Università degli Studi di Milano

SCUOLA DI DOTTORATO IN INFORMATICA
DIPARTIMENTO DI SCIENZE DELL'INFORMAZIONE
CORSO DI DOTTORATO IN INFORMATICA – CICLO XXIII

Tesi di Dottorato di Ricerca

Fast Learning on Graphs

Settore Scientifico Disciplinare: INF/01

Fabio Vitale

Relatore:
Prof. Nicolò Cesa-Bianchi

Correlatore:
Prof. Claudio Gentile

Il Direttore della Scuola di Dottorato in Informatica:
Prof. Ernesto Damiani

Anno Accademico 2009/2010

To my parents

Acknowledgements

My supervisors Nicolò Cesa-Bianchi and Claudio Gentile deserve my deepest gratitude. Most of what I have learned in the past three years is a direct result of their intuition and skillful guidance. Working with them I have acquired a variety of research skills, including the ability to select interesting problems, make progress on difficult ones and present results in an engaging manner. They profoundly influenced my research by showing me the bridge between intuition and formality, asking insightful questions and pushing ideas to their full development in numerous enlightening discussions. Finally, I thank them for providing me with motivation and encouragement and instilling in me the confidence to do research on my own. I am very glad I had the opportunity to work with two of the leading researchers in my field.

I also would like to express my appreciation and gratitude to Giovanni Zappella for his collaboration and hard work.

Most importantly, I want to thank my parents for all the support, constant encouragement and unconditional trust they have given me throughout my life. This thesis is dedicated to them.

Abstract

We carry out a systematic study of classification problems on networked data, presenting novel techniques with good performance both in theory and in practice.

We assess the power of node classification based on class-linkage information only. In particular, we propose four new algorithms that exploit the homiphilic bias (linked entities tend to belong to the same class) in different ways.

The set of the algorithms we present covers diverse practical needs: some of them operate in an active transductive setting and others in an on-line transductive setting. A third group works within an explorative protocol, in which the vertices of an unknown graph are progressively revealed to the learner in an on-line fashion.

Within the mistake bound learning model, for each of our algorithms we provide a rigorous theoretical analysis, together with an interpretation of the obtained performance bounds. We also design adversarial strategies achieving matching lower bounds. In particular, we prove optimality for all input graphs and for all fixed regularity values of suitable labeling complexity measures. We also analyze the computational requirements of our methods, showing that our algorithms can to handle very large data sets.

In the case of the on-line protocol, for which we exhibit an optimal algorithm with constant amortized time per prediction, we validate our theoretical results carrying out experiments on real-world datasets.

Contents

List of Figures

Chapter 1

Introduction

1.1 Networked data

The development of pattern recognition methods has traditionally focused on problems where each data element can be represented by a point in a linear space. Many algorithms have been devised in this context, both in on-line and batch settings. Two popular examples are the family of Perceptron algorithms and the Maximum Likelihood methods.

The possibility of describing each data element as a numerical feature vector led to the construction of a flurry of techniques with which the algorithms' performance can be characterized through powerful analytical tools.

Though many of these methods have had a remarkable impact on Machine Learning, they are neither sufficient nor appropriate for dealing with inputs coming from more complex domains, in which the relational structure of the available information plays a crucial role. Many real-world datasets are indeed rich in structure, involving heterogenous data samples related to each other. This kind of data continue to gain attention in several research areas for various reasons.

One of these reasons concerns the attempt to describe a complex object through the interaction of its components. This need emerged in the last decades in bioinformatics, where the components are molecules, genes, or gene products, but it is also common in research areas like computer vision or language processing. The problems addressed in these fields naturally highlight the expressive limitations of the approaches describing data just as points of a linear space.

Another important reason is related to the growing interest in internet and social networks. The World Wide Web is probably the most active, fast-growing, and heterogenous source of information currently available. The tremendous scale of the World Wide Web makes it one of the most interesting, rich and high-throughput dataset but, at the same time, calls for new fast and accurate machine learning/data mining techniques. Automatic web-page categorization ([93], [28], [88]), web page ranking ([72]), analysis of political web-blog datasets ([98]), clustering and discovering authoritative web-pages for improving web search ([56]), are some of the most common machine learning and data mining applications that are most wanted in these years. Users are becoming more demanding, which entails the need of very fast web-searches and categorization methods. Automatic ontology creation with different granularity levels motivates the development of new classification algorithms that take into account even the personal profile of each single user.

A major web phenomenon, which has attracted considerable attention recently, is the popularity of on-line social networks. Day after day new communication tools are replacing more conventional systems of interaction, and web users consider on-line social networks as being one of their preferred ways for interacting. Such a widespread use of social networks poses new challenges concerning practical problems like link prediction ([62]), analysis of the spreading of diseases in epidemiological models ([5], [68], [35]), targeted advertising ([97]), prediction of user interests ([92]), and analysis of the network's structural properties and evolution over time ([16], [59], [21]). These challenges show the limitations of algorithms operating on geometric domains.

In the web and social networks data samples are interdependent, since each web page is connected to others through hyperlinks, and the users of a social network are interconnected through different types of relationships. In particular, the connections of a social network are the channels used for spreading influence and ideas among its individuals, giving rise to concerted behaviours widely analyzed in sociology. Moreover, new user-centric families of applications are currently emerging on the web, where relational dependencies play an even more crucial role. Some examples are publishing and knowledge management platforms like youtube^{TM} (www.youtube.com), delicious^{TM} (www.delicious.com) and, more in general, Blogs and Wikis.

In addition to the World Wide Web, there are several other kinds of domains, which are natively described as a graph, for which inferences need to be made. Recent work has in fact concentrated on applications, for example for predicting movie evaluations ([96]) or which movie will be a blockbuster ([64]) in graphs representing movie rating datasets, for categorizing scientific papers that are linked to their authors and to each other via citations ([89]), for performing counterterrorism analysis ([66]), or for studying the evolution through time of large dynamic graphs for telecommunications fraud detection ([30]).

Though recently the interest in graph-based learning problem has remarkably increased, the topic has been studied for a longtime. In Statistical Physics, the Ising model (1925) ([50]) is used for singling out the properties of physical systems with pairwise interacting components associated with binary states, computing the free energy of the relative networks. Undirected graphical models have been used also in Spatial Statistics ([79]). For example, in ([12]) lattice analysis schemes are employed for modeling conditional probability in finite systems of spatially interacting random variables. Graph-based algorithms have been applied also in for reconstructing images mapping each pixel to a node of a grid graph ([11], [37], [31], [43], [95]).

1.2 Graph construction

Besides the expansion of World Wide Web and on-line social networks, there is a number of other significant motivations for the growing interest in problems concerning graph-based datasets, like the possibility to convert "flat" data, that can be represented with feature vectors, into data interaction networks. In fact, a widespread approach to the solution of several classification problems is representing the data through an undirected weighted graph in which edge weights quantify the similarity between data points. This technique for coding input data has been applied to several domains, including classification of genomic data ([90]), face recognition ([27]), and text categorization ([39]). In most applications, edge weights are computed through a complex data-modeling process and convey crucially important information for classifying nodes, which makes it possible to infer information related to each data sample even exploiting the graph topology solely.

There are several methods commonly used for transforming a given dataset

in a graph. Once each data entity is mapped to a point in a linear space, the first step consists in modeling the interaction between each pair of data samples through a suitable metric distance, so that the edge weights are simply determined by the distance between all pairs of points in the considered linear space. A distance widely employed for such purpose is the Gaussian function ([9]).

Embedding the dataset in a linear space for these purposes is not always possible. In such cases, the full adjacency matrix is generated by employing suitable similarity functions chosen through a deep understanding of the problem structure. For example Pandey et al. ([71]) adopt weighting schemes for modeling the interactions of protein-protein networks, based on likelihood values. These scores have been assigned through various techniques estimating the experimental reproducibility of these interactions in ([58]). Balcan et al. ([6]) construct graphs from the images of surveillance videos for solving person identification problems. In this case three kinds of edges have been generated: "face edges" (determined with "pixelwise" Euclidean distance between face images), "color edges" (defined in terms of cosine similarity on the color histograms) and "time edges" (based on the time difference between two images). Finally, for TF-IDF representation of documents, the affinity between pairs of samples is often estimated through the cosine measure or the χ^2 distance.

After the generation of the full adjacency matrix, the second phase for obtaining the final graph consists in an edge sparsification/reweighting operation. Some of the edges of the clique obtained in the first step are pruned and the remaining ones can be reweighted to meet the specific requirements of the given classification problem. One of the most common approaches for this pruning operation is the construction of k-*nearest neighbors* graphs, which simply instantiates k indirect edges between each node to the k nodes corresponding to its k nearest samples, according to the distance/similarity function previously computed. Another sparsification technique is generating ϵ-*balls*, which consists in pruning, in the initial clique, all edges whose weights are smaller than a threshold value ϵ. A suitable choice of the parameters governing the degree of sparsification in both these approaches, namely k and ϵ, leads to an improvements over the fully-connected graph for tasks like clustering and manifold learning (see [20] and references therein). However, an improper choice of these parameters may cause the presence of discon-

nected components. Jebara et al. ([52]) propose an alternative sparsification method called b-*matching* that, by generalizing the solution of a *maximum weight matching* problem, ensures that the final graph is b-regular. They also experimentally demonstrated the robustness of this approach in prediction tasks. A further technique for generating a graph ([20]) consists in building a graph ensemble that combines multiple minimum spanning trees, each fitting to a perturbed version of the dataset.

Constructing a graph with these methods obviously entails various kinds of loss of information. However, in problems like node classification, the use of graphs generated from several datasets can lead to an improvement in accuracy performance. Hence, the transformation of a dataset into a graph may, at least in some cases, partially remove various kinds of irregularities present in the original datasets, while keeping some of the most useful information for classifying the data samples. Moreover, as shown in this thesis, it is possible to accomplish classification tasks on the obtained graph using a running time remarkably lower than is needed by algorithms exploiting the initial datasets.

Hence, data interconnections undoubtedly play a crucial role in pattern recognition problems related to this kind of structured domains, spurring the development of graph-based algorithms able to take into account the topology of the underlying representation network.

1.3 Node classification without side information

The abundance and heterogeneity of networked data presents special challenges for a wide range of tasks, from the analysis of structural properties of graphs to the study of disease spread in epidemiological models. The most common and perhaps important class of problems involving graph-based datasets is *node classification*. In this context, each vertex belongs to one or more classes and the learner is asked to classify a given subset of nodes usually in an on-line or batch setting.

In many cases, the algorithms devised for solving node classification problems are driven by the following assumption: linked entities tend to be assigned to the same class. This assumption, in the context of social networks, is known as *homophily* ([13], [65]) and involves ties of every type,

including friendship, work, marriage, age, gender, and so on. In social networks, homophily naturally implies that a set of individuals can be parted into subpopulations that are more cohesive. In fact, the presence of homogenous groups sharing interests is one of the most significant reasons for affinity among interconnected individuals, which suggests that, in spite of its simplicity, this principle turns out to be very powerful for node classification problems in general networks. This motivates the development of several classification techniques based on this bias and the graph topology solely. A further motivation is represented by the various existing methods for transforming flat data into networks (see Section 1.2), codifying part of the information associated with the data samples into edge weights. Finally, it is worth pointing out that gathering attribute information associated with the networked data samples may be very expensive in some cases, and it is therefore worthwhile validating and analyzing the predictive capabilities of methods that do not use side information associated with the vertices.

We now survey some of the most popular node classification methods able to operate without side information —namely, based on the graph topology and the homophilic bias solely.

Formally, in this classification problem we are given an undirected, connected, and weighted graph $G(V, E, W)$ with node set V, edge set E, and positive edge weights $w_{i,j} > 0$ for each $(i, j) \in E$. We focus on the case in which the labels are binary. Hence, a *labeling* of G can be defined as an assignment $\mathbf{y} = (y_1, \ldots, y_n) \in \{-1, +1\}^n$, where $n = |V|$. The algorithm's performance for this problem is usually measured simply with the total number of prediction mistakes occurred.

There are two main settings for this problem: *passive* and *active*.

In the passive setting the learner is a pure observer, since it obtains some of the node labels without being able to choose how this information is provided. Passive learning approaches can be subdivided into *batch* and *on-line*. In the batch paradigm, the temporal order in which the training data information are obtained does not typically matter for the learning task. The learner knows the labels y_i for all i belonging to some node subset $K \subset V$ (*training set*) and the goal is to predict the labels y_j of all j of the set $U \equiv V \setminus K$ (*test set*).

In the on-line paradigm, vertices are presented in an arbitrary order.

More precisely, at each time step, the algorithm is required to predict the label of a new arbitrarily chosen node. After each prediction, the true label is revealed. As real-world applications typically involve large graphs, on-line learners play an important role because of their good scaling properties. Moreover, on-line classification enables a clear characterization of the optimal performance through the analysis of the interplay between the learner and an adversary generating the node permutation and the labeling (typically under some regularity constraint). The study of this interaction provides theoretical performance guarantees for each individual input graph, thus guiding the choice of a suitable algorithm for a given problem. Furthermore, in many applications, methods having good on-line performances are often competitive even when used in batch settings.

In the active setting of the node classification problem the learner is allowed to choose the subset of training nodes. The choice of these vertices is driven by the goal of minimizing the number of mistakes made on the non-queried nodes. The active setting is motivated by several practical needs.

First of all it is worthwhile to note that the batch node classification problem can be especially hard if the training set is formed by a subset of nodes of V connected to the rest of the graph through very few edges. In this case, if all training set nodes are assigned to the same label, say $+1$, and all test set nodes are assigned to the same random label, equal to $+1$ or -1 with probability $1/2$, then any algorithm would make in expectation half mistake per required prediction. However, the labels of all linked test nodes are equal, as well as that ones of all linked training nodes. Hence, the labeling is regular according to the homophilic bias, since in the input graph very few pairs of adjacent nodes are assigned to different labels. A suitable choice of the training set plays therefore a crucial role in the prediction task.

Moreover, in real-world web-based problems, for example, though the datasets can be very large, only a few labels may be obtained. More specifically the off-line (non-adaptive) active protocol can be motivated by considering that, in many situations, requiring a label is time consuming, since it may involve expensive experiments [41]. Hence, it may be significantly less costly to run a single batch of experiments in parallel as compared to running experiments in series. Active learning on graphs has been investigated from a theoretical viewpoint by Guillory and Bilmes [41], and by Afshani et al. [1].

As pointed out in ([100]), several methods operating in the passive batch setting can be naturally cast in a regularization framework. These algorithms estimate a function f on set V in such a way that it satisfies two requirements: (i) f is close to the given labels y_i for all $i \in K$ (*fitting constraint*), and (ii) is smooth on the whole node set (*smoothness constraint*). This can be expressed by a linear combination of two terms: a loss function and a regularization term, relative to the first and the second constraints respectively. Usually, the loss function depends on the difference, for all node $i \in K$, between $f(i)$ and y_i, and the regularization term is expressed as a function of difference, for all pair $(j, k) \in E$, between $f(j)$ and $f(k)$.

These graph methods differ only for the choice of these two terms.

Blum and Chawla ([14]) reduce the node classification to the *mincut* problem, also known as *st-cut* (source-target) problem. In the graph obtained via a suitable transformation of the initial dataset, the positive labels act as sources and must be separated from the negative labels, which acts as targets, minimizing the cutsize. The intuition underlying this reduction is simply that nodes that are strongly connected are unlikely to be separated by the mincut. Hence the mincut provides a solution also for the original classification problem, taking into account the homophilic principle. Clearly the sources and targets are fixed in the mincut problem. This imply that in the linear combination of loss and regularization term, the loss function can be seen as multiplied by an infinite coefficient, so that the minimization problem involves the regularization term solely.

This technique can be viewed as giving the most probable configuration to the labeling if each label y_i is seen as a random variable depending only on the labels of the nodes adjacent to i. The random function whose arguments are drawn from the set of the random labels $y_1, \ldots, y_n$ is a Markov Random Field. More in general, a Markov Random Field is random process defined on the node set of a graph G, in such a way that the random variable v_i associated with node i depends only on all v_j such that $(i, j) \in E$. Hence two random variables v_i are mutually dependent only through a combination of local interactions on the graph structure.

A shortcoming of the method used by Blum et al. is that the solution gives hard classification without confidence. This problem is addressed in [15] extending the mincut approach by adding randomness to the graph structure.

The learning problem can also be formulated in terms of a Gaussian Random Field on the given graph. In general, a Gaussian Random Field can be defined as a probability measure P of a sample space on which is mapped a set X of random variables, satisfying the following property: for each variable subset $\{x_1, x_2, \ldots\}$, the vector with components $P(x_1), P(x_2), \ldots$ is distributed as a multivariate Gaussian. Zhu et al. in [102] adopt Gaussian fields assigning a probability distribution P on functions $f : V \rightarrow \mathbb{R}$. P is defined as a function of the loss and regularization terms of f, representing respectively the fitting and smoothness constraints of the labeling of the input graph. Their approach is a continuous relaxation to the discrete Markov Random Field and it is intimately connected to random walks and spectral graph theory. The solution function $f : V \rightarrow \mathbb{R}$ is forced to take values $f(i) = y_i$ for all nodes $i \in L$, which is equivalent to having a infinite loss coefficient in the linear combination of loss and regularization term (like for the mincut of approach of Blum and Chawla – [14]). The smoothness penalty term depends only on the differences between $f(i)$ and $f(j)$ for all $i, j \in V$ such that $(i, j) \in E$. The energy function f minimizing the regularization terms is harmonic, i.e., it satisfies $\Delta f = 0$ on the unlabeled nodes U and is equal to the given labels on the nodes of K, where the Laplace's operator Δ coincides in this case with the Laplacian matrix of G. An interesting property of this function is that $f(i)$ is equal, for each node $i \in U$, to the weighted average of the values of f for all the i's neighbors. Moreover f is unique, it satisfies $0 \leq f(i) \leq 1$ for all $i \in V$. More precisely $f(i)$ is equal to the probability of hitting, with a random walk starting from node i, a node labeled with 1.

Zhou et al. ([99]) propose an algorithm inspired by diffusion kernels ([54], [57], [85]) to find a smooth function that can be defined in terms of spreading activation networks ([4], [82]). Their approach, unlike the two methods described above, is characterized by a parametric trade-off between fitting and smoothness constraint.

Belkin et al. [8] propose a Tikhonov regularization algorithm using a loss function and a smoothness penalty term subject to the harmonic constraint $\sum_i f(i) = 0$.

Bengio et al. [10], besides showing how different node classification algorithms can be cast into a common framework where the objective is to minimize a quadratic cost criterion, propose a new method for "propagating" the label values through the edge connections from labeled examples to

the whole dataset. This method is inspired by the Jacobi iterative method for linear systems. The approaches spreading the known labels on the other nodes using the graph structure (see also [101]) clearly reflect the homophilic bias, and are known as *label propagation* methods.

Using a maximum margin and maximal average margin based argument, Pelckmans at al. ([75]) prove a generalization bound for approaches whose prediction of each label y_i is based on the weighted average of the labels of the i's neighbors. They also construct a convex formulation and implemented this scheme as a linear program.

Szummer and Jaakkola ([87]) classify the vertices of the considered graph through the use of a t-step random walk.

On-line linear learners, such as the Perceptron algorithm, have been applied to the binary classification of the vertices of a graph by embedding the vertices of the graph in $\mathbb{R}^{|V|}$ through a map transforming node i to the i-th coordinate versor $\boldsymbol{e}_i \in \mathbb{R}^{|V|}$. For example, the graph Perceptron algorithm [49, 47] predicts the label y_i (associated with versor $\boldsymbol{e}_i$) using the linear kernel $K = L_G^{\dagger} + \mathbf{1}\mathbf{1}^{\top}$, where L_G is the Laplacian of G, $L_G^{\dagger}$ is its pseudoinverse, and $\mathbf{1} = (1, \ldots, 1)^{\top}$.

The idea of using a spanning tree to reduce the *cutsize* induced by the labeling, i.e., the number of edges incident to nodes with different labels, has been investigated in [48], where the graph Perceptron is applied to a spanning tree T of G. A different technique [46] attempts to control the cutsize by linearizing T via a depth-first visit so as to preserve some regularity property of the labeling and running a Nearest Neighbor predictor on the obtained line graph.

In [46] the authors classify the vertices of the considered graph through a combination of Perceptron and Nearest-Neighbor previously proposed in [44].

Herbster and Lever in [45] give a p-seminorm on the space of graph labellings. On every step their algorithm predicts using the labelling minimizing the p-seminorm and is consistent with the revealed labels. When $p = 2$ this is the harmonic energy minimization procedure of [102] that we described above. In the limit as $p \to 1$ this is equivalent to predicting with a label-consistent mincut.

Another on-line algorithm for binary classification is the so-called GRAPHTRON algorithm [74]. This algorithm predicts at time t with the majority

vote of the labels of previously mistaken nodes that are adjacent to i_t.

Finally, it is worthwhile mentioning the work of Smola and Kondor [85], where the authors introduced a family of kernels on graphs based on the notion of regularization operators.

1.4 Statistical relational learning methods

The homophilic bias assumption can be considered very powerful for classification problems in which the network is formed by only one type of nodes and links. However, in other kind of graph-based problems, the network is formed by an heterogeneous set of objects which are usually associated with a list of attributes, and incorporate a much richer relational structure. The properties of each entity depend probabilistically on those of the linked entities. In this section we briefly mention some of the methods able to handle this kind of heterogeneous datasets, capturing probabilistic interactions between attributes of related entities.

In the last years, statistical approaches, suitable for learning from a collection of homogeneous independent instances, have been combined with relational learning methods. For the task of hypertext classification Slattery and Craven in [83] devised a method involving a statistical learner, a feature selection procedure based on word frequencies and the relational structure induced by the hyperlinks connecting the documents.

Chakrabarti et al. in [26] combined textual and linkage features into a general statistical model for predicting the topics of documents, considering at the same time the local features of the document being classified, together with the text of the document's neighbor.

Popescul et al. in [77] propose an approach integrating inductive logic programming techniques ([60]) for generating features exploring the relational structure of the dataset, together with a logistic regression model which takes into account also local features of the data being classified.

Slattery and Mitchell in [84] propose an iterative algorithm for the web page classification task. Their method exploits the presence of additional regularity in the test set and combines the use of recursive rules for identifying the most informative pages with text-based classifiers in an iterative relaxation scheme.

Other techniques are based on probabilistic relational models ([76], [69])

which represent a joint probability distribution over a relational dataset and extend the standard attribute-based Bayesian network representation ([73]) quantifying probabilistic interactions between attributes of related entities. This kind of structure exploits the possibility that a variable used in one instantiation of a Bayesian network may refer to the same variable in another Bayesian network.

In order to use the whole structure of knowledge encoded by this relational representation, Friedman at al. ([34]) show how to extend statistical methods for learning Bayesian network to the task of learning these more complex models.

Getoor at al. ([38]) extend probabilistic relational models, which originally focused on modeling the distribution over the entity attributes, by a unified statistical framework for nodes and links which defines probability distributions over the presence of links between objects.

Since Bayesian networks must be acyclic, probabilistic relational model cannot handle directly homophily. Relational dependency networks ([67]) is a form of graphical model able to model cyclic dependence and homophily. It estimates a set of conditional distributions independently, avoiding therefore the complexity of estimating a full joint distribution.

One of the main advantages of the approaches we mentioned is the capability to handle structured data with a very complex relational nature. However, it is also important to note that it is difficult to provide, for this kind of algorithms, theoretical analyses which guarantee accuracy performances as for the methods handling homogeneous datasets whose complexity is governed by the homophilic bias solely.

1.5 Overview of our main contributions

Many algorithms developed for node classification and based only on the homophilic principle are heuristic methods whose accuracy performance are not formally analyzed. Such an analysis would require, first of all, a rigorous definition of a regularity measure related to the considered bias. The few techniques with theoretical performance guarantees almost always do not provide a lower bound analysis validating the quality of the exhibited results, nor they are computationally efficient.

Moreover, an increasing amount of relational data is becoming available every day, and the tremendous volume of information provided by data sources like the World Wide Web demands a more severe point of view on what can be considered as being a "computationally efficient algorithm". Several prediction algorithms developed for the node classification problem perform matrix operations or linear programming applications, requiring a total time cubic (or worse) in the number of predictions. These methods are clearly impractical when handling very large-scale datasets, such as on-line social networks, on-line movie rating, and paper coauthors networks.

In this thesis we assess the power of class-linkage alone proposing four node classification algorithms based on the homophilic bias and the graph topology. For *all* these methods we provide

- An accuracy analysis and a careful comparison against other methods together with an interpretation of the obtained mistake bounds.
- A *matching* lower bound, which implies *optimality* (up to constant or $\mathcal{O}(\log |V|)$ factors) for *every* input network.
- Time and space complexity analysis showing that our algorithms are able to handle very large datasets.

Moreover, in Chapter 2, 3 and 5, we also give two new regularity measures for this problem, carrying out a comparison to other existing measures.

Finally, for one of our methods, which requires amortized *constant* time per prediction (for most input graphs —see Chapter 3), we validate the theoretical results with careful experimental comparisons against other competing methods. In particular, we show that our algorithm compares well to all competitors while using, in most cases, the least amount of time and memory resources.

We thus carry out a systematic study of the node classification problem achieving the difficult objective of presenting novel techniques able to satisfy, at the same time, desirable theoretical and practical requirements. More precisely, whilst designing each algorithm, we take into account the running time and memory space required and the performance accuracy. In order to accomplish this goal, the accuracy analysis of these methods is based on a careful study of the game between the forecaster and an adversary

deciding the labeling, which stems directly from the homophilic bias and the considered protocol rules.

The set of the algorithms we present covers diverse practical needs. In fact, some of them operate in an active transductive setting, others in an on-line passive transductive setting and others still in an *explorative* model (which is somehow related to the graph exploration problem introduced in [32]).

One of the main obstacles in developing these methods arises from the need of having very good scalability properties. This leads us to avoid the use of many powerful but computational expensive analytical tools, which are ubiquitous in the literature related to these problems. Some of these tools involve carrying out matrix operations or linear programming applications. In fact, in order to develop very scalable algorithms, we depart from traditional approaches, resorting to both novel and well-established combinatorial methods related to graph-based problems. Some of these are ad hoc techniques devised for this work.

Hence, although there is a wide literature addressing the node classification problem, our methodologies for proving optimality, and the implementation techniques we employ for improving the computational resource use, are not "mainstream", which implies a further difficulty for achieving our objective.

This thesis originates from the following publications:

- ([22]) N. Cesa-Bianchi, C. Gentile, F. Vitale, **Fast and optimal prediction of a labeled tree**. Proceedings of the 22nd Annual Conference on Learning Theory, Omnipress, 2009 – (COLT 2009).

- ([25]) N. Cesa-Bianchi, C. Gentile, F. Vitale, and G. Zappella. **Random spanning trees and the prediction of weighted graphs**. Proceedings of the 27th International Conference on Machine Learning, Omnipress 2010 – (ICML 2010).

- ([24]) N. Cesa-Bianchi, C. Gentile, F. Vitale, and G. Zappella. **Active learning on trees and graphs**. Proceedings of the 23rd Annual Conference on Learning Theory, Omnipress 2010 – (COLT 2010).

- ([23]) N. Cesa-Bianchi, C. Gentile, F. Vitale. **Learning unknown Graphs**. Accepted for publication in *Theoretical Computer Science*, special issue on Algorithmic Learning Theory.

The last paper is based on

- N. Cesa-Bianchi, C. Gentile, F Vitale: **Learning Unknown Graphs**. 20th International Conference on Algorithmic Learning Theory. Springer, 2009 – (ALT 2009).

Preliminary versions of [22], [25] and [24] have been presented respectively in

- N. Cesa-Bianchi, C. Gentile, F. Vitale. **Online Graph Prediction with Random Trees**. Workshop: New Challanges in Theoretical Machine Learning: Data Dependent Concept Spaces – (NIPS 2008).

- N. Cesa-Bianchi, C. Gentile, F. Vitale. **Fast and Optimal Algorithms for Weighted Graph Prediction**. Workshop: Analyzing Networks and Learning with Graphs – (NIPS 2009).

- N. Cesa-Bianchi, C. Gentile, F. Vitale, G. Zappella. **Scalable algorithms for learning on graphs**. Workshop: Learning in Non-(geo)metric Spaces – (ICML 2010).

We now describe in detail the main contributions of our work.

Transductive and explorative settings

We consider the transductive batch setting (the entire graph is known in advance) both within the on-line (passive) and the active protocol. In Chapter

3 we present an algorithm operating within an on-line protocol and carry out an experimental evaluation of this method in a batch setting. The positive results obtained suggest that good predictive capabilities in the on-line protocol correspond, for this problem, to good practical performance in the batch setting.

In Chapter 5 we present a novel prediction protocol that we call *exploorative*. In this protocol we drop the transductive assumption and study the graph prediction problem from a purely sequential standpoint, where the vertices (and their incident edges) of an unknown graph are progressively revealed to the learner in an online fashion. As soon as a new vertex is revealed, the learner is required to predict its label. Before the next vertex is observed, the true label of the new vertex is fed back to the learner. In this setting the learner is allowed to actively explore the graph in directions that are judged easier to predict. This new setting is interesting from several points of view. For example, in many real-world problem, the whole network is so large that it cannot be considered as the problem instance in its entirety. Ideally, one should be able to choose in a suitable way the parts of the graph to operate which. A typical example related to this issue is the target advertising problem in an on-line social network, in which the goal is to target each member of a social network with the product he/she is most likely to buy.

Complexity measures

We present two novel complexity measures for the node classification problem. In Chapter 5 we describe a new measure called *merging degree*. The homophilic principle implies that labeling can be considered as being *regular* when the graph can be partitioned into a small number of weakly interconnected clusters (subgroups of network members) such that labels in each cluster are all roughly similar. The merging degree is inherently related to the degree of interaction among the clusters which the graph can be partitioned into. We carry out a careful comparison between this regularity measure and the cutsize, showing that, in general, the merging degree is able to quantify the labeling regularity in a more refined and robust way.

In Chapter 2 and 3 we propose a new natural complexity measure for binary labeling, namely the number of edges connecting nodes with different

labels that happen to be included in a uniformly generated random spanning tree. When the input graph is unweighted, this regularity measure coincides with sum of the *effective resistances* between each pair of nodes i and j such that $(i,j) \in E$ and $y_i \neq y_j$. The effective resistance between any pair of nodes is a connectivity measure widely used in graph theory (see, e.g., [63]), ad is equivalent to the resistance in an electrical network represented by the considered graph, in which current flows along each edges (see Section 1.6). The use of spanning trees has the advantage of retaining relevant spectral information from the original graph. This regularity measure takes into account the smoothness of the labeling on V, but, unlike the commonly used cutsize, satisfies several desirable properties: it is invariant to uniform weight scaling, it is locally density-independent since the contribution of each edge (i,j) depends in inverse proportion on how strongly i and j are connected, and it scales linearly in the number of nodes of the input graph (global density independence).[1]

Optimal accuracy

For each algorithm that we present, we provide a rigorous theoretical analysis together with an interpretation of the results obtained. We also design adversarial strategies in order to exhibit *matching* lower bounds. In particular, through the complex interplay between the hard combinatorial constraints related to our algorithms and the adversarial strategies, we prove optimality *for all* input graphs and *for all* fixed regularity values.[2]

This kind of optimality is clearly "stronger" than other optimalities proposed in the literature of node classification problem. For example, in [1] Afshani et al. proved that their algorithm is optimal providing a "matching" lower bound obtained by the analysis of a single graph constructed for this purpose, which clearly may be different from the real input graph. Hence,

[1] Even simple measures based on the standard cutsize, like $(|V|/|E|) \cdot$ cutsize, scale linearly in the number of nodes of the input graph. However it is immediate to see that such measures do not satisfies the other properties.

[2] For the algorithm described in Chapter 3, the optimality actually holds for all graphs save for some pathological inputs, like, for example, binary labeling such that the total weight of the edges connecting nodes labeled in different way almost coincides with the total weight of all the edges in E. Such a labeling patently contrasts with the homophilic bias. Other pathological examples stem simply from the need of extending the connectivity notion to *weighted* graphs. (see Chapter 3).

in this case upper and lower bounds refers in general to different graphs and their optimality does not hold *for all* input graphs.

