varying ASE parameters and measured classification Micro F_1 accuracy for `PPI` with 10% labeling rate. The aim is to assess the sensitivity of ASE wrt its basic parameters. The plot on the left shows how increasing λ (cf. (4.32)) may decrease accuracy by forcing the entries of $\boldsymbol{\theta}^*$ to be close to uniform, thus losing the benefits of graph-specific adaptation. Regarding the number of sampled edges N_s, results (middle plot) indicate relative robustness of ASE embeddings, given a minimum number of samples. As expected, sampling a large number of edges may cause noticeable perturbation on the graph (even using the minimally-perturbing Algorithm 2); this may be causing a slight decrease in accuracy. Finally, the plot on the left depicts accuracy across a range of embedding dimensions d.

Runtime. Finally, we compared different embedding methods in terms of runtime. Results for all graphs are reported in Fig. 4.8. All experiments were run on a personal workstation with a quad-core i5 processor, and 16 GB of RAM. For our proposed ASE, we provide a light-weight yet highly portable implementation [7] that uses the SVDLIBC library [3] for sparse SVD. We also developed a more scalable implementation [8] that relies on (and requires installation of)

[7] https://github.com/DimBer/ASE-project/tree/master/portable
[8] https://github.com/DimBer/ASE-project/tree/master/slepc_based

the SLEPc package [58]; this scalable version can perform large-scale sparse SVD on multiple processes and distributed memory environments using the message-passing interface (MPI) [17]. We used the high-performance implementation for the five larger graphs, and the portable one for the five smaller ones. Evidently, ASE and HOPE that are SVD-based are orders of magnitudes faster than VERSE, Deepwalk, and LINE. The main factor that slows the latter down seems to be the large number of stochastic optimization iterations that these methods must perform to reach accurate embeddings. Nevertheless, it should be noted that sampling based methods enjoy nearly-full parallelization and could thus benefit more from highly multi-threaded environments. On the other hand, methods that rely on SVD (and EVD) can greatly benefit from decades of research on how to efficiently perform these decompositions, and a suite of stable and highly optimized software tools.

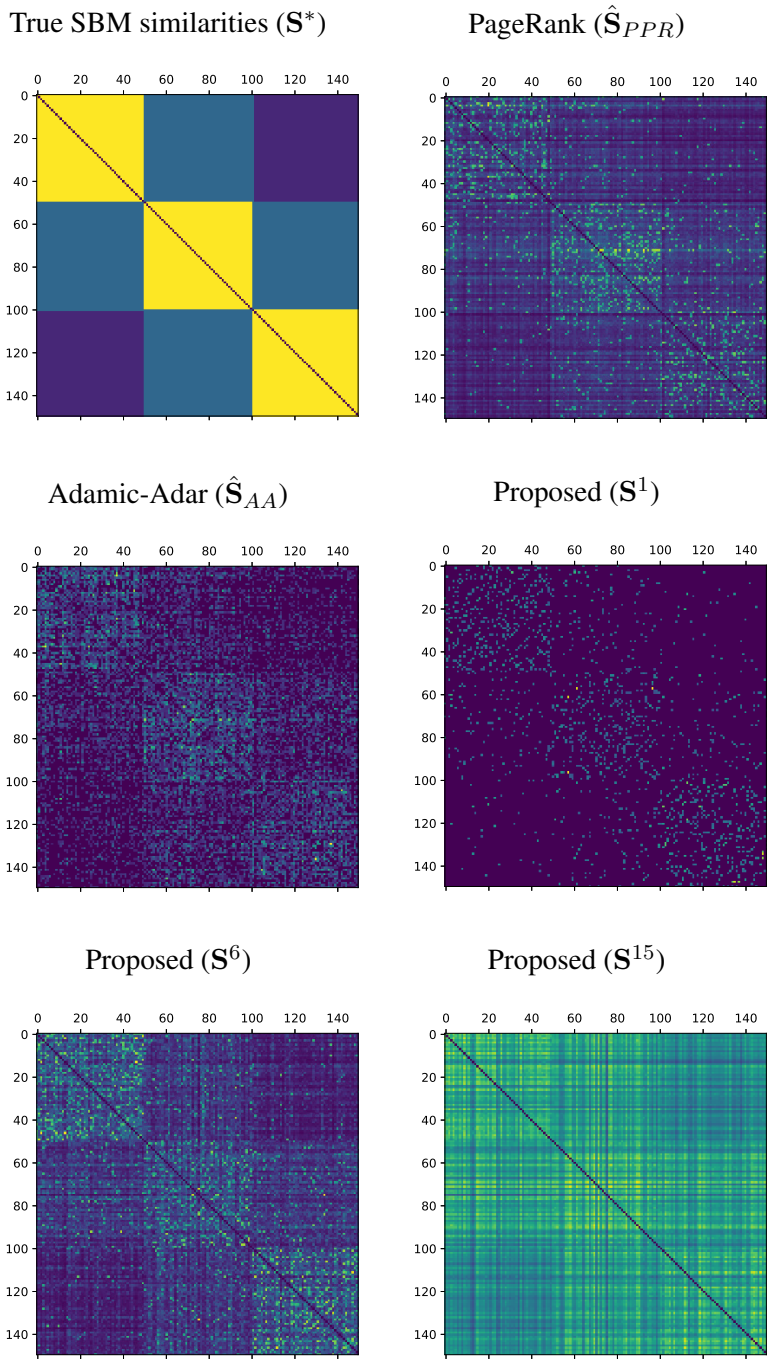

Figure 4.1: Depiction of groundtruth and estimated similarity matrices, as yielded from an instance of the numerical experiments described in Section 4.1.1.

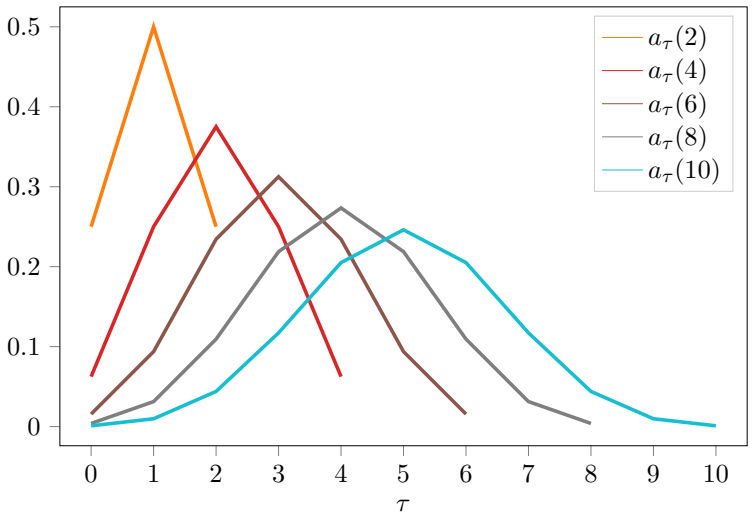

Figure 4.2: Matrix $\mathbf{S}^k$ is equivalent to applying "wavelet"-type weights $\alpha_\tau(k)$ over walks with hops $\leq k$.

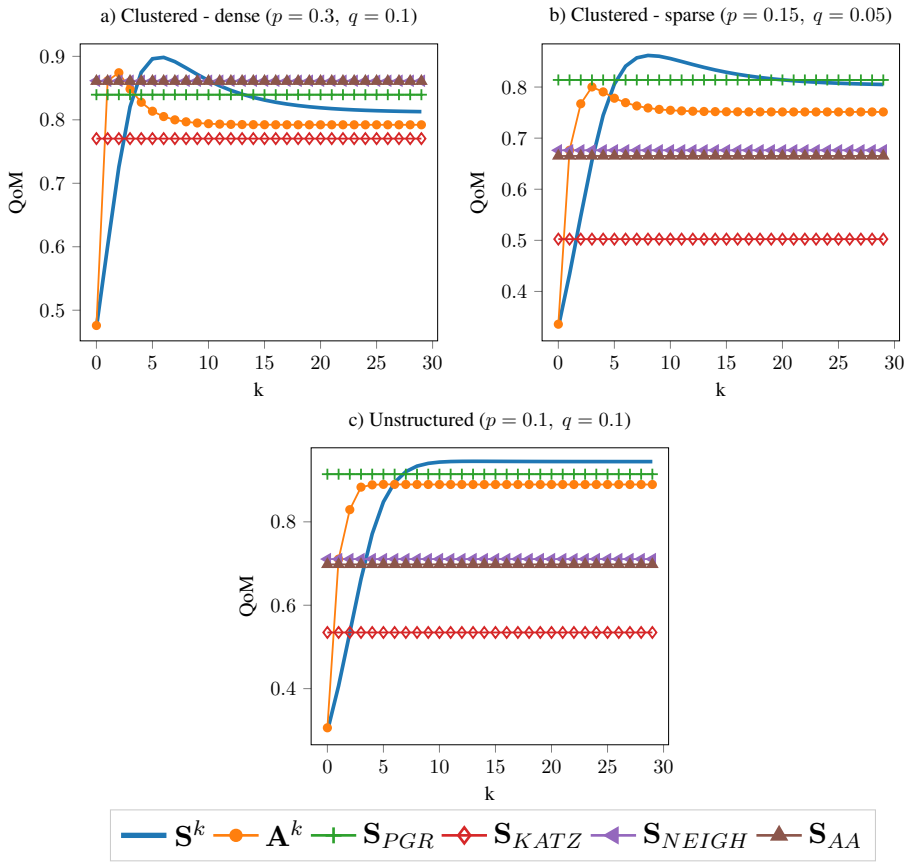

Figure 4.3: Quality of match between true SBM similarity and various estimates, as yielded from experiments of Section 4.1.1.