Computational efficiency

Scalability is one of the main issues of this work. For *all* our algorithms, we provide computational analyses ensuring that our methods are able to handle very large datasets. In order to achieving our objective we resort to other methodologies and develop new ad hoc combinatorial methods. In particular, In Chapter 2 and 3 we use uniformly generated random spanning trees for condensing the structural information of the original dataset, enabling the use of some fast prediction techniques.

Finally, in Chapter 3 we reduce node classification to the problem of predicting on a weighted line graph (inspired by [41]) and we devise an innovative method solving this problem in *constant* amortized time per prediction and space *linear* in the number of nodes.

Experiments

In Chapter 3 we describe an algorithm which is both computationally very efficient and able to operate on weighted graphs. To evaluate the performance of our algorithm and validate our theoretical analysis, we present the results of an experimental comparison on a number of real-world weighted graphs from different domains: text categorization, optical character recognition, and bioinformatics. We compare our algorithm to online prediction methods, intended as representatives of two different ways of facing the graph prediction problem: global (Perceptron) vs. local (label-propagation) prediction. The experimental results show that our method compares well to both global and local prediction techniques, while being much faster.

1.6 Preliminaries

In this section we formally provide the basic definitions and notations used throughout the thesis.

1.6.1 Graphs and Laplacian matrix

Formally an *unweighted* **graph** $G(V, E)$ is defined to be a pair (V, E), where V is a non-empty finite set of elements called *nodes* (or *vertices*) and E is a finite family of pair of elements of V called *edges* (or *links*). If (i, j) belongs to E, we say that i is *adjacent* to j and that (i, j) is *incident* to nodes i and j. When the pairs of V's elements contained in E are unordered then G is called *undirected*. If each pair of elements of V is linked only through at most one edge and there is no edge connecting a node with itself, the graph is called *simple*. Throughout the thesis, we will focus on undirected and simple graphs solely.

With each unweighted graph $G(V, E)$ is associated its **adjacency matrix** A. Each element $a_{i,j}$ of A is equal to 1 if $(i, j) \in E$ and 0 otherwise. Observe that when a graph is simple all elements of the main diagonal of A are null. Clearly, the adjacency matrix of a undirected graph is always symmetric.

The **degree** of a node $i \in V$ is the number of edges incident to i.

The **Laplacian matrix** L_G of an unweighted graph $G(V, E)$ is equal to $D - A$, where D is the diagonal matrix such that $D_{i,i}$ is equal the degree of node i.

When each edge (i, j) of a graph G is associated with a positive *weight* $w_{i,j}$ (in general real-valued), then G is said to be **weighted**, and is denoted by $G(V, E, W)$. The matrix W contains the values of all weights $w_{i,j}$. If $(i, j) \notin E$ then $w_{i,j} = 0$.

The **Laplacian matrix** L_G of a weighted graph $G(V, E, W)$ is equal to $D - W$, where D here is the diagonal matrix such that $D_{i,i}$ is equal the sum of the weights of all edges incident to node i.

A **subgraph** $G'(V', E', W')$ of a given graph $G(V, E, W)$ is a graph where $V' \subseteq V$, $E' \subseteq E$ and $w'_{i,j}$ is equal to $w_{i,j}$ if $(i, j) \in E'$ and null otherwise. This definition extends for unweighted graphs simply observing that each unweighted graph $G(V, E)$ can be seen as a weighted graph $G'(V, E, W)$ in which the weight matrix W coincides with the adjacency matrix A of G.

A **walk** is an alternate sequence of edges and vertices of the form $i_1, (i_1, i_2), i_2, (i_2, i_3), i_3, \ldots$. The number of (not necessarily distinct) edges present in the sequence is the **lenght** of the walk.

If all nodes of the sequence are distinct, the walk is said to be a **path**, and is usually denoted by the sequence of vertices solely $i_1, i_2, \ldots$. The length of

a path is therefore equal to its number of nodes minus one.

If any two vertices of a given undirected graph G are connected via at least one path, then G is **connected**.

The **diameter** of a connected graph is the number of edges present in the longest path.

A **bridge** is a edge whose removal disconnects the graph.

We now give the definitions of some of the most important classes of graphs. For the sake of simplicity we will refer to unweighted graphs. The definitions of these graphs can be easily extended to the weighted case associating a positive weight value with each edge.

A **clique** $C(V, E)$ (or **complete** graph) is a graph where each node is adjacent to every other node.

A **tree** $T(V, E)$ is a connected graph with $|V| - 1$ edges.

A **forest** $\mathcal{F}$ is a set of trees having disjoint node sets.

A **star**-graph $S(V, E)$ is a tree in which one of the nodes is adjacent to all the others.

A **line**-graph $L(V, E)$ is a tree with diameter equal to $|V| - 1$.

The **terminal** nodes of a line-graph are the two nodes having degree equal to 1.

A **lollipop**-graph $G(V, E)$ is a graph formed by a clique C and a line-graph L having disjoint node sets, together with an edge incident to a node of C and a terminal node of L.

A **spanning-tree** $T(V, E')$ of a given graph $G(V, E)$ is a subgraph of G that is a tree and has the same node set of G.

1.6.2 Resistance and random spanning tree

The **effective resistance** $r_{i,j}^W$ between two nodes of a given undirected and simple graph $G(V, E, W)$ is a connectivity measure widely used in graph theory, which can be defined in terms of the Laplacian matrix of G as follows:

$$r_{i,j}^W = L_{ii}^\dagger + L_{jj}^\dagger - 2L_{i,j}^\dagger \ ,$$

where $L^\dagger$ is the pseudoinverse of the Laplacian matrix of G.

The **resistor** of any edge $(i, j) \in E$ is defined as $1/w_{i,j}$.

In the interpretation of the graph as an electric network where the weights $w_{i,j}$ are the edge **conductances**, the effective resistance $r_{i,j}^W$ is the voltage between i and j when a unit current flow is maintained through them.

Given a graph $G(V, E, W)$, a **uniformly generated random spanning tree** is a random spanning tree taken with probability proportional to the product of its edge weights. The probability that each edge $(i, j) \in E$ is included in such spanning tree is equal to $w_{i,j} r_{i,j}^W$ (see, e.g., [63]).

The concepts of effective resistance and random spanning trees are intimately connected with *random walks*, which can be defined as follows.

Given a graph $G(V, E, W)$, a **random walk** is a walk generated randomly through a sequence of steps in the following way. In the first step a vertex $i_1 \in V$ is added to the random sequence. At each time step t the edge (i_{t-1}, i_t) together with vertex i_t is added to the random walk with probability $w_{i_{t-1},i_t} / \sum_{i:\ (i_{t-1},i)\in E} w_{i_{t-1},i}$.

If we construct a spanning tree T of G performing a random walk on G and adding to T, at each time step t, each randomly selected pair edge-vertex $((i_{t-1}, i_t), i_t)$ only if node i_t have never been added previously to the random walk, then T is a uniformly generated random spanning tree (see, e.g. [63]).

1.6.3 Labeling and cutsize

A **labeling** of G is any assignment $\mathbf{y} = (y_1, \ldots, y_n) \in \{-1, +1\}^n$ of binary labels to its nodes.[3] We use $(G, \mathbf{y})$ to denote the resulting labeled weighted graph.

A standard notion of label regularity is the **cutsize** of a labeled graph, defined as follows. A **ϕ-edge** of a labeled graph $(G, \mathbf{y})$ is any edge (i, j) such that $y_i \neq y_j$. Similarly, an edge (i, j) is **ϕ-free** if $y_i = y_j$. The **cutsize** $\Phi_G(\mathbf{y})$ of $(G, \mathbf{y})$ is the number of ϕ-edges in E, i.e. $\Phi_G(\mathbf{y}) = |(i, j) :\ y_i \neq y_j|$ (independent of the edge weights).

[3] We focus on binary labeling. However, in Chapter 5 we will describe an algorithm able to operate also with real labeling.

1.7 Outline of the thesis

The thesis is organized as follows.

In **Chapter 2** ([22]) we characterize, up to constant factors, the number of mistakes necessary and sufficient for sequentially predicting a given tree with binary labeled nodes. We provide an efficient algorithm achieving this number of mistakes on any tree. In order to cope with adversarial assignments of labels over a general graph, we advocate the use of random spanning trees.

In **Chapter 3** ([25]) we show that the mistake bound for predicting the nodes of an arbitrary weighted graph is characterized (up to logarithmic factors) by the the number of edges connecting nodes with different labels of a random spanning tree of the input graph. In deriving our characterization, we obtain a simple randomized algorithm achieving the optimal mistake bound on any weighted graph. We validate the theoretical results carrying out experiments on real-world datasets.

In **Chapter 4** ([24]) we investigate the problem of active learning on a given tree whose nodes are assigned binary labels in an adversarial way. We characterize (up to constant factors) the optimal placement of queries so to minimize the mistakes made on the non-queried nodes. The optimal number of mistakes on the non-queried nodes is achieved by a simple and efficient mincut classifier. Through a simple modification of the query selection algorithm we also show optimality (up to constant factors) with respect to the trade-off between number of queries and number of mistakes on non-queried nodes. Finally, we provide a lower bound on the number of mistakes made on arbitrary graphs by any active learning algorithm using a number of queries which is up to a constant fraction of the graph size.

In **Chapter 5** ([23]) we introduce a new model of online learning on labeled graphs where the graph is initially unknown and the algorithm is free to choose which vertex to predict next. For this learning model, we define an appropriate measure of regularity of a graph labeling called the merging degree. After observing that natural nonadaptive exploration/prediction

strategies, like depth-first with majority vote, do not behave satisfactorily on graphs with small merging degree, we introduce an efficiently implementable adaptive strategy whose cumulative loss is provably controlled by the merging degree. Finally, we provide a matching lower bound demonstrating that in the case of binary labels our analysis cannot be improved.

In **Chapter 6** we conclude by mentioning some promising research directions for future work.

Chapter 2

Learning on unweighted trees and graphs

2.1 Introduction

In the (passive) on-line setting for the node classification problem, bounds on the number of prediction mistakes are naturally expressed in terms of the cutsize. This immediately suggests a simple regularization technique: if the mistakes of a prediction algorithm are bounded in terms of the cutsize of the graph, then it should be beneficial to run the algorithm on a thinned version of the original graph where some of the edges have been dropped. Since dropping edges that cause the graph to disconnect is intuitively throwing away too much structural information, we are naturally led to the idea of running the learner on a spanning tree of the original graph.

This approach leaves us with the problem of choosing a good spanning tree. Because of the adversarial nature of the on-line setting, the presentation of vertices and the assignment of labels are both arbitrary. This suggests to pick a tree at random among all spanning trees of the graph so as to prevent the adversary from concentrating the cutsize on the chosen tree. Moreover, we can exploit Kirchoff's equivalence between the effective resistance of an edge and its probability of being included in a random spanning tree. This equivalence allows us to express the expected cutsize of the random spanning tree in a simple form, namely, as the sum of resistances over all edges in the cut of G induced by the adversarial label assignment. On the other hand, the resistance-weigthed cutsize is a very natural measure of complexity

for labeled graphs, and this is precisely the fact that led us to consider random spanning trees. In Section 2.2 we exhibit a lower bound showing that this measure captures the hardness of the node classification problem even on weighted graphs: We demonstrate that given *any* weighted graph, *any* prediction algorithm can be forced to make a number of mistakes which is at least as big as the cutsize of the graph's random spanning tree.

There is a vast literature on the problem of drawing random spanning trees from a graph (see, e.g., the recent monograph [63]). For "most" graphs, a random spanning tree can be sampled with a random walk in time $\mathcal{O}(|V| \ln |V|)$ [18], or even $\mathcal{O}(|V|)$ [3, 94], although all known techniques take $\Theta(|V|^3)$ in the worst case. As a matter of fact, this cubic worst-case bound is a theoretical limitation only, since the bound is hardly met in practice. The space complexity for generating a random spanning tree is always linear in the graph size. Finally, although we exploit random spanning trees to reduce the cutsize, similar approaches can also be used to approximate the cutsize of a weighted graph (see, e.g., [86]).

Based on the above argument for using random spanning trees in graph prediction tasks, we mainly focus on the problem of designing a good algorithm for predicting an arbitrary tree.

In this chapter we present an algorithm that is both optimal (up to constant factors) and efficient. Optimality is meant in the following sense: Given any tree T, the worst-case (over labeling and node presentation order) number of mistakes made by our algorithm can only be improved by a factor which is constant with respect to the relevant parameters. As for efficiency, we show that the overall running time of our algorithm is of order $\min\{K, n_f\}K + n \log D_T$, where K is the cutsize of the (labeled) tree T, D_T is the diameter of T, $n = |V|$ and $n_f < n/2$ is the number of nodes in T with degree bigger than two.

2.1.1 Preliminaries

We now give some basic definitions used throughout this chapter.

We define $\mathcal{Y}(T, k)$ as the set of all labelings of T with exactly k ϕ-edges. We will say that $\mathcal{Y}(T, k)$ has *cutsize* k.

A **cluster** C of a labeled tree $(T, \mathbf{y})$ is a maximal subtree of T containing no ϕ-edges.

With $\pi(i,j)$ we denote the (unique) path connecting node i to node j, and $d(i,j)$ denote the number of edges in $\pi(i,j)$.

If A is a tree prediction algorithm and $(T,\mathbf{y})$ is a labeled tree, then $m(A,T,\mathbf{y})$ denotes the worst-case number of prediction mistakes made by A over all presentations $i_1,\ldots,i_n$ of nodes of T. With a slight abuse of language we define

$$m(A,T,\leq K) = \max_{k=1,\ldots,K} \max_{\mathbf{y}\in\mathcal{Y}(T,k)} m(A,T,\mathbf{y}) \ .$$

This is the number of mistakes made by A on the worst-case choice of a labeling of T with *cutsize budget* K. The maximization over k is needed because $\max_{\mathbf{y}\in\mathcal{Y}(T,k)} m(A,T,\mathbf{y})$ is in general not monotonic in k. We define the *minimax mistake bound* on a tree T with cutsize budget K by

$$\textsc{opt}(T,K) = \min_A m(A,T,\leq K) \ ,$$

where the minimum is over all deterministic prediction algorithms.

Finally, when focusing on the nodes of a given cluster C throughout the proofs, without loss of generality we assume they are all given label $+1$.

2.2 Lower bounds

We now describe an adversarial strategy that, given a tree T with n nodes and cutsize K (for $1 \leq K < n$) forces any deterministic prediction algorithm A to make a certain number of mistakes that depends both on K and T. This lower bound is achieved for a worst-case choice (depending on A) of both labeling and node presentation order.

The lower bound is based on the following fact. Given a line graph ℓ (i.e., a "list") with $n+1$ nodes $1,\ldots,n+1$ and $|\ell| = n$ edges, a simple dichotomic adversarial strategy can always force $\lfloor \log_2(n+1) \rfloor$ mistakes using a cutsize of at most 1. In order to achieve this, the adversary initially assigns an arbitrary label to one of the two terminal nodes of ℓ, say node 1. Now let i_1 be the node of ℓ such that there are exactly $\lceil n/2 \rceil$ edges between 1 and i_1. The adversary chooses node i_1 first, and forces a mistake by picking y_{i_1} to be different from the algorithm's prediction. Now let $\ell_1 \subseteq \ell$ be the (sub-)line having as terminal nodes 1 and i_1 if $y_{i_1} \neq y_1$, or nodes i_1 and $n+1$, otherwise. Let i_2 be the node of ℓ_1 such that there are $\lceil |\ell_1|/2 \rceil$ edges

between i_1 and i_2. The adversary then chooses node i_2 and another mistake is forced as in the previous step. The adversary proceeds recursively in this way until the chosen sub-line contains a single edge. Then, irrespective to the algorithm's predictions, all the remaining nodes are labeled in such a way that the cutsize does not increase. It is then easy to check that $\lfloor \log_2\big(n+1\big) \rfloor$ mistakes are forced. Moreover, it is important to observe that the above adversarial strategy works even if y_1 is already known to the algorithm. On the other hand, this strategy cannot be applied if the known label is on an internal node of ℓ. This fact is used in the proof of Theorem 3.1 below.

The above adversarial strategy is extended to trees in the following way. The adversary looks for a certain set L of K edge-disjoint line graphs contained in T, and then applies the dichotomic strategy *independently* on each line. To this end, it suffices the set L is a *blanket*, a notion which we now define. Given a set L of edge-disjoint lines contained in a tree T, we say that $\ell \in L$ is a **grafted line** if one of the two terminal nodes of ℓ is also an internal node of another line $\ell' \in L$. This shared node is called the **graft** node of ℓ. We say that L is a **connected blanket** if: (i) The union of all lines in L forms a (connected) tree, (ii) every node in this (connected) tree can be internal node of at most one such line, and (iii) Every grafted line in L shares with the remaining lines in L no nodes but the graft, and Finally, L is a **blanket** if it is either a connected blanket or it has been obtained by a connected blanket after removing one or more of its lines.[1] See Figure 2.1 for an example. The **size** of a blanket L is the number of its lines $|L|$. Note that a blanket need not include all edges of the original tree T. Also, observe that for any size $K < n$, a size-K blanket over a tree T always exists: take L to be any set of K distinct edges in T; then no lines of L have internal nodes and the blanket property trivially holds. On the other hand, a given tree T clearly admits many size-K blankets.

Let $\mathcal{L}(T, K)$ be the set of all size-K blankets over T, and define the function LB ("lower bound") as follows:

$$\text{LB}(T, K) = \max_{L \in \mathcal{L}(T,K)} \sum_{\ell \in L} m_\ell$$

where we use the abbreviation $m_\ell = \Big\lfloor \log_2(|\ell| + 1) \Big\rfloor$.

[1] Observe that a given L might be generated by many connected blankets.

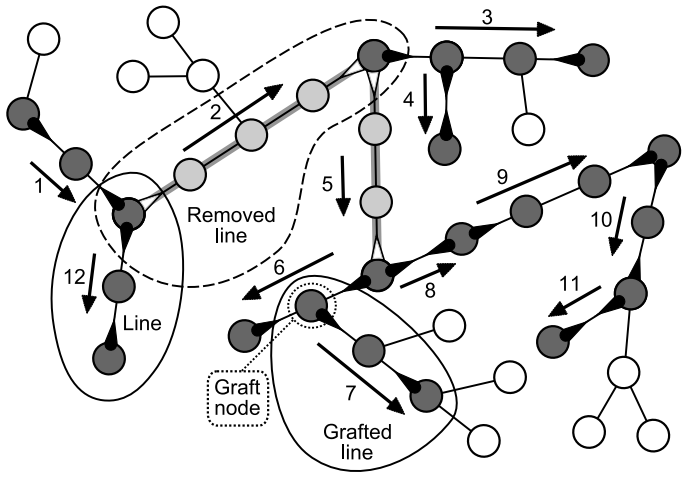

Figure 2.1: A tree T whose nodes have been divided into three types: dark shaded, light shaded, and white. A connected blanket is shown including dark shaded and light shaded nodes only, along with their connecting edges. Another blanket L, of size 9, is formed by 3 connected blankets (made up of 2, 2 and 6 lines, respectively), and is obtained from the connected blanket by removing the two lines indicated by the light shaded edges and nodes. The terminal nodes of each line are indicated by a bulbous endpoint of the incoming edge. The edges directly connected to white nodes are not part of the underlying connected blanket. The numbers denote a possible depth-first presentation order followed by an adversary that starts from, say, the dark shaded node on the top-left. This adversary assigns to the white nodes the same label as their closest (dark or light) shaded nodes. Similarly, the light shaded nodes belonging to the two removed lines are labeled as the corresponding line terminal nodes.

Theorem 2.1 *For any tree* T *with* n *nodes and any cutsize* $K = 1, \ldots, n-1$*, we have* $\textsc{opt}(T, K) \geq \textsc{lb}(T, K)$.

PROOF. Given any size-K blanket L over T, we need to exhibit an adversarial strategy that allows the adversary to apply the logarithmic lower bounding argument for line graphs to each line $\ell \in L$ *independently*, by using at most K ϕ-edges. A key fact here is that each line of L can be processed by the adversary even if one of the two terminal nodes has already been revealed to the learning algorithm.

Since L is a blanket, we know there exists a connected blanket L_0 such

that $L \subseteq L_0$. The adversary initially finds a line $\ell_1 \in L$ which is not grafted (ℓ_1 must exist since T has no cycles) and performs a depth-first visit over the lines in L_0 starting from a terminal node of ℓ_1. The adversary processes the lines in L_0 in the order determined by the visit. If the current line ℓ belongs to L then the adversary applies the strategy for line-graphs spending one (*at most* one when $|\ell| = 1$) ϕ-edge, and causing the learning algorithm to make m_ℓ mistakes on ℓ. Our argument gives no guarantees on the number of mistakes forced on the lines in $L_0 \setminus L$ (e.g., the light shaded lines in Figure 2.1). Thus, irrespective to the algorithm's predictions, the non-terminal nodes of a non-grafted line in $L_0 \setminus L$ are given the same label as the terminal node shared with the line in L that precedes in the depth-first order. For instance, in Figure 2.1 the three light shaded internal nodes in Line 2 are labeled like the dark shaded terminal node shared with Line 1. This allows the adversary to avoid using ϕ-edges on the removed lines $\ell \in L_0 \setminus L$, at the cost of being forced to set the label of the terminal nodes of one or more lines that follow ℓ in the depth-first order (for instance, assigning labels to the non-terminal nodes of Line 2 determines the labels of the left terminal node of Line 3). However, we know that this constraint is compatible with the lower bounding argument for line graphs.

If $\ell \in L_0$ is a grafted line, the depth-first order insures that ℓ will be processed only after the (unique) line ℓ is grafted onto (in Figure 2.1, Line 7 is guaranteed to be processed after Line 6). Note that, again, this is key to enabling the application of the lower bounding strategy for line graphs independently on each line in L. Finally, the parts of T not in L_0 (indicated by white nodes in Figure 2.1) are labeled at the very end. The adversary does not employ any further ϕ-edge by assigning to each such node the same label as the closest labeled node (for instance, the three white nodes on the bottom-right of Figure 2.1 are assigned the same label as the upper terminal node of Line 11). □

2.3 The optimal tree algorithm

In this section we describe a tree prediction algorithm that achieves, up to constant factors, the lower bound proven in the previous section even without knowing the cutsize budget K. Our algorithm, TREEOPT, predicts a

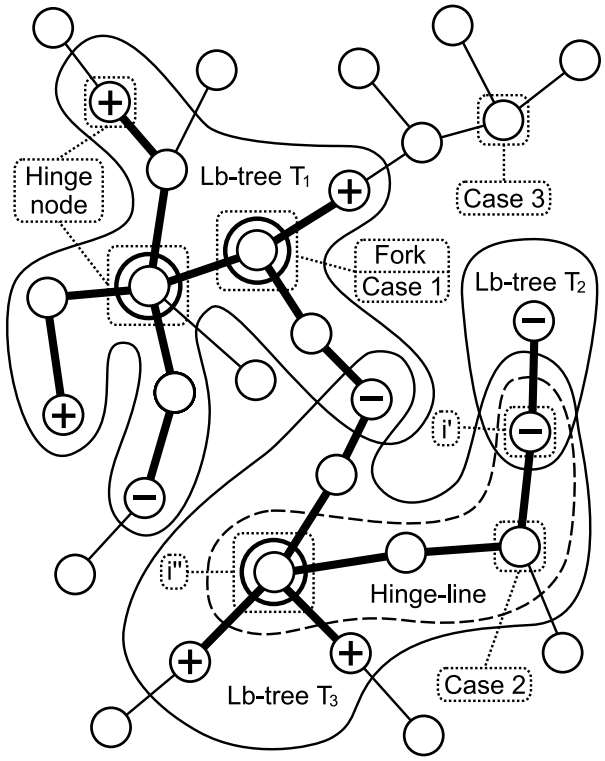

Figure 2.2: A tree T with 9 revealed labels inducing 3 lb-trees T_1, T_2, and T_3. Fork nodes are denoted by double circles. T_1 has two forks, T_2 has none, T_3 has one. The outer white nodes do not belong to any lb-tree. This figure also explains the behavior of TREEOPT by illustrating examples of the three itemized cases (1, 2, and 3 in the box), depending on the position of the node i_t to be predicted. For instance, in Case 2, TREEOPT determines i' and i'' as indicated, computes $\widetilde{y}_{i'}(t) = -1$, and $\widetilde{y}_{i''}(t) = 0$ (after running the fork label estimation procedure on i''), and then predicts $\widehat{y}_{i_t} = -1$ with rule 2.b.

node with the label minimizing the cutsize consistent with all labels seen so far. If this label is not unique, then the algorithm predicts using a nearest neighbor method. As we show in Section 2.4.1, TREEOPT can be viewed as an approximate and efficient implementation of the Halving algorithm for trees.

We say that a label (or node) is **revealed** at time t if the adversary already selected that node (thus causing its label to be observed by the algorithm). At any time step, the set of revealed labels defines a collection of edge-disjoint subtrees of T, which we call label-bordered trees (or lb-trees, for

short). Formally, given a labeled tree $(T, \mathbf{y})$ with revealed labels $y_{i_1}, \ldots, y_{i_t}$, an **lb-tree** is any maximal subtree of T whose leaves are all revealed and no internal node is. Clearly, a non-revealed node can belong to at most one lb-tree. A **fork** node is any node of an lb-tree T' having degree greater than two in T'. Figure 2.2 gives an example. Note that the set of lb-trees, together with their fork nodes, depends on the set of revealed labels, and is therefore changing with time. For brevity, call a node that is either a fork or a revealed node a **hinge node**. Also, call **hinge line** any line whose terminals are hinge nodes, and such that no internal node is a hinge node. Given a hinge node i, we compute its *estimated label* in such a way that the cutsize of T given the past revealed labels is minimized. The procedure for computing this estimate, called *Fork Label Estimation Procedure* (FLEP), is the core of our algorithm. When there is no unique minimizing label, the procedure assigns the fork a value of 0 ("undecided"), rather than $+1$ or -1. Let $\tilde{y}_i(t)$ be the label of i estimated by FLEP at time t. If i is revealed at time t then $\tilde{y}_i(t) = y_i$. Otherwise, $\tilde{y}_i(t)$ is computed as follows: Let T' be the (unique) lb-tree i belongs to. FLEP performs a depth-first visit of T' rooted at i. The visit starts at i and, when backtracking to a node j after all the children of j have been visited, FLEP assigns a *temporary label* to j given by the majority vote among the temporary or revealed labels of its children. Note that temporary labels set to 0 do not influence this vote. If the vote is a tie, i.e., the sum over all involved labels is 0, then the temporary label of j is set to 0, too. Once all nodes of T' have been visited (and the visit is back to node i) FLEP returns the temporary label $\tilde{y}_i(t)$ assigned to i. Figure 2.3 gives an example.

In the box is the pseudocode of our algorithm. This algorithm takes in input a tree T, a set $y_{i_1}, \ldots, y_{i_{t-1}}$ of revealed labels, and a node i_t to be predicted. The algorithm then returns its prediction $\widehat{y}_{i_t}$ for the label of i_t. In particular, if i_t is a fork node inside some lb-tree (Case 1), then TREEOPT just outputs the label $\tilde{y}_{i_t}(t)$ returned by FLEP, unless FLEP returns 0. In this latter case TREEOPT outputs the default value -1. On the other hand, if i_t is not a fork, but it is still contained in some lb-tree (Case 2), then the algorithm determines the opposite hinge nodes (i' and i'') closest to i_t, computes (again through FLEP) estimated values $\tilde{y}_{i'}(t)$ and $\tilde{y}_{i''}(t)$, and uses these values to compute its prediction. If i_t lies between nodes with estimated (or revealed) labels $+1$ and -1 (Case 2.d) then TREEOPT returns

Algorithm TREEOPT
Parameters : Tree T, revealed node labels $y_{i_1}, \ldots, y_{i_{t-1}}$, selected node i_t.

1. **If** i_t is a fork in an lb-tree T' then:

$$\widehat{y}_{i_t} \leftarrow \begin{cases} \widetilde{y}_{i_t}(t) & \text{if } \widetilde{y}_{i_t}(t) \neq 0 \quad \textbf{[1.a]} \\ -1 & \text{otherwise} \quad \textbf{[1.b]} \end{cases}$$

2. **Else if** i_t is contained in a lb-tree T' but it is not a fork then:

   - Let i' be the closest hinge node to i_t in T';
   - Let i'' be the second closest hinge node to i_t in T' such that the paths connecting i' and i'' to i have no edges in common (i'' always exists);

$$\widehat{y}_{i_t} \leftarrow \begin{cases} +1 & \text{if } \widetilde{y}_{i'}(t) + \widetilde{y}_{i''}(t) \geq 1 \quad \textbf{[2.a]} \\ -1 & \text{if } \widetilde{y}_{i'}(t) + \widetilde{y}_{i''}(t) \leq -1 \quad \textbf{[2.b]} \\ -1 & \text{if } \widetilde{y}_{i'}(t) = \widetilde{y}_{i''}(t) = 0 \quad \textbf{[2.c]} \\ \widetilde{y}_{i'}(t) & \text{otherwise} \quad \textbf{[2.d]} \\ & (\text{i.e. } \widetilde{y}_{i'}(t)\widetilde{y}_{i''}(t) = -1) \end{cases}$$

3. **Else** (i_t is not contained in any lb-tree)

   - Let s be the closest node to i_t in an lb-tree

   [3.a] If y_s is revealed at time t then $\widehat{y}_{i_t} \leftarrow y_s$

   [3.b] Else recursively call TREEOPT with parameters T, $y_{i_1}, \ldots, y_{i_{t-1}}$, and s.
   Obtain $\widehat{y}_s$ and set $\widehat{y}_{i_t} \leftarrow \widehat{y}_s$.

the label of the closer node. Finally, if i_t is not contained in any lb-tree (Case 3), the algorithm determines the closest node s inside some lb-tree, and then either predicts through the label of s (if y_s is revealed) or acts as if s were the label to be predicted at time t (i.e., TREEOPT recursively[2] invokes itself

[2]Note that after the recursive call, TREEOPT will not recur any more in that time step,

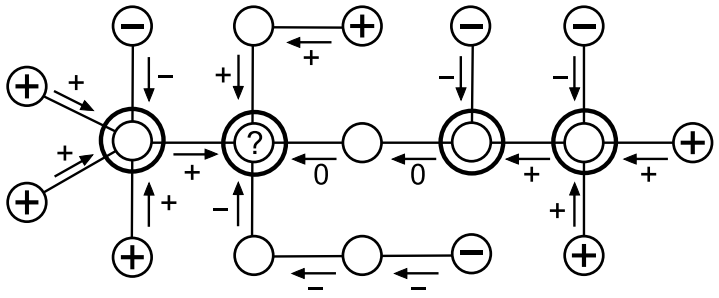

Figure 2.3: Fork label estimation procedure (FLEP) within the displayed lb-tree. The question mark indicates the fork node whose label has to be estimated. The arrows indicate the backtracking steps of the depth-first visit, where the majority vote (the arrow tags) among the temporary (or revealed) child labels is calculated. For instance, the right-most fork node receives two $+1$ and one -1 from its three incoming neighbors, and thus sends $+1$ to its left. The second fork node from the right receives one $+1$ and one -1, thereby sending out 0. The fork node tagged with "?" is estimated $+1$ (note that 0 is immaterial for the majority vote).

with $i_t = s$).

Figure 2.2 contains examples of the algorithm functioning. Note that TREEOPT reduces to a standard 1-Nearest Neighbor algorithm when the tree T is a line graph (namely, when fork nodes are absent).

2.4 Mistake bound analysis

This section contains the analysis of TREEOPT. We will prove the algorithm is optimal up to (multiplicative) constant factors.

The following simple property of the function LB is of primary importance.

Lemma 2.1 *For any tree* T *with* n *nodes and for any* $1 \leq K \leq K' < n$*, we have* $\mathrm{LB}(T, K') \leq \frac{K'}{K}\mathrm{LB}(T, K)$.