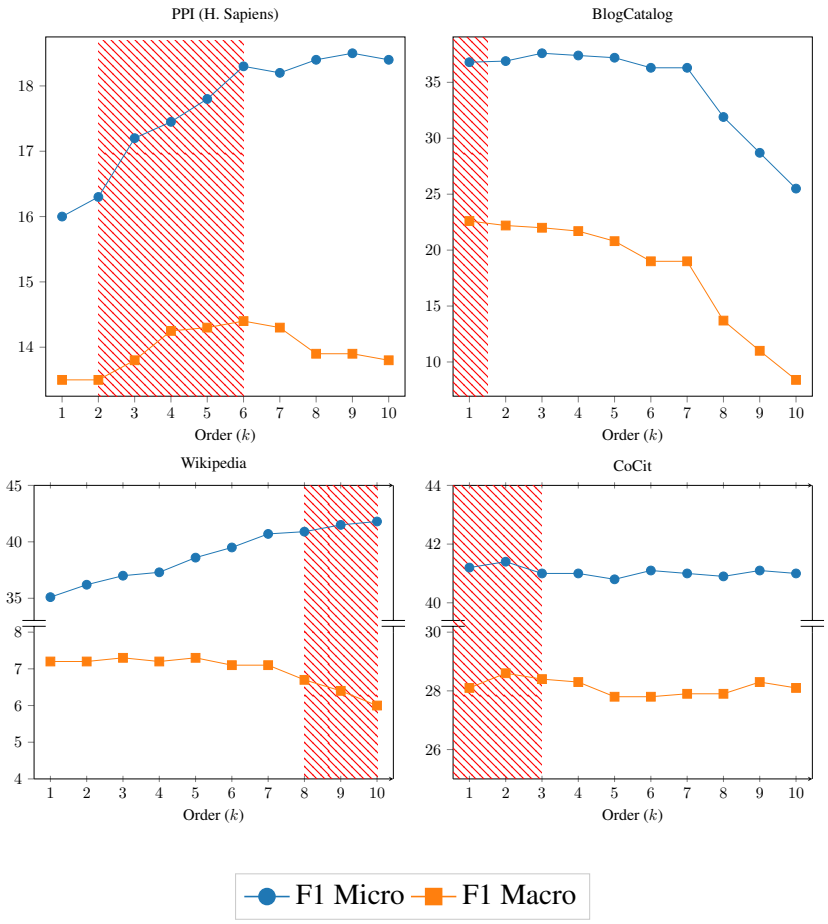

Figure 4.4: Micro and Macro F_1 scores for the four labeled graphs, when the "pure" $k-$order $\mathbf{S}^k$ is used for embedding, given as a function of k. Red shade denotes the corresponding k's where ASE assigned non-zero $\boldsymbol{\theta}_k$'s; see also Table 2.

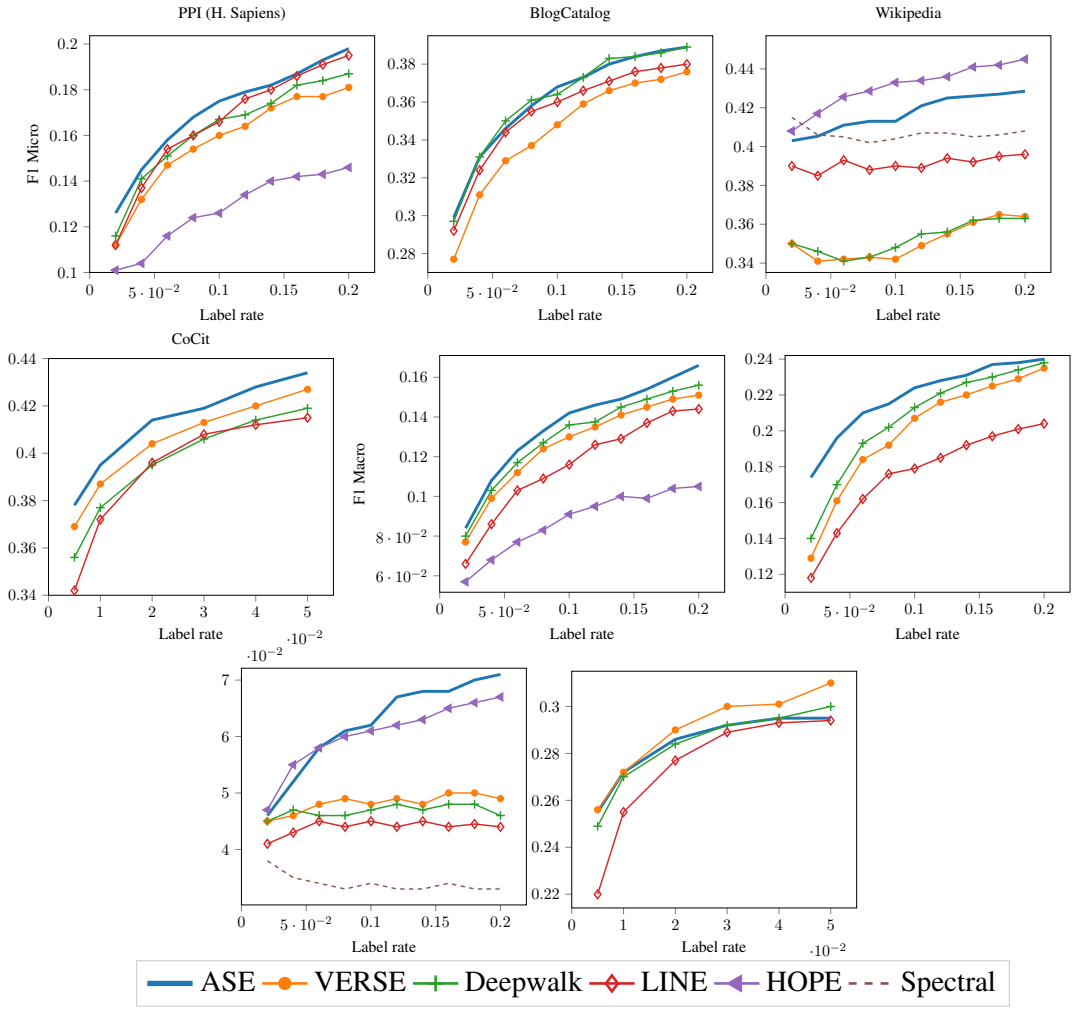

Figure 4.5: Micro (upper row) and Macro (lower row) F_1 scores that different embeddings + logistic regression yield on labeled graphs, as a function of the labeling rated (percentage of training data)

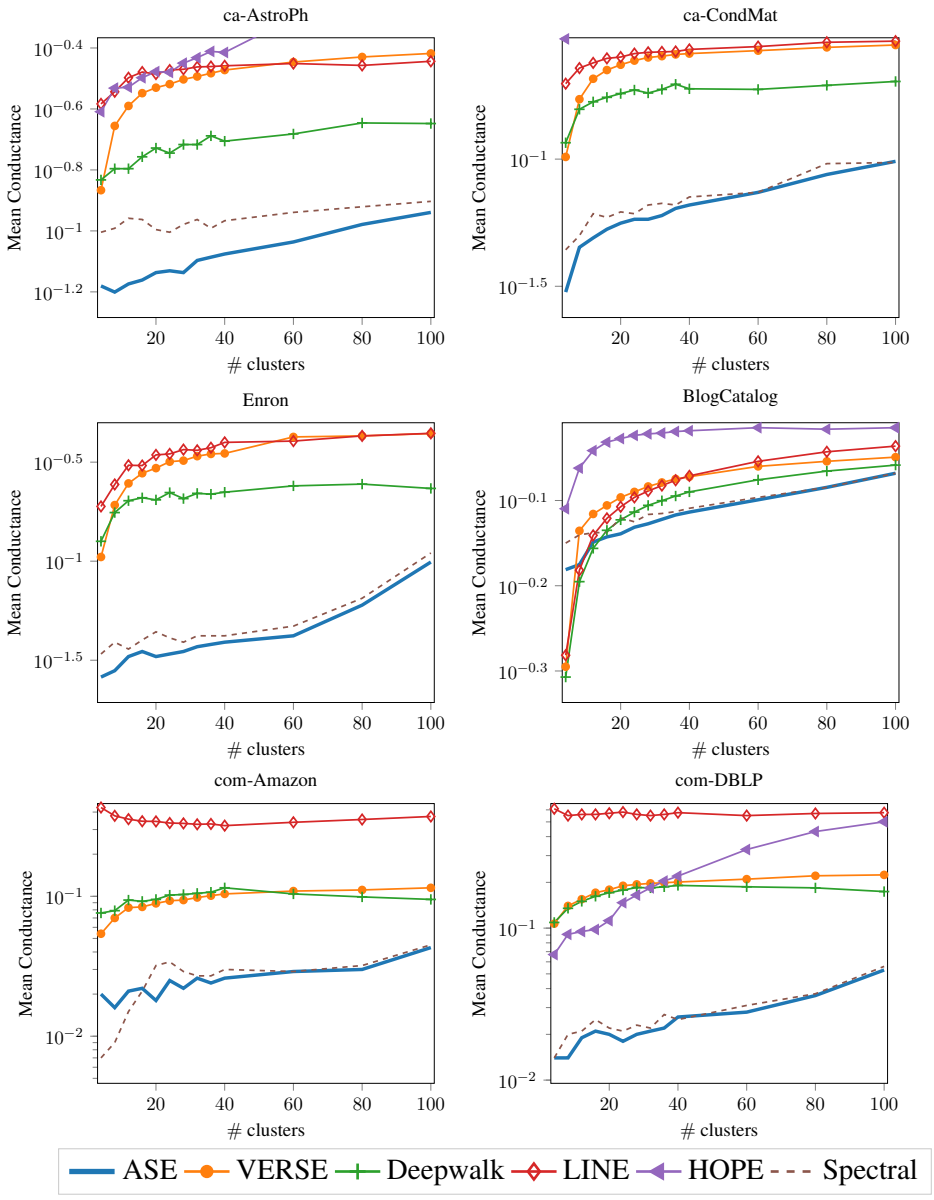

Figure 4.6: Average conductance of different embeddings used by kmeans for clustering, as a function of number of clusters.

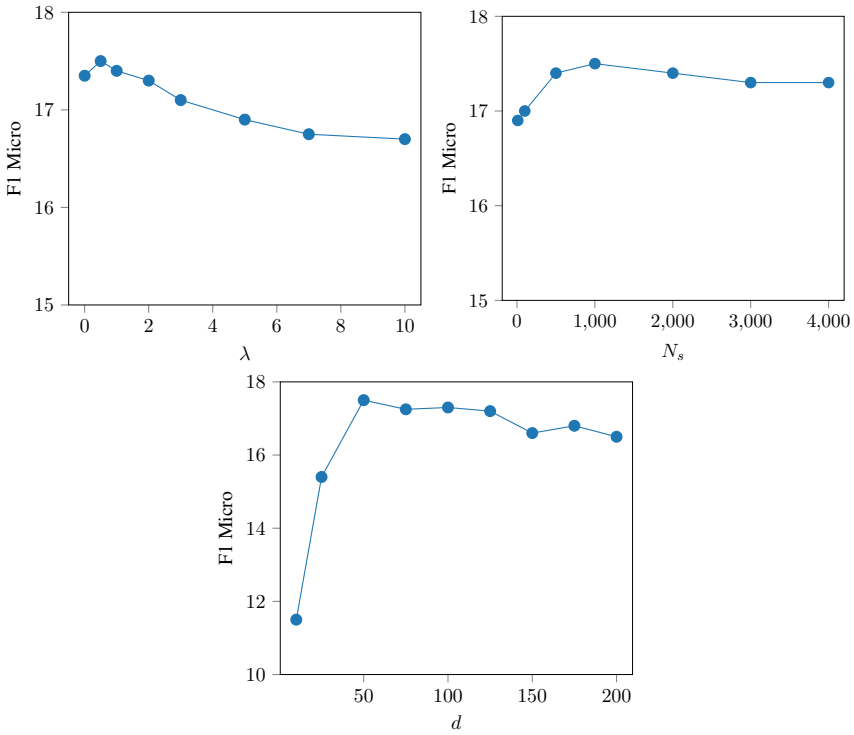

Figure 4.7: Sensitivity of ASE on PPI graphs wrt various parameters.

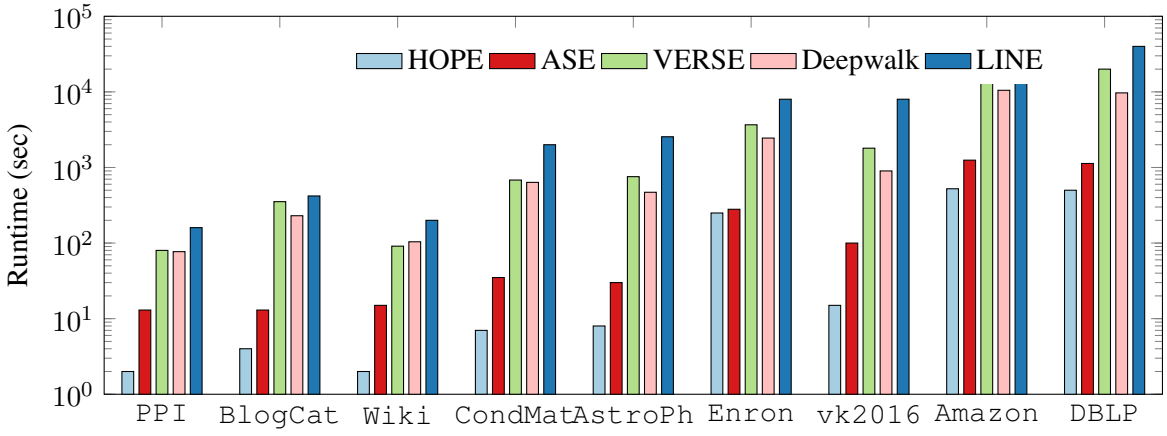

Figure 4.8: Runtime of various embedding methods across different graphs

Chapter 5

Summary and Future Directions

The thesis contributes to semi-supervised and unsupervised adaptive learning over large-scale graphs. In this final chapter, we provide a summary of the main results discussed in this thesis, and also point out a few possible directions for future research.

5.1 Thesis Summary

Aiming at active learning for semi-supervised classification, Chapter 2 introduces a sampling strategy that is based on querying the node whose label disclosure is expected to inflict the largest change on the GMRF, and in this sense it is the most informative on average. Connections are established to other sampling methods including uncertainty sampling, variance minimization, and sampling based on the $\Sigma-$optimality criterion. A simple yet effective heuristic is also introduced for increasing the exploration capabilities of the sampler, and reducing bias of the resultant classifier, by adjusting the confidence on the model label predictions. The novel sampling strategies are based on quantities that are readily available without the need for model retraining, rendering them computationally efficient and scalable to large graphs.

Further focusing on improving the classifier itself, Chapter 3 improves the performance of random-walk-based classifiers by *adapting the diffusion functions of every class* 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.

Finally, Chapter 4 deals with the unsupervised task of node embedding, and puts forth an adaptive node embedding framework that adjusts the embedding process to a given underlying graph, in a fully automated manner (no cross validation is needed). This is achieved by leveraging the notion of a tunable node similarity matrix that assigns weights on multihop paths. The design of multihop similarities ensures that the resultant embeddings also inherit interpretable spectral properties. The proposed model is thoroughly investigated, interpreted, and numerically evaluated using stochastic block models. Moreover, an unsupervised algorithm is developed for training the model parameters effieciently. Extensive node classification, link prediction, and clustering experiments are carried out on many real-world graphs from various domains, along with comparisons with state-of-the-art scalable and unsupervised node embedding alternatives. The proposed method enjoys superior performance in many cases, while also yielding interpretable information on the underlying graph structure.

5.2 Future Research

The results in this thesis open up interesting directions for a number of future research topics. Next, we outline a couple of them that we are currently pursuing.