PROOF. Let L' be a blanket of size K' over T achieving the maximum in the definition of LB. Let L be the subset of L' obtained by keeping only the K

since rule 3.b will subsequently rely on rules 1 or 2 only.

longest lines in L'. Since L' is a blanket, so is L. By definition of LB,

$$\sum_{\ell \in L} m_\ell \leq \max_{L \in \mathcal{L}(T,K)} \sum_{\ell \in L} m_\ell = \textsc{lb}(T,K). \tag{2.1}$$

Besides, since L contains the K longest lines in L', for any $\ell' \in L' \setminus L$ we can write

$$m_{\ell'} \leq \frac{1}{K} \sum_{\ell \in L} m_\ell \leq \frac{\textsc{lb}(T,K)}{K} \ . \tag{2.2}$$

Hence

$$\begin{aligned} \textsc{lb}(T,K') &= \sum_{\ell' \in L'} m_{\ell'} = \sum_{\ell' \in L} m_{\ell'} + \sum_{\ell' \in L' \setminus L} m_{\ell'} \\ &\leq \textsc{lb}(T,K) + \frac{K'-K}{K} \textsc{lb}(T,K) = \frac{K'}{K} \textsc{lb}(T,K) \end{aligned}$$

the inequality following from (2.1) and (2.2). □

At a high level, the proof of optimality hinges on showing $m(\textsc{treeOpt}, T, \leq K) = \mathcal{O}\Big(\textsc{ub}(T,K) + K\Big)$, where UB is a function that bounds the number of mistakes made by TREEOPT in terms of a size-$\mathcal{O}(K)$ blanket $L(T, \mathbf{y})$ over T. This blanket is obtained by dividing T into subparts, and then by mapping each subpart to a set of lines. The union of these lines forms the blanket. Then we show that $\textsc{ub}(T,K) \leq K + 1 + \textsc{lb}(T, \mathcal{O}(K))$. Since by Lemma 2.1 we have $\textsc{lb}(T, \mathcal{O}(K)) = \mathcal{O}\Big(\textsc{lb}(T,K)\Big)$, and $K \leq \textsc{lb}(T,K)$ holds by definition of LB, we immediately get $m(\textsc{treeOpt}, T, \leq K) = \mathcal{O}\Big(\textsc{lb}(T,K)\Big)$. Combined with Theorem 3.1, this implies the optimality condition $m(\textsc{treeOpt}, T, \leq K) = \mathcal{O}\Big(\textsc{opt}(T,K)\Big)$.

The proof is a bit involved and requires us to step through several auxiliary definitions and intermediate results. We first introduce some notions related to the (inner and outer) structure of a cluster.

The blanket L over T used in the proof is the union of sets L_C of edge-disjoint lines from each cluster C. The sets L_C will be defined later on.

Let C be a (non-degenerate) cluster containing at least two nodes. We will define a covering $\mathcal{P}(C)$ of the nodes of cluster C —i.e., each node of C belongs to at least one subset in $\mathcal{P}(C)$. Then, we will construct a mapping f that, for each C, bijectively associates elements of $\mathcal{P}(C)$ with elements of subsets of lines in L_C, in such a way that for any non-degenerate C the number of mistakes TREEOPT makes on each element $Q \in \mathcal{P}(C)$ is $\mathcal{O}\Big(\sum_{\ell \in f(Q)} m_\ell\Big)$. If

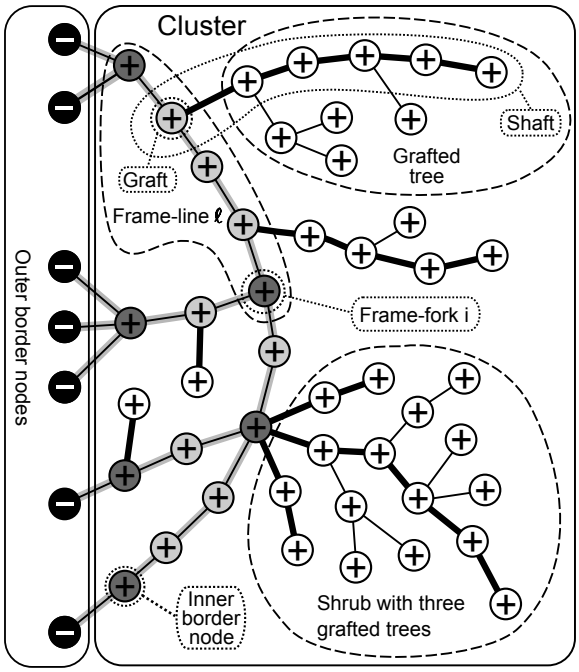

Figure 2.4: Cluster structure. All nodes within the cluster are labeled $+1$. The displayed cluster C has 7 outer border nodes (hence $\Phi_C = 7$) and 4 inner border nodes. The frame F_C is made up of all dark or light shaded (and $+1$-labeled) nodes. Dark shaded nodes are either frame-forks or inner border nodes (i.e., the terminal nodes of a frame-line). The gray shaded edges indicate paths connecting pairs of outer border nodes, identifying the cluster frame. The tagged frame-fork i has $d_i = 3$. Thick black edges identify the shaft of each grafted tree. The shaft $\sigma(\ell)$ contoured by dotted lines is associated with the tagged frame-line ℓ. Examples of grafted shrubs are also displayed.

C is a degenerate cluster (i.e., C contains only one node), then f will not be defined.

In order to define f over non-degenerate clusters, we need to introduce a specific cluster structure terminology. See Figure 2.4 for reference.

Definition 2.1 *Let a cluster C of a labeled tree $(T, \mathbf{y})$ be given.*

- *The **outer border nodes** are all nodes of T not in C that are adjacent to (exactly) one node of C. We denote by Φ_C the number of outer border nodes of cluster C, i.e., the number of ϕ-edges connecting nodes in C to the outside.*

- *The* ***inner border nodes*** *are all nodes of* C *that are adjacent to at least an outer border node of* C*.*

- *The* ***frame*** F_C *is the subtree of* C *whose nodes are on a path connecting any two outer border nodes. We denote by* d_i *the maximum number of edge-disjoint paths connecting* i *with outer border nodes.*

- *A* ***frame-fork*** *is a node* i *of* F_C *such that* $d_i \geq 3$*.*

- *A* ***frame-line*** *is a line* $\ell \subseteq F_C$ *where each terminal node is either a frame-fork or an inner border node, and such that no internal node of* ℓ *is a frame-fork. Notice that* $d_i = 2$ *for all internal nodes* i *of* ℓ*.*

- *A tree* ***grafted on a frame*** *is*[3] *any connected component of* C *that remains after deleting all nodes of the frame* F_C *and all edges incident to them. Notice that* $d_i = 1$ *for all nodes* i *of such trees. One can define, more generally, a tree grafted on a subtree* T' *in a similar way.*

- *A* ***graft*** *node* i *of* T *is any node of* F_C *adjacent to a node of a grafted tree* T'*; we will say that* T' *is grafted on* i*;*

- *A* ***grafted shrub*** *is a set of one or more trees grafted on the same node;*

- *The* ***shaft*** *of a grafted tree* T'*, denoted by* $\sigma(T')$*, is the line connecting the graft node* i *of* T' *to the farthest node in* T'*. We define* $\sigma(\ell)$ *to be the shaft of maximal length among all trees grafted on internal nodes of a frame-line* ℓ*. Moreover, for any shrub* S *grafted on a node* i*, we define* $\sigma(S)$ *to be the set containing the*[4] $d_i + 1$ *longest shafts of trees in* S*.*

[3]The reader might expect a grafted line, as defined in Section 2.2, be a special case of a grafted tree. In fact, the two definitions are slightly divergent, in the sense that the former includes the graft node, while the latter does not. For the sake of presentation, we find it more convenient here to keep the graft node out of the grafted tree.

[4]Obviously, the number of shafts in $\sigma(S)$ will actually be $\min\{|S|, d_i + 1\}$

We now define $\mathcal{P}(C)$ for each cluster C, and the bijective mapping f from $\bigcup_C \mathcal{P}(C)$ to the set of all lines of T. More precisely, f maps each $Q \in \mathcal{P}(C)$ to a subset of lines in C.

A subset Q of nodes in C belongs to $\mathcal{P}(C)$ if and only if one of the following two cases is true:

1. Q is the set of nodes of a frame-line ℓ, together with all the nodes of shrubs grafted on internal nodes of ℓ;

2. Q is the set of nodes of a shrub grafted on either a frame-fork or an inner border node in C, together with the graft node.

For sets Q of type 1 we define $f(Q) = \{\ell, \sigma(\ell)\}$. For sets Q of type 2 we define $f(Q) = \sigma(S)$. Now, if we extend the mapping f by viewing it as defined over $\bigcup C$ (the union including non-degenerate clusters only) one can easily verify its bijectivity: In fact, for any cluster C and any $Q \in \mathcal{P}(C)$, the set of nodes contained in a line $\ell \in f(Q)$ is a subset of Q only. Let $\mathcal{C}_1 = \mathcal{C}_1(T, \mathbf{y})$ be the subset of degenerate (singleton) clusters. Given a labeled tree $(T, \mathbf{y})$ with cluster set $\mathcal{C} = \mathcal{C}(T, \mathbf{y})$, define

$$L(T, \mathbf{y}) = \bigcup_{C \in \mathcal{C} \setminus \mathcal{C}_1} L_C = \bigcup_{C \in \mathcal{C} \setminus \mathcal{C}_1} \bigcup_{Q \in \mathcal{P}(C)} f(Q) \ . \tag{2.3}$$

Note that $L = L(T, \mathbf{y})$ is a union of lines that do not contain ϕ-edges. If we add to L all ϕ-edges of T we obtain a set of edge-disjoint lines whose only grafted lines are the shafts. Those, in turn, share with the other lines the graft nodes only. Hence this augmented set of lines is a connected blanket, implying that the original L is indeed a blanket over T. We show below (proof of Theorem 2.2) that $|L| = \mathcal{O}(K)$, being K the maximum cutsize of $(T, \mathbf{y})$.

The following sequence of lemmas, show the announced key property of mapping f, as related to the behavior of TREEOPT. Namely, for any non-degenerate C, TREEOPT makes on each element $Q \in \mathcal{P}(C)$ at most $\mathcal{O}\left(\sum_{\ell \in f(Q)} m_\ell\right)$ mistakes. For this purpose, we find it convenient to introduce the function UB ("upper bound"):

$$\text{UB}(T, K) = \max_{\mathbf{y} \in \mathcal{Y}(T,K)} \left(\left| \mathcal{C}_1(T, \mathbf{y}) \right| + \sum_{\ell \in L(T,\mathbf{y})} m_\ell \right).$$

$\text{UB}(T, K)$ will be shown to be an upper bound (up to a constant factor) on $m(\text{TREEOPT}, T, \leq K)$.

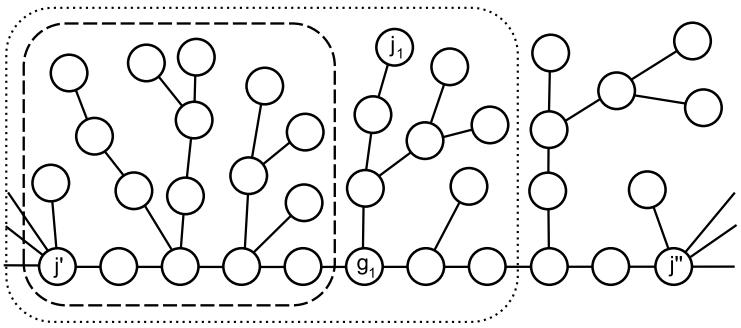

Figure 2.5: A line ℓ with terminal nodes j' and j'' and grafted trees above. Dichotomic behavior: after j' is revealed, the algorithm makes no more mistakes on the nodes in the dashed rectangle (ties are broken as in prediction rule 1.b). After j_1 is revealed, the algorithm makes no more mistakes on the nodes in the dotted rectangle, etc. Node g_1 is the graft node involved in Case 3.b of TREEOPT when predicting the label of j_1.

Lemma 2.2 *Let* C *be a cluster and* $\ell \subseteq \ell'$ *be a sub-line of some frame-line* $\ell' \in F_C$*. Assume at time* t *one of the two terminal nodes of* ℓ *are revealed. Then after time* t *the total number of mistakes made by* TREEOPT *on either internal nodes of* ℓ *or trees grafted on* ℓ *is bounded by* $\lfloor \log_2 |\ell| \rfloor \leq m_\ell$ *(see Figure 2.5 for reference).*

PROOF. Let j' be the terminal node of ℓ whose label is revealed, and j'' be the other terminal node. After time t, as soon as the algorithm makes the first mistake on a tree T' grafted on an internal node g of ℓ, the majority vote in the fork label estimation procedure ensures that the algorithm's prediction on g will be correct. Moreover the prediction rules 2.a and 3.a of TREEOPT ensure that no other mistake will be made in the whole shrub grafted on g. In both cases we have used the hypothesis that ℓ contains no frame-forks which could change the outcome of the majority vote. Since the four prediction rules 2.a–2.d of TREEOPT make no distinction between estimated fork labels and true (revealed) labels, for the purpose of this analysis a mistake made on g in ℓ is equivalent to a mistake made on a tree grafted on that node. These observations, combined with the prediction rule 2.c, imply the two following facts. Given a node r of ℓ, denote by $n(r)$ the closest hinge node to r on $\pi(r, j')$. Then:

1. Each node r of ℓ on which a mistake is made after time t satisfies $d(r, n(r)) \geq d(r, j'')$.
2. Let T' be a tree grafted on an internal node s of ℓ. A mistake can be made on T' only if $d(s, n(s)) \geq d(s, j'')$.

From the above, it is then easy to see that the number of internal nodes of ℓ on which the algorithm can make mistakes is at least halved after every new mistake. Since correct predictions on nodes of ℓ imply correct predictions on the whole shrub grafted on those nodes (see Figure 2.5), this halving process implies that total number of mistakes made after time t on internal nodes of ℓ, or on trees grafted on ℓ, is at most $\lfloor \log_2 |\ell| \rfloor \leq m_\ell$. □

The next three lemmas hold for any frame-line ℓ belonging to a cluster C of a labeled tree $(T, \mathbf{y})$.

Lemma 2.3 *The total number of mistakes* TREEOPT *makes on internal nodes of ℓ is at most $2m_\ell$.*

PROOF. As soon as the first node i gets revealed, line ℓ is split into the two sub-lines ℓ_1 and ℓ_2 sharing i as terminal node. By Lemma 2.2 the number of mistakes made on the internal nodes of ℓ is therefore bounded by $1 + \lfloor \log_2 |\ell_1| \rfloor + \lfloor \log_2 |\ell_2| \rfloor \leq 1 + \lfloor \log_2 |\ell_1||\ell_2| \rfloor \leq 1 + \lfloor 2 \log_2 |\ell| - 2 \rfloor \leq 2m_\ell$. □

Lemma 2.4 *The total number of trees grafted on ℓ on which* TREEOPT *makes mistakes is at most $2m_\ell + 1$.*

PROOF. As soon as three nodes of ℓ from three different trees grafted on ℓ have been revealed, on all subsequent time steps there will be at least one node i of ℓ with estimated label $\tilde{y}_i = +1$. As in Lemma 2.3, node i splits the line into two sub-lines, enabling the application of Lemma 2.2 to both. The proof is concluded by noting that the number of trees on which at least a mistake occurs must be bounded by the number of mistakes made on trees. □

Lemma 2.5 *There exists at most one tree grafted on ℓ where* TREEOPT *makes more than one mistake.*

PROOF. Let T_0 be the first tree grafted on ℓ where a mistake occurs. Call t the time step when this happens. Let $T_1 \neq T_0$ be another tree grafted on ℓ where a mistake occurs at some later time step $t' > t$. The majority vote of the fork label estimation procedure ensures that after time t' the graft node of T_1 will always be correctly estimated (if not yet revealed). Finally, the prediction rule 1.a implies that no additional mistakes will be made on T_1. This implies that T_0 is the only tree grafted on ℓ on which more than a mistake may occur. □

The next two lemmas bound the number of mistakes made on trees and shrubs grafted on the frame of a cluster C.

Lemma 2.6 *The number of mistakes made by* TREEOPT *on a tree* T_0 *grafted on the frame* F_C *of* C *is at most* $m_{\sigma(T_0)} + 1$.

PROOF. Let i_t be the first revealed node in T_0, g be the graft node of T_0, and g' be the node in T_0 adjacent to g. If we drop from T_0 both i_t and all edges incident to it, then T_0 gets decomposed into subtrees. Denote by T_1 the subtree containing g', and by $\mathcal{F}$ the forest of the remaining subtrees. After time t, any lb-tree created on connected components of $\mathcal{F}$ will be bordered by $+1$ nodes. Hence, either prediction rule 3.a or prediction rule 3.b combined with 2.a guarantee no more mistakes afterwards.

Let us now focus on nodes in T_1. We can invoke Lemma 2.2 (where j' and j'' in that proof are here i_t and g) even if the line ℓ connecting i_t to g is not part of any frame. Indeed, the result of Lemma 2.2 only depends on the fact that during the backtracking phase of the fork label estimation procedure, no temporary label -1 will be assigned to nodes grafted on that line. Thus the number of mistakes made after time t on nodes in T_1 is bounded by m_ℓ which, in turn, is bounded by $m_{\sigma(T_0)}$. This concludes the proof. □

Lemma 2.7 *The number of grafted trees of a shrub* S *grafted on* i *on which* TREEOPT *can make mistakes is at most* $d_i + 1$.

PROOF. If $|S| \leq d_i$ the claim is trivial. Hence, we continue by assuming $|S| > d_i$. Suppose that at least one mistake has been made on $d_i + 1$ trees grafted on i. If y_i is revelead at time t, then the prediction rules 3.b and 2.a ensure that no more mistakes will be made on S. On the contrary, if y_i is not

revelead, the majority vote in the fork label estimation procedure guarantees that y_i will always be correctly estimated in the future (i.e., $\tilde{y}_i(s) = y_i$ for any $s > t$), and the same prediction rules 3.b and 2.a guarantee no more mistakes. □

The next key lemma bounds the number of mistakes made on any element of $\mathcal{P}(C)$.

Lemma 2.8 *Let C be a non-degenerate cluster, $\mathcal{P}(C)$ be the corresponding covering. Let f be the bijective mapping defined above. Then the number of mistakes made by* TREEOPT *on any $Q \in \mathcal{P}(C)$ is bounded by $\mathcal{O}\left(\sum_{\ell \in f(Q)} m_\ell\right)$.*

PROOF. We first consider the case when Q is of type 1. In this case, the total number of mistakes made on Q can be simply bounded by summing: (i) the mistakes on the internal nodes of ℓ (Lemma 2.3); (ii) the mistakes on the tree grafted on ℓ on which the algorithm can make more than one mistake (Lemma 2.5 and Lemma 2.6); (iii) the number of the remaining trees grafted on ℓ where the algorithm can make at most one mistake (Lemma 2.4 and Lemma 2.5). Putting together, the total number of mistakes made on Q can be bounded by $\mathcal{O}\left(\sum_{\ell \in f(Q)} m_\ell\right)$.

Let us now consider a subset Q of type 2, and let S be the shrub referred to in the definition of such Q. By Lemma 2.6, Lemma 2.7, and the definition of $\sigma(S)$, the total number of mistakes made on Q can be bounded as

$$\sum_{\ell \in \sigma(S)} (m_\ell + 1) = \sum_{\ell \in f(Q)} (m_\ell + 1)\,,$$

which is again $\mathcal{O}\left(\sum_{\ell \in f(Q)} m_\ell\right)$. □

Before proving the main result of this section, we need one more lemma establishing a key property of the function UB.

Lemma 2.9 *For all $K = 1, \ldots, n-1$, the function* UB *satisfies* $\text{UB}(T, K-1) \leq \text{UB}(T, K) + 1$.

PROOF. Fix T. Any labeling $\mathbf{y}$ of T with cutsize $K-1$ can always be obtained from a labeling $\mathbf{y}'$ with cutsize K by merging two clusters C_1 and C_2. After this merge, $L(T, \mathbf{y})$ contains at most a new line that was not already

in $L(T, \mathbf{y}')$. This new line ℓ is the ϕ-edge deleted in the merge. Since $|\ell| = 1$, $\textsc{ub}(T, K-1) \leq \textsc{ub}(T, K) + m_\ell = \textsc{ub}(T, K) + 1$. $\square$

Theorem 2.2 *For any tree* T *with* n *nodes and any cutsize budget* K, $m(\textsc{treeOpt}, T, \leq K) = \mathcal{O}\Big(\textsc{lb}(T, K)\Big)$.

PROOF. Pick any labeled tree $(T, \mathbf{y})$ and let $k \leq K$ be its cutsize. Let $L = L(T, \mathbf{y})$ be the blanket (2.3). Pick a non-degenerate cluster $C \in \mathcal{C}(T, \mathbf{y})$. Let T_C be the tree obtained by augmenting the frame F_C with the Φ_C outer border nodes of C as leaves (referring to Figure 2.4, the resulting T_C is the tree including all non-white nodes). Then observe that the number of inner border nodes is $\mathcal{O}(\Phi_C)$. Since in any tree the number of nodes of degree larger than 2 cannot be greater than the number of leaves, the total number of frame-forks in C is also $\mathcal{O}(\Phi_C)$. Finally, since a frame-line in F_C is terminated by either a frame-fork or an inner border node, the total number of frame-lines is also $\mathcal{O}(\Phi_C)$. This can be seen by noting that collapsing each frame-line to a single edge turns F_C into a tree with a number of nodes linear in Φ_C. This tree then has $\mathcal{O}(\Phi_C)$ edges, which implies that the frame-lines in F_C are also $\mathcal{O}(\Phi_C)$. Now, by definition of f: (i) the number of lines in L_C deriving from subsets Q of type 1 is linear in the number of frame-lines in C; (ii) since the number of shafts having as terminal node an inner border node (i.e., a leaf of F_C) is linear in Φ_C, and the total number of remaining shafts (i.e., those grafted on frame-forks that are internal nodes of F_C) is linear in the number of frame-lines, the number of lines in L_C deriving from subsets Q of type 2 is $\mathcal{O}(\Phi_C)$. Therefore,

$$|L| = \sum_{C \in \mathcal{C} \setminus \mathcal{C}_1} |L_C| = \sum_{C \in \mathcal{C} \setminus \mathcal{C}_1} \mathcal{O}(\Phi_C) = \mathcal{O}(K) \ .$$

To finish the proof observe that, by Lemma 2.8 and by definition of f,

$$\begin{aligned} & m(\textsc{treeOpt}, T, \mathbf{y}) \\ & \quad \leq \Big|\mathcal{C}_1(T, \mathbf{y})\Big| + \mathcal{O}\left(\sum_{C \in \mathcal{C} \setminus \mathcal{C}_1} \sum_{Q \in \mathcal{P}(C)} \sum_{\ell \in f(Q)} m_\ell \right) \\ & \quad = \Big|\mathcal{C}_1(T, \mathbf{y})\Big| + \mathcal{O}\left(\sum_{\ell \in L(T, \mathbf{y})} m_\ell \right) \\ & \quad = \mathcal{O}\Big(\textsc{ub}(T, k)\Big) \end{aligned}$$

where $k \leq K$ is the cutsize of $\mathbf{y}$. Since the above holds for all labelings $\mathbf{y}$ of T with cutsize at most K,

$$\begin{aligned} m(\textsc{treeOpt}, T, \leq K) &= \mathcal{O}\Big(\max_{k=1,\dots,K} \textsc{ub}(T,k)\Big) \\ &= \mathcal{O}\left(\textsc{ub}(T,K) + K\right) \ , \end{aligned}$$

using Lemma 2.9 in the last step. Now, $\textsc{ub}$ is defined in terms of a specific blanket L, with $|L| = \mathcal{O}(K)$, and $\left|\mathcal{C}_1(T,\mathbf{y})\right| \leq K+1$ when $\mathbf{y}$ has cutsize bounded by K. These facts imply $\textsc{ub}(T,K) \leq K+1+\textsc{lb}\Big(T,\mathcal{O}(K)\Big)$. Finally, using Lemma 2.1 and $\textsc{lb}(T,K) \geq K$, we obtain $m(\textsc{treeOpt}, T, \leq K) = \mathcal{O}\Big(K + \textsc{lb}\Big(T,\mathcal{O}(K)\Big)\Big) = \mathcal{O}\Big(\textsc{lb}(T,K)\Big)$. $\square$

In order to compare the optimal bound achieved by our algorithm to the bounds of other methods present in the literature, we note that, for any given labeled tree $(T,\mathbf{y})$, our algorithm makes a number of prediction mistakes whose upper bound can be re-written as

$$\mathcal{O}\Big(\Phi_T(\mathbf{y})\,\overline{m}_\ell\Big) \tag{2.4}$$

where $\Phi_T(\mathbf{y})$ is the cutsize of $(T,\mathbf{y})$ and $\overline{m}_\ell$ is the average of m_ℓ over all lines ℓ in the blanket L of size $\Phi_T(\mathbf{y})$ maximizing $\sum_{\ell\in L} m_\ell$.

Note that $m_\ell < \log_2 D_T + \mathcal{O}(1)$ for all ℓ. Moreover, for many classes of trees T, if the cutsize is not too small then it is not even possible to find a blanket of size $\Phi_T(\mathbf{y})$ whose lines have average length linear in D_T. In these cases $\overline{m}_\ell$ can be much smaller than $\log D_T$. As for the time complexity, since $m(\textsc{treeOpt}, T, \leq K) \geq K$ and $m = m(A, T, \leq K) = \Omega\Big(m(\textsc{treeOpt}, T, \leq K)\Big)$ for any deterministic algorithm A, if the cutsize is $\Omega(\log D_T)$ our algorithm is faster than the one in [44], which predicts all labels in time $\Theta(n\,m)$. Note also that $\textsc{treeOpt}$ does not require any explicit (and costly) pre-computation. Moreover, unlike Perceptron-like algorithms which use $n \times n$ matrices, the space required by $\textsc{treeOpt}$ is always linear in n.

2.4.1 Comparison to the Halving algorithm

We now compare $\textsc{treeOpt}$ to the so-called $\textsc{Halving}$ algorithm (applied to trees). This is a standard version space algorithm defined as follows. Let $\mathcal{Y}_t$

be the set of labelings $\mathbf{y} \in \{-1,+1\}^n$ consistent with all labels revelead up to time t. Define now $\mathcal{Y}_t^{\min} \subseteq \mathcal{Y}_t$ as those labelings with minimum cutsize in $\mathcal{Y}_t$. HALVING predicts the label of i_t with the value $y \in \{-1,+1\}$ that maximizes $\left|\left\{\mathbf{u} \in \mathcal{Y}_{t-1}^{\min} : u_{i_t} = y\right\}\right|$. For example, if assigning a certain label to i_t increases the current cutsize (irrespective to the value of the remaining non-revealed labels), then HALVING always predicts the opposite label, i.e., the cut-minimizing label.

Proving tight mistake bounds for HALVING is in general not straightforward. As a simple example, the best bound for HALVING on a star graph with n nodes and cutsize $K < n/2$ is $\mathcal{O}(K)$. This in contrast with the more intuitive "version space bound" $\mathcal{O}\big(K \log(n/K)\big)$ one might think of at first glance. In this section, we prove the optimality of HALVING (up to constant factors), but because of the very difficult combinatorics involved, we do so only indirectly, by exploiting the optimality of TREEOPT.

The following lemma shows that when the fork label estimation procedure (FLEP) of TREEOPT returns a nondefault value (as in prediction rule 1.a), then this value is the same cut-minimizing label predicted by HALVING.

Lemma 2.10 *Let $\tilde{y}_r(t)$ be the value returned by* FLEP *run by* TREEOPT *at time t to evaluate the label of node r. If $\tilde{y}_r(t) \neq 0$ then $u_r = \tilde{y}_r(t)$ for each $\mathbf{u} \in \mathcal{Y}_t^{\min}$.*

PROOF. Let T_r be the lb-tree rooted at r at time t. Recall that the fork estimation procedure works by assigning temporary labels while backtracking in the depth-first visit of T_r. We prove the following claim: each temporary label $y'_i(t) \neq 0$ assigned to node i of T_r is such that the cutsize is at least as small as the cutsize when i is assigned label $-y'_i(t)$. The proof is by induction on the maximum distance between i and its descendants in T_r. When $y'_i(t) = 0$ we show that the cutsize-minimizing label for i is the same as i's parent. Finally, by applying the claim to the children of r, we obtain that the cutsize-minimizing label of r is the majority vote over the children's temporary (or revealed) labels. □

The same equivalence between TREEOPT and HALVING predictions holds in other cases, for instance when i_t does not belong to any lb-tree. In general, however, the predictions of the two algorithms may differ. Nevertheless, it is possible to prove that the number of nodes where the two predictions differ is small, as stated by the following theorem.

Theorem 2.3 *For any labeled tree* $(T, \mathbf{y})$ *with cutsize* K*, and any presentation* $i_1, \ldots, i_n$ *of the nodes of* T*, the number of times when* TREEOPT *and* HALVING *output a disagreeing prediction is bounded by* $\mathcal{O}(\text{LB}(T, K))$*.*

PROOF. The predictions of TREEOPT and HALVING differs only when: (i) TREEOPT estimates a fork as 0 (prediction rule 1.b); (ii) TREEOPT predicts a node between two forks estimated as 0 (prediction rule 2.c); (iii) Node i_t does not belong to any lb-tree and the closest node in a lb-tree is a fork estimated as 0 (prediction rule 3.b together with 1.b); (iv) i_t is on a hinge line whose terminal nodes i' and i'' are such that the label of i'' (estimated or revealed) is different from 0 and the label of i' is estimated as 0 (subcases in prediction rules 2.a and 2.b).

The nodes in which cases (i) to (iii) may occur are easily seen to be $\mathcal{O}(K)$. In case (iv) the two predictions differs only when i_t is closer to i'. This fact makes it possible to find a size-$\mathcal{O}(K)$ blanket L such that the number of disagreeing predictions is $\mathcal{O}\left(\sum_{\ell \in L} m_\ell\right)$. □

Theorem 2.3 implies that TREEOPT approximates HALVING, the two algorithms making the same number of mistakes up to constant factors. A close examination of the two algorithms reveals that when TREEOPT predicts a default value, HALVING apparently needs to perform a certain amount of computation. In this respect, we can view TREEOPT as a "lighter" implementation of HALVING. In fact, in the next section we show that TREEOPT can be implemented in a quite efficient manner.

2.5 Efficient implementation

A naive implementation of TREEOPT requires space linear in the total number n of nodes. It is also easy to check that predicting a single label requires time $\mathcal{O}(n)$, since each lb-tree has $\mathcal{O}(n)$ nodes. In this section we describe a more sophisticated implementation that improves significantly the amortized time per time step, while still using space linear in n.

Theorem 2.4 *The total time* TREEOPT *requires to predict all labels of a labeld tree* $(T, \mathbf{y})$ *with* n *nodes is*

$$\mathcal{O}\left(\min\{n_f, K\}\, K + n \log D_T\right)$$

where K *is the cutsize of* $(T, \mathbf{y})$, n_f *is the number of internal nodes of* T *with degree greater than* 2, *and* D_T *is the diameter of* T.

Note that whenever $K = \mathcal{O}\left(\sqrt{n}\right)$, the amortized time per step is at most logarithmic in the diameter[5] of T. In order to achieve this speed up, we maintain the following data structures (see Figure 2.6).

Signals and signal values. We store extra links connecting neighboring hinge nodes so as to avoid running the depth-first visit involved in FLEP. For each hinge line ℓ with terminal nodes i and j we store an extra directed link $[i \rightarrow j]$ connecting i to j, and a second one $[j \rightarrow i]$ connecting j to i. We call these links *signals*. All signals of the form $[i \rightarrow j]$ are stored together with node i. Each signal $[i \rightarrow j]$ is linked to its twin $[j \rightarrow i]$ and to the node adjacent to i in ℓ. Hence, when traversing ℓ for predicting with rule 2, it is possibile to find both signals associated with ℓ in constant time just after reaching one of the two terminal nodes. Each signal $[i \rightarrow j]$ has a value $\nu([i \rightarrow j]) \in \{-1, 0, 1, \square\}$. If j is a frame-fork, $\nu([i \rightarrow j])$ is equal to the temporary label that FLEP would assign to i when estimating y_j. In the special case when y_i is already revealed and j is a fork node, $\nu([i \rightarrow j])$ is simply equal to y_i. Finally, if y_j is revealed then $\nu([i \rightarrow j])$ is equal to $\square$, and we say that the signal is *empty*. Recall that, in order to return a label for the fork node j, FLEP assigns temporary labels to each internal node of the hinge line connecting i to j. These labels are $\nu([i \rightarrow j])$.

Fork values. We associate with each fork i a numerical value ν_i given by the sum of the temporary or revelead labels of its children in the lb-tree rooted at i. Observe that FLEP always returns $\operatorname{sgn}(\nu_i)$ as the value of a fork label y_i (where we define $\operatorname{sgn}(0) = 0$). Moreover, ν_i is equal to $\sum_{j \in N(i)} \nu([j \rightarrow i])$ where $N(i)$ is the set of hinge nodes j such that i is linked to j via a signal; note also that $\nu([i \rightarrow j]) = \operatorname{sgn}(\nu_i - \nu([j \rightarrow i]))$ for each signal $[i \rightarrow j]$ where i and j are both forks.

Other auxiliary structures. By means of an initial depth-first visit of T, we associate with each edge $(i, j) \in E$ a direction given by the relationship

[5] Though we do not prove it here, the above computational bound can be further refined by replacing $\log D_T$ with a smaller structural parameter (independent of K). For some trees the value of this parameter can be constant even when $\log D_T = \Theta(\log n)$.

child $\rightarrow$ parent in the tree T rooted at node i_1, i.e., the node whose label is revealed at the end of the first time step. Starting from any node i not contained in any lb-tree, it is then possibile to find the nearest node j belonging to an lb-tree in time linear in the distance between i and j by simply following these edge directions. We associate with each pair of adjacent nodes i and j in any given hinge line ℓ an extra directed link $[i, j]$, along with its twin link $[j, i]$. These links are useful when traversing ℓ. Each node has a mark that allows the algorithm to know whether the node belongs to an lb-tree, or if it is a fork node or whether its label has been revealed or not.

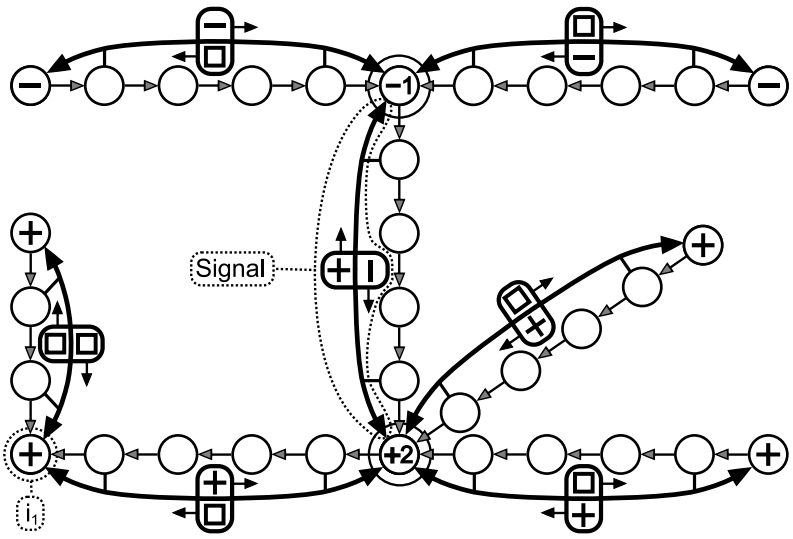

Figure 2.6: Two lb-trees with the main auxiliary data-structures for the efficient implementation. The numbers inside the fork nodes (the two doubly-circled nodes) indicate the fork values v_i. Node i_1 is located at the bottom-left. The gray arrows, directed towards i_1, are aimed at supporting a quick implementation of prediction rule 3 of TREEOPT when finding the nearest node contained in an lb-tree. The bidirectional black arrows denote signals exchanged between pairs of terminal nodes of hinge lines.

We now describe the key concept of *signal change propagation*. Suppose that a signal $[i \rightarrow j]$ changes its value in such a way that $v([i \rightarrow j]) \neq \square$ both before and after the signal modification. This modifies the value v_j which, in turn, may affect the values of some signals departing from j. Therefore, any signal modification can propagate through the signal links in the lb-tree. It

is important to observe that an increase (decrease) of signal value $\nu([i \to j])$ will not propagate if, before the change, $\nu_j \geq 2$ ($\nu_j \leq -2$) (all values of outgoing non-empty signal will remain equal to $\mathrm{sgn}(\nu_j)$).

We continue by describing how the algorithm uses and updates the auxiliary structures when predicting node i_t. The reader is referred to the three prediction rules in the pseudocode of TREEOPT.

1) i_t is a fork. The algorithm predicts with $\mathrm{sgn}(\nu_i)$ (or -1 if $\nu_i = 0$), sets the value of all signals incoming to i equal to $\square$ and that of all signals outgoing from i equal to y_i, propagating them if necessary.

2) i_t is contained in an lb-tree but it is not a fork. Let i' and i'' be defined as in prediction rule 2. The algorithm finds the nearest hinge node i' by traversing the hinge line in both directions (using a breadth-first visit on that line). Then it uses the signs of $\nu_{i'}$ and $\nu_{i''}$ for predicting with rule 2, creates the signals $[i_t \to i']$ and $[i_t \to i'']$, and propagates them if necessary. Finally, the algorithm replaces the two old signals linking i' to i'' with $[i' \to i_t]$ and $[i'' \to i_t]$, and sets both values to $\square$.

3) i_t is not contained in any lb-tree. The algorithms finds the nearest node s contained in an lb-tree using the extra-links directed towards i_1 and creates the auxiliary information for the new hinge line connecting i_t to s. Then the algorithm predicts as if the adversary had asked for label y_s, and creates the signals $[i \to j]$ and $[j \to i]$. If j is not a hinge node, then a new signal is created. This signal is updated and propagated analogously to the previous case.

The next lemma is useful for the complexity analysis. First of all, we define a *phase* to be a maximal non-empty interval of time steps where no label revelation increases the minimal cutsize consistent with the labels seen so far. Hence a time step where the current minimal cutsize increases does not belong to any phase.

Lemma 2.11 *Let t belong to a phase and let $\nu_i(t)$ be the value of a fork node i at the beginning of time t. If y_i is not revelead at time t, then $\nu_i(t+1) \geq \nu_i(t)$ if $\nu_i(t) > 0$, and $\nu_i(t+1) \leq \nu_i(t)$ if $\nu_i(t) < 0$.*

PROOF. Considering without loss of generality that $\mathrm{sgn}\big(\nu_i(t)\big) = +1$, it is enough to prove that if $\nu_i(t+1) \geq \nu_i(t)$. For each hinge node j connected

with i by a non-negative signal, we have $\nu_j(t) \geq 1$ (or y_j has been revealed before time t). By Lemma 2.10, $\nu_i(t+1)$ can be smaller than $\nu_i(t)$ only if a non-negative numerical signal $[j \to i]$ decreases its value in the signal change propagation at time t, which implies the decrease of ν_j. Hence, decreasing a positive fork value implies a previous decrease of another positive fork value in the same change signal propagation, which concludes the proof. □

We can now give the proof of the worst case time bound for predicting the labels in T.

PROOF OF THEOREM 2.4. Each internal node i of a hinge line ℓ can be visited only $\mathcal{O}\big(\log|\ell|\big) = \mathcal{O}(\log D_T)$ times through prediction rule 2. As a matter of fact, for each of the two traversing directions, the distance between i and the node from which the breadth-first visit over ℓ starts is at least halved each time i gets visited. This fact accounts for the $\mathcal{O}(n \log D_T)$ term in the bound.

Now observe that a node with degree smaller than 3 can never become a fork. Moreover, the number of forks involved in a signal propagation process in each tree grafted on a cluster frame is constant. The number of trees grafted on a frame-line ℓ on which a signal change can propagate is again constant. For each shrub S grafted on a node i, the number of trees of S involved in the signal propagation is $\mathcal{O}(d_i)$. Lemma 2.11 applied to each fork j, together with these observations, allows us to deduce that in a single phase the signal propagation process takes time $\mathcal{O}\big(\min\{n_f, K\}\big)$. This is also the time required by a signal propagation in each step where the minimal cutsize gets increased. Finally, the number of phases is equal to $\mathcal{O}(K)$.

The proof is concluded by considering that the total time required for creating and emptying all signals, as well as for creating the other auxiliary structures, is linear in n. □

2.6 Application to labeled graph prediction

We now discuss the application of our tree prediction algorithm to the general problem of predicting the labels of an undirected graph, and compare our results to the existing literature. As mentioned in the introduction of this

chapter, when given a graph $G = (V, E)$ with n nodes and arbitrary binary labels $\mathbf{y}$, we suggest running TREEOPT on a (uniformly generated) random spanning tree of G.

The expected cutsize of T is the resistance-weighted cutsize of G $\Phi_R(\mathbf{y}) = \frac{1}{4}\sum_{(i,j)\in E} r_{i,j}(y_i - y_j)^2$, which is significantly better than G's cutsize $\Phi_G(\mathbf{y})$ in most cases. In fact, on an unweighted graph with n nodes, the effective resistance $r_{i,j}$ of an edge (i,j) always lies in $[2/n, 1]$. In particular, $r_{i,j}$ is very small when (i,j) is located in a densely connected area of the graph, while $r_{i,j} = 1$ when (i,j) is a bridge edge. For instance, in a dense graph where $r_{i,j} = \mathcal{O}(1/n)$ for all $(i,j) \in E$, the adversary may choose $\mathbf{y}$ so as to concentrate $\Theta(n)$ ϕ-edges on any specific tree, and yet $\Phi_R(\mathbf{y}) = \mathcal{O}(1)$.

The above argument immediately leads to the following general result. Let TREEOPT+ be the (randomized) graph prediction algorithm that, on input G, first generates a random spanning tree T of G, and then runs TREEOPT on T. Define $(G, \mathbf{y})$ and $m(A, G, \mathbf{y})$ similarly to what we did for trees.

Corollary 2.1 *For any undirected labeled graph $(G, \mathbf{y})$, and for any presentation order $i_1, \ldots, i_n$ of the nodes of G, the expected (over the random choice of the spanning tree T) number of mistakes* TREEOPT+ *makes on $(G, \mathbf{y})$ is bounded as* $\mathbb{E}\big[m(\text{TREEOPT}+, G, \mathbf{y})\big] = \mathcal{O}\big(\Phi_R(\mathbf{y}) \log n\big)$.

PROOF. We have

$$\begin{aligned}\mathbb{E}\big[m(\text{TREEOPT}+, G, \mathbf{y})\big] &= \mathbb{E}\big[m(\text{TREEOPT}, T, \mathbf{y})\big] \\ &= \mathbb{E}\big[\mathcal{O}\big(\Phi_T(\mathbf{y})\overline{m}_\ell\big] = \mathcal{O}\big(\Phi_R(\mathbf{y}) \log n\big)\end{aligned}$$

where the second equality is (2.4), and the last one follows after (crudely) upper bounding $\overline{m}_\ell$ by $\log n$. □

Similar bounds could also be shown to hold with high probability, rather than in expectation, by exploiting known concentration properties of random spanning trees. See, e.g., [40] and references therein.

As mentioned in the introduction of this chapter, in Section 2.2 we demonstrate that *any* prediction algorithm can be forced to make a number of mistakes which is at least as big as the cutsize of the graph's random spanning tree. This lower bound, together with Corollary 2.1, clearly implies that TREEOPT is optimal (up to a $\mathcal{O}(\log |V|)$ factors) on *any* labeled graph.

The best mistake bound we know of for the general graph prediction problem has the form $\min_\rho\big(\mathcal{N}(G,\rho) + \Phi_G(\mathbf{y})\rho\big)$, where $\mathcal{N}(G,\rho)$ is the covering number of G in the resistance metric [46]. It is easy to see that this bound gets large when the diameter D_G is large. Moreover, real-world graphs G (such as parts of the web graph) have dense regions that can cause a large cutsize. In some of these cases, (take the lollipop graph as an extreme situation), it is just impossible to find a small-sized covering using balls of small radius. A uniformly generated random spanning tree T of G guarantees, instead, that the presence of dense parts of G will not dramatically increase the cutsize of T. Hence the use of TREEOPT on a random tree ensures an appealing (expected) mistake bound where the cutsize factor cannot get too large, except for degenerate and very irregular labelings.

Chapter 3

Learning on weighted trees and graphs

3.1 Introduction

This chapter focuses on the on-line transductive version of the node classification problem.

Although in the unweighted case previous studies use the cutsize to prove several interesting upper bounds [46, 47, 48], no general lower bounds on the number of prediction mistakes are known, leaving fully open the question of characterizing the complexity of learning a labeled graph. In Chapter ?? we bounded the expected number of mistakes by the cutsize of a random spanning tree of the graph, a quantity typically much smaller than the cutsize of the whole graph. In this chapter we show that this quantity captures the hardness of the graph learning problem. Given any weighted graph, we prove that any prediction algorithm must err on a number of nodes which is at least as big as the cutsize of the graph's random spanning tree (which is defined in terms on the graph's weights). Moreover, we exhibit a simple algorithm achieving the optimal mistake bound to within logarithmic factors on any sufficiently connected weighted graph whose weighted cutsize is not an overwhelming fraction of the total weight.

Following [22] (see Chapter 2), our algorithm first extracts a random spanning tree of the original graph. Then, it predicts all nodes of this tree using a generalization of the method proposed by [46]. Our tree prediction procedure is extremely efficient: it only requires *constant* amortized time per prediction and space *linear in the number of nodes*. Note that computational efficiency is a central issue in practical applications where the involved

datasets can be very large. In such contexts, learning algorithms whose time complexity scales, say, more than quadratically with the number of data points should be considered impractical.

A further significant contribution of this work is the experimental evaluation of our method, as compared to methods recently proposed in the literature on graph prediction. In particular, we compare our algorithm to the Perceptron algorithm with Laplacian kernel by [47, 48], and to the label propagation algorithm by [102], including its on-line version. The experiments have been carried out on four medium-sized real-world datasets. The two tree-based algorithms (ours and the Perceptron algorithm) have been tested using spanning trees generated in various ways. Our experimental comparison shows that our on-line algorithm compares well to all competitors while using, in most cases, the least amount of time and memory resources.

3.2 Preliminaries and basic notation

Let $E^\phi \subseteq E$ be the set of ϕ-edges in $(G, \mathbf{y})$. The *weighted* cutsize $\Phi_G^W(\mathbf{y})$ of $(G, \mathbf{y})$ is $\Phi_G^W(\mathbf{y}) = \sum_{(i,j)\in E^\phi} w_{i,j}$.

Fix $(G, \mathbf{y})$. For $(i, j) \in E$, let also $p_{i,j} = w_{i,j} r_{i,j}^W$ be the probability that (i, j) belongs to a random spanning tree T — see, Section 1.6 or, e.g., [63]. Then we have

$$\mathbb{E}\,\Phi_T(\mathbf{y}) = \sum_{(i,j)\in E^\phi} p_{i,j} = \sum_{(i,j)\in E^\phi} w_{i,j} r_{i,j}^W \tag{3.1}$$

where the expectation $\mathbb{E}$ is over the random choice of spanning tree T. Since $\sum_{(i,j)\in E} p_{i,j}$ is equal to $n - 1$, irrespective of the edge weighting, we have $0 \le \mathbb{E}\,\Phi_T(\mathbf{y}) \le n - 1$. Hence the ratio $\frac{1}{n-1}\mathbb{E}\,\Phi_T(\mathbf{y}) \in [0, 1]$ provides a *density-independent* measure of the cutsize in G, and even allows to compare labelings on different graphs. It is also important to note that $\mathbb{E}\,\Phi_T(\mathbf{y})$ can be much smaller than $\Phi_G^W(\mathbf{y})$ when there are strongly connected regions in G contributing prominently to the weighted cutsize. To see this, consider the following scenario: If $(i, j) \in E^\phi$ and $w_{i,j}$ is large, then (i, j) gives a big contribution to[1] $\Phi_G^W(\mathbf{y})$. However, this might not happen in $\mathbb{E}\,\Phi_T(\mathbf{y})$. In fact, if i and j are strongly connected (i.e., if there are many disjoint paths connecting them), then $r_{i,j}^W$ is very small and the terms $w_{i,j} r_{i,j}^W$ in (3.1) are small too. Therefore, the effect of the large weight $w_{i,j}$ may often be com-

[1] It is easy to see that in such cases $\Phi_G^W(\mathbf{y})$ can be much larger than n.

pensated by the small probability of including (i,j) in the random spanning tree.

3.3 A general lower bound

We start by stating a general lower bound, showing that any prediction algorithm must err at least $\frac{1}{2}\mathbb{E}\,\Phi_T(\mathbf{y})$ times on any weighted graph.

Theorem 3.1 *Let $G = (V, E, W)$ be a weighted undirected graph with n nodes and weights $w_{i,j} > 0$ for $(i,j) \in E$. Then for all $K \leq n$ there exists a randomized labeling $\mathbf{y}$ of G such that for all (deterministic or randomized) algorithms A, the expected number of prediction mistakes made by A is at least $K/2$, while $\mathbb{E}\,\Phi_T(\mathbf{y}) < K$.*

PROOF. The adversary uses the weighting P induced by W and defined by $p_{i,j} = w_{i,j} r_{i,j}^W$. By (1) $p_{i,j}$ is the probability that edge (i,j) belongs to a random spanning tree T of G. Let $P_i = \sum_j p_{i,j}$ be the sum over the induced weights of all edges incident to node i. We call P_i the *weight* of node i. Let $S \subseteq V$ be the set of K nodes i in G having the smallest weight P_i. The adversary assigns a random label to each node $i \in S$. This guarantees that, no matter what, the algorithm A will make on average $K/2$ mistakes on the nodes in S. The labels of the remaining nodes in $V \setminus S$ are set either all $+1$ or all -1, depending on which one of the two choices yields the smaller $\Phi_G^P(\mathbf{y})$. We now show that the weighted cutsize $\Phi_G^P(\mathbf{y})$ of this labeling $\mathbf{y}$ is less than K, *independent of* the labels of the nodes in S. Since the nodes in $V \setminus S$ have all the same label, the ϕ-edges induced by this labeling can only connect either two nodes in S or one node in S and one node in $V \setminus S$. Hence $\Phi_G^P(\mathbf{y}) = \Phi_G^{P,\mathrm{int}}(\mathbf{y}) + \Phi_G^{P,\mathrm{ext}}(\mathbf{y})$, where $\Phi_G^{P,\mathrm{int}}(\mathbf{y})$ is the cutsize contribution within S, and $\Phi_G^{P,\mathrm{ext}}(\mathbf{y})$ is the one from edges between S and $V \setminus S$. Let

$$P_S^{\mathrm{int}} = \sum_{(i,j)\in E\,:\,i,j\in S} p_{i,j}$$

and

$$P_S^{\mathrm{ext}} = \sum_{(i,j)\in E\,:\,i\in S, j\in V\setminus S} p_{i,j}\ .$$

From the very definition of P_S^{int} and $\Phi_G^{P,\mathrm{int}}(\mathbf{y})$ we have $\Phi_G^{P,\mathrm{int}}(\mathbf{y}) \leq P_S^{\mathrm{int}}$. Moreover, from the way the labels of nodes in $V \setminus S$ are selected, it follows

that $\Phi_G^{P,ext}(\mathbf{y}) \leq P_S^{ext}/2$. Finally,

$$\sum_{i \in S} P_i = 2P_S^{int} + P_S^{ext}$$

holds, since each edge connecting nodes in S is counted twice in the sum $\sum_{i \in S} P_i$. Putting everything together we obtain

$$\begin{aligned} 2P_S^{int} + P_S^{ext} &= \sum_{i \in S} P_i \leq \frac{K}{n} \sum_{i \in V} P_i \\ &= \frac{2K}{n} \sum_{(i,j) \in E} p_{i,j} \\ &= \frac{2K(n-1)}{n}, \end{aligned}$$

the inequality following from the definition of S. Hence

$$\begin{aligned} \mathbb{E}\, \Phi_T(\mathbf{y}) &= \Phi_G^{P}(\mathbf{y}) \\ &= \Phi_G^{P,int}(\mathbf{y}) + \Phi_G^{P,ext}(\mathbf{y}) \\ &\leq P_S^{int} + \frac{P_S^{ext}}{2} \\ &\leq \frac{K(n-1)}{n} < K\ , \end{aligned}$$

thereby concluding the proof. □

3.4 The Weighted Tree Algorithm

We now describe the Weighted Tree Algorithm (WTA) for predicting the labels of a weighted tree. In Section 3.5 we show how to apply WTA to the more general weighted graph prediction problem. WTA first turns the tree into a line graph (i.e., a list), then runs a fast nearest neighbor method to predict the labels of each node in the line. Though this technique is similar to that one used in [46], the fact that the tree is weighted makes the analysis significantly more difficult, and the practical scope of our algorithm significantly wider. Our experimental comparison in Section 3.7 confirms that exploiting the weight information is often beneficial in graph prediction.

Given a labeled weighted tree $(T, \mathbf{y})$, the algorithm initially creates a weighted line graph L' containing some duplicates of the nodes in T. Then,

each duplicate node (together with its incident edges) is replaced by a single edge with a suitably chosen weight. This results in the final weighted line graph L which is then used for prediction. In order to create L from T, WTA performs the following *tree linearization* steps:

1. An arbitrary node r of T is chosen, and a line L' containing only r is created.
2. Starting from r, a depth-first visit of T is performed. Each time an edge (i,j) is traversed (even in a backtracking step), the edge is appended to L' with its weight $w_{i,j}$, and j becomes the current terminal node of L'. Note that backtracking steps can create in L' at most one duplicate of each edge in T, while nodes in T may be duplicated several times in L'.
3. L' is traversed once, starting from terminal r. During this traversal, duplicate nodes are eliminated as soon as they are encountered. This works as follows. Let j be a duplicate node, and (j',j) and (j,j'') be the two incident edges. The two edges are replaced by a new edge (j',j'') having weight[2] $w_{j',j''} = \min\{w_{j',j}, w_{j,j''}\}$. Let L be the resulting line.

The analysis of Section 3.4.1 shows that this choice of $w_{j',j''}$ guarantees that the weighted cutsize of L is smaller than twice the weighted cutsize of T. Once L is created from T, the algorithm predicts the label of each node i_t using a nearest-neighbor rule operating on L with a *resistance distance* metric. That is, the prediction on i_t is the label of i_{s^*}, being $s^* = \arg\min_{s<t} d(i_s, i_t)$ the previously revealed node closest to i_t, and $d(i,j) = \sum_{s=1}^{k} 1/w_{v_s,v_{s+1}}$ is the sum of the resistors (i.e., reciprocals of edge weights) along the unique path $i = v_1 \to v_2 \to \cdots \to v_{k+1} = j$ connecting node i to node j.

3.4.1 Analysis of WTA

The following lemma gives a mistake bound on WTA run on any weighted line graph. Let $R_G^W = \sum_{(i,j)\in E\setminus E^\Phi} 1/w_{i,j}$ the sum of resistors of ϕ-free edges in a labeled graph $(G,\mathbf{y})$. Let also $f \overset{\mathcal{O}}{=} g$ denote $f = \mathcal{O}(g)$.

Lemma 3.1 *If* WTA *is run on a weighted line graph* $(L,\mathbf{y})$*, then the total number* m_L *of mistakes satisfies*

[2]By iterating this elimination procedure, it might happen that more than two adjacent nodes get eliminated. In this case, the two surviving terminal nodes are connected in L by the lightest edge among the eliminated ones in L'.

$$m_L \stackrel{\mathcal{O}}{=} \Phi_L(\mathbf{y}) \left(1 + \log\left(1 + \frac{\tilde{R}_L^W \Phi_L^W(\mathbf{y})}{\Phi_L(\mathbf{y})}\right)\right) + K$$

where $\tilde{R}_L^W$ *is the sum of the resistors of any arbitrary set including all but* K ϕ*-free edges of* L*.*

Note that the bound of Lemma 3.1 implies that, for any $K \geq 0$, one can drop from the bound the contribution of any set of K resistors in R_L^W at the cost of adding K extra mistakes.

We now proceed with the proof of Lemma 3.1. Some preliminary definitions and auxiliary results are first needed.

From the description of WTA in Section 5.4, we see that when L' is transformed into L the pairs of edges (j', j) and (j, j'') of L' which are incident to a repeated node j get replaced in L (together with j) by a single edge (j', j'') —step 3. We call these edges *spurious* edges. Assume that (j', j'') is spurious in L. When $y_{j'} \neq y_{j''}$ we have created a spurious ϕ-edge by eliminating a ϕ-edge and a ϕ-free edge from L'. When $y_{j'} = y_{j''} = y_j$, we have created a spurious ϕ-free edge by eliminating two ϕ-free edges from L'. Finally, when $y_{j'} = y_{j''} \neq y_j$, we have created a spurious ϕ-free edge by eliminating two ϕ-edges from L'. Let R_0^W be the sum of resistors of all spurious ϕ-free edges created during the elimination of pairs of ϕ-edges in L'.

Lemma 3.2 *Let* $(T, \mathbf{y})$ *be a labeled tree,* $(L, \mathbf{y})$ *be a linearized version of it, and* L' *be the line graph with duplicates (as described above). Then the following holds:*

1. $R_L^W \leq R_{L'}^W + R_0^W \leq 2R_T^W + R_0^W$;
2. $\Phi_L^W(\mathbf{y}) \leq \Phi_{L'}^W(\mathbf{y}) \leq 2\Phi_T^W(\mathbf{y})$;
3. $\Phi_L(\mathbf{y}) \leq \Phi_{L'}(\mathbf{y}) \leq 2\Phi_T(\mathbf{y})$.

PROOF. Note that each edge of T occurs in L' at least once and at most twice. This proves $\Phi_{L'}^W(\mathbf{y}) \leq 2\Phi_T^W(\mathbf{y})$ and $\Phi_{L'}(\mathbf{y}) \leq 2\Phi_T(\mathbf{y})$. Note further that L contains some non-spurious edges from L' plus a number of spurious edges. Each spurious ϕ-free edge (j', j'') can be created (by eliminating a node j) when either (i) $y_{j'} = y_{j''} = y_j$, which implies that $w_{j',j''}$ corresponds to the weight of a ϕ-free edge eliminated in L' together with node j, thus $w_{j',j''}$ is not included in R_0^W; or (ii) $y_{j'} = y_{j''} \neq y_j$, which implies that $w_{j',j''}$ is included in R_0^W. This proves the first inequality. To prove the remaining

inequalities, first note that a spurious edge (j', j'') cannot be a ϕ-edge in L unless either (j, j') or (j, j'') is a ϕ-edge in L'. Moreover, if (j', j'') is a ϕ-edge in L, then its weight is not larger than the weight of the associated ϕ-edge in L' —Step 3 in the description of WTA. □

We are now ready to prove Lemma 3.1.

PROOF OF LEMMA 3.1. Let a *cluster* be any maximal sub-line of L whose edges are all ϕ-free. Then L contains exactly $\Phi_L(\mathbf{y}) + 1$ clusters, which we number consecutively, starting from one of the two terminal nodes. Consider the k-th cluster c_k. Let v_0 be the first node of c_k whose label is predicted by WTA. After y_{v_0} is revealed, the cluster splits into two edge-disjoint sub-lines[3] c'_k and c''_k, both having v_0 as terminal node. Let v'_k and v''_k be the closest nodes to v_0 such that (i) $y_{v'_k} = y_{v''_k} \neq y_{v_0}$ and (ii) v'_k is adjacent to a terminal node of c'_k, and v''_k is adjacent to a terminal node of c''_k. The nearest neighbor prediction rule of WTA guarantees that the first mistake made on c'_k (respectively, c''_k) must occur on a node v_1 such that $d(v_0, v_1) \geq d(v_1, v'_k)$ (respectively, $d(v_0, v_1) \geq d(v_1, v''_k)$). By iterating this argument for the subsequent mistakes we see that the total number of mistakes made on cluster c_k is bounded by

$$1 + \left\lfloor \log_2 \frac{R'_k + (w'_k)^{-1}}{(w'_k)^{-1}} \right\rfloor + \left\lfloor \log_2 \frac{R''_k + (w''_k)^{-1}}{(w''_k)^{-1}} \right\rfloor$$

where R'_k is the resistance diameter of sub-line c'_k, and w'_k is the weight of the ϕ-edge between v'_k and the terminal node of c'_k closest to it (R''_k and w''_k are defined similarly). Hence, summing the above displayed expression over

[3]W.l.o.g., we assume that neither of the two sub-lines is empty, so that v_0 is not a terminal node of c_k.

clusters $k = 1, \ldots, \Phi_L(\mathbf{y}) + 1$ we obtain

$$\begin{aligned} m_L &\overset{\mathcal{O}}{=} \Phi_L(\mathbf{y}) + \sum_k \left(\log\left(1 + R'_k w'_k\right) + \log\left(1 + R'_k w''_k\right) \right) \\ &\overset{\mathcal{O}}{=} \Phi_L(\mathbf{y}) \left(1 + \log\left(1 + \frac{1}{\Phi_L(\mathbf{y})} \sum_k R'_k w'_k \right) \right. \\ &\qquad \left. + \log\left(1 + \frac{1}{\Phi_L(\mathbf{y})} \sum_k R''_k w''_k \right) \right) \\ &\overset{\mathcal{O}}{=} \Phi_L(\mathbf{y}) \left(1 + \log\left(1 + \frac{R^W_L \Phi^W_L(\mathbf{y})}{\Phi_L(\mathbf{y})} \right) \right) \end{aligned}$$

where in the second step we used Jensen's inequality and in the last one the fact that $\sum_k (R'_k + R''_k) = R^W_L$ and $\sum_k w'_k = \sum_k w''_k = \mathcal{O}\left(\Phi^W_L(\mathbf{y})\right)$. This proves the lemma in the case $K = 0$.

In order to conclude the proof, we simply observe that if we take any semi-cluster c'_k (obtained, as before, by splitting cluster c_k, being $v_0 \in c_k$ the first node whose label is predicted by WTA), and pretend to split it into two sub-clusters connected by a ϕ-free edge, we could repeat the above dichotomic argument almost verbatim on the two sub-clusters at the cost of adding an extra mistake. □

We now provide an upper bound on the number of mistakes made by WTA on any weighted tree $T = (V, E, W)$ in terms of the number of ϕ-edges, the weighted cutsize, and R^W_T.

Theorem 3.2 *If* WTA *is run on a weighted and labeled tree* $(T, \mathbf{y})$*, then the total number* m_T *of mistakes satisfies*

$$m_T \overset{\mathcal{O}}{=} \Phi_T(\mathbf{y}) \left(1 + \log\left(1 + \frac{R^W_T \Phi^W_T(\mathbf{y})}{\Phi_T(\mathbf{y})} \right) \right) \ .$$

PROOF. Recall that R^W_0 is the sum of resistors on all spurious ϕ-free edges obtained by eliminating pairs of ϕ-edges in L'. Hence, we can injectively associate with each such edge two distinct ϕ-edges in L', and therefore the total number of spurious edges giving contribution to R^W_0 is bounded by $\Phi_{L'}(\mathbf{y})/2$, which in turn can be bounded by $\Phi_T(\mathbf{y})$ via Lemma 3.2. Applying Lemma 3.1 (setting $\tilde{R}^W_L$ to $R^W_L - R^W_0$) along with Lemma 3.2 concludes the proof. □

The logarithmic factor in the above bound shows that the algorithm takes advantage of labelings such that the weights of ϕ-edges are small (thus making $\Phi_T^W(\mathbf{y})$ small) and the weights of ϕ-free edges are high (thus making R_T^W small). This somehow matches the intuition behind WTA's nearest-neighbor rule according to which nodes that are close to each other are expected to have the same label. In particular, observe that the way the above quantities are combined makes the bound independent of rescaling of the edge weights. Again, this has to be expected, since WTA's prediction is scale insensitive. On the other hand, it may appear less natural that the mistake bound also depends linearly on the cutsize $\Phi_T(\mathbf{y})$, *independent of the edge weights*. As a matter of fact, this linear dependence on the unweighted cutsize *cannot be eliminated* (this is a simple consequence of Theorem 3.1 in Section 3.3).