5.2.1 Tracking and sampling time-varying label distributions on graphs

Recent years have seen increased research effort to deal with tracking dynamically evolving signals on graphs; see, for instance [60],[105],[87], [61],[66]. Meanwhile, AL in the presence of "concept drift" (i.e., data that are generated from a time-evolving distribution) has also been gaining momentum lately; see, e.g., [76],[132],[135]. In this context, we are motivated to develop *tracking* and *adaptive sampling* methods based on RWR for label distributions $\{y_i(t)\}_{i\in\mathcal{V}}$ that may evolve across time t, while the topology of the graph remains invariant. Towards this goal, given the per-class stationary distribution $\boldsymbol{\pi}_{t-1}(j)$ available from $t-1$, and label $y_{i(t)}$ obtained

Algorithm 10 Graph-based class-tracking with RWR

Input: $\mathbf{G}, d, \lambda$
Initialize: $\{\boldsymbol{\pi}_0(j) = \mathbf{0}\}_{j \in \mathcal{C}}$
for $t = 1 : T$ **do**
  Select sample set $\mathcal{L}_t$
  Obtain labels $y_{\mathcal{L}_t}$
  Update $\boldsymbol{\pi}_t(j)$ according to (5.2)
  Predict y_i $\forall i \in \mathcal{U}_t$ as in (2.7)
end for

at time t, the new stationary distributions can be obtained recursively as

$$\boldsymbol{\pi}_t(j) = d(1-\lambda)\boldsymbol{\pi}_{t-1}(j) + d\lambda \mathbf{g}_{i(t)} \mathbb{1}_{\{y_{i(t)}=j\}} \tag{5.1}$$

where $0 < \lambda < 1$ is the forgetting factor (similar in spirit to the LMS and RLS algorithms [67]) that tunes how quickly the influence of a given observation on the current model decays as time progresses. The larger λ is the faster the model "forgets" old observations. Note that in (5.1) we only sample one node at each time t with index $i(t)$. The labels of all other nodes at time t are effectively unknown, since even the ones we have observed in previous time slots might have changed in the meantime. This is a point where the dynamic scenario is clearly different from the static case. Furthermore, we will generalize (5.1) to accommodate multiple labels observed per slot, in which case

$$\boldsymbol{\pi}_t(j) = d(1-\lambda)\boldsymbol{\pi}_{t-1}(j) + d\lambda \frac{1}{|\mathcal{L}_t(j)|} \sum_{k \in \mathcal{L}_t(j)} \mathbf{g}_k \tag{5.2}$$

where $\mathcal{L}_t(j) := \{i \in \mathcal{L}_t : y_i(t) = j\}$, and $\mathcal{L}_t$ is the set of nodes that were labeled at slot t; see also the tabulated Algorithm 10 for a summary of the tracking approach. Our preliminary tests on the time evolving temperature data (temperature measurements were thresholded to "high" and "low" classes) from [60] indicate that Algorithm 10 with random sampling of $\mathcal{L}_t$ and for a wide range of values of λ can successfully track the seasonal and geographical changes of temperature. We will further employ Algorithm 10 as a stepping stone towards developing schemes for judiciously and adaptively selecting the time-varying sample set $\mathcal{L}_t$ such that the prediction accuracy per time slot is maximized.

5.2.2 Personalized Diffusions for Top-n Recommendation

Item-based methods are among the most popular approaches for top-n recommendation. Such methods work by building a model that captures the relations between the items, which is then used to recommend new items that are "close" to the ones each user has consumed in the past. Item-based models have been shown to achieve high top-n recommendation accuracy [111, 97], while being scalable and easy to interpret [35]. Nevertheless, item-model-based methods typically consider only direct inter-item relations that can impose fundamental limitations to their quality and make them brittle to the presence of sparsity—leading to poor itemspace coverage and significant decay in performance [9]. *Random-walk-based methods* on the other hand, are particularly well-suited for alleviating such problems. Having the innate ability to relate items that are not directly connected by propagating information along the edges of the underlying graph, random walk methods are more robust to the effects of sparsity and they can afford better coverage of the itemspace.

Having established the efficiency of recommendations based on diffusions over sparse item-graphs, we will proceed to *personalize* the diffusion pattern for each user. Following the adaptive-diffusion approach we introduced in Chapter 3, we postulate identifying user-specific diffusion parameters $\hat{\boldsymbol{\theta}}_u$ by solving per user $u \in \mathcal{U}$ for

$$\hat{\boldsymbol{\theta}}_u = \arg\min_{\boldsymbol{\theta} \in \mathcal{S}^K} \ell(\mathbf{h}_u, \mathbf{f}_u(\boldsymbol{\theta})) \tag{5.3}$$

where $\mathcal{U}$ is the set of users, and $\mathbf{f}_u(\boldsymbol{\theta}) = \mathbf{P}_u^{(K)}\boldsymbol{\theta}$ is (cf. 3.2) the per-user diffusion vector defined over the items, with $\mathbf{P}_u^{(K)}$ concatenating landing probabilities $\{\mathbf{p}_u^{(k)}\}_{k=1}^K$ of the random walk of user u on the item-model graph $\mathbf{W}_{\text{itm}}$. The starting (a.k.a. seed) distribution $\mathbf{p}_u^0$ is given by the past implicit feedback (viewing history) as the normalized indicator vector of past observed actions of user u, while the target vector $\mathbf{h}_u$ contains the indexes of hold-out validation items that $\mathbf{f}_u(\boldsymbol{\theta})$ aims to predict. The basic premise behind this work is that the latent mechanism for propagating historic preferences along the edges of the item graph can be *specific to each user*. Thus, finding a disciplined way to uncover such a mechanism can prove beneficial both in terms of recommendation accuracy as well as in providing useful information about the latent behavioral patterns that govern users' interaction within the system. Our preliminary results on real datasets indicate pronounced variability on the diffusion patterns $\hat{\boldsymbol{\theta}}_u$ exhibited by different users. Furthermore, personalizing can yield significant improvements on recommendation

accuracy, while also providing side-information on the behavior of individual users, and user-model interactions.

5.2.3 Node Hashing for Fast Queries in Very Large Graphs

As demonstrated in Chapter 4, node embedding methods can extract informative features over the node of a graph, that can markedly improve the performance of downstream learning tasks. Nodes v_i are typically embedded in Euclidean spaces, as $d-$dimensional real-valued vectors $\mathbf{e}_i \in \mathbb{R}^d$. A downside of this approach is the fact that it scales poorly to large graphs of several millions of nodes. Especially in terms of storage memory requirements, Euclidean embedding may even inflate the size of certain graphs. This becomes apparent upon considering that a typical $N-$node graph with average degree $\bar{\boldsymbol{\delta}}$, and $N \leq 2^{32} \approxeq 4.3 \times 10^9$, requires storage of $N\bar{\boldsymbol{\delta}}$ unsigned $32-$bit integers. For a typical average degree $\bar{\mathbf{d}} = 50$, this translates to 200 bytes-per-node on average. A Euclidean embedding of dimension d, requires the storage of d (typically) double-precision $64-$bit floating point numbers per node. Typically, the embedding dimension is at least $d = 100$, meaning that the embedding requires 800 bytes-per-node, a $4-$fold increase compared to the original graph format. This size inflation is even worse for sparser graphs or higher-dimensional embeddings, which can make the latter a less attractive alternative for graph-learning and mining practitioners. More importantly, running downstream machine learning methods that rely on $d-$dimensional features, such as $K-$means and logistic regression, may quickly become prohibitive (on shared memory platforms) if N grows to several millions, a typical range for many real-world graphs.

These considerations point to the need for developing and exploring more parsimonious embedding paradigms. Towards this direction, the notion of *node hashing* (or binary embedding) has gradually been gaining traction [93], [82]. As its name suggests, binary embedding aims to represent every node with d bits comprising the so-termed *hash code* for which $v_i \rightarrow \mathbf{b}_i \in \{0,1\}^d$. Apart from considerably decreasing storage requirements, representing each node with a few bits can also largely accelerate learning algorithms by replacing inner products and distances in the Euclidean space with highly efficient bit-wise operations (e.g., OR, AND, XOR) and distances (e.g., Hamming). The latter may allow for very fast $k-$nearest-neighbor queries, which can be further accelerated using recent advances in hash-code indexing [98]. Nevertheless, while the approaches in [93] and [82] yield usable binary embeddings, they do so by implicitly computing continuous-valued embeddings that are then thresholded or quantized. This leads

to poor scalability in the embedding method itself. To mitigate this shortcoming, we will draw upon the notion of spherical hashing developed for binary representation of images [57]. Our preliminary results indicate that spherical hashing on graphs is highly scalable, prompting us to pursue further research on this promising direction.

References

[1] [Online]. Available: https://en.wikipedia.org/wiki/List_of_social_networking_websites

[2] [Online]. Available: https://snap.stanford.edu/data/index.html

[3] [Online]. Available: https://tedlab.mit.edu/~dr/SVDLIBC/

[4] "https://archive.ics.uci.edu/ml/index.html."

[5] "https://linqs.soe.ucsc.edu/data."

[6] "https://www.csie.ntu.edu.tw/ cjlin/libsvmtools/datasets/binary.html."

[7] "http://www-personal.umich.edu/ mejn/netdata/."

[8] L. A. Adamic and E. Adar, "Friends and neighbors on the web," *Social Networks*, vol. 25, no. 3, pp. 211–230, 2003.

[9] G. Adomavicius and A. Tuzhilin, "Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions," *IEEE Transactions on Knowledge & Data Engineering*, no. 6, pp. 734–749, 2005.

[10] A. Ahmed, N. Shervashidze, S. Narayanamurthy, V. Josifovski, and A. J. Smola, "Distributed large-scale natural graph factorization," *Proc. of the World Wide Web Conf.*, pp. 37–48, 2013.

[11] A. Argyriou, M. Herbster, and M. Pontil, "Combining graph laplacians for semi–supervised learning," Vancouver, Canada, 2006, pp. 67–74.

[12] D. Arthur and S. Vassilvitskii, "k-means++: The advantages of careful seeding," *SIAM*, pp. 1027–1035, 2007.