3.5 Predicting a weighted graph

In order to solve the more general problem of predicting the labels of a weighted graph G, one can first generate a spanning tree T of G and then run WTA directly on T. In this case it is possible to rephrase Theorem 3.2 in terms of properties of G. Note that for each spanning tree T of G, $\Phi_T^W(\mathbf{y}) \le \Phi_G^W(\mathbf{y})$ and $\Phi_T(\mathbf{y}) \le \Phi_G(\mathbf{y})$. Specific choices of the spanning tree T control in different ways the quantities in the mistake bound of Theorem 3.2. For example, a minimum spanning tree tends to reduce the value of R_T^W, betting on the fact that ϕ-edges are light. The next theorem relies on *random* spanning trees.

Theorem 3.3 *If* WTA *is run on a random spanning tree* T *of a labeled weighted graph* $(G, \mathbf{y})$*, then the total number* m_G *of mistakes satisfies*

$$\mathbb{E}\, m_G \overset{\mathcal{O}}{=} \mathbb{E}\Big[\Phi_T(\mathbf{y})\Big]\left(1 + \log\left(1 + w_{\max}^{\phi}\mathbb{E}\Big[R_T^W\Big]\right)\right) \tag{3.2}$$

where $w_{\max}^{\phi} = \max_{(i,j)\in E^{\phi}} w_{i,j}$.

PROOF. Using Theorem 3.2 we can write

$$\begin{aligned}
\mathbb{E}\, m_G &\overset{\mathcal{O}}{=} \mathbb{E}\left[\Phi_T(\mathbf{y})\left(1 + \log\left(1 + \frac{R_T^W \Phi_T^W(\mathbf{y})}{\Phi_T(\mathbf{y})}\right)\right)\right] \\
&\overset{\mathcal{O}}{=} \mathbb{E}\Big[\Phi_T(\mathbf{y})\Big(1 + \log\Big(1 + R_T^W w_{\max}^{\phi}\Big)\Big)\Big] \\
&\overset{\mathcal{O}}{=} \mathbb{E}\Big[\Phi_T(\mathbf{y})\Big]\left(1 + \log\left(1 + \mathbb{E}\Big[R_T^W\Big] w_{\max}^{\phi}\right)\right)
\end{aligned}$$

where the second equality follows from the fact that $\Phi_T^W(\mathbf{y}) \leq \Phi_T(\mathbf{y}) w_{\max}^{\phi}$, and the third one follows from Jensen's inequality applied to the concave function

$$(x, y) \mapsto x\left(1 + \log\left(1 + y\, w_{\max}^{\phi}\right)\right) \qquad x, y \geq 0.$$

□

Note that the mistake bound in (3.2) is scale-invariant, since $\mathbb{E}\left[\Phi_T(\mathbf{y})\right] = \sum_{(i,j)\in E^{\phi}} w_{i,j} r_{i,j}^W$ cannot be affected by a uniform rescaling of the edge weights, and so is the product $w_{\max}^{\phi}\mathbb{E}\left[R_T^W\right] = w_{\max}^{\phi} \sum_{(i,j)\in E\setminus E^{\phi}} r_{i,j}^W$.

We now compare the mistake bound (3.2) to the lower bound stated in Theorem 3.1. In particular, we prove that WTA is optimal (up to $\mathcal{O}(\log n)$ factors) on every weighted connected graph in which the ϕ-edges weights are not "superpolynomially overloaded" w.r.t. the ϕ-free edge weights. In order to rule out pathological cases, when the weighted graph is nearly disconnected, we impose the following mild assumption on the graphs being considered.

We say that a graph is **polynomially connected** if the ratio of any pair of effective resistances (even those between nonadjacent nodes) in the graph is polynomial in the total number of nodes n. This definition essentially states that a weighted graph can be considered connected if no pair of nodes can be found which is substantially less connected than any other pair of nodes. Again, as one would naturally expect, this definition is independent of uniform weight rescaling. The following corollary shows that if WTA is not optimal on a polynomially connected graph, then the labeling must be so irregular that the total weight of ϕ-edges is an overwhelming fraction of the overall weight.

Corollary 3.4 *Pick any polynomially connected weighted graph* G *with* n *nodes. If the ratio of the total weight of* ϕ*-edges to the total weight of* ϕ*-free edges is bounded by a polynomial in* n*, then the total number of mistakes* m_G *made by* WTA *when run on a random spanning tree* T *of* G *satisfies* $\mathbb{E}\, m_G \overset{\mathcal{O}}{=} \mathbb{E}\left[\Phi_T(\mathbf{y})\right] \log n$.

PROOF. Let $f > \text{poly}(n)$ denote a function growing faster than any polynomial in n. Choose a polynomially connected graph G and a labeling $\mathbf{y}$. For the sake of contradiction, assume that WTA makes more than $\mathcal{O}(\mathbb{E}\left[\Phi_T(\mathbf{y})\right] \log n)$ mistakes on $(G, \mathbf{y})$. Then Theorem 5 implies

$w^{\phi}_{\max}\mathbb{E}\left[R^W_T\right] > \mathrm{poly}(n)$. Since $\mathbb{E}\left[R^W_T\right] = \sum_{(i,j)\in E\setminus E^{\phi}} r^W_{i,j}$, we have that $w^{\phi}_{\max}\max_{(i,j)\in E\setminus E^{\phi}} r^W_{i,j} > \mathrm{poly}(n)$. Together with the assumption of polynomial connectivity for G, this implies $w^{\phi}_{\max} r^W_{i,j} > \mathrm{poly}(n)$ for all ϕ-free edges (i,j). By definition of effective resistance, $w_{i,j} r^W_{i,j} \leq 1$ for all $(i,j) \in E$. This gives $w^{\phi}_{\max}/w_{i,j} > \mathrm{poly}(n)$ for all ϕ-free edges (i,j), which in turn implies

$$\frac{\sum_{(i,j)\in E^{\phi}} w_{i,j}}{\sum_{(i,j)\in E\setminus E^{\phi}} w_{i,j}} > \mathrm{poly}(n),$$

thereby concluding the proof. □

Note that when the hypothesis of this corollary is not satisfied the bound of WTA is not necessarily vacuous. For example, $\mathbb{E}\left[R^W_T\right] w^{\phi}_{\max} = n^{\mathrm{polylog}(n)}$ implies an upper bound which is optimal up to $\mathrm{polylog}(n)$ factors. In particular, having a constant number of ϕ-free edges with exponentially large resistance contradicts the assumption of polynomial connectivity, but it need not lead to a vacuous bound in Theorem 3.3. In fact, one can use Lemma 3.1 to drop from the mistake bound of Theorem 3.3 the contribution of any set of $\mathcal{O}(1)$ resistances in $\mathbb{E}\left[R^W_T\right] = \sum_{(i,j)\in E\setminus E^{\phi}} r^W_{i,j}$ at the cost of adding just $\mathcal{O}(1)$ extra mistakes. This could be interpreted as a robustness property of WTA's bound against graphs that do not fully satisfy the connectedness assumption.

Corollary 3.4 can be compared to the expected mistake bound of the graph Perceptron algorithm GPA on the same random spanning tree —see Section 3.7 for more details on GPA. This bound depends on the expectation of the product $\Phi^W_T(y) D^W_T$, where D^W_T is the diameter of T in the resistance distance metric. Note that these two factors are negatively correlated because $\Phi^W(y)$ dipends linearly on the edge weights whereas D^W_T dipends linearly on the reciprocal of these weights —see the definition of resistance distance in Section 3.2. Moreover, for any given scale of the edge weights, D^W_T can be linear in the number n of nodes.

Finally, in light of the performance guarantees provided by Corollary 3.4, it is worthwhile to compare WTA with the TREEOPT algorithm described in Chapter 2. Both algorithms predict on a general input graph condensing its structural information with a random spanning tree. WTA is in general faster than TREEOPT and is able to operate also with weighted graphs. However, TREEOPT is able to achieve optimality up to *constant* factors on *any* labeled tree, while Corollary 3.4 only ensures optimality up to $\mathcal{O}(\log n)$ factors, even when the input graph is a tree.

3.6 Implementation

A direct implementation of WTA operating on a tree T with n nodes runs in time $\mathcal{O}(n \log n)$ and requires linear memory space. We now describe how to implement WTA to run in time $\mathcal{O}(n)$, i.e., in *constant* amortized time per step.

Once the given tree T is linearized into an n-node line L, we initially traverse L from left to right. Call j_0 the left-most terminal node of L. During this traversal, the resistance distance $d(j_0, i)$ is incrementally computed for each node i in L. This makes it possible to calculate $d(i, j)$ in constant time for any pair of nodes, since $d(i, j) = |d(j_0, i) - d(j_0, j)| \quad \forall i, j \in L$. On top of line L a complete binary tree T' with $2^{\lceil \log_2 n \rceil}$ leaves is constructed.[4] The k-th leftmost leaf (in the usual tree representation) of T' is the k-th node in L (numbering the nodes of L from left to right). The algorithm maintains this data-structure in such a way that at time t: (i) the subsequence of leaves whose labels are revealed at time t are connected through a (bidirectional) list B, and (ii) all the ancestors in T' of the leaves of B are marked. See Figure 3.1.

When WTA is required to predict the label y_{i_t}, the algorithm looks for the two closest leaves i' and i'' oppositely located in L with respect to i_t. The above data structure supports this operation as follows. WTA starts from i_t and goes upwards in T' until the first marked ancestor $\mathrm{anc}(i_t)$ of i_t is reached. During this upward traversal, the algorithm marks each internal node of T' on the path connecting i_t to $\mathrm{anc}(i_t)$. Then, WTA starts from $\mathrm{anc}(i_t)$ and goes downwards in order to find the leaf $i' \in B$ closest to i_t. Note how the algorithm uses node marks for finding its way down: For instance, in Figure 3.1 the algorithm goes left since $\mathrm{anc}(i_t)$ was reached from below through the right child node, and then keeps right all the way down to i'. Node i'' (if present) is then identified via the links in B. The two distances $d(i_t, i')$ and $d(i_t, i'')$ are compared, and the closest node to i_t within B is then determined. Finally, WTA updates the links of B by inserting i_t between i' and i''.

In order to quantify the amortized time per trial, the key observation is that each internal node k of T' gets visited only twice during *upward*

[4] For simplicity, this description assumes n is a power of 2. If this is not the case, we could add dummy nodes to L before building T'.

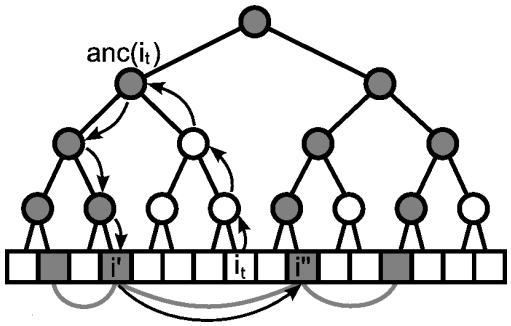

Figure 3.1: Constant amortized time implementation of WTA. The line L has $n = 16$ nodes (the adjacent squares at the bottom). Shaded squares are the revealed nodes, connected through a dark grey doubly-linked list B. The depicted tree T' has both unmarked (white) and marked (shaded) nodes. The arrows indicate the traversal operations performed by WTA when predicting the label of node i_t: The upward traversal stops as soon as a marked ancestor $\mathrm{anc}(i_t)$ is found, and then a downward traversal begins. Note that WTA first descends to the left, and then keeps going right all the way down. Once i' is determined, a single step within B suffices to determine i''.

traversals over the n trials: The first visit takes place when k gets marked for the first time, the second visit of k occurs when a subsequent upward visit also marks the other (unmarked) child of k. Once both of k's children are marked, we are guaranteed that no further upward visits to k will be performed. Since the preprocessing operations take $\mathcal{O}(n)$, this shows that the total running time over the n trials is linear in n, as anticipated.[5]

3.7 Experiments

We now present the results of an experimental comparison on a number of real-world weighted graphs from different domains: text categorization, optical character recognition, and bioinformatics.

Our goal is to compare the prediction accuracy of WTA to the one achieved by known baseline algorithms for weighted (and unweighted) graph prediction. We compare our algorithm to the following two other on-line prediction methods, intended as representatives of two different ways of facing the graph

[5]Note, however, that the worst-case time per trial is $\mathcal{O}(\log n)$. For instance, on the very first trial T' has to be traversed all the way up and down.

prediction problem: global vs. local prediction.

The **Perceptron algorithm with graph Laplacian kernel** by [47], abbreviated as GPA (graph Perceptron algorithm). This algorithm predicts the nodes of a weighted graph $G = (V, E)$ after mapping V via the linear kernel based on $L_G^+ + \mathbf{1}\mathbf{1}^\top$, where L_G is the laplacian matrix of G. Following [48] we run GPA on a spanning tree T of the original graph. We do so because computing the pseudoinverse L_G^+ when G is a tree takes time and space quadratic in the number of nodes n (this in contrast to WTA that runs in linear time and linear space). GPA is a global approach in the sense that the graph topology affects, via the inverse Laplacian, the prediction on all nodes.

The **On-line Majority Vote algorithm** (abbreviated as OMV). Since the common underlying assumption to graph prediction algorithms is that nearby nodes are labeled similarly, a very intuitive and fast algorithm for sequentially predicting the label of a node i_t is via a weighted majority vote on all labels of the adjacent nodes seen so far, i.e., $\mathrm{sgn}\left(\sum_{s<t\,:\,(i_s,i_t)\in E} y_{i_s} w_{i_s,i_t}\right)$. The overall time and space requirements are both of order $\Theta(|E|)$, since we need to read (at least once) the weights of all edges. OMV-like algorithms are local approaches, in the sense that prediction at one node is affected only by adjacent nodes. OMV, as presented above, is the most natural on-line version of the **label propagation** (or energy minimization) algorithm [102], abbreviated as LABPROP, which we keep as an accuracy baseline throughout our experiments.[6] LABPROP is a batch transductive learning method and is computed by solving a (possibly sparse) linear system of equations which requires $\mathcal{O}(kn^2)$ time on an n-node graph with k neighbors per node. This bad scalability, which prevented us from carrying out comparative experiments on larger graphs of 10^5 nodes, should be taken into account when comparing LABPROP to fast on-line (i.e., one-sweep) algorithms.

In our experiments, we combined WTA and GPA with spanning trees generated in different ways (note that OMV and LABPROP do not operate on spanning trees).

[6] Many other algorithms have been proposed in the literature for graph prediction problems, including the label-consistent mincut approach of [14] and a number of other "energy minimization" methods —e.g., the ones in [8, 45]. See [10] for a relatively recent survey on this subject.

Random Spanning Tree (RST). Each spanning tree is taken with probability proportional to the product of its edge weights — see Section 1.6 or, e.g., Ch. 4 of [63]. In addition, we also tested WTA combined with RST generated ignoring the edge weights (which were restored before running WTA). As shown in [94, 3], it is possible to generate unweighted random spanning trees in time *linear* in the number n of nodes for many and important classes of graphs. This gives a prediction algorithm whose total running time (including the generation of the spanning tree) is $\mathcal{O}(n)$ for many graphs. We abbreviate this spanning tree as NWRST (non-weighted RST).

Depth-first spanning tree (DFST). The spanning tree is created via the following randomized depth-first visit: A root is selected at random, then each newly visited node is chosen with probability proportional to the weights of the edges connecting the current vertex with the adjacent nodes that have not been visited yet. This spanning tree is faster to generate than RST, and can be viewed as an approximate version of RST.

Minimum Spanning Tree (MST). The spanning tree minimizing the sum of the resistors of all edges. This is the tree whose Laplacian best approximates the Laplacian of G according to the trace norm criterion —see, e.g., [48].

Shortest Path Spanning Tree (SPST). [48] use the shortest path tree for its small diameter (at most twice the diameter of G), which allows them to better control the theoretical performance of GPA. We generated n shortest path spanning trees by varying the choice of the root node, and then took the one having minimal diameter among them.

Finally, in order to check whether the information carried by the edge weight has predictive value for a nearest neighbor rule like WTA, we also performed a test by ignoring the edge weights during both the generation of the spanning tree and the running of WTA's nearest neighbor rule. This is essentially the algorithm analyzed in [46], and we denote it with NWWTA (non-weighted WTA). We combined NWWTA with (weighted) MST, that is the spanning tree on which WTA performs best.

We ran our experiments on four medium size real-world datasets: (1) The first 10,000 documents (in chronological order) of **RCV1**, with TF-IDF

Table 3.1: Macro-averaged and cross-validated classification error rates (percentages) achieved by the various algorithms on the five datasets/graphs mentioned in the main text. We compare WTA, GPA, and the algorithm by [46] combined with different spanning trees, to LABPROP and OMV. TRAIN:TEST denotes the training and test set size ratio (for instance, 3:1 means 75% train and 25% test). In boldface are the lowest errors on each dataset/graph among the on-line algorithms (thus excluding LABPROP). Standard deviations (averaged over the binary problems) are quite small. For instance, in Krogan and Comb, the average standard deviations are below 1.0%.

DATASET	RCV1-100		USPS-10		USPS-100		KROGAN		COMBINED	
TRAIN:TEST	1:1	3:1	1:1	3:1	1:1	3:1	1:1	3:1	1:1	3:1
ALGORITHM										
WTA+RST	23.2	21.5	2.1	1.8	5.2	4.5	19.0	18.4	19.7	19.2
WTA+DF	20.4	18.7	2.0	1.6	4.2	3.5	18.7	18.1	19.7	19.1
WTA+MST	**11.8**	**10.2**	**1.0**	0.8	**1.0**	**0.9**	18.3	17.6	19.6	19.1
WTA+SPST	21.4	20.1	2.3	1.9	4.2	3.6	19.5	18.9	19.9	19.5
WTA+NWRST	23.8	22.0	2.4	2.0	5.8	5.0	19.4	18.7	19.9	19.5
GPA+RST	32.5	31.2	4.7	4.4	9.6	8.6	21.7	22.0	21.4	21.5
GPA+DF	41.2	40.1	24.0	18.7	28.8	23.6	24.1	22.6	23.8	22.7
GPA+MST	20.4	18.2	2.0	1.7	2.0	1.8	20.7	20.9	21.1	20.7
GPA+SPST	24.5	24.4	2.9	2.7	5.2	4.5	20.8	20.6	21.1	20.2
GPA+NWRST	32.1	31.4	6.0	5.4	10.1	9.9	21.8	21.5	22.0	22.0
NWWTA+NWDFST	21.4	20.3	2.5	2.3	5.2	4.8	19.3	19.0	19.9	19.7
NWWTA+MST	12.8	11.9	1.2	1.2	1.2	1.2	18.8	18.4	19.8	19.6
OMV	25.4	20.9	1.1	**0.7**	1.9	1.6	**16.3**	**16.0**	**17.5**	**17.3**
LABPROP	10.9	9.5	0.8	0.7	2.0	1.7	15.1	15.3	16.0	16.2

preprocessing and vector normalization; (2) the **USPS** dataset with features normalized into $[0, 2]$; (3) the dataset of [58, 71] abbreviated as **KROGAN**; (4) a second dataset [71], abbreviated as **COMBINED**, resulting from a combination of three datasets from [36, 51, 91];

On the RCV1 and USPS datasets we generated graphs with as many nodes as the total number of examples $(\mathbf{x}_i, y_i)$, that is, 10,000 nodes for RCV1 and 7291+2007 = 9298 for USPS. Following previous experimental settings [102, 8], we used k-NN based on the standard Euclidean distance $\|\mathbf{x}_i - \mathbf{x}_j\|$ between node i and node j. The weight $w_{i,j}$ was set as $w_{i,j} = e^{-\|\mathbf{x}_i - \mathbf{x}_j\|/\sigma^2}$, if j is one of the k nearest neighbors of i, and 0 otherwise. To set σ^2, we first computed the average square distance between i and its k nearest neighbors, and then took a further average over i. On USPS we generated two graphs, **USPS-10** and **USPS-100**, by running k-NN with $k = 10$ and $k = 100$. On RCV1 we generated a single graph, **RCV1-100**, by

setting $k = 100$. We selected the four most frequent categories in RCV1 and all 10 categories in USPS.

KROGAN and COMBINED are high-throughput protein-protein interaction networks of budding yeast taken from [71]. We only consider the biggest connected components of both datasets, obtaining 2,169 nodes and 6,102 edges for KROGAN, and 2,871 nodes and 6,407 edges for COMBINED. In these graphs, each node belongs to one or more classes, each class representing a protein function. We selected the set of functional labels at depth one in the *FunCat* classification scheme of the MIPS database [80], resulting in 17 classes per dataset.

In order to associate binary classification tasks with the five datasets/graphs (RCV1-100, USPS-10, USPS-100, KROGAN, and COMBINED) we binarized the corresponding multiclass problems via a standard one-vs-rest scheme. We thus obtained: 4 binary classification tasks for RCV1-100, 10 binary tasks for USPS-10 and USPS-100, 17 binary tasks for both KROGAN and COMBINED. For a given a binary task and dataset, we tried different proportions of training and test set sizes. On all datasets we used both the two-fold 50% train – 50% test and the four-fold 75% train – 25% test. The error rate results we report in Table 3.1 are obtained by (either two or four)-fold cross validation over the entire datasets after macro-averaging over the corresponding binary tasks.

In our experimental setup we tried to control the sources of variance as follows: (i) We first generated 10 random permutations of the node indices for each of the five graphs/datasets; (ii) on each permutation we generated the training/test splits, (iii) we computed MST and SPST for each graph and made (for WTA, GPA, OMV, and LABPROP) one run per permutation on each of the $4+10+10+17+17 = 58$ binary problems, averaging results over permutations and splits; (iv) we generated 10 RST's and 10 DFST's for each graph (possibly disregarding edge weights at either generation time or prediction time), and operated as in (ii), with a further averaging over the randomness in the tree generation.

Table 3.1 gives the average fraction of prediction mistakes achieved by the various algorithms on the five datasets/graphs. Though the experiments are not conclusive, several interesting observations can be made.

1. WTA outperforms GPA on all datasets and with all spanning tree combinations. In particular, though we only reported aggregated results, the

same relative performance pattern among the two algorithms repeats systematically over all binary classification problems. In addition, WTA runs significantly faster than GPA, requires less memory storage (linear in n, rather than quadratic), and is also fairly easy to implement.

2. The best performing combination for both WTA and GPA is MST. This might be explained by the fact that MST tends to select light ϕ-edges of the original graph.
3. By comparing NWWTA to WTA, we see that the edge weight information in the nearest neighbor rule is beneficial.
4. On RCV1 and USPS the prediction performance of WTA+MST is comparable to that of LABPROP, whereas on KROGAN and COMBINED WTA+MST is slightly inferior. However, recall that LABPROP takes time $\mathcal{O}(kn^2)$, where k is the node degree, whereas a single sweep of WTA+MST over the graph just takes[7] time $\mathcal{O}(kn \log n)$.

   Moreover, a simple way of making WTA outperform LABPROP on the two biological datasets is to let WTA predict through a *committee* of RST's aggregated via a majority vote. For instance, using WTA with a committee of 11 RST's generated independently (either considering or disregarding the edge weights) gets the following figures.

DATASET	KROGAN		COMBINED	
TRAIN:TEST	1:1	3:1	1:1	3:1
ALGORITHM				
WTA+11RST	14.9	14.4	14.9	14.6
WTA+11NWRST	15.0	14.6	14.9	14.7
OMV	16.3	16.0	17.5	17.3
LABPROP	15.1	15.3	16.0	16.2

   Similar improvements are likely to occur on the other datasets. On USPS, WTA+MST, LABPROP, and OMV tend to perform comparably.
5. NWRST and DFST are fast approximations to RST. Though the use of NWRST and DFST does not provide the same theoretical performance guarantees as RST, in our experiments the three do actually perform comparably. Hence, in practice, NWRST and DFST might be viewed as fast and practical ways to generate spanning trees for WTA.

[7] The MST of a graph $G = (V, E)$ can be computed in time $\mathcal{O}(|E| \log |V|)$.

Finally, in the next figure we show, for each algorithm, the performance accuracy and the time used for predicting on the COMBINED dataset. The times are calculated averaging over 10 executions and include uploading the graph data into memory, drawing the spanning tree (if required) and running the algorithm.

Algorithm	Mistake [%]	Time [s]
labprop	15.96	1.8
omv	17.47	0.2
wta+rst	19.66	0.69
wta+dfst	19.65	0.78
wta+nwrst	19.93	0.64
wta+mst	19.64	1.92
gpa+rst	21.41	18.26
gpa+dfst	23.83	25.42
gpa+nwrst	22.01	18.14
gpa+mst	21.14	18.82
wta+11rst	14.88	4.97
wta+11dfst	15.44	5.81
wta+11nwrst	14.94	4.4
gpa+11rst	14.84	198.32
gpa+11dfst	15.7	273.89
gpa+11nwrst	14.84	194.97

3.8 Conclusions

We introduced and analyzed wta, an on-line prediction algorithm for weighted graph prediction. The algorithm uses random spanning trees and has nearly optimal (expected) performance guarantees in terms of both prediction accuracy and running time. Our initial experimental evaluation shows that wta outperforms other previously proposed on-line predictors. Moreover, when combined with an aggregation of random spanning trees, wta also tends to beat standard batch predictors, such as label propagation. These features make wta (and combinations thereof) suitable to large scale applications.

Chapter 4

Active learning on trees and graphs

4.1 Introduction

In the active learning version of node classification problem, the learner is allowed to choose the subset of training nodes. Similarly to standard feature-based learning, one expects active methods to provide a significant boost of predictive ability compared to a noninformed (e.g., random) draw of the training set. The following simple example provides some intuition of why this could happen when the labels are chosen by an adversary, which is the setting considered in this chapter. Consider a "binary star system" of two star-shaped graphs whose centers are connected by a bridge, where one star is a constant fraction bigger than the other. The adversary draws two random binary labels and assigns the first label to all nodes of the first star graph, and the second label to all nodes of the second star graph. Assume that the training set size is two. If we choose the centers of the two stars and predict with a mincut strategy (see Section 1.3) we are guaranteed to make zero mistakes on all unseen vertices. On the other hand, if we query two nodes at random, then with constant probability both of them will belong to the bigger star, and all the unseen labels of the smaller star will be mistaken. This simple example shows that the gap between the performance of passive and active learning on graphs can be made arbitrarily big.

In general, one would like to devise a strategy for placing a certain budget of queries on the vertices of a given graph. This should be done so as to

minimize the number of mistakes made on the non-queried nodes by some reasonable classifier like mincut. This question has been investigated from a theoretical viewpoint by Guillory and Bilmes [41], and by Afshani et al. [1]. Our work is related to an elegant result from [41] which bounds the number of mistakes made by the mincut classifier on the worst-case assignment of labels in terms of $\Phi(\mathbf{y})/\Psi(L)$. Here $\Psi(L)$ is a function of the query (or training) set L, which depends on the structural properties of the (unlabeled) graph. For instance, in the above example of the binary system, the value of $\Psi(L)$ when the query set L includes just the two centers is 1. This implies that for the binary system graph, Guillory and Bilmes' bound on the mincut strategy is $\Phi(\mathbf{y})$ mistakes in the worst case (note that in the above example $\Phi(\mathbf{y}) \leq 1$). Since $\Psi(L)$ can be efficiently computed on any given graph and query set L, the learner's task might be reduced to finding a query set L that maximizes $\Psi(L)$ given a certain query budget (size of L). Unfortunately, no feasible general algorithm for solving this maximization problem is known, and so one must resort to heuristic methods —see [41].

In this chapter we investigate the active learning problem on graphs in the important special case of trees. We exhibit a simple iterative algorithm which, combined with a mincut classifier, is optimal (up to constant factors) on any given labeled tree. This holds even if the algorithm is not given information on the actual cutsize $\Phi(\mathbf{y})$. Our method is extremely efficient, requiring $\mathcal{O}(n \ln Q)$ time for placing Q queries in an n-node tree, and space linear in n. As a byproduct of our analysis, we show that Ψ can be efficiently maximized over trees to within constant factors. Hence the bound $\min_L \Phi(\mathbf{y})/\Psi(L)$ can be achieved efficiently.

Another interesting question is what kind of trade-off between queries and mistakes can be achieved if the learner is not constrained by a given query budget. We show that a simple modification of our selection algorithm is able to trade-off queries and mistakes in an optimal way up to constant factors.

Finally, we prove a general lower bound for predicting the labels of any given graph (not necessarily a tree) when the query set is up to a constant fraction of the number of vertices. Our lower bound establishes that the number of mistakes must then be at least a constant fraction of the cutsize weighted by the effective resistances. This lower bound apparently yields a contradiction to the results of Afshani et al. [1], who constructs the query set adaptively. This apparent contradiction is also obtained via a simple

counterexample that we detail in Section 4.5.

4.2 Preliminaries and basic notation

We measure the label regularity of the input tree $(T, \mathbf{y})$ by the cutsize $\Phi_T(\mathbf{y})$ induced by $\mathbf{y}$ on T. We consider the following *active* learning protocol: given a tree T with unknown labeling $\mathbf{y}$, the learner obtains all labels in a *query set* $L \subseteq V$, and is then required to predict the labels of the remaining nodes $V \setminus L$. Active learning algorithms work in two-phases: a *selection* phase, where a query set of given size is constructed, and a *prediction* phase, where the algorithm receives the labels of the query set and predicts the labels of the remaining nodes. Note that the only labels ever observed by the algorithm are those in the query set. In particular, no labels are revealed during the prediction phase.

We measure the ability of the algorithm by the number of prediction mistakes made on $V \setminus L$, where it is reasonable to expect this number to depend on both the uknown cutsize $\Phi_T(\mathbf{y})$ and the number $|L|$ of requested labels. A slightly different prediction measure is considered in Section 4.4.3.

Given a tree T and a query set $L \subseteq V$, a node $i \in V \setminus L$ is a **fork node generated by** L if and only if there exist three distinct nodes $i_1, i_2, i_3 \in L$ that are connected to i through edge disjoint paths. Let $\textsc{fork}(L)$ be the set of all fork nodes generated by L. Then L^+ is the query set obtained by adding to L all the generated fork nodes, i.e., $L^+ \triangleq L \cup \textsc{fork}(L)$. We say that $L \subseteq V$ is **0-forked** iff $L^+ \equiv L$. Note that L^+ is 0-forked. That is, $\textsc{fork}(L^+) \equiv \emptyset$ for all $L \subseteq V$.

Given a node subset $S \subseteq V$, we use $T \setminus S$ to denote the forest obtained by removing from the tree T all nodes in S and all edges incident to them. Moreover, given a second tree T', we denote by $T \setminus T'$ the forest $T \setminus V'$, where V' is the set of nodes of T'. Given a query set $L \subseteq V$, a **hinge-tree** is any connected component of $T \setminus L^+$. We call **connection node** of a hinge-tree a node of L adjacent to any node of the hinge tree. We distinguish between 1-hinge and 2-hinge trees. A **1-hinge-tree** has one connection node only, whereas a **2-hinge-tree** has two (note that a hinge tree cannot have more than two connection nodes because L^+ is zero-forked, see Figure 4.1).

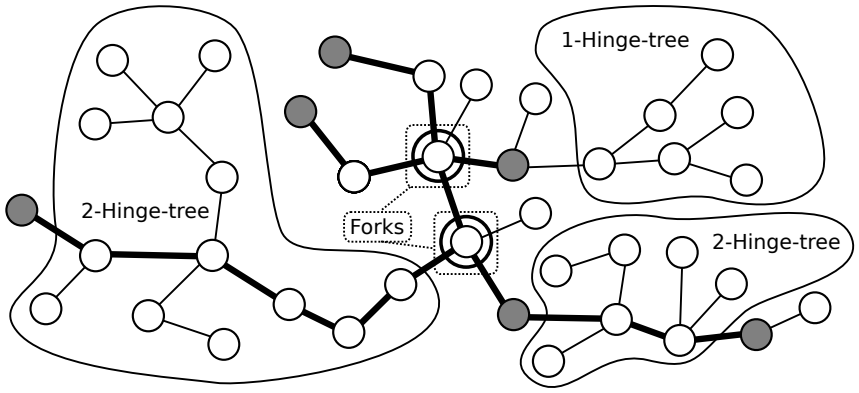

Figure 4.1: A tree $T = (V, E)$ whose nodes are shaded (the query set L) or white (the set $V \setminus L$). The shaded nodes are also the connection nodes of the depicted hinge trees (not all hinge trees are contoured). The fork nodes generated by L are denoted by double circles. The thick black edges connect the nodes in L.