[13] J. Atwood and D. Towsley, "Diffusion-convolutional neural networks," Barcelona, Spain, 2016, pp. 1993–2001.

[14] K. Avrachenkov, A. Mishenin, P. Gonçalves, and M. Sokol, "Generalized optimization framework for graph-based semi-supervised learning," Anaheim, CA, 2012, pp. 966–974.

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

[16] M. Balasubramanian and E. L. Schwartz, "The isomap algorithm and topological stability," *Science*, vol. 295, no. 5552, 2002.

[17] B. Barker, "Message passing interface (mpi)," 2015.

[18] 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, pp. 2399–2434, 2006.

[19] Y. Bengio, O. Delalleau, and N. L. Roux, "Label propagation and quadratic criterion," *Semi-Supervised Learning*, 2006.

[20] A. N. Berberidis, D. Nikolakopoulos and G. B. Giannakis, "Adadif: Adaptive diffusions for efficient semi-supervised learning over graphs," Seattle, WA, 2018, pp. 92–99.

[21] ——, "Random walks with restarts for graph-based classification: Teleportation tuning and sampling design," Calgary, Can., 2018.

[22] D. P. Bertsekas, *Nonlinear Programming*. Belmont: Athena Scientific, 1999.

[23] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko, "Translating embeddings for modeling multirelational data," Lake Tahoe, CA, 2016, pp. 2787–2795.

[24] 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.

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

[26] H. Cai, V. W. Zheng, and K. Chang, "A comprehensive survey of graph embedding: problems, techniques and applications," *IEEE Trans. on Knowledge and Data Engineering*, vol. 30, no. 9, pp. 1616–1637, 2018.

[27] S. Cao, W. Lu, and Q. Xu, "Deep neural networks for learning graph representations," Phoenix, AZ, 2016, pp. 1145–1152.

[28] W. Cao, S. Lu and Q. Xu, "Grarep: Learning graph representations with global structural information," 2015, pp. 891–900.

[29] N. Cesa-Bianchi, C. Gentile, F. Vitale, and G. Zappella, "Active learning on trees and graphs," *arXiv preprint arXiv:1301.5112*, 2013.

[30] O. Chapelle, B. Schölkopf, and A. Zien, *Semi-Supervised Learning*. MIT Press, 2006.

[31] 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. Sig. Proc.*, vol. 62, no. 11, pp. 2879–2893, 2014.

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

[33] L. Condat, "Fast projection onto the simplex and the ℓ_1 ball," *Mathematical Programming*, vol. 158, no. 1-2, pp. 575–585, 2016.

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

[35] C. Desrosiers and G. Karypis, "A comprehensive survey of neighborhood-based recommendation methods," in *Recommender Systems Handbook*, F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, Eds. Springer US, 2011, pp. 107–144. [Online]. Available: http://dx.doi.org/10.1007/978-0-387-85820-3_4

[36] C. Donnat, M. Zitnik, D. Hallac, and J. Leskovec, "Spectral graph wavelets for structural role similarity in networks," *arXiv preprint arXiv:1710.10321*, 2017.

[37] S. Fortunato and M. Barthelemy, "Resolution limit in community detection," *Proc. of the National Academy of Sciences*, vol. 104, no. 1, pp. 36–41, 2007.

[38] S. Fortunato, "Community detection in graphs," *Physics reports*, vol. 486, no. 3-5, pp. 75–174, 2010.

[39] K. Fujii and H. Kashima, "Budgeted stream-based active learning via adaptive submodular maximization," in *Proc. of Adv. in Neural Inf. Proc. Systems*, Barcelona, Spain, Dec. 2016.

[40] E. E. Gad, A. Gadde, A. S. Avestimehr, and A. Ortega, "Active learning on weighted graphs using adaptive and non-adaptive approaches," in *Proc. of Intl. Conf. on Acoustics, Speech and Signal Proc.*, Shanghai, China, March 2016.

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

[42] D. F. Gleich and M. W. Mahoney, "Using local spectral methods to robustify graph-based learning algorithms," Sidney Australia, 2015.

[43] G. H. Golub and C. Reinsch, "Singular value decomposition and least squares solutions," *Numer. Math.*, vol. 14, no. 5, pp. 403–420, 1970.

[44] ——, "Singular value decomposition and least squares solutions," *Numerische mathematik*, vol. 14, no. 5, pp. 403–420, 1970.

[45] 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, 2007.

[46] P. Goyal and E. Ferrara, "Graph embedding techniques, applications, and performance: A survey," *Knowledge-Based Systems*, vol. 151, pp. 78–94, 2018.

[47] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," San Francisco, CA, 2016, pp. 855–864.

[48] Q. Gu, C. Aggarwal, J. Liu, and J. Han, "Selective sampling on graphs for classification," in *Proc. of Intl. Conf. on Knowledge, Discovery and Data Mining*, Chicago, IL, Aug. 2013.

[49] Q. Gu and J. Han, "Towards active learning on graphs: An error bound minimization approach," in *Proc. of Intl. Conf. on Data Mining*, Brussels, Belgium, Dec. 2012.

[50] P. Gupta, A. Goel, J. Lin, A. Sharma, D. Wang, and R. Zadeh, "Wtf: The who to follow service at twitter," in *Proceedings of the 22nd international conference on World Wide Web*. ACM, 2013, pp. 505–514.

[51] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," Long Beach, CA, 2017, pp. 1024–1034.

[52] Y. Han and Y. Shen, "Partially supervised graph embedding for positive unlabeled feature selection," New York, NY, 2016, pp. 1548–1554.

[53] A. Hauser and P. Bühlmann, "Two optimal strategies for active learning of causal models from interventions," in *Proc. of Europ. Work. on Prob. Graph. Models*, Granada, Spain, Sept. 2012.

[54] K. He, P. Shi, J. E. Hopcroft, and D. Bindel, "Local spectral diffusion for robust community detection," San Francisco CA, 2016.

[55] K. He, Y. Sun, D. Bindel, J. E. Hopcroft, and Y. Li, "Detecting overlapping communities from local spectral subspaces," Atlantic City NJ, 2015, pp. 769–774.

[56] X. He and P. Niyogi, "Locality preserving projections," *Advances in Neural Inf. Proc. Systems*, vol. 486, no. 3-5, pp. 153–160, 2003.

[57] J.-P. Heo, Y. Lee, J. He, S.-F. Chang, and S.-E. Yoon, "Spherical hashing," in *Proc. of IEEE Conference on Computer Vision and Pattern Recognition*, 2012, pp. 2957–2964.

[58] V. Hernandez, J. E. Roman, and V. Vidal, "Slepc: A scalable and flexible toolkit for the solution of eigenvalue problems," in *ACM Trans. Math. Software*, vol. 31, no. 3, 2005, pp. 351–362.

[59] T. Hofmann and J. M. Buhmann, "Multidimensional scaling and data clustering," 1994, pp. 459–466.

[60] V. N. Ioannidis, D. Romero, and G. B. Giannakis, "Kernel-based reconstruction of space-time functions via extended graphs," in *in Proc. of Asilomar Conf. on Signals, Systems and Computers*, Monterey, CA, Nov 2016, pp. 1829–1833.

[61] E. Isufi, A. Loukas, A. Simonetto, and G. Leus, "Autoregressive moving average graph filtering," *IEEE Trans. Sig. Proc.*, vol. 65, no. 2, pp. 274–288, 2017.

[62] M. Ji and J. Han, "A variance minimization criterion to active learning on graphs." in *Intl. Conf. on Artif. Intel. and Stat.*, La Palma, Canary Islands, April 2012.

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

[64] T. Joachims, "Transductive learning via spectral graph partitioning," Washington DC, 2003, pp. 290–297.

[65] K.-S. Jun and R. Nowak, "Graph-based active learning: A new look at expected error minimization," *arXiv preprint arXiv:1609.00845*, 2016.

[66] V. Kalofolias, A. Loukas, D. Thanou, and P. Frossard, "Learning time varying graphs," in *in Proc. of Intl. Conf. on Acoustics, Speech and Signal Proc.*, New Orleans, LA, March 2017.

[67] S. M. Kay, *Fundamentals of Statistical Signal Processing, Vol. I: Estimation Theory*. Englewood Cliffs: Prentice Hall PTR, 1993.

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

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

[70] K. Kloster and D. F. Gleich, "Heat kernel based community detection," New York, NY, 2014, pp. 1386–1395.

[71] 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.

[72] E. D. Kolaczyk, *Statistical Analysis of Network Data*. Springer, 2009.

[73] R. I. Kondor and J. Lafferty, "Diffusion kernels on graphs and other discrete input spaces," Syndey, Australia, 2002, pp. 315–322.

[74] Y. Koren, R. M. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *IEEE Computer*, no. 8, pp. 30–37, 2009.

[75] A. Krause, A. Singh, and C. Guestrin, "Near-optimal sensor placements in gaussian processes: Theory, efficient algorithms and empirical studies," *Journal of Machine Learning Research*, vol. 9, no. Feb, pp. 235–284, 2008.