4.3 The active learning algorithm

We now describe the two phases of our active learning algorithm. For the sake of exposition, we call SEL the selection phase and PRED the prediction phase. SEL returns a 0-forked query set $L^+_{\text{SEL}} \subseteq V$ of desired size. PRED takes in input the query set L^+_{SEL} and the set of labels y_i for all $i \in L^+_{\text{SEL}}$. Then PRED returns a prediction for the labels of all remaining nodes $V \setminus L^+_{\text{SEL}}$.

In order to see the way SEL operates, we formally introduce the function Ψ^*. This is the reciprocal of the Ψ function introduced in [41] and mentioned in Section 5.1.

Definition 4.1 *Given a tree* $T = (V, E)$ *and a set of nodes* $L \subseteq V$*,*

$$\Psi^*(L) \triangleq \max_{\emptyset \neq V' \subseteq V \setminus L} \frac{|V'|}{\left|\{(i,j) \in E : i \in V', j \in V \setminus V'\}\right|} \ .$$

In words, $\Psi^*(L)$ measures the largest set of nodes not in L that share the least number of edges with nodes in L. From the adversary's viewpoint, $\Psi^*(L)$ can be described as the largest return in mistakes per unit of cutsize invested. We now move on to the description of the algorithms SEL and PRED.

The **selection algoritm** SEL greedily computes a query set that minimizes Ψ^* to within constant factors. To this end, SEL exploits Lemma 4.6 (a) (see Section 4.4.2) stating that, for any fixed query set L, the subset $V' \subseteq V$

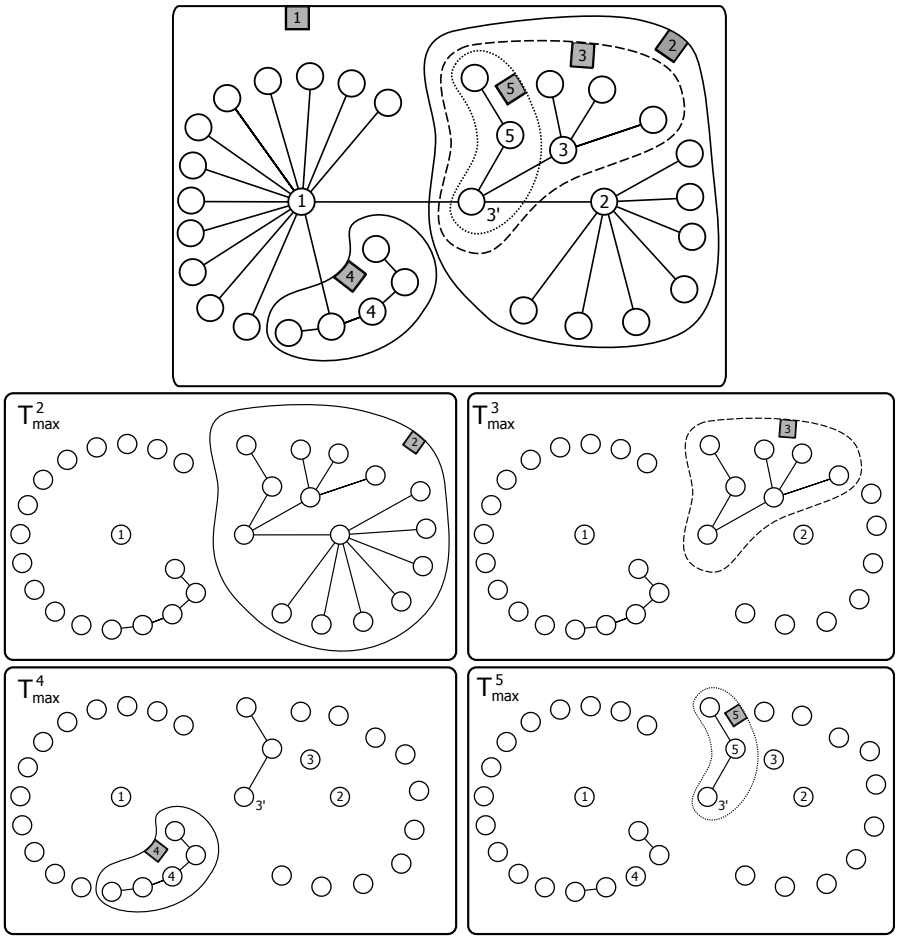

Figure 4.2: The SEL algorithm at work. The upper pane shows the initial tree $T = T^1_{max}$ (in the box tagged with "1"), and the subsequent subtrees T^2_{max}, T^3_{max}, T^4_{max}, and T^5_{max}. The left pane also shows the nodes selected by SEL in chronological order. The four lower panes show the connected components of $T \setminus L_t$ resulting from this selection. Observe that at the end of round 3, SEL detects the generation of fork node $3'$. This node gets stored, and is added to L_{SEL} at the end of the selection process.

maximizing $\frac{|V'|}{\left|\{(i,j)\in E : i\in V', j\in V\setminus V'\}\right|}$ is always included in a connected component of $T \setminus L$. Thus SEL places its queries in order to end up with a query set L^+_{SEL} such that the largest component of $T \setminus L^+_{SEL}$ is as small as possible.

SEL operates as follows. Let $L_t \subseteq L$ be the set including the first t nodes chosen by SEL, T^t_{max} be the largest connected component of $T \setminus L_{t-1}$, and $\sigma(T', i)$ be the size (number of nodes) of the largest component of the forest

$T' \setminus \{i\}$, where T' is any tree. At each step $t = 1, 2, \ldots,$ SEL simply picks the node $i_t \in T^t_{max}$ that minimizes $\sigma(T^t_{max}, i)$ over i and sets $L_t = L_{t-1} \cup \{i_t\}$. During this iterative construction, SEL also maintains a set containing all fork nodes generated in each step by adding nodes i_t to the sets L_{t-1}.[1] After the desired number of queries is reached (also counting the queries that would be caused by the stored fork nodes), SEL has terminated the construction of the query set L_{SEL}. The final query set L^+_{SEL}, obtained by adding all stored fork nodes to L_{SEL}, is then returned.

The **Prediction Algorithm pred** receives in input the labeled nodes of the 0-forked query set L^+_{SEL} and computes a mincut assignment. Since each component of $T \setminus L^+_{SEL}$ is either a 1-hinge-tree or a 2-hinge-tree, PRED is simple to describe and is also very efficient. The algorithm predicts all the nodes of hinge-tree $\mathcal{T}$ using the same label $\widehat{y}_{\mathcal{T}}$. This label is chosen according to the following two cases:

1. If $\mathcal{T}$ is a 1-hinge-tree, then $\widehat{y}_{\mathcal{T}}$ is set to the label of its unique connection node;

2. If $\mathcal{T}$ is a 2-hinge-tree and the labels of its two connection nodes are equal, then $\widehat{y}_{\mathcal{T}}$ is set to the label of its connection nodes, otherwise $\widehat{y}_{\mathcal{T}}$ is set as the label of the closer connection node (ties are broken arbitrarily).

In Section 4.6 we show that SEL requires overall $\mathcal{O}(|V| \log Q)$ time and $\mathcal{O}(|V|)$ memory gg for selecting Q query nodes. Also, we will see that the total running time taken by PRED for predicting all nodes in $V \setminus L$ is linear in $|V|$.

4.4 Analysis

For a given tree T, we denote by $m_A(L, \mathbf{y})$ the number of prediction mistakes that algorithm A makes on the labeled tree $(T, \mathbf{y})$ when given the query set L. Introduce the function

$$m_A(L, K) = \max_{\mathbf{y} : \Phi_T(\mathbf{y}) \le K} m_A(L, \mathbf{y})$$

[1] In Section 4.6 we will see that during each step $L_{t-1} \to L_t$ at most a single new fork node may be generated.

denoting the number of prediction mistakes made by A with query set L on all labeled trees with cutsize bounded by K. We will also find it useful to deal with the "lower bound" function $\textsc{lb}_L(K)$. This is the maximum expected number of mistakes that any prediction algorithm A can be forced to make on the labeled tree $(T, \mathbf{y})$ when the query set is L and the cutsize is not larger than K. [2]

We show that the number of mistakes made by PRED on any labeled tree when using the query set $L^+_{\textsc{sel}}$ satisfies

$$m_{\textsc{pred}}(L^+_{\textsc{sel}}, K) \leq 10\, \textsc{lb}_L(K)$$

for all query sets $L \subseteq V$ of size up to $\frac{1}{8}|L^+_{\textsc{sel}}|$. Though neither SEL nor PRED do know the actual cutsize of the labeled tree $(T, \mathbf{y})$, the combined use of these procedures is competitive against any algorithm that knows the cutsize budget K beforehand.

While this result implies the optimality (up to constant factors) of our algorithm, it does not relate the mistake bound to the cutsize, which is a clearly interpretable measure of the label regularity. In order to address this issue, we show that our algorithm also satisfies the bound

$$m_{\textsc{pred}}(L^+_{\textsc{sel}}, \mathbf{y}) \leq 4\, \Psi^*(L)\, \Phi_T(\mathbf{y})$$

for all query sets $L \subseteq V$ of size up to $\frac{1}{8}|L^+_{\textsc{sel}}|$. The proof of these results needs a number of preliminary lemmas.

Lemma 4.1 *For any tree $T = (V, E)$ it holds that $\min_{v \in V} \sigma(T, v) \leq \frac{1}{2}|V|$.*

PROOF. Let $i \in \operatorname{argmin}_{v \in V} \sigma(T, v)$. For the sake of contradiction, assume there exists a component $T_i = (V_i, E_i)$ of $T \setminus \{i\}$ such that $|V_i| > |V|/2$. Let s be the sum of the sizes all other components. Since $|V_i| + s = |V| - 1$, we know that $s \leq |V|/2 - 1$. Now let j be the node adjacent to i which belongs to V_i and $T_j = (V_j, E_j)$ be the largest component of $T \setminus \{j\}$. There are only two cases to consider: either $V_j \subset V_i$ or $V_j \cap V_i \equiv \emptyset$. In the first case, $|V_j| < |V_i|$. In the second case, $V_j \subseteq \{i\} \cup \left(T \setminus V_i\right)$, which implies $|V_j| \leq 1 + s \leq |V|/2 < |V_i|$. In both cases, $i \notin \operatorname{argmin}_{v \in V} \sigma(T, v)$, which provides the desired contradiction.

Lemma 4.2 *For all subsets $L \subset V$ of the nodes of a tree $T = (V, E)$ we have $\left|L^+\right| \leq 2|L|$.*

[2] Observe that function LB is used in also Section 2.2, but with a different meaning.

PROOF. Pick an arbitrary node of T and perform a depth-first visit of all nodes in T. This visit induces an ordering $\mathcal{T}_1, \mathcal{T}_2, \dots$ of the connected components in $T \setminus L$ based on the order of the nodes visited first in each component. Now let $\mathcal{T}_1', \mathcal{T}_2', \dots$ be such that each $\mathcal{T}_i'$ is a component of $\mathcal{T}_i$ extended to include all nodes of L adjacent to nodes in $\mathcal{T}_i$. Then the ordering implies that, for $i \geq 2$, $\mathcal{T}_i'$ shares exactly one node (which must be a leaf) with all previously visited trees. Since in any tree the number of nodes of degree larger than two must be strictly smaller than the number of leaves, we have $|\textsc{fork}(\mathcal{T}_i')| < |\Lambda_i|$ where, with slight abuse of notation, we denote by $\textsc{fork}(\mathcal{T}_i')$ the set of all fork nodes in subtree $\mathcal{T}_i'$. Also, we let Λ_i be the set of leaves of $\mathcal{T}_i'$. This implies that, for $i = 1, 2, \dots$, each fork node in $\textsc{fork}(\mathcal{T}_i')$ can be injectively associated with one of the $|\Lambda_i| - 1$ leaves of $\mathcal{T}_i'$ that are not shared with any of the previously visited trees. Since $|\textsc{fork}(L)|$ is equal to the sum of $|\textsc{fork}(\mathcal{T}_i)|$ over all indices i, this implies that $|\textsc{fork}(L)| \leq |L|$.

Lemma 4.3 *Let $L_{t-1} \subseteq L_{\textsc{sel}}$ be the set of the first $t-1$ nodes chosen by* SEL*. Given any tree $T = (V, E)$, the largest subtree of $T \setminus L_{t-1}$ contains no more than $\frac{2}{t}|V|$ nodes.*

PROOF. Recall that i_s denotes the s-th node selected by SEL during the incremental construction of the query set $L_{\textsc{sel}}$, and that $T^s_{\max}$ is the largest component of $T \setminus L_{s-1}$. The first t steps of the recursive splitting procedure performed by SEL can be associated with a splitting tree T' defined in the following way. The internal nodes of T' are $T^s_{\max}$, for $s \geq 1$. The children of $T^s_{\max}$ are the connected components of $T^s_{\max} \setminus \{i_s\}$, i.e., the subtrees of $T^s_{\max}$ created by the selection of i_s. Hence, each leaf of T' is bijectively associated with a tree in $T \setminus L_t$.

Let T'_{nol} be the tree obtained from T' by deleting all leaves. Each node of T'_{nol} is one of the t subtrees split by SEL during the construction of L_t. As $T^t_{\max}$ is split by i_t, it is a leaf in T'_{nol}. We now add a second child to each internal node s of T'_{nol} having a single child. This second child of s is obtained by merging all the subtrees belonging to leaves of T' that are also children of s. Let T'' be the resulting tree.

We now compare the cardinality of $T^t_{\max}$ to that of the subtrees associated with the leaves of T''. Let Λ be the set of all leaves of T'' and $\Lambda_{\text{add}} = T'' \setminus T'_{\text{nol}} \subset \Lambda$ be the set of all leaves added to T'_{nol} to obtain T''. First of all, note that

$|T^t_{max}|$ is not larger than the number of nodes in any leaf of T'_{nol}. This is because the selection rule of SEL ensures that T^t_{max} cannot be larger than any subtree associated with a leaf in T'_{nol}, since it contains no node selected before time t. In what follows, we write $|s|$ to denote the size of the forest or subtree associated with a node s of T''. We now prove the following claim:

Claim. For all $\ell \in \Lambda$, $|T^t_{max}| \leq |\ell|$, and for all $\ell \in \Lambda_{add}$, $|T^t_{max}| - 1 \leq |\ell|$.

Proof of Claim. The first part just follows from the observation that any $\ell \in \Lambda$ was split by SEL before time t. In order to prove the second part, pick a leaf $\ell \in \Lambda_{add}$. Let ℓ' be its unique sibling in T'' and let p be the parent of ℓ and ℓ', also in T''. Lemma 4.1 applied to the subtree p implies $|\ell'| \leq \frac{1}{2}|p|$. Moreover, since $|\ell| + |\ell'| = |p| - 1$, we obtain $|\ell| + 1 \geq \frac{1}{2}|p| \geq |\ell'| \geq |T^t_{max}|$, the last inequality using the first part of the claim. This implies $|T^t_{max}| - 1 \leq |\ell|$, and the claim is proven.

Let now $N(\Lambda)$ be the number of nodes in subtrees and forests associated with the leaves of T''. With each internal node of T'' we can associate a node of L_{SEL} which does not belong to any leaf in Λ. Moreover, the number $|T'' \setminus \Lambda|$ of internal nodes in T'' is bigger than the number $|\Lambda_{add}|$ of internal nodes of T'_{nol} to which a child has been added. Since these subtrees and forests are all distinct, we obtain $N(\Lambda) + |T'' \setminus \Lambda| < N(\Lambda) + |\Lambda_{add}| \leq |V|$. Hence, using the above claim we can write $N(\Lambda) \geq \Big(|\Lambda| - |\Lambda_{add}|\Big)|T^t_{max}| + |\Lambda_{add}|\Big(|T^t_{max}| - 1\Big)$, which implies $|T^t_{max}| \leq \Big(N(\Lambda) + |\Lambda_{add}|\Big)/|\Lambda| \leq |V|/|\Lambda|$. Since each internal node of T'' has at least two children, we have that $|\Lambda| \geq |T''|/2 \geq |T'_{nol}|/2 = t/2$. Hence, we can conclude that $|T^t_{max}| \leq 2|V|/t$.

4.4.1 Lower bounds

We now state and prove a lower bound on the number of mistakes that any prediction algorithm (even knowing the cutsize budget K) makes on any given tree, when the query set L is 0-forked. The bound depends on the following quantity: Given a tree $T(V, E)$, a node subset $L \subseteq V$ and an integer K, the **component function** $\Upsilon(L, K)$ is the sum of the sizes of the K largest components of $T \setminus L$, or $|V \setminus L|$ if $T \setminus L$ has less than K components.

Theorem 4.1 *For all trees $T = (V, E)$, for all 0-forked subsets $L^+ \subseteq V$, and for all cutsize budgets $K = 0, 1, \ldots, |V| - 1$, we have that $\mathrm{LB}_{L^+}(K) \geq \frac{1}{2}\Upsilon(L^+, K)$.*

PROOF. We describe an adversarial strategy causing any algorithm to make at least $\Upsilon(L^+, K)/2$ mistakes even when the cutsize budget K is known beforehand. Since L^+ is 0-forked, each component of $T \setminus L^+$ is a hinge-tree. Let F_{max} be the set of the K largest hinge-trees of $T \setminus L^+$, and $E(\mathcal{T})$ be the set of all edges in E incident to at least one node of a hinge-tree $\mathcal{T}$. The adversary creates at most one ϕ-edge[3] in each edge set $E(\mathcal{T}_1)$ for all 1-hinge-trees $\mathcal{T}_1 \in F_{max}$, exactly one ϕ-edge in each edge set $E(\mathcal{T}_2)$ for all 2-hinge-trees $\mathcal{T}_2 \in F_{max}$, and no ϕ-edges in the edge set $E(\mathcal{T})$ of any remaining hinge-tree $\mathcal{T} \notin F_{max}$. This is done as follows. By performing a depth-first visit of T, the adversary can always assign disagreeing labels to the two connection nodes of each 2-hinge-tree in F_{max}, and agreeing labels to the two connection nodes of each 2-hinge-tree not in F_{max}. Then, for each hinge-tree $\mathcal{T} \in F_{max}$, the adversary assigns a unique random label to all nodes of $\mathcal{T}$, forcing $|\mathcal{T}|/2$ mistakes in expectation. The labels of the remaining hinge-trees not in F_{max} are chosen in agreement with their connection nodes.

Remark 1 *Note that Theorem 4.1 holds for all query sets, not only those that are 0-forked, since any adversarial strategy for a query set L^+ can force at least the same mistakes on the subset $L \subseteq L^+$. Note also that it is not difficult to modify the adversarial strategy described in the proof of Theorem 4.1 in order to deal with algorithms that are allowed to adaptively choose the query nodes in L depending on the labels of the previously selected nodes. The adversary simply assigns the same label to each node in the query set and then forces, with the same method described in the proof, $\frac{1}{2}\Upsilon\left(L^+, \frac{K}{2}\right)$ mistakes in expectation on the $\frac{K}{2}$ largest hinge-trees. Thus there are at most two ϕ-edges in each edge set $E(\mathcal{T})$ for all hinge-trees $\mathcal{T}$, yielding at most K ϕ-edges in total. The resulting (slightly weaker) bound is $\mathrm{LB}_{L^+}(K) \geq \frac{1}{2}\Upsilon\left(L^+, \frac{K}{2}\right)$. Theorem 4.2 and Corollary 4.1 can also be easily rewritten in order to extend the results in this direction.*

4.4.2 Upper bounds

We now bound the total number of mistakes that PRED makes on any labeled tree when the queries are decided by SEL. We use Lemma 4.1 and 4.2, to-

[3] A ϕ-edge (i, j) is one where $y_i \neq y_j$.

gether with the two lemmas below, to prove that $m_{\text{PRED}}(L^+_{\text{SEL}}, K) \leq 10\,\text{LB}_L(K)$ for all cutsize budgets K and for all node subset $L \subseteq V$ such that $|L| \leq \frac{1}{8}|L^+_{\text{SEL}}|$.

Lemma 4.4 *For all labeled trees* $(T, \mathbf{y})$ *and for all 0-forked query sets* $L^+ \subseteq V$*, the number of mistakes made by* PRED *satisfies* $m_{\text{PRED}}(L^+, \mathbf{y}) \leq \Upsilon\Big(L^+, \Phi_T(\mathbf{y})\Big)$.

PROOF. As in the proof of Theorem 4.1, we first observe that each component of $T \setminus L^+$ is a hinge-tree. Let $E(\mathcal{T})$ be the set of all edges in E incident to nodes of a hinge-tree $\mathcal{T}$, and F_ϕ be the set of hinge-trees such that, for all $\mathcal{T} \in F_\phi$, at least one edge of $E(\mathcal{T})$ is a ϕ-edge. Since $E(\mathcal{T}) \cap E(\mathcal{T}') \equiv \emptyset$ for all $\mathcal{T}, \mathcal{T}' \in T \setminus L^+$, we have that $|F_\phi| \leq \Phi_T(\mathbf{y})$. Moreover, since for any $\mathcal{T} \notin F_\phi$ there are no ϕ-edges in $E(\mathcal{T})$, the nodes of $\mathcal{T}$ must be labeled as its connections nodes. This, together with the prediction rule of PRED, implies that PRED makes no mistakes over any of the hinge-trees $\mathcal{T} \notin F_\phi$. Hence, the number of mistakes made by PRED is bounded by the sum of the sizes of all hinge-trees $\mathcal{T} \in F_\phi$, which (by definition of Υ) is bounded by $\Upsilon\Big(L^+, \Phi_T(\mathbf{y})\Big)$.

The next lemma, whose proof is a bit involved, provides the relevant properties of the component function $\Upsilon(\cdot, \cdot)$. Figure 4.3 helps visualizing the main ingredients of the proof.

Lemma 4.5 *Given a tree* $T = (V, E)$*, for all node subsets* $L \subseteq V$ *such that* $|L| \leq \frac{1}{2}|L_{\text{SEL}}|$ *and for all integers* k*, we have: (a)* $\Upsilon(L_{\text{SEL}}, k) \leq 5\Upsilon(L, k)$*; (b)* $\Upsilon(L_{\text{SEL}}, 1) \leq \Upsilon(L, 1)$.

PROOF. We prove part (a) by constructing, via SEL, three bijective mappings $\mu_1, \mu_2, \mu_3 : \mathcal{P}_{\text{SEL}} \to \mathcal{P}_L$, where $\mathcal{P}_{\text{SEL}}$ is a suitable partition of $T \setminus L_{\text{SEL}}$, $\mathcal{P}_L$ is a subset of 2^V such that any $S \in \mathcal{P}_L$ is all contained in a single connected component of $T \setminus L$, and the union of the domains of the three mappings covers the whole set $T \setminus L_{\text{SEL}}$. The mappings μ_1, μ_2 and μ_3 are shown to satisfy, for all forests[4] $F \in \mathcal{P}_{\text{SEL}}$,

$$|F| \leq |\mu_1(F)|, \qquad |F| \leq 2|\mu_2(F)|, \qquad |F| \leq 2|\mu_3(F)| \quad .$$

[4]In this proof, $|\mu(A)|$ denotes the number of nodes in the set (of nodes) $\mu(A)$. Also, with a slight abuse of notation, for all forests $F \in \mathcal{P}_{\text{SEL}}$, we denote by $|F|$ the sum of the number of nodes in all trees of F. Finally, whenever $F \in \mathcal{P}_{\text{SEL}}$ contains a single tree, we refer to F as it were a tree, rather than a (singleton) forest containing only one tree.

Since each $S \in \mathcal{P}_L$ is all contained in a connected component of $T \setminus L$, this we will enable us to conclude that, for each tree $T' \in T \setminus L$, the forest of all trees $T \setminus L_{\text{SEL}}$ mapped (via any of these mappings) to any node subset of T' has at most five times the number of nodes of T'. This would prove the statement in (a).

The construction of these mappings requires some auxiliary definitions. We call ζ-component each connected component of $T \setminus L_{\text{SEL}}$ containing at least one node of L. Let i_t be the t-th node selected by SEL during the incremental construction of the query set L_{SEL}. We distinguish between four kinds of nodes chosen by SEL—see Figure 4.3 for an example.

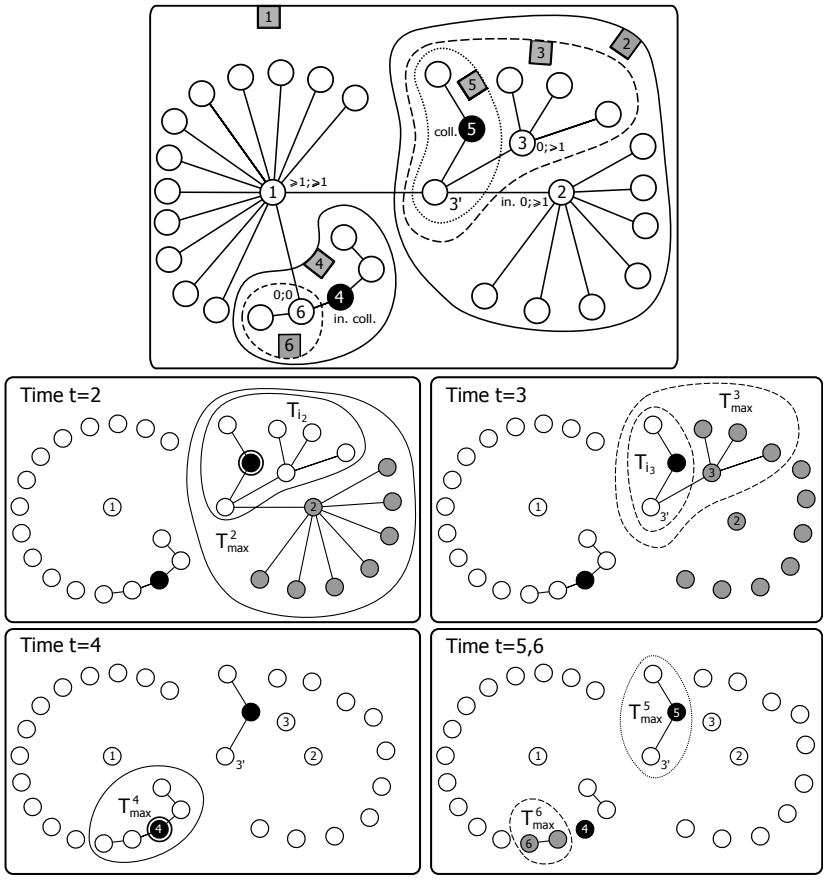

Figure 4.3: The upper pane illustrates the different kinds of nodes chosen by SEL. Numbers in the square tags indicate the first six subtrees T^t_{max}, and their associated nodes i_t, selected by SEL. Node i_1 is a $[\geq 1; \geq 1]$-node, i_2 is an initial $[0; \geq 1]$-node, i_3 is a (noninitial) $[0; \geq 1]$-node, i_4 is an initial collision node, i_5 is a (noninitial) collision node, and i_6 is a $[0; 0]$-node. As in Figure 4.2, we denote by $3'$ the fork node generated by the inclusion of i_3 into L_{SEL}. Note that node i_6 may be chosen arbitrarily among the four nodes in $T^4_{max} \setminus i_4$. The two black nodes are the set of nodes we are competing against, i.e., the nodes in the query set L. Forest $T \setminus L$ is made up of one large subtree and two small subtrees. Time $t = 2$: Trees T^2_{max} and T_{i_2} are shown. As explained in the proof, $|T_{i_2}| \leq |T^2_{max} \setminus T_{i_2}|$. The circled black node is captured by i_2. The nodes of tree $T^2_{max} \setminus T_{i_2}$ are shaded, and can be used for mapping any ζ-component through μ_2. In the lower panes we illustrate some steps of the proof of Lemma 4.5, with reference to the upper pane. Time $t = 3$: Trees T^3_{max} and T_{i_3} are shown. Again, one can easily verify that $|T_{i_3}| \leq |T^3_{max} \setminus T_{i_3}|$. As before, the nodes of $T^3_{max} \setminus T_{i_3}$ are shaded, and can be used for mapping any ζ-component via μ_2. The reader can see that, according to the injectivity of μ_2, these grey nodes are well separated from the ones in $T^2_{max} \setminus T_{i_2}$. Time $t = 4$: T^4_{max} and the initial collision node i_4 are depicted. The latter is enclosed in a circled black node since it captures itself. Time $t = 5, 6$: We depicted trees T^5_{max} and T^6_{max}, together with nodes i_5 and i_6. Node i_5 is a collision node, which is not initial since it was already captured by the $[0; \geq 1]$-node i_2. Node i_6 is a $[0; 0]$ node, so that the whole tree T^6_{max} is completely included in a component (the largest, in this case) of $T \setminus L$. Tree T^6_{max} can be used for mapping via μ_3 any ζ-component. The resulting forest $T \setminus L_6$ includes several single-node trees and one two-node tree. If i_6 is the last node selected by L_{SEL}, then each component of $T \setminus L_6$ can be exploited by mapping μ_1, since in this specific case none of these components contains nodes of L, i.e., there are no ζ-components left.

Node i_t is:

1. A *collision node* if it belongs to $L_{SEL} \cap L$;
2. a $[0; 0]$-*node* if, at time t, the tree T^t_{max} does not contain any node of L;
3. a $[0; \geq 1]$-*node* if, at time t, the tree T^t_{max} contains $k \geq 1$ nodes $j_1, \ldots, j_k \in L$ all belonging to the same connected component of $T^t_{max} \setminus \{i_t\}$;
4. a $[\geq 1; \geq 1]$-*node* if $i_t \notin L$ and, at time t, the tree T^t_{max} contains

$k \geq 2$ nodes $j_1, \ldots, j_k \in L$, which do not belong to the same connected component of $T^t_{max} \setminus \{i_t\}$.

We now turn to building the three mappings.

μ_1 simply maps each tree $T' \in T \setminus L_{\text{SEL}}$ that is *not* a ζ-component to the node set of T' itself. This immediately implies $|F| \leq |\mu_1(F)|$ for all forests F (which are actually single trees) in the domain of μ_1. Mappings μ_2 and μ_3 deal with the ζ-components of $T \setminus L_{\text{SEL}}$. Let Z be the set of all such ζ-components, and denote by $V_{0;0}$, $V_{0;1}$, and $V_{1;1}$ the set of all $[0;0]$-nodes, $[0;\geq 1]$-nodes, and $[\geq 1;\geq 1]$-nodes, respectively. Observe that $|V_{1;1}| < |L|$. Combined with the assumption $|L_{\text{SEL}}| \geq 2|L|$, this implies that $|V_{0;0}|+|V_{0;1}|$ plus the total number of collision nodes must be larger than $|L|$; as a consequence, $|V_{0;0}| + |V_{0;1}| > |Z|$. Each node $i_t \in V_{0;1}$ chosen by SEL splits the tree T^t_{max} into one component T_{i_t} containing at least one node of L and one or more components all contained in a single tree T'_{i_t} of $T \setminus L$. Now mapping μ_2 can be constructed incrementally in the following way. For each $[0;\geq 1]$-node selected by SEL at time t, μ_2 sequentially maps any ζ-component generated to the set of nodes in $T^t_{max} \setminus T_{i_t}$, the latter being just a subset of a component of $T \setminus L$. A future time step $t' > t$ might feature the selection of a new $[0;\geq 1]$-node within T_{i_t}, but mapping μ_2 would cover a different subset of such component of $T \setminus L$. Now, applying Lemma 4.1 to tree T^t_{max}, we can see that $|T^t_{max} \setminus T_{i_t}| \geq |T^t_{max}|/2$. Since the selection rule of SEL guarantees that the number of nodes in T^t_{max} is larger than the number of nodes of any ζ-component, we have $|F| \leq 2|\mu_2(F)|$, for any ζ-component F considered in the construction of μ_2.