[76] B. Krawczyk and M. Wozniak, "Online query by committee for active learning from drifting data streams," in *in Proc. of Intl. Conf. on Neural Networks*, Anchorage, AL, May 2017, pp. 2120–2127.

[77] B. Kveton, M. Valko, A. Rahimi, and L. Huang, "Semi-supervised learning with max-margin graph cuts," Sardinia, Italy, 2010, pp. 421–428.

[78] A. Lancichinetti, S. Fortunato, and F. Radicchi, "Benchmark graphs for testing community detection algorithms," *Physical review E*, vol. 78, no. 4, pp. 046–110, 2008.

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

[80] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney, "Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters," *Internet Mathematics*, vol. 6, no. 1, pp. 29–123, 2009.

[81] D. A. Levin and Y. Peres, *Markov Chains and Mixing Times*. Amer. Math. Soc., 2017.

[82] D. Lian, K. Zheng, V. W. Zheng, Y. Ge, L. Cao, I. W. Tsang, and X. Xie, "High-order proximity preserving information network hashing," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2018, pp. 1744–1753.

[83] D. Liben-Nowell and J. Kleinberg, "The link-prediction problem for social networks," *Journal of the American society for information science and technology*, vol. 58, no. 7, pp. 1019–1031, 2007.

[84] F. Lin and W. W. Cohen, "Semi-supervised classification of network data using very few labels," Odense, Denmark, 2010, pp. 192–199.

[85] 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.

[86] J. Long, J. Yin, W. Zhao, and E. Zhu, "Graph-based active learning based on label propagation," in *Intl. Conf. on Modeling Decisions for Artif. Intel.*, Catalonia, Spain, Oct. 2008.

[87] A. Loukas and D. Foucard, "Frequency analysis of time-varying graph signals," in *in Proc. of Global Conf. on Signal and Inf. Proc.*, Washington, DC, Dec 2016, pp. 346–350.

[88] E. I. M. Contino and G. Leus, "Distributed edge-variant graph filters," Curacao, Dutch Antilles, 2017, pp. 1–5.

[89] Y. Ma, R. Garnett, and J. Schneider, "$\sigma-$optimality for active learning on Gaussian random fields," in *Proc. of Adv. in Neural Inf. Proc. Systems*, Lake Tahoe, Dec. 2013.

[90] C. D. Manning, P. Raghavan, and H. Schutze, *Introduction to Information Retrieval*. Cambridge, MA: Cambridge University Press, 2208.

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

[92] A. Milanese, J. Sun, and T. Nishikawa, "Approximating spectral impact of structural perturbations in large networks," *Physical Review E*, vol. 81, no. 4, pp. 046–112, 2010.

[93] V. Misra and S. Bhatia, "Bernoulli embeddings for graphs," *arXiv preprint arXiv:1803.09211*, 2018.

[94] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Intl Conf. on Learn. Repres. (ICLR)*, Toulon, France, April 2017.

[95] A. N. Nikolakopoulos and J. D. Garofalakis, "Ncdawarerank: A novel ranking method that exploits the decomposable structure of the web," Rome, Italy, 2013, pp. 143–152.

[96] A. N. Nikolakopoulos, A. Korba, and J. D. Garofalakis, "Random surfing on multipartite graphs," Washington DC, 2016, pp. 736–745.

[97] X. Ning and G. Karypis, "Slim: Sparse linear methods for top-n recommender systems," in *2011 11th IEEE International Conference on Data Mining*. IEEE, 2011, pp. 497–506.

[98] M. Norouzi, A. Punjani, and D. J. Fleet, "Fast search in hamming space with multi-index hashing," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE, 2012, pp. 3108–3115.

[99] M. Ou, P. Cui, J. Pei, Z. Zhang, and W. Zhu, "Asymmetric transitivity preserving graph embedding," San Fransisco, CA, 2016, pp. 1105–1114.

[100] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web." Stanford InfoLab, Tech. Rep., 1999.

[101] B. Perozzi, R. A. Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," New York, NY, 2014, pp. 701–710.

[102] J. J. Pfeiffer III, J. Neville, and P. N. Bennett, "Active sampling of networks," in *Proc. of Intl. Work. on Mining and Learning with Graphs*, Edinburgh, Scotland, July 2012.

[103] 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.

[104] J. Qiu, Y. Dong, H. Ma, J. Li, K. Wang, and J. Tang, "Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec," Los Angeles, CA, 2018, pp. 459–467.

[105] K. Qiu, X. Mao, X. Shen, X. Wang, T. Li, and Y. Gu, "Time-varying graph signal reconstruction," *IEEE J. Sel. Topics Sig. Proc.*, July 2017.

[106] L. F. R. Ribeiro, P. H. P. Saverese, and D. R. Figueiredo, "struc2vec: Learning node representations from structural identity," Halifax, Canada, 2017, pp. 385–394.

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

[108] S. T. Roweis and L. K. Saul, "Nonlinear dimensionality reduction by locally linear embedding," *Science*, vol. 290, no. 5500, pp. 2323–2326, 2000.

[109] R. A.-R. S. Abu-El-Haija, B. Perozzi and A. Alemi, "Watch your step: Learning graph embeddings through attention," 2017.

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

[111] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Item-based collaborative filtering recommendation algorithms," in *Proceedings of the 10th international conference on World Wide Web*. ACM, 2001, pp. 285–295.

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

[113] B. Settles, "Active learning," *Synthesis Lectures on Artif. Intel. and Machine Learning*, vol. 6, no. 1, pp. 1–114, 2012.

[114] B. Shaw and T. Jebara, "Structure preserving embedding," Montreal, Canada, 2009, pp. 937–944.

[115] M. Simonovsky and N. Komodakis, "Dynamic edge-conditioned filters in convolutional neural networks on graphs," in *Proc of. IEEE Conf. on Computer Vision and Pattern Recognition*, 2017.

[116] A. Smola and B. Scholkopf, "A tutorial on support vector regression," *Statistics and Computing*, vol. 14, no. 3, pp. 199–222, 2004.

[117] H. Su, Z. Yin, T. Kanade, and S. Huh, "Active sample selection and correction propagation on a gradually-augmented graph," in *Proc. of Conf. on Computer Vision and Pattern Recognition*, Boston, MA, June 2015, pp. 1975–1983.

[118] P. P. Talukdar and K. Crammer, "New regularized algorithms for transductive learning," 2009, pp. 442–457.

[119] J. Tang, M. Qu, and Q. Mei, "Pte: Predictive text embedding through large-scale heterogeneous text networks," Sidney, Australia, 2015, pp. 1165–1174.

[120] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line: Large-scale information network embedding," Florence, Italy, 2015, pp. 1067–1077.

[121] F. Tian, B. Gao, Q. Cui, E. Chen, and T. Liu, "Learning deep representations for graph clustering," Quebec, Canada, 2014, pp. 1293–1299.

[122] A. Tsitsulin, D. Mottin, P. Karras, and E. Muller, "Verse: Versatile graph embeddings from similarity measures," Florence, Italy, 2018.

[123] J. Ugander and L. Backstrom, "Balanced label propagation for partitioning massive graphs," Rome, Italy, 2013, pp. 507–516.

[124] U. Von Luxburg, "A tutorial on spectral clustering," *Statistics and Computing*, vol. 17, no. 4, pp. 395–416, 2007.

[125] D. Wang, P. Cui, and W. Zhu, "Structural deep network embedding," San Fransisco, CA, 2016, pp. 1225–1234.

[126] Q. Wang, Z. Mao, B. Wang, and L. Guo, "Knowledge graph embedding: A survey of approaches and applications," *IEEE Transactions on Knowledge & Data Engineering*, no. 12, pp. 2724–2743, 2017.

[127] X.-M. Wu, Z. Li, A. M. So, J. Wright, and S.-F. Chang, "Learning with partially absorbing random walks," Lake Tahoe, CA, 2012, pp. 3077–3085.

[128] Z. G. X. Zhu, "Learning from labeled and unlabeled data with label propagation," *CMU CALD tech report CMU-CALD-02-107*, 2002.

[129] R. Xie, Z. Liu, and M. Sun, "Representation learning of knowledge graphs with hierarchical types," New York, NY, 2016, pp. 2965–2971.

[130] S. Yan, D. Xu, B. Zhang, H.-J. Zhang, Q. Yang, and S. Lin, "Graph embedding and extensions: A general framework for dimensionality reduction," *IEEE Trans. on Pattern Analysis and Machine Intel.*, vol. 29, no. 1, pp. 40–51, 2007.

[131] C. Yang, Z. Liu, D. Zhao, M. Sun, and E. Y. Chang, "Network representation learning with rich text information," 2015, pp. 2111–2117.

[132] L. Yang, "Active learning with a drifting distribution," in *in Proc. of Adv. in Neural Inform. Processing Systems*, Granada, Spain, Dec. 2011, pp. 2079–2087.

[133] W. Yang, Z. Cohen and R. Salakhutdinov, "Revisiting semi-supervised learning with graph embeddings," New York City, NY, Feb. 2016.

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

[135] D. Zambon, C. Alippi, and L. Livi, "Concept drift and anomaly detection in graph streams," *arXiv preprint arXiv:1706.06941*, 2017.

[136] Z. Zhang, P. Cui, X. Wang, J. Pei, X. Yao, and W. Zhu, "Arbitrary-order proximity preserved network embedding," London, UK, 2018, pp. 2778–2786.

[137] Y. Zhao, E. Levina, and j. Zhu, "Consistency of community detection in networks under degree-corrected stochastic block models," *The Annals of Statistics*, vol. 40, no. 4, pp. 2266–2292, 2012.

[138] Y. Zhao, Z. Liu, and M. Sun, "Representation learning for measuring entity relatedness with rich information," Buenos aires, Argentina, 2015, pp. 1412–1418.

[139] J. Zhou and S. Sun, "Active learning of Gaussian processes with manifold-preserving graph reduction," *J. of Neural Computing and Applications*, vol. 25, no. 7-8, pp. 1615–1625, 2014.

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

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

Appendix A

Proofs for Chapter 2

A.1 Proof of relation (2.4)

Since $\mathbf{C}^{-1} = \mathbf{L}$ and upon partitioning the two matrices according to labeled and unlabeled nodes, we have

$$\begin{bmatrix} \mathbf{L}_{\mathcal{U}\mathcal{U}} & \mathbf{L}_{\mathcal{U}\mathcal{L}} \\ \mathbf{L}_{\mathcal{L}\mathcal{U}} & \mathbf{L}_{\mathcal{L}\mathcal{L}} \end{bmatrix} \begin{bmatrix} \mathbf{C}_{\mathcal{U}\mathcal{U}} & \mathbf{C}_{\mathcal{U}\mathcal{L}} \\ \mathbf{C}_{\mathcal{L}\mathcal{U}} & \mathbf{C}_{\mathcal{L}\mathcal{L}} \end{bmatrix} = \begin{bmatrix} \mathbf{I}_{|\mathcal{U}|} & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_{|\mathcal{L}|} \end{bmatrix} \tag{A.1}$$

which gives rise to four matrix equations. Specifically, the equation that corresponds to the upper right part of (A.1) is

$$\mathbf{L}_{\mathcal{U}\mathcal{U}}\mathbf{C}_{\mathcal{U}\mathcal{L}} + \mathbf{L}_{\mathcal{U}\mathcal{L}}\mathbf{C}_{\mathcal{L}\mathcal{L}} = \mathbf{0}. \tag{A.2}$$

Multiplying (A.2) from the left by $\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}$ and from the right by $\mathbf{C}_{\mathcal{L}\mathcal{L}}^{-1}$ yields $\mathbf{C}_{\mathcal{U}\mathcal{L}}\mathbf{C}_{\mathcal{L}\mathcal{L}}^{-1} = -\mathbf{L}_{\mathcal{U}\mathcal{U}}^{-1}\mathbf{L}_{\mathcal{U}\mathcal{L}}$, which verifies (2.4).

A.2 Proof of relation (2.26)

Let $\mathbf{x}_1 \sim \mathcal{N}(\mathbf{m}_1, \mathbf{C})$ and $\mathbf{x}_2 \sim \mathcal{N}(\mathbf{m}_2, \mathbf{C})$, and assume that $\mathbf{x}_1$ and $\mathbf{x}_2$ are uncorrelated. Then,

$$\begin{aligned} \mathrm{MSD}(\mathbf{x}_1, \mathbf{x}_2) &:= \mathbb{E}\left[\|\mathbf{x}_1 - \mathbf{x}_2\|_2^2\right] \\ &= \mathbb{E}\left[\|\mathbf{x}_1\|_2^2 + \|\mathbf{x}_2\|_2^2 - 2\mathbf{x}_1^T\mathbf{x}_2\right] \end{aligned} \tag{A.3}$$

where

$$\begin{aligned}\mathbb{E}\left[\|\mathbf{x}_1\|_2^2\right] &= \mathbb{E}\left[\|(\mathbf{x}_1-\mathbf{m}_1)+\mathbf{m}_1\|_2^2\right] \\ &= \mathbb{E}\left[\|\mathbf{x}_1-\mathbf{m}_1\|_2^2\right] + 2\mathbb{E}\left[(\mathbf{x}_1-\mathbf{m}_1)^T\mathbf{m}_1\right] + \|\mathbf{m}_1\|_2^2 \\ &= \mathrm{tr}\left(\mathbf{C}\right) + \|\mathbf{m}_1\|_2^2 \end{aligned} \tag{A.4}$$

and similarly for $\mathbb{E}\left[\|\mathbf{x}_2\|_2^2\right]$. Finally, note that

$$\begin{aligned}\mathbb{E}\left[(\mathbf{x}_1-\mathbf{m}_1)^T(\mathbf{x}_2-\mathbf{m}_2)\right] &= \mathbb{E}\big[\mathbf{x}_1^T\mathbf{x}_2 - \mathbf{x}_1^T\mathbf{m}_2 - \mathbf{m}_1^T\mathbf{x}_2 + \mathbf{m}_1^T\mathbf{m}_2\big] \\ &= \mathbb{E}\left[\mathbf{x}_1^T\mathbf{x}_2\right] - \mathbf{m}_1^T\mathbf{m}_2 \end{aligned} \tag{A.5}$$

and since $\mathbf{x}_1$ and $\mathbf{x_2}$ are uncorrelated it follows that (A.5) equals to 0; hence,

$$\mathbb{E}\left[\mathbf{x}_1^T\mathbf{x}_2\right] = \mathbf{m}_1^T\mathbf{m}_2. \tag{A.6}$$

Substituting (B.2) and (A.6) into (A.3) yields

$$\begin{aligned}\mathrm{MSD}(\mathbf{x}_1,\mathbf{x}_2) &= 2\mathrm{tr}(\mathbf{C}) + \|\mathbf{m}_1\|_2^2 + \|\mathbf{m}_2\|_2^2 - 2\mathbf{m}_1^T\mathbf{m}_2 \\ &= 2\mathrm{tr}(\mathbf{C}) + \|\mathbf{m}_1-\mathbf{m}_2\|_2^2\end{aligned}$$

which implies that the MSD between two Gaussian fields with the same covariance matrix is proportional to the Euclidean norm of the difference of their means.

Appendix B

Proofs for Chapter 3

B.1 Proof of Proposition 1

For $\lambda \to \infty$, the effect of $\ell(\cdot)$ in (3.10) vanishes, and the optimization problem becomes equivalent to solving

$$\min_{\boldsymbol{\theta} \in \mathcal{S}^K} \boldsymbol{\theta}^\mathsf{T} \mathbf{A} \boldsymbol{\theta} \tag{B.1}$$

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 $\boldsymbol{\pi} = \mathbf{d}/(2|\mathcal{E}|)$ [81]. The landing probability vector for class c is thus

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

where $\gamma_n := \mathbf{v}_n^\mathsf{T}\mathbf{v}_c$, and the approximation in (B.2) holds because $\lambda_2^K \gg \lambda_n^K$, for $n \in [3, N]$, and K large enough but finite. Using (B.2), A_{ij} can be rewritten as

$$\begin{aligned} A_{ij} =& \left[\frac{1}{2|\mathcal{E}|}\mathbf{d}^\mathsf{T} + \lambda_2^i \mathbf{u}_2^\mathsf{T}\gamma_2\right]\mathbf{D}^{-1}\mathbf{L}\mathbf{D}^{-1}\left[\frac{1}{2|\mathcal{E}|}\mathbf{d} + \lambda_2^j \mathbf{u}_2\gamma_2\right] \\ =& \left[\frac{1}{2|\mathcal{E}|}\mathbf{1}^\mathsf{T} + \lambda_2^i \mathbf{u}_2^\mathsf{T}\mathbf{D}^{-1}\gamma_2\right]\mathbf{L}\left[\frac{1}{2|\mathcal{E}|}\mathbf{1} + \lambda_2^j \mathbf{D}^{-1}\mathbf{u}_2\gamma_2\right] \\ =& \frac{1}{4|\mathcal{E}|^2}\mathbf{1}^\mathsf{T}\mathbf{L}\mathbf{1} + \frac{\lambda_2^i\gamma_2}{2|\mathcal{E}|}\mathbf{u}_2^\mathsf{T}\mathbf{D}^{-1}\mathbf{L}\mathbf{1} + \frac{\lambda_2^j\gamma_2}{2|\mathcal{E}|}\mathbf{1}^\mathsf{T}\mathbf{L}\mathbf{D}^{-1}\mathbf{u}_2 \\ & + \gamma_2^2\lambda_2^{i+j}\mathbf{u}_2^\mathsf{T}\mathbf{D}^{-1}\mathbf{L}\mathbf{D}^{-1}\mathbf{u}_2 \\ =& C\lambda_2^{i+j} \end{aligned} \tag{B.3}$$