Mapping μ_3 maps all the remaining ζ-components that are not mapped through μ_2. Let $\sim$ be an equivalence relation over $V_{0;0}$ defined as follows: $i \sim j$ iff i is connected to j by a path containing only $[0;0]$-nodes and nodes in $V \setminus (L_{\text{SEL}} \cup L)$. Let $i_{t_1}, i_{t_2}, \ldots, i_{t_k}$ be the sequence of nodes of any given equivalence class $[C]_\sim$, sorted according to SEL's chronological selection. Lemma 4.3 applied to tree $T^{t_1}_{max}$ shows that $|T^{t_k}_{max}| \leq 2|T^{t_1}_{max}|/k$. Moreover, the selection rule of SEL guarantees that the number of nodes of $T^{t_k}_{max}$ cannot be smaller than the number of nodes of any ζ-component. Hence, for each equivalence class $[C]_\sim$ containing k nodes of type $[0;0]$, we map through μ_3 a set F_ζ of k arbitrarily chosen ζ-components to $T^{t_1}_{max}$. Since the size of each ζ-component is $\leq |T^{t_k}_{max}|$, we can write $|F_\zeta| \leq k|T^{t_k}_{max}| \leq 2|T^{t_1}_{max}|$, which implies $|F_\zeta| \leq 2|\mu_3(F_\zeta)|$ for all F_ζ in the domain of μ_3. Finally, observe that the

number of ζ-components that are not mapped through μ_2 cannot be larger than $|V_{0;0}|$, thus the union of mappings μ_2 and μ_3 do actually map all ζ-components. This, in turn, implies that the union of the domains of the three mappings covers the whole set $T \setminus L_{\text{SEL}}$, thereby concluding the proof of part (a).

The proof of (b) is built on the definition of collision nodes, $[0;0]$-nodes, $[0;\geq 1]$-nodes and $[\geq 1;\geq 1]$-nodes given in part (a). Let $L_t \subseteq L_{\text{SEL}}$ be the set of the first t nodes chosed by SEL. Here, we make a further distinction within the collision and $[0;\geq 1]$-nodes. We say that during the selection of node $i_t \in V_{0;1}$, the nodes in $L \cap T^t_{\max}$ are *captured* by i_t. This notion of capture extends to collision nodes by saying that a collision node $i_t \in L \cap L_{\text{SEL}}$ just *captures itself*. We say that i_t is an *initial* $[0;\geq 1]$-node (resp., *initial* collision node) if i_t is a $[0;\geq 1]$-node (resp., collision node) such that the whole set of nodes in L captured by i_t contains no nodes captured so far. See Figure 4.3 for reference. The simple observation leading to the proof of part (b) is the following. If i_t is a $[0;0]$-node, then $T^t_{\max}$ cannot be larger than the component of $T \setminus L$ that contains $T^t_{\max}$, which in turn cannot be larger than $\Upsilon(L,1)$. This would already imply $\Upsilon(L_{t-1},1) \leq \Upsilon(L,1)$. Let now i_t be an initial $[0;\geq 1]$-node and T_{i_t} be the unique component of $T^t_{\max} \setminus \{i_t\}$ containing one or more nodes of L. Applying Lemma 4.1 to tree $T^t_{\max}$ we can see that $|T_{i_t}|$ cannot be larger than $|T^t_{\max} \setminus T_{i_t}|$, which in turn cannot be larger than $\Upsilon(L,1)$. If at time $t' > t$ the procedure SEL selects $i_{t'} \in T_{i_t}$ then $|T^{t'}_{\max}| \leq |T_{i_t}| \leq \Upsilon(L,1)$. Hence, the maximum integer q such that $\Upsilon(L_q,1) > \Upsilon(L,1)$ is bounded by the number of $[\geq 1;\geq 1]$-nodes plus the number of initial $[0;\geq 1]$-nodes plus the number of initial collision nodes. We now bound this sum as follows. The number of $[\geq 1;\geq 1]$-nodes is clearly bounded by $|L|-1$. Also, any initial $[0;\geq 1]$-node or initial collision node selected by SEL captures at least a new node in L, thereby implying that the total number of initial $[0;\geq 1]$-node or initial collision node must be $\leq |L|$. After $q = 2|L|-1$ rounds, we are sure that the size of the largest tree of $T^q_{\max}$ is not larger than the size of the largest component of $T \setminus L$, i.e., $\Upsilon(L,1)$.

We now put the above lemmas together to prove our main result concerning the number of mistakes made by PRED on the query set chosen by SEL.

Theorem 4.2 *For all trees* T *and all cutsize budgets* K*, the number of*

mistakes made by PRED *on the query set* L^+_{SEL} *satisfies*

$$m_{\text{PRED}}(L^+_{\text{SEL}}, K) \leq \min_{L \subseteq V : |L| \leq \frac{1}{8}|L^+_{\text{SEL}}|} 10\,\text{LB}_L\Big(K\Big) \ .$$

PROOF. Pick any $L \subseteq V$ such that $|L| \leq \frac{1}{8}|L^+_{\text{SEL}}|$. Then

$$m_{\text{PRED}}(L^+_{\text{SEL}}, K) \overset{\text{(Lem. 4.4)}}{\leq} \Upsilon(L^+_{\text{SEL}}, K) \overset{\text{(A)}}{\leq} \Upsilon(L_{\text{SEL}}, K) \overset{\text{(Lem. 4.5 (a))}}{\leq}$$

$$5\Upsilon(L^+, K) \overset{\text{(Thm. 4.1)}}{\leq} 10\,\text{LB}_{L^+}(K) \overset{\text{(B)}}{\leq} 10\,\text{LB}_L(K) \ .$$

Inequality (A) holds because $L_{\text{SEL}} \subseteq L^+_{\text{SEL}}$, and thus $T \setminus L^+_{\text{SEL}}$ has connected components of smaller size than L_{SEL}. In order to apply Lemma 4.5 (a), we need the condition $|L^+| \leq \frac{1}{2}|L_{\text{SEL}}|$. This condition is seen to hold after combining Lemma 4.2 with our assumptions: $|L^+| \leq 2|L| \leq \frac{1}{4}|L^+_{\text{SEL}}| \leq \frac{1}{2}|L_{\text{SEL}}|$. Finally, inequality (B) holds because any adversarial strategy using query set L can also be used with the larger query set $L^+ \supseteq L$.

Note also that Theorem 4.1 and Lemma 4.4 imply the following statement about the optimality of PRED over 0-forked query sets.

Corollary 4.1 *For all trees* T*, for all cutsize budgets* K*, and for all 0-forked query sets* $L^+ \subseteq V$*, the number of mistakes made by* PRED *satisfies* $m_{\text{PRED}}(L^+, K) \leq 2\text{LB}_{L^+}\Big(K\Big)$.

In the rest of this section we derive a more intepretable bound on $m_{\text{PRED}}(L^+, \mathbf{y})$ based on the function Ψ^* introduced in [41]. To this end, we prove that L_{SEL} minimizes Ψ^* up to constant factors, and thus is an optimal query set according to the analysis of [41].

For any subset $V' \subseteq V$, let $\Gamma(V', V \setminus V')$ be the number of edges between nodes of V' and nodes of $V \setminus V'$. Using this notation, we can write

$$\Psi^*(L) = \max_{\emptyset \neq V' \subseteq V \setminus L} \frac{|V'|}{\Gamma(V', V \setminus V')} \ .$$

Lemma 4.6 *For any tree* $T = (V, E)$ *and any* $L \subseteq V$ *the following holds.*

(a) A maximizer of $\frac{|V'|}{\Gamma(V', V \setminus V')}$ *exists which is included in the node set of a single component of* $T \setminus L$*;*

(b) $\Psi^*(L) \leq \Upsilon(L, 1)$.

PROOF. Let V'_{max} be any maximizer of $\frac{|V'|}{\Gamma(V', V \setminus V')}$. For the sake of contradiction, assume that the nodes of V'_{max} belong to $k \geq 2$ components $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_k \in T \setminus L$. Let $V'_i \subset V'_{max}$ be the subset of nodes included in the node set of $\mathcal{T}_i$, for $i = 1, \ldots, k$. Then $|V'| = \sum_{i \leq k} |V'_i|$ and $\Gamma(V', V \setminus V') = \sum_{i \leq k} \Gamma(V'_i, V \setminus V'_i)$. Now let $i^* = \text{argmax}_{i \leq k} |V'_i| / \Gamma(V'_i, V \setminus V'_i)$. Since $\left(\sum_i a_i\right) \Big/ \left(\sum_i b_i\right) \leq \max_i a_i / b_i$ for all $a_i, b_i \geq 0$, we immediately obtain $\Psi(V'_{i^*}) \geq \Psi(V'_{max})$, contradicting our assumption. This proves (a). Part (b) is an immediate consequence of (a).

Lemma 4.7 *For any tree* $T = (V, E)$ *and any* 0*-forked subset* $L^+ \subseteq V$ *we have* $\Upsilon(L^+, 1) \leq 2\Psi^*(L^+)$.

PROOF. Let $\mathcal{T}_{max}$ be the largest component of $T \setminus L^+$ and V_{max} be its node set. Since L^+ is a 0-forked query set, $\mathcal{T}_{max}$ must be either a 1-hinge-tree or a 2-hinge-tree. Since the only edges that connect a hinge-tree to external nodes are the edges leading to connection nodes, we find that $\Gamma(V_{max}, V \setminus V_{max}) \leq 2$. We can now write

$$\Psi^*(L^+) = \max_{\emptyset \neq V' \subseteq V \setminus L^+} \frac{|V'|}{\Gamma(V', V \setminus V')} \geq \frac{|V_{max}|}{\Gamma(V_{max}, V \setminus V_{max})} \geq \frac{|V_{max}|}{2} = \frac{\Upsilon(L^+, 1)}{2}$$

thereby concluding the proof.

Lemma 4.8 *For any tree* $T = (V, E)$ *and any subset* $L \subseteq V$ *we have* $\Psi^*(L^+) \leq \Psi^*(L)$.

PROOF. Let V'_{max} be any set maximizing $\Psi^*(L^+)$. Since $V'_{max} \in V \setminus L^+$, V'_{max} cannot contain any node of $L \subseteq L^+$. Hence

$$\Psi^*(L) = \max_{\emptyset \neq V' \subseteq V \setminus L} \frac{|V'|}{\Gamma(V', V \setminus V')} \geq \frac{|V'_{max}|}{\Gamma(V'_{max}, V \setminus V'_{max})} = \Psi^*(L^+)$$

which concludes the proof.

We now put together the previous lemmas to show that the query set L_{SEL} minimizes Ψ^* up to constant factors.

Theorem 4.3 *For any tree* $T = (V, E)$ *we have*

$$\Psi^*(L_{\text{SEL}}) \leq \min_{L \subseteq V : |L| \leq \frac{1}{4}|L_{\text{SEL}}|} 2\Psi^*(L)$$

PROOF. Let L be a query set such that $|L| \leq |L_{\text{SEL}}|/4$. Then we have the following chain of inequalities:

$$\Psi^*(L_{\text{SEL}}) \overset{\text{(Lemma 4.6 (b))}}{\leq} \Upsilon(L_{\text{SEL}}, 1) \overset{\text{(Lemma 4.5 (b))}}{\leq}$$

$$\Upsilon(L^+, 1) \overset{\text{(Lemma 4.7)}}{\leq} 2\Psi^*(L^+) \overset{\text{(Lemma 4.8)}}{\leq} 2\Psi^*(L) \ .$$

In order to apply Lemma 4.5 (b), we need the condition $|L^+| \leq \frac{1}{2}|L_{\text{SEL}}|$. This condition holds because, by Lemma 4.2, $|L^+| \leq 2|L| \leq \frac{1}{2}|L_{\text{SEL}}|$.

Finally, as promised, the following corollary contains an interpretable mistake bound for PRED run with a query set returned by SEL.

Corollary 4.2 *For any labeled tree $(T, \mathbf{y})$, the number of mistakes made by* PRED *when run with query set L^+_{SEL} satisfies*

$$m_{\text{PRED}}(L^+_{\text{SEL}}, \mathbf{y}) \leq 4 \min_{L \subseteq V : |L| \leq \frac{1}{8}|L^+_{\text{SEL}}|} \Psi^*(L)\, \Phi_T(\mathbf{y}) \ .$$

PROOF. Observe that PRED assigns labels to nodes in $V \setminus L^+_{\text{SEL}}$ so as to minimize the resulting cutsize given the labels in the query set L^+_{SEL}. We can then invoke [41, Lemma 1], which bounds the number of mistakes made by the mincut strategy in terms of the functions Ψ^* and the cutsize. This yields

$$m_{\text{PRED}}(L^+_{\text{SEL}}, \mathbf{y}) \overset{\text{[41, Lemma 1]}}{\leq} 2\,\Psi^*(L^+_{\text{SEL}})\, \Phi_T(\mathbf{y}) \overset{\text{(A)}}{\leq}$$

$$2\,\Psi^*(L_{\text{SEL}})\, \Phi_T(\mathbf{y}) \overset{\text{(Theorem 4.3)}}{\leq} 4\,\Psi^*(L)\, \Phi_T(\mathbf{y}) \ .$$

Inequality (A) holds because $L_{\text{SEL}} \subseteq L^+_{\text{SEL}}$, and thus $T \setminus L^+_{\text{SEL}}$ has connected components of smaller size than L_{SEL}. In order to apply Theorem 4.3, we need the conditon $|L| \leq \frac{1}{4}|L_{\text{SEL}}|$, which follows from a simple combination of Lemma 4.2 and our assumptions: $|L| \leq \frac{1}{8}|L^+_{\text{SEL}}| \leq \frac{1}{4}|L_{\text{SEL}}|$.

Remark 2 *A mincut algorithm exists which efficiently predicts even when the query set L is not 0-forked (thereby gaining a factor of 2 in the cardinality of the competing query sets L – see Theorem 4.2 and Corollary 4.2). This algorithm is a "batch" variant of the* TREEOPT *algorithm analyzed in Chapter 2. The algorithm can be implemented in such a way that the total time for predicting $|V| - |L|$ labels is $\mathcal{O}(|V|)$.*

4.4.3 Automatic calibration of the number of queries

A key aspect to the query selection task is deciding when to stop asking queries. Since the more queries are asked the less mistakes are made afterwards, a reasonable way to deal with this trade-off is to minimize the number of queries issued during the selection phase plus the number of mistakes made during the prediction phase. For a given pair $A = \langle S, P \rangle$ of prediction and selection algorithms, we denote by $[q+m]_A$ the sum of queries made by S and prediction mistakes made by P. Similarly to m_A introduced in Section 4.4, $[q+m]_A$ has to scale with the cutsize $\Phi_T(\mathbf{y})$ of the labeled tree $(T, \mathbf{y})$ under consideration.

As a simple example of computing $[q+m]_A$, consider a line graph $T = (V, E)$. Since each query set on T is 0-forked, Theorem 4.1 and Corollary 4.1 ensure that an optimal strategy for selecting the queries in T is choosing a sequence of nodes such that the distance between any pair of neighbor nodes in L is equal. The total number of mistakes that can be forced on $V \setminus L$ is, up to a constant factor, $\Big(|V|/|L|\Big)\Phi_T(\mathbf{y})$. Hence, the optimal value of $[q+m]_A$ is about

$$|L| + \frac{|V|}{|L|}\Phi_T(\mathbf{y}) \ . \tag{4.1}$$

Minimizing the above expression over $|L|$ clearly requires knowledge of $\Phi_T(\mathbf{y})$, which is typically unavailable. In this section we investigate a method for choosing the number of queries when the labeling is known to be sufficiently regular, that is when a bound K is known on the cutsize $\Phi_T(\mathbf{y})$ induced by the adversarial labeling.[5]

We now show that when a bound K on the cutsize is known, a simple modification of SEL(we call it SEL$\star$) exists which optimizes the $[q+m]_A$ criterion. This means that the combination of SEL$\star$ and PRED can trade-off optimally (up to constant factors) queries against mistakes.

Given a selection algorithm S and a prediction algorithm P, define

[5] In [1] a labeling $\mathbf{y}$ of a graph G is said to be α-balanced if, after the elimination of all ϕ-edges, each connected component of G is not smaller than $\alpha|V|$ for some known constant $\alpha \in (0,1)$. In the case of labeled trees, the α-balancing condition is stronger than our regularity assumption. This is because any α-balanced labeling $\mathbf{y}$ implies $\Phi_T(\mathbf{y}) \leq 1/\alpha - 1$. In fact, getting back to the line graph example, we immediately see that, if $\mathbf{y}$ is α-balanced, then the optimal number of queries $|L|$ is order of $\sqrt{|V|(1/\alpha - 1)}$, which is also $\inf_A [q+m]_A$.

$[q+m]_{\langle S,P\rangle}$ by

$$[q+m]_{\langle S,P\rangle} = \min_{Q\geq 1}\Big(Q + m_P(L_{S(Q)}, K)\Big)$$

where $L_{S(Q)}$ is the query set output by S given query budget Q, and $m_P(L_{S(Q)}, K)$ is the maximum number of mistakes made by P with query set $L_{S(Q)}$ on any labeling $\mathbf{y}$ with $\Phi_T(\mathbf{y}) \leq K$ —see definition in Section 4.4. Define also $[q+m]_{\text{OPT}} = \inf_{S,P}[q+m]_{\langle S,P\rangle}$, where $\text{OPT} = \langle S^*, P^*\rangle$ is an optimal pair of selection and prediction algorithms. If SEL knows the size of the query set L^* selected by S^*, so that SEL can choose a query budget $Q = 8|L^*|$, then a direct application of Theorem 4.2 guarantees that $|L^+_{\text{SEL}}| + m_{\text{PRED}}(L^+_{\text{SEL}}, K) \leq 10\,[q+m]_{\text{OPT}}$. We now show that SEL$\star$, the announced modification of SEL, can efficiently search for a query set size Q such that $Q + m_{\text{PRED}}(L^+_{\text{SEL}(Q)}, K) = \mathcal{O}\Big([q+m]_{\text{OPT}}\Big)$ when only K, rather than $|L^*|$, is known. In fact, Theorem 4.1 and Corollary 4.1 ensure that $m_{\text{PRED}}(L^+_{\text{SEL}}, K) = \Theta\Big(\Upsilon(L^+_{\text{SEL}}, K)\Big)$. When K is given as side information, SEL$\star$ can operate as follows. For each $t \leq |V|$, the algorithm builds the query set L^+_t and computes $\Upsilon(L^+_t, K)$. Then it finds the smallest value t^* minimizing $t + \Upsilon(L^+_t, K)$ over all $t \leq |V|$, and selects $L_{\text{SEL}\star} \equiv L_{t^*}$. We stress that the above is only possible because the algorithm can estimate within constant factors its own future mistake bound (Theorem 4.1 and Corollary 4.1), and because the combination of SEL and PRED is competitive against all query sets whose size is a constant fraction of $|L^+_{\text{SEL}}|$ —see Theorem 4.2. Putting together, we have shown the following result.

Theorem 4.4 *For all trees $(T, \mathbf{y})$, for all cutsize budgets K, and for all labelings $\mathbf{y}$ such that $\Phi_T(\mathbf{y}) \leq K$, the combination of* SEL$\star$ *and* PRED *achieves $|L_{\text{SEL}\star}| + m_{\text{PRED}}(L^+_{\text{SEL}\star}, K) = \mathcal{O}\Big([q+m]_{\text{OPT}}\Big)$ when K is given to* SEL$\star$ *as input.*

Just to give a few simple examples of how SEL$\star$ works, consider a star graph. It is not difficult to see that in this case $t^* = 1$ *independent* of K, i.e., SEL$\star$ always selects the center of the star, which is intuitively the optimal choice. If T is the "binary system" mentioned in the introduction of this chapter, then $t^* = 2$ and SEL$\star$ always selects the centers of the two stars, again independent of K. At the other extreme, if T is a line graph, then SEL$\star$ picks the query nodes in such a way that the distance between two consecutive nodes of L in T is (up to a constant factor) equal to $\sqrt{|V|/K}$. Hence $|L| = \Theta(\sqrt{|V|K})$, which is the minimum of (4.1) over $|L|$ when $\Phi_T(\mathbf{y}) \leq K$.

4.5 On the prediction of general graphs

In this section we provide a general lower bound for prediction on arbitrary labeled graphs $(G, \mathbf{y})$. We then contrast this lower bound to some results contained in Afshani et al. [1].

Let $\Phi_G^R(\mathbf{y})$ be the sum of the effective resistances (see Se) on the ϕ-edges of $G = (V, E)$. The theorem below shows that any prediction algorithm using any query set L such that $|L| \leq \frac{1}{4}|V|$ makes at least order of $\Phi_G^R(\mathbf{y})$ mistakes. This lower bound holds even if the algorithm is allowed to use a randomized adaptive strategy for choosing the query set L, that is, a randomized strategy where the next node of the query set is chosen after receiving the labels of all previously chosen nodes.

Theorem 4.5 *Given a labeled graph $(G, \mathbf{y})$, for all $K \leq |V|$, there exists a randomized labeling strategy such that for all prediction algorithms A choosing a query set of size $|L| \leq \frac{1}{4}|V|$ via a possibly randomized adaptive strategy, the expected number of mistakes made by A on the remaining nodes $V \setminus L$ is at least $K/8$, while $\Phi_G^R(\mathbf{y}) < K$.*

PROOF. Define the *cost* $c(i)$ of any node i as the sum of the effective resistances between i and all adjacent nodes:

$$c(i) = \sum_{j:\ (i,j)\in E} r_{i,j} \ .$$

Let V' be the subset of V formed by the $|V|/2$ nodes having the smallest cost.

Since, for all $(i, j) \in E$, $r_{i,j}$ is equal to the probability of including the edge in a uniformly generated random spanning tree, we know that $\sum_{(i,j)\in E} r_{i,j} = |V| - 1$. Hence we have

$$\sum_{i\in V} c(i) = \sum_{i\in V} \sum_{j:\ (i,j)\in E} r_{i,j} = 2|V| - 2 \ ,$$

because for each edge $(i, j) \in E$ the resistance $r_{i,j}$ contributes twice in the sum. This immediately implies that $c(i) < 4$ for all $i \in V'$. As a matter of fact, if there were a vertex $i \in V'$ such that $c(i) \geq 4$, then $c(j) \geq 4$ for all $j \in V \setminus V'$ by definition of V', which would in turn imply $\sum_{i\in V} c(i) \geq 2|V|$.

Now let $\mathbf{y}$ be the labeling in which $K/4$ nodes chosen uniformly at random from the node subset V' are labeled with $+1$, and all the remaining nodes of V are labeled with -1. Since each edge connecting nodes with different labels must be incident to a node labeled with $+1$, we have

$$\Phi^R_G(\mathbf{y}) \leq \sum_{i:\ y_i=+1} c(i) < 4 \cdot |\{i \in V :\ y_i = +1\}| < K\ .$$

Since $|L| \leq |V|/4 = |V'|/2$ and $|\{i \in V :\ y_i = +1\}| = K/4 \leq |V|/4 = |V'|/2$, $V' \setminus L$ contains in expectation at least $K/8$ nodes with label $+1$ and at least $K/8$ nodes with label -1. Hence, every algorithm makes at least $K/8$ mistakes on the node subset V', no matter how it selects the query set L. $\square$

The above lower bound appears to contradict an argument by Afshani et al. [1, Section 5]. This argument establishes that for any $\varepsilon > 0$ there exists a randomized algorithm using at most $K\ln(3/\varepsilon)+K\ln(|V|/K)+\mathcal{O}(K)$ queries on any given graph $G = (V, E)$ with cutsize K, and making at most $\varepsilon|V|$ mistakes on the remaining vertices. This contradiction is easily obtained through the following simple counterexample: assume G is a line graph where all node labels are $+1$ but for $K = o\left(|V|/\ln|V|\right)$ randomly chosen nodes, which are also given random labels. For all $\varepsilon = o\left(\frac{K}{|V|}\right)$, the above argument implies that order of $K\ln|V| = o(|V|)$ queries are sufficient to make at most $\varepsilon|V| = o(K)$ mistakes on the remaining nodes, among which $\Omega(K)$ have random labels —which is clearly impossible.

4.6 Efficient Implementation

In this section we describe an efficient implementation of SEL and PRED. We will show that the total time needed for selecting Q queries is $\mathcal{O}(|V|\log Q)$, the total time for predicting $|V| - Q$ nodes is $\mathcal{O}(|V|)$, and that the overall memory space is again $\mathcal{O}(|V|)$.

In order to locate the largest subtree of $T \setminus L_{t-1}$, the algorithm maintains a priority deque [55] D containing at most Q items. This data-structure enables to find (resp. eliminate) the item with the smallest or largest key in time $\mathcal{O}(1)$ (resp., time $\mathcal{O}(\log Q)$). In addition, the insertion of a new element takes time $\mathcal{O}(\log Q)$.

Each item in D has two records: a reference to a node in T and the priority key associated with that node. Just before the selection of the[6] t-th query node i_t, the Q references point to nodes contained in the Q largest subtrees in $T \setminus L_{t-1}$, while the corresponding keys are the sizes of such subtrees. Hence at time t the item top of D having the largest key points to a node in T^t_{max}.

First, during an initialization step, SEL creates, for each edge $(i,j) \in E$, a directed edge $[i,j]$ from i to j and the twin directed edge $[j,i]$ from j to i. During the construction of L_{SEL} the algorithm also stores and maintains the current size $\sigma(D)$ of D, i.e., the total number of items contained in D. We first describe the way SEL finds node i_t in T^t_{max}. Then we will see how SEL can efficiently augment the query set L_{SEL} to obtain L^+_{SEL}.

Starting from the node r of T^t_{max} referred to by[7] D, SEL performs a depth-first visit of T^t_{max}, followed by the elimination of the item with the largest key in D. For the sake of simplicity, consider T^t_{max} as rooted at node r. Given any edge (i,j), we let T_i and T_j be the two subtrees obtained from T^t_{max} after removing edge (i,j), where T_i contains node i, and T_j contains node j. During each backtracking step of the depth-first visit from a node i to a node j, SEL stores the number of nodes $|T_i|$ contained in T_i. This number gets associated with $[j,i]$. Observe that this task can be accomplished very efficiently, since $|T_i|$ is equal to 1 plus the number of nodes of the union of $T_{c(i)}$ over all children $c(i)$ of i. These numbers can be recursively calculated by summing the size values that SEL associates with all direct edges $[i,c(i)]$ in the previous backtracking steps. Just after storing the value $|T_i|$, the algorithm also stores $|T_j| = |T^t_{max}| - |T_i|$ and associates this value with the twin directed edge $[i,j]$. The size of T^t_{max} is then stored in D as the key record of the pointer to node r.

It is now important to observe that the quantity $\sigma(T^t_{max}, i)$ used by SEL (see Section 4.3) is simply the largest key associated with the directed edges $[i,j]$ over all j such that (i,j) is an edge of T^t_{max}. Hence, a new depth-first visit is enough to find in time $\mathcal{O}(|T^t_{max}|)$ the t-th node $i_t = \arg\min_{i \in T^t_{max}} \sigma(T^t_{max}, i)$ selected by SEL. Let $N(i_t)$ be the set of all nodes adjacent to node i_t in T^t_{max}. For all nodes $i' \in N(i_t)$, SEL compares $|T_{i'}|$ to the smallest key bottom stored in D. We have three cases:

1. If $|T_{i'}| \leq$ bottom and $\sigma(D) \geq Q - t$ then the algorithm does nothing,

[6] If $t = 1$ the priority deque D is empty.

[7] In the initial step $t = 1$ (i.e., when $T^t_{max} \equiv T$) node r can be chosen arbitrarily .

since $T_{i'}$ (or subtrees thereof) will never be largest in the subsequent steps of the construction of L_{SEL}, i.e., there will not exist any node $i_{t'}$ with $t' > t$ such that $i_{t'} \in T_{i'}$.

2. If $|T_{i'}| \leq \text{bottom}$ and $\sigma(D) < Q - t$, or if $|T_{i'}| > \text{bottom}$ and $\sigma(D) < Q$ then SEL inserts a pointer to i' together with the associated key $|T_{i'}|$. Note that, since D is not full (i.e., $\sigma(D) < Q$), the algorithm need not eliminate any item in D.
3. If $|T_{i'}| > \text{bottom}$ and $\sigma(D) = Q$ then SEL eliminates from D the item having the smallest key, and inserts a pointer to i', together with the associated key $|T_{i'}|$.

Finally, SEL eliminates node i_t and all edges (both undirected and directed) incident to it. Note that this elimination implies that we can easily perform a depth-first visit within $T^s_{\max}$ for each $s \leq Q$, since $T^s_{\max}$ is always completely disconnected from the rest of the tree T.

In order to turn L_{SEL} into L^+_{SEL}, the algorithm proceeds incrementally, using a technique borrowed from [22]. Just after the selection of the first node i_1, a depth-first visit starting from i_1 is performed. During each backtracking step of this visit, the algorithm associates with each edge (i, j), the closer node to i_1 between the two nodes i and j. In other words, SEL assigns a direction to each undirected edge (i, j) so as to be able to efficiently find the path connecting each given node i to i_1. When the t-th node i_t is selected, SEL follows these edge directions from i_t towards i_1. Let us denote by $\pi(i, j)$ the path connecting node i to node j. During the traversal of $\pi(i_1, i_t)$, the algorithm assigns a special mark to each visited node, until the algorithm reaches the first node $j \in \pi(i_1, i_t)$ which has already been marked. Let $\eta(i, L)$ be the maximum number of edge disjoint paths connecting i to nodes in the query set L. Observe that all nodes i for which $\eta(i, L_t) > \eta(i, L_{t-1})$ must necessarily belong to $\pi(i_t, j)$. We have $\eta(i_t, L_t) = 1$, and $\eta(i, L_t) = 2$, for all internal nodes i in the path $\pi(i_t, j)$. Hence, j is the unique node that we may need to add as a new fork node (if $j \notin \text{FORK}(L_{t-1})$). In fact, j is the unique node such that the number of edge-disjoint paths connecting it to query nodes may increase, and be actually larger than 2.

Therefore if $j \in L^+_{t-1}$ we need not add any fork node during the incremental construction of L^+_{SEL}. On the other hand, if $j \notin L^+_{t-1}$ then $\eta(i, L_{t-1}) = 2$, which implies $\eta(i, L_t) = 3$. This is the case when SEL views j as new fork node to be added to the query set L_{SEL} under consideration.

In order to bound the total time required by SEL for selecting Q nodes, we rely on Lemma 4.3, showing that $|T^t_{max}| \leq 2|V|/t$. The two depth-first visits performed for each node i_t take $\mathcal{O}(|T^t_{max}|)$ steps. Hence the overall running time spent on the depth-first visits is $\mathcal{O}(\sum_{t \leq Q} 2|V|/t) = \mathcal{O}(|V| \log Q)$. The total time spent for incrementally finding the fork nodes of L_{SEL} is linear in the number of nodes marked by the algorithm, which is equal to $|V|$. Finally, handling the priority deque D takes $|V|$ times the worst-case time for eliminating an item with the smallest (or largest) key or adding a new item. This is again $\mathcal{O}(|V| \log Q)$.

We now turn to the implementation of the prediction phase. PRED operates in two phases. In the first phase, the algorithm performs a depth-first visit of each hinge-tree $\mathcal{T}$, starting from each connection node (thereby visiting the nodes of all 1-hinge-tree once, and the nodes of all 2-hinge-tree twice). During these visits, we add to the nodes a tag containing (i) the label of node $i_{\mathcal{T}}$ from which the depth-first visit started, and (ii) the distance between $i_{\mathcal{T}}$ and the currently visited node. In the second phase, we perform a second depth-first visit, this time on the whole tree T. During this visit, we predict each node $i \in V \setminus L$ with the label coupled with smaller distance stored in the tags of[8] i. The total time of these visits is linear in $|V|$ since each node of T gets visited at most 3 times.

4.7 Conclusions

The results proven in this chapter characterize, up to constant factors, the optimal algorithms for adversarial active learning on trees in two main settings. In the first setting the goal is to minimize the number of mistakes on the non-queried vertices under a certain query budget. In the second setting the goal is to minimize the sum of queries and mistakes under no restriction on the number of queries.

An important open question is the extension of our results to the general case of active learning on graphs. While a direct characterization of optimality on general graphs is likely to require new analytical tools, an alternative line of attack is reducing the graph learning problem to the tree learning

[8] If i belongs to a 1-hinge-tree, we simply predict y_i with the unique label stored in the tag.

problem via the use of spanning trees. Certain types of spanning trees, such as random spanning trees, are known to summarize well the graph structure relevant to passive learning — see Chapter 3. In the case of active learning, however, we want good query sets on the graph to correspond to good query sets on the spanning tree, and random spanning trees may fail to do so in simple cases. For example, consider a set of m cliques connected through bridges, so that each clique is connected to, say, k other cliques. The breadth-first spanning tree of this graph is a set of connected stars. This tree clearly reveals a query set (the star centers) which is good for regular labelings (cfr., the binary system example of Section 5.1). On the other hand, for certain choices of m and k a random spanning tree has a good probability of hiding the clustered nature of the original graph, thus leading to the selection of bad query sets.

We also believe that an extension to general graphs of our algorithm does actually exist. However, the complexity of the methods employed in [41] suggests that techniques based on minimizing Ψ^* on general graphs are computationally very expensive.

Chapter 5

Adaptive exploration of unknown graphs

5.1 Introduction

The learning model studied in the previous chapters is "transductive" in nature, in the sense that the graph is assumed to be known, and the task is to sequentially predict the labels of an adversarially chosen permutation of the vertices.

In this chapter we drop the transductive assumption and study the graph prediction problem from a purely sequential standpoint, where the vertices (and their incident edges) of an unknown graph are progressively revealed to the learner in an on-line fashion. As soon as a new vertex is revealed, the learner is required to predict its label. Before the next vertex is observed, the true label of the new vertex is fed back to the learner.

In order to allow the learner to actively *explore* the graph in directions that are judged easier to predict, we assume the underlying graph is connected, and force each newly revealed vertex to be adjacent to some vertex dynamically chosen by the learner in the subgraph so far observed.

More formally (see Section 5.2 for a complete description the protocol): at each time step $t = 1, 2, \ldots$, the learner selects a known node q_t having unexplored edges, receives a new vertex i_t adjacent to q_t, and is required to output a prediction $\widehat{y}_t$ for the (unknown) label y_t associated with i_t. Then y_t is revealed, and the algorithm incurs a loss $\ell(\widehat{y}_t, y_t)$ measuring the discrepancy between prediction and true label. Our basic measure of

performance is the learner's cumulative loss $\ell(\widehat{y}_1, y_1) + \cdots + \ell(\widehat{y}_n, y_n)$, over a sequence of n predictions.

As a motivating application for this exploration/prediction protocol, consider the advertising problem of targeting each member of a social network (where ties between individuals indicate a certain degree of similarity in tastes and interests) with the product he/she is most likely to buy. Suppose, for the sake of simplicity, that the network and the preferences of network members are initially unknown, apart from those of a single "seed member". It is reasonable to assume the existence of a mechanism that allows exploration of the social network by revealing new members connected (i.e., with similar interests) to members that are already known. This mechanism could be implemented in different ways, e.g., by providing incentives or rewards to members with unrevealed connections. Alternatively, if the network is hosted by a social network service (like Facebook$^{\text{TM}}$), the service provider itself may release the needed pieces of information. Since each discovery of a new network member is presumably costly, the goal of the marketing strategy is to minimize the number of new members not being offered their preferred product.

This social network advertising task can be naturally cast in our exploration/prediction protocol: at each step t, find the member q_t, among those whose preferred product y_t we already know, who is most likely to have undiscovered connections i_t with the same preferred product as q_t.

In order to leverage on the assumption that connected members tend to prefer the same products (see [92]), we design a learning/exploration strategy that perform well to the extent that the underlying graph labeling $\mathbf{y} = (y_1, \ldots, y_n)$ (which is not necessarily binary as in the previous chapters) is *regular* in the following sense: The graph can be partitioned into a small number of weakly interconnected clusters (subgroups of network members) such that labels in each cluster are all roughly similar.

The cumulative loss bound we prove in this chapter holds for general (real-valued) labels, and is expressed in terms of a measure of regularity we call *merging degree*. The merging degree of a labeled graph G is inherently related to the degree of interaction among the clusters which G can be partitioned into. In the special case of binary labels, this measure is often significantly smaller than the cutsize $\Phi_G(\mathbf{y})$, and never larger than $2\Phi_G(\mathbf{y})$. Furthermore, unlike $\Phi_G(\mathbf{y})$, which may even be quadratic in the number of

nodes, the merging degree is never larger than n, implying that our bound is never vacuous.

The main results of this chapter are the following. We prove that, for every binary-labeled graph G, the number of mistakes made on G by our learning/exploration algorithm is at most equal to the merging degree of G (Theorem 1). As a complementary result, we also show that, on *any* connected graph it is possible to force *any* algorithm to make a number of mistakes equal to half the merging degree (Theorem 2). We generalize the upper bound result by giving a cumulative loss bound holding for any loss function (Theorem 3). Finally, we show that our algorithm has small time and space requirements, which makes it suitable to large scale applications.

The chapter is organized as follows. In the next subsection we briefly overview some related work. The exploration/prediction protocol is introduced in Section 5.2. We define our measure of graph regularity in Section 5.3. In Section 5.4 we point out the weakness of some obvious exploration strategies (such as depth-first or breadth-first) and describe our algorithm, which is analyzed in Section 5.5. In Section 5.6 we describe time and space efficient implementations of our algorithm. We conclude in Section 5.7 with some comments and a few open questions.

5.1.1 Related work

on-line prediction of labeled graphs has often been studied in a "transductive" learning model different from the one studied here. Since the techniques proposed for transductive prediction assume knowledge of the entire graph in advance, they do not have any mechanism for guiding the exploration of the graph, hence they do not work well on the exploration/prediction problem studied in this chapter.

On the other hand, our exploration/prediction model bears some similarities to the graph exploration problem introduced in [32], where the measure of performance is the overall number edge traversals sufficient to ensure that each edge has been traversed at least once. Unlike that approach, we do not charge any cost for visits of the same node beyond the first visit. Moreover, in our setting depth-first exploration performs badly on simple graphs with binary labels (see discussion in Section 5.2), whereas depth-first traversal is optimal in the setting of [32] for any undirected graph —see [2].

As explained in Section 5.4, our strategy works by incrementally building a spanning tree whose total cost is equal to the algorithm's cumulative loss. The problem of constructing a minimum spanning tree on-line is also considered in [78], although only for graphs with random edge costs.

5.2 The exploration/prediction protocol

We use $\mathbf{y} = (y_1, \ldots, y_n) \in \mathcal{Y}^n$ to denote an unknown assignment of labels $y_i \in \mathcal{Y}$ to the vertices $i \in V$, where $\mathcal{Y}$ is a given label space, e.g., $\mathcal{Y} = \mathbb{R}$ or $\mathcal{Y} = \{-1, +1\}$.

We consider the following protocol between a graph exploration/prediction algorithm and an adversary. Initially, the algorithm receives an arbitrary vertex $i_0 \in V$ and its corresponding label y_0. For all subsequent steps $t = 1, \ldots, n-1$, let $V_{t-1} \subseteq V$ be the set of vertices visited in the first $t-1$ steps, where we conventionally set $V_0 = \{i_0\}$. We assume that the algorithm is told which nodes of V_{t-1} have unexplored neighbors; i.e., which nodes of V_{t-1} are adjacent to nodes in $V \setminus V_{t-1}$. Then:

1. The algorithm chooses a node $q_t \in V_{t-1}$ among those with unexplored neighbors.

2. The adversary chooses a node $i_t \in V \setminus V_{t-1}$ adjacent to q_t;

3. All edges $(i_t, j) \in E$ connecting i_t to *previously visited* vertices $j \in V_{t-1}$ are revealed, including edge (q_t, i_t);

4. The algorithm predicts the label y_t of i_t with $\widehat{y}_t \in \mathcal{Y}$;

5. The label y_t is revealed and the algorithm incurs a loss.

At each step $t = 1, \ldots, n-1$, the loss of the algorithm is $\ell(\widehat{y}_t, y_t)$, where $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}^+$ is a fixed and known function measuring the discrepancy between $\widehat{y}_t$ and y_t. For example, if $\mathcal{Y} = \mathbb{R}$, then we may set $\ell(\widehat{y}_t, y_t) = |\widehat{y}_t - y_t|$. The algorithm's goal is to minimize its cumulative loss $\ell(\widehat{y}_1, y_1) + \cdots + \ell(\widehat{y}_n, y_n)$. Note that the edges (q_t, i_t), for $t = 1, \ldots, n-1$, form a spanning tree for G. This is key to understanding the way our algorithm works —see Section 5.4.

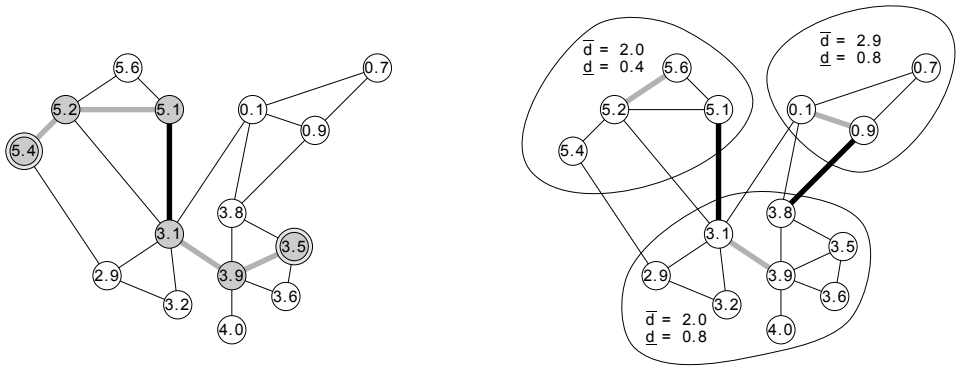

Figure 5.1: Two copies of a graph with real labels y_i associated with each vertex i. On the left, a shortest path connecting the two nodes enclosed in double circles is shown. The path length is $\max_t \ell(s_{k-1}, s_k)$, where $\ell(i,j) = |y_i - y_j|$. The thick black edge is incident to the nodes achieving the max in the path length expression. On the right, the vertices of the same graph have been clustered to form a regular partition. The diameter of a cluster C (the maximum of the pairwise distances between nodes of C) is denoted by $\underline{d}$. Similarly, $\overline{d}$ denotes the minimum of the pairwise distances (i,j), where $i \in C$ and $j \in V \setminus C$. Note that each $\overline{d}$ is determined by the thick black edge connecting the cluster to the rest of the graph, while $\underline{d}$ is determined by the two nodes incident to the thick gray edge. The partition is regular, hence $\underline{d} < \overline{d}$ holds for each cluster. Also, the three subgraphs induced by the clusters are connected.

5.3 Regular partitions and the merging degree

We are interested in designing exploration/prediction strategies that work well to the extent that the underlying graph G can be partitioned into a small number of weakly connected regions (the "clusters") such that labels on the vertices in each cluster are similar. Before defining this property formally, we need a few key auxiliary definitions.

Given a path $s_1, \ldots, s_d$ in G, a notion of path length $\lambda(s_1, \ldots, s_d)$ can be defined which is naturally related to the prediction loss. A reasonable choice might be $\lambda(s_1, \ldots, s_d) = \max_{k=2,\ldots,d} \ell(s_{k-1}, s_k)$, where we conventionally write $\ell(s_{t-1}, s_t)$ instead of $\ell(y_{s_{t-1}}, y_{s_t})$ when the labeling is understood from the context. Note that, in the binary classification case, when $\mathcal{Y} = \{-1, +1\}$ and $\ell(\widehat{y}, y) = \mathbb{I}_{\{\widehat{y} \neq y\}}$ (zero-one loss), if the labels of nodes $s_1, \ldots, s_d$ are either all positive or all negative, then $\lambda(s_1, \ldots, s_d) = 0$, oth-

erwise $\lambda(s_1, \ldots, s_d) = 1$.

In general, we say that λ is a **path length assignment** if it satisfies

$$\lambda(s_1, \ldots, s_{d-1}, s_d) \geq \lambda(s_1, \ldots, s_{d-1}) \geq 0 \tag{5.1}$$

for each path $s_1, \ldots, s_{d-1}, s_d$ in G. As we see in Section 5.6, condition (5.1) helps in designing efficient algorithms.

Given a path length assignment λ, denote by $P_t(i, j)$ the set of all paths connecting node i to node j in $G_t = (V_t, E_t)$, the subgraph containing all nodes V_t and edges E_t that have been observed during the first t steps. The distance $d_t(i, j)$ between i and j is the length of the shortest path between i and j in G_t,

$$d_t(i, j) = \min_{\pi \in P_t(i,j)} \lambda(\pi) \ .$$

We assume the path length $\lambda(\pi)$ is 0 if π consists of one node only, (i.e., $\pi = s_1$), which implies $d(i, i) = 0$ for all d.

A partition $\mathcal{P}$ of V in subsets C is **regular** if, for all $C \in \mathcal{P}$ and for all $i \in C$,

$$\max_{j \in C} d(i, j) < \min_{k \notin C} d(i, k)$$

where $d(i, j)$, without subscript, denotes the length of the shortest path between i and j in the whole graph G. See Figure 5.1 for an example.

We call **cluster** each element of a regular partition. Note that in a regular partition each node is closer to every node in its cluster than to any other node outside. When $-d(\cdot, \cdot)$ is taken as *similarity function*, our notion of regular partition becomes equivalent to the *Apresjan clusters* in [19] and to the *strict separation property* of [7].

It is easy to see that, because of (5.1), all subgraphs induced by the clusters on a regular partition are connected graphs. This simple fact is key to the proof of Lemma 1 in Section 5.5.

Note that every labeled graph $G = (V, E)$ has at least two regular partitions, since both $\mathcal{P} = \{V\}$ and $\mathcal{P} = \big\{\{1\}, \{2\}, \ldots, \{|V|\}\big\}$ are regular. Moreover, as depicted in Figure 5.2, if labels are binary then the notion of regular partition includes the (natural) partition made up of the smallest number of clusters C, each one including only nodes with the same label.

Now, for any given subset $C \subseteq V$, define the **inner border** ∂C of C to be the set of all nodes $i \in C$ that are adjacent to any node $j \notin C$. The **outer**

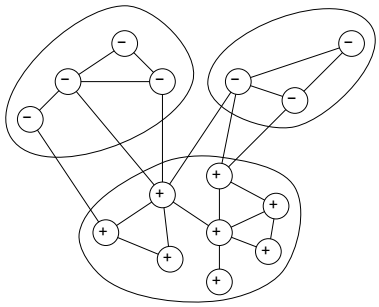

Figure 5.2: A (natural) regular partition for a graph with labels in $\{-1,+1\}$. The path length is measured as $\max_k \ell(s_{k-1}, s_k)$, where $\ell(i,j) = |y_i - y_j|$. The diameter of each cluster C (the maximum of the pairwise distances between nodes of C) is equal to 0, whereas the minimum of the pairwise distances (i,j), where $i \in C$ and $j \in V \setminus C$, is equal to 2.

border $\overline{\partial C}$ of C is the set of all nodes $j \notin C$ that are adjacent to at least one node in the inner border of C. See Figure 5.3 for an example.

Given the above, we are ready to introduce our measure of graph label regularity, which will be tightly related to the predictive ability of our algorithm.

Given a regular partition $\mathcal{P}$ of the vertices V of an undirected, connected and labeled graph $G = (V, E)$, for each $C \in \mathcal{P}$ the **merging degree** $\delta(C)$ of cluster C is defined as

$$\delta(C) = \min\left\{|\partial C|, |\overline{\partial C}|\right\} .$$

The overall merging degree of the partition, denoted by $\delta(\mathcal{P})$ is given by

$$\delta(\mathcal{P}) = \sum_{C \in \mathcal{C}} \delta(C) .$$

The merging degree $\delta(C)$ of a cluster $C \in \mathcal{P}$ quantifies the amount of interaction between C and the remaining clusters in $\mathcal{P}$.

In the binary case, it is not difficult to compare the merging degree of a partition to the graph custsize. Since at least one edge contributing to the cutsize $\Phi_G(\mathbf{y})$ must be incident to each node in an inner or outer border of a cluster, $\delta(\mathcal{P})$ is never larger than $2\Phi_G(\mathbf{y})$. On the other hand, as suggested for example by Figure 5.4, $\delta(\mathcal{P})$ is often much smaller $\Phi_G(\mathbf{y})$. This is directly

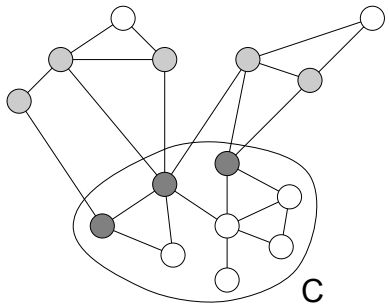

Figure 5.3: The inner border of the depicted subset C is the set of dark grey nodes, the outer border is made up of the light grey nodes, hence $|\partial C| = 3$ and $|\overline{\partial C}| = 5$.

implied by the two basic differences between merging degree and cutsize: (i) The merging degree counts subsets of nodes, and thus $\delta(\mathcal{P})$ is never larger than n; on the contrary, the cutsize counts subsets of edges, and thus on dense graphs $\Phi_G(\mathbf{y})$ can even be quadratic in n. (ii) The merging degree of a cluster is the *minimum* between two quantities (the cardinalities of inner and outer borders) related to the interaction among clusters. Hence, even on sparse graphs (where $\Phi_G(\mathbf{y})$ is close to the total number of border nodes of G), the merging degree can take advantage of clusters having unbalanced borders.

More importantly, as hinted again by Figure 5.4, $\delta(\mathcal{P})$ is typically more robust to label noise than $\Phi_G(\mathbf{y})$. For instance, if we flip the label of the black node, the merging degree of the depicted partition gets affected only by a small amount, whereas the cutsize can increase in a significant way. A more detailed study of the robustness of merging degree and cutsize against label flipping follows.

Let i be the node whose label y_i has been flipped. We write $\delta(\mathcal{P}, \mathbf{y})$ to emphasize the dependence of the merging degree on the labeling $\mathbf{y} \in \{-1, +1\}^n$. Let $\mathbf{y}_{\text{old}}$ be the labeling before the flip of y_i and $\mathbf{y}_{\text{new}}$ be the one after the flip. The following statement is easily verified. It provides *sufficient* conditions to insure that, after the label flip, $\delta(\mathcal{P}, \mathbf{y})$ cannot change by more than 2.

Fact 1 *Given a graph* $G = (V, E)$ *with labeling* $\mathbf{y} \in \{-1, +1\}^n$ *and a node* $i \in V$*, denote by* $G_i \subseteq G$ *the maximal connected subgraph containing* i *and made up of nodes labeled as* y_i *(so that* $V_i \subseteq V$ *is the*

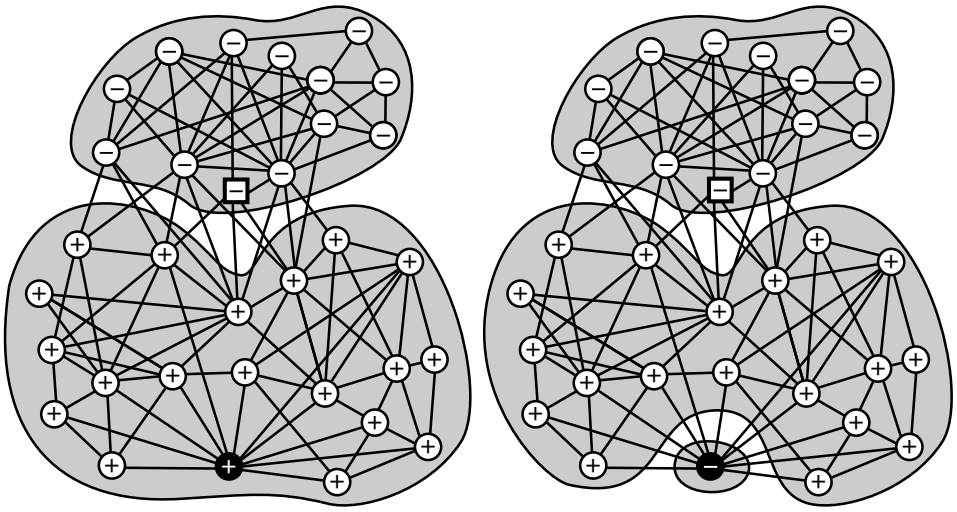

Figure 5.4: A relatively dense graph G (repeated twice) with two clusters C_1 and C_2 (left-hand side, from top to bottom), or three clusters C_1, C_2, and C_3 (right-hand side, from top to bottom), depending on the label of the black node at the bottom. If negative, this label might naturally be viewed as a *noisy* label. When we flip the label of the black node from positive to negative, the cutsize increases (as it is often the case in dense graphs) whereas the merging degree remains small. In particular, for the graph on the left $\Phi_G(\mathbf{y}) = 14$ and $\delta(\mathcal{P}) = \delta(C_1)+\delta(C_2) = 5+5 = 10$, while for the graph on the right $\Phi_G(\mathbf{y}) = 25$ and $\delta(\mathcal{P}) = \delta(C_1)+\delta(C_2)+\delta(C_3) = 5+6+1 = 12$. Note that the black node in the left graph satisfies the assumptions of Fact 1, while the square node in cluster C_1 does not. Indeed, flipping this square node might cause $\delta(\mathcal{P})$ to change significantly, whereas, in this case, $\Phi_G(\mathbf{y})$ would remain unchanged.

cluster containing node i*). If* i *is neither a border node of the cluster nor an articulation node*[1] *of* G_i*, then* $\left|\delta(\mathcal{P}, \mathbf{y}_{new}) - \delta(\mathcal{P}, \mathbf{y}_{old})\right| \leq 2$ *while* $\left|\Phi_G(\mathbf{y}_{new}) - \Phi_G(\mathbf{y}_{old})\right|$ *is always equal to the degree of* i.

See again Figure 5.4 for an illustration of the above statement.

A couple of observations are in order. First, when G is a dense graph it is fairly unlikely that a node exists which is an articulation node for its own cluster. In addition, since the most part of nodes in a real graph are not border nodes for any cluster, we tend to consider the case of the black node

[1]Recall that an *articulation* node of a connected graph is a node whose removal disconnects the graph.

shown in Figure 5.4 as the most common situation. Second, the two conditions on node i contained in Fact 1 are sufficient in order for the statement to hold, but are not necessary. As a matter of fact, there are important classes of labeled graphs (even sparse ones) where Fact 1 need not apply, still something interesting could be said about $\delta(\mathcal{P}, \mathbf{y})$ as compared to $\Phi_G(\mathbf{y})$. For example, if G is a labeled tree, then all vertices i that are not border nodes for any cluster are articulation nodes for the clusters which they belong to. In such cases, it is straightforward to verify that

$$\left|\delta(\mathcal{P}, \mathbf{y}_{\text{new}}) - \delta(\mathcal{P}, \mathbf{y}_{\text{old}})\right| \leq \left|\Phi_G(\mathbf{y}_{\text{new}}) - \Phi_G(\mathbf{y}_{\text{old}})\right| + 1 \ .$$

That is, a high variation in merging degree must correspond to a similar (or higher) variation in the cutsize.

The merging degree $\delta(\mathcal{P})$ is used to bound the total loss of our algorithm, which is described in the following section.

5.4 The Clustered Graph Algorithm

Before describing our algorithm, we would like to stress that in our exploration/prediction protocol, standard *nonadaptive* graph exploration strategies (combined with simple prediction rules) are suboptimal, meaning that their cumulative loss is not controlled by the merging degree. To this end, consider the strategy DEPTHFIRST, performing a depth-first visit of G (partially driven by the adversarial choice of i_t) and predicting the label of i_t through the adjacent node q_t in the spanning tree generated by the visit. In the binary classification case with zero-one loss, the graph cutsize $\Phi_G(\mathbf{y})$ is an obvious mistake bound achieved by such a strategy. Figure 5.5 shows an example where $\delta(\mathcal{P}) = \mathcal{O}(1)$ while DEPTHFIRST makes $\Phi_G(\mathbf{y}) = \Omega(|V|)$ mistakes. This high number of mistakes is not due to the choice of the prediction rule. Indeed, the same large number of mistakes is achieved by variants of DEPTHFIRST where the predicted label is determined by the majority vote of all labels (or just of the mistaken ones) among the adjacent nodes seen so far.

Another algorithm which we may consider is the so-called GRAPHTRON algorithm [74] for binary classification. As mentioned in Section 1.3, this algorithm predicts at time t just with the majority vote of the labels of

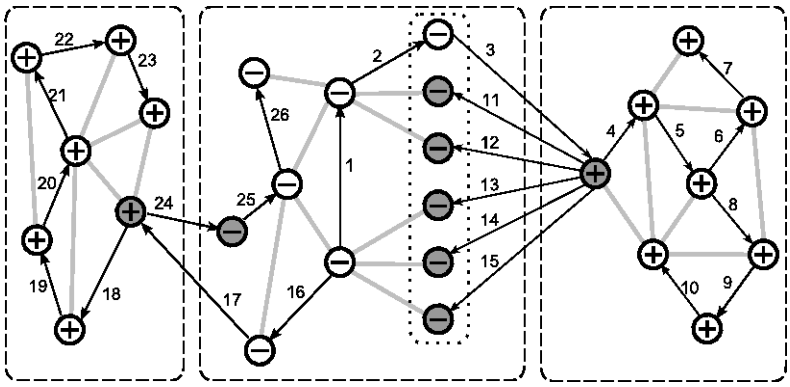

Figure 5.5: A binary labeled graph with three clusters such that $\delta(\mathcal{P}) = 4$ and $\Phi_G(\mathbf{y}) = 8$. We show that DEPTHFIRST makes order of $\Phi_G(\mathbf{y})$ mistakes. Arrow edges indicate predictions, and numbers on such edges denote the adversarial order of presentation. For instance edge 3 (connecting a -1 node to a $+1$ node) indicates that DEPTHFIRST uses the -1 label associated with the start node (the current q_t node) to predict the $+1$ label associated with the end node (the current i_t node). Dark grey nodes are the mistaken nodes (in this figure ties are mistakes). Note that in the dotted area we could add as many (mistaken) nodes as we like, thus making the graph cutsize $\Phi_G(\mathbf{y})$ arbitrarily close to $|V|$ without increasing $\delta(\mathcal{P})$. These nodes would still be mistaken if DEPTHFIRST predicted y_t through a majority vote among previously observed adjacent nodes, and they would remain mistaken if this majority vote were only restricted to previously mistaken adjacent nodes. This is because DEPTHFIRST is forced to err on the left-most node of the right-most cluster.

previously mistaken nodes that are adjacent to i_t. The number of mistakes satisfies $|E_{\mathcal{M}}| \leq 2\,\Phi_G(\mathbf{y})$, where $E_{\mathcal{M}} \subseteq E$ are all edges of G whose endpoints are both mistaken points. As a matter of fact, GRAPHTRON has been designed for a harder protocol where the adversary is not restricted to choose i_t adjacent to a previously observed node q_t. The example in Figure 5.5 shows that, even in our easier protocol, this algorithm makes order of $\Phi_G(\mathbf{y})$ mistakes. This holds even when the graph labeling is consistent with the majority vote predictor based on the entire graph.

Similar examples can be constructed to show that visiting the graph in breadth-first order can still cause $\Omega(|V|)$ mistakes.

These algorithms fail mainly because their exploration strategy is oblivious to the sequence of revealed labels. In fact, an *adaptive* exploration strategy taking advantage of the revealed structure of the labeled graph can

make a substantially smaller number of mistakes under our cluster regularity assumptions. Our algorithm, called CGA (Clustered Graph Algorithm), *learns* the next "good" node $q_t \in V_{t-1}$ to explore, and is able to take advantage of regular partitions. As we show in Section 5.5, the cumulative loss of CGA can be expressed in terms of the *best* regular partition of G with respect to the unknown labeling $\mathbf{y} \in \mathbb{R}^n$, i.e., the partition having minimum merging degree.

At each time step t, CGA sets $\widehat{y}_t$ to be the (known) label y_{q_t} of the selected vertex $q_t \in V_{t-1}$. Hence, the algorithm's cumulative loss is the cost of the spanning tree with edges $\{(q_t, i_t) : t = 1, \ldots, |V| - 1\}$ where edge (q_t, i_t) has cost $\ell(i,j) = \ell(y_i, y_j)$. The key to controlling this cost, however, is the specific rule the algorithm uses to select the next q_t based on G_{t-1}. The approach we propose is simple. If there exists a regular partition of G with few elements, then it does not really matter how the spanning tree is built within each element, since the cost of all these different trees will be small anyway. What matters the most is the cost of the edges of the spanning tree that join two distinct elements of the partition. In order to keep this cost small, our algorithm learns to select q_t so as to avoid going back to the same region many times. More precisely, at each time t, CGA selects and predicts the label of a node adjacent to the node in the inner border of V_{t-1} which is closest to the previously predicted node i_{t-1}. Formally,

$$\widehat{y}_t = y_{q_t} \quad \text{where} \quad q_t = \operatorname*{arg\,min}_{q \in \partial V_{t-1}} d_{t-1}(i_{t-1}, q) \ . \tag{5.2}$$

We say that cluster C is **exhausted** at time t if at time t the algorithm has already selected all nodes in C together with its outer border, i.e., if $C \cup \overline{\partial C} \subseteq V_t$. In the special but important case when labels are binary and the path length is $\lambda(s_1, \ldots, s_d) = \max_k \ell(s_{k-1}, s_k)$ (being ℓ the zero-one loss), the choice of node q_t in (5.2) can be defined as follows: If the cluster C where i_{t-1} lies is not exhausted at the beginning of time t, then CGA picks any node q_t connected to i_{t-1} by a path all contained in $V_{t-1} \cap C$. On the other hand, if C is exhausted, CGA chooses an arbitrary node in V_{t-1}.

Figure 5.6 contains a pictorial explaination of the behavior of CGA, as compared to DEPTHFIRST on the same binary labeled graph as in Figure 5.5. As we argue in the next section (Lemma 1 in Section 5.5), a key property of CGA is that when choosing q_t causes the algorithm to move out of a cluster of a regular partition, then the cluster must have been exhausted.

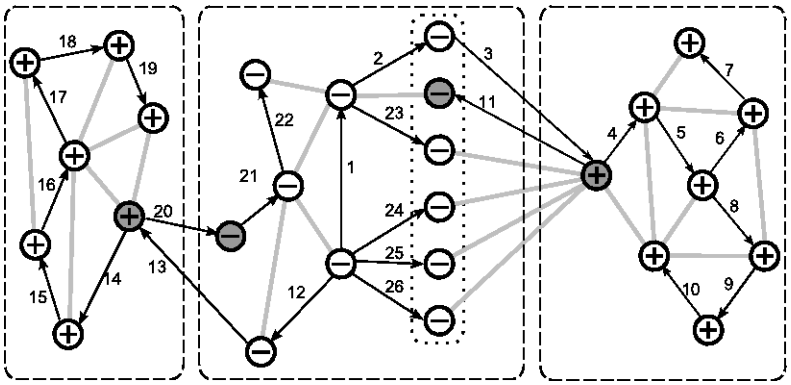

Figure 5.6: The behavior of CGA displayed on the binary labeled graph of Figure 5.5. The length of a path $s_1, \ldots, s_d$ is measured by $\max_k \ell(s_{k-1}, s_k)$ and the loss is the zero-one loss. The pictorial conventions are as in Figure 5.5. As in that figure, the cutsize $\Phi_G(\mathbf{y})$ of this graph can be made as close to $|V|$ as we like, still CGA makes $\delta(\mathcal{P}) = 4$ mistakes. For the sake of comparison, recall that the various versions of DEPTHFIRST can be forced to err $\Phi_G(\mathbf{y})$ times on this graph.

This suggests a fundamental difference between CGA and simple algorithms like DEPTHFIRST. Evidence of that is provided by comparing Figure 5.5 to Figure 5.6. CGA is seen to make a *constant* number of binary prediction mistakes on simple graphs where DEPTHFIRST makes order of $|V|$ mistakes. In this figure, the leftmost cluster has merging degree 1, the middle one has merging degree 2, and the rightmost one has merging degree 1. Hence