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 (B.3), 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 (B.1) reduces to

$$\min_{\boldsymbol{\theta}\in\mathcal{S}^K}\left(\boldsymbol{\lambda}_2^\mathsf{T}\boldsymbol{\theta}\right)^2. \tag{B.4}$$

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 (B.4) are identical to those of

$$\min_{\boldsymbol{\theta}\in\mathcal{S}^K}\boldsymbol{\lambda}_2^\mathsf{T}\boldsymbol{\theta}. \tag{B.5}$$

Therefore, (B.4) admits minimizer(s) identical to (B.5). Finally, we will show that the minimizer of (B.5) 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

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

$$\Leftrightarrow \sum_{k=1}^{K} \theta_k \left(\lambda_2^{k-K} - 1 \right) \geq 0$$

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.2 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

$$\begin{aligned} \|\mathbf{y} - \check{\mathbf{y}}\| &= \|\mathbf{X}_+\boldsymbol{\theta} - \mathbf{X}_-\boldsymbol{\theta} - \check{\mathbf{X}}_+\boldsymbol{\theta} + \check{\mathbf{X}}_-\boldsymbol{\theta}\| \leq \\ &\leq \|\theta_K \mathbf{p}_+^{(K)} - \theta_K \mathbf{p}_-^{(K)}\| + \|\theta_K \mathbf{p}_+^{(K+1)} - \theta_K \mathbf{p}_-^{(K+1)}\| \\ &\leq \|\mathbf{H}^K \mathbf{p}_+ - \mathbf{H}^K \mathbf{p}_-\| + \|\mathbf{H}^{K+1} \mathbf{p}_+ - \mathbf{H}^{K+1} \mathbf{p}_-\| \end{aligned} \tag{B.6}$$

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 (B.6) 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

$$\mathbf{H} = \mathbf{D}^{\frac{1}{2}} (\mathbf{I} - \tilde{\mathbf{L}}) \mathbf{D}^{-\frac{1}{2}}. \tag{B.7}$$

For the first term of the RHS of (B.6), we have

$$\begin{aligned} \|\mathbf{H}^K \mathbf{p}_+ - \mathbf{H}^K \mathbf{p}_-\| &\leq \|\mathbf{H}^K \mathbf{p}_+ - \boldsymbol{\pi}\| + \|\mathbf{H}^K \mathbf{p}_- - \boldsymbol{\pi}\| \\ &= \|\mathbf{D}^{\frac{1}{2}} (\mathbf{I} - \tilde{\mathbf{L}})^K \mathbf{D}^{-\frac{1}{2}} \mathbf{p}_+ - \frac{\mathbf{D1}}{2|\mathcal{E}|}\| \\ &+ \|\mathbf{D}^{\frac{1}{2}} (\mathbf{I} - \tilde{\mathbf{L}})^K \mathbf{D}^{-\frac{1}{2}} \mathbf{p}_- - \frac{\mathbf{D1}}{2|\mathcal{E}|}\|. \end{aligned} \tag{B.8}$$

Since $\mathbf{q}_1 = \frac{\mathbf{D}^{\frac{1}{2}}\mathbf{1}}{\sqrt{2|\mathcal{E}|}}$ [81], we have for $c \in \{+, -\}$ that

$$\begin{aligned} \mathbf{D}^{\frac{1}{2}} \mathbf{q}_1 \langle \mathbf{q}_1, \mathbf{D}^{-\frac{1}{2}} \mathbf{p}_c \rangle &= \mathbf{D}^{\frac{1}{2}} \frac{\mathbf{D}^{\frac{1}{2}}\mathbf{1}}{\sqrt{2|\mathcal{E}|}} \left\langle \frac{\mathbf{D}^{\frac{1}{2}}\mathbf{1}}{\sqrt{2|\mathcal{E}|}}, \mathbf{D}^{-\frac{1}{2}} \mathbf{p}_c \right\rangle \\ &= \frac{\mathbf{D1}}{\sqrt{2|\mathcal{E}|}} \frac{\langle \mathbf{1}, \mathbf{p}_c \rangle}{\sqrt{2|\mathcal{E}|}} = \frac{\mathbf{D1}}{2|\mathcal{E}|} \ . \end{aligned} \tag{B.9}$$

Upon defining $\mathbf{M} := (\mathbf{I} - \tilde{\mathbf{L}})^K - \mathbf{q}_1\mathbf{q}_1^\mathsf{T}$, and taking into account (B.9), inequality (B.8) can be written as

$$\|\mathbf{H}^K\mathbf{p}_+ - \mathbf{H}^K\mathbf{p}_-\| \leq \|\mathbf{D}^{\frac{1}{2}}\|\|\mathbf{M}\| \left(\|\mathbf{D}^{-\frac{1}{2}}\mathbf{p}_+\| + \|\mathbf{D}^{-\frac{1}{2}}\mathbf{p}_-\|\right). \tag{B.10}$$

The factors in (B.10) can be bounded as

$$\begin{aligned}
\|\mathbf{D}^{-\frac{1}{2}}\mathbf{p}_+\| &= \sqrt{\sum_{i\in\mathcal{L}_+}\left(\frac{1}{|\mathcal{L}_+|}d_i^{-\frac{1}{2}}\right)^2} \\
&= \sqrt{\sum_{i\in\mathcal{L}_+}\frac{1}{|\mathcal{L}_+|^2}d_i^{-1}} \leq \frac{1}{\sqrt{d_{\min_+}|\mathcal{L}_+|}}, && \text{(B.11)}\\
\|\mathbf{D}^{-\frac{1}{2}}\mathbf{p}_-\| &= \sqrt{\sum_{i\in\mathcal{L}_-}\frac{1}{|\mathcal{L}_-|^2}d_i^{-1}} \leq \frac{1}{\sqrt{d_{\min_-}|\mathcal{L}_-|}}, && \text{(B.12)}\\
\|\mathbf{M}\| &= \sup_{\mathbf{v}}\frac{\langle\mathbf{M}\mathbf{v},\mathbf{v}\rangle}{\mathbf{M}\mathbf{v}} = \max_{i\neq 1}|1-\mu_i|^K, && \text{(B.13)}\\
\|\mathbf{D}^{\frac{1}{2}}\| &= \sqrt{d_{\max}} && \text{(B.14)}
\end{aligned}$$

where (B.13) follows from the properties of the normalized Laplacian. Therefore, (B.10) becomes

$$\begin{aligned}
\|\mathbf{H}^K\mathbf{p}_+ - \mathbf{H}^K\mathbf{p}_-\| \leq &\left(\frac{1}{\sqrt{d_{\min_-}|\mathcal{L}_-|}} + \frac{1}{\sqrt{d_{\min_+}|\mathcal{L}_+|}}\right) \\
&\cdot \max_{i\neq 1}|1-\mu_i|^K \cdot \sqrt{d_{\max}}.
\end{aligned} \tag{B.15}$$

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

$$(1-\mu')^K \leq e^{-K\mu'} \tag{B.16}$$

we obtain

$$\|\mathbf{H}^K\mathbf{p}_+ - \mathbf{H}^K\mathbf{p}_-\| \leq \left(\sqrt{\frac{d_{\max}}{d_{\min_-}|\mathcal{L}_-|}} + \sqrt{\frac{d_{\max}}{d_{\min_+}|\mathcal{L}_+|}}\right) e^{-K\mu'}. \tag{B.17}$$

Likewise, we can bound the second term in (B.6) as

$$\|\mathbf{H}^{K+1}\mathbf{p}_+ - \mathbf{H}^{K+1}\mathbf{p}_-\| \leq \left(\sqrt{\frac{d_{\max}}{d_{\min_-}|\mathcal{L}_-|}} + \sqrt{\frac{d_{\max}}{d_{\min_+}|\mathcal{L}_+|}}\right) e^{-(K+1)\mu'}. \tag{B.18}$$

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

$$e^{-K\mu'} + e^{-(K+1)\mu'} < 2e^{-K\mu'}. \tag{B.19}$$

Upon substituting (B.17) and (B.18) into (B.6), and also using (B.19), we arrive at

$$\|\mathbf{y} - \check{\mathbf{y}}\| \leq 2\left(\sqrt{\frac{d_{\max}}{d_{\min_-}|\mathcal{L}_-|}} + \sqrt{\frac{d_{\max}}{d_{\min_+}|\mathcal{L}_+|}}\right) e^{-K\mu'}. \tag{B.20}$$

To determine an upper bound on the γ-distinguishability threshold, it suffices to find the smallest integer K for which (B.20) becomes less than γ; that is,

$$2\left(\sqrt{\frac{d_{\max}}{d_{\min_-}|\mathcal{L}_-|}} + \sqrt{\frac{d_{\max}}{d_{\min_+}|\mathcal{L}_+|}}\right) e^{-K\mu'} \leq \gamma. \tag{B.21}$$

Multiplying both sides of (B.21) 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.

B.2.1 Bound for PageRank

Substituting PageRank's diffusion coefficients in the proof of Theorem 1, inequality (B.21) 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

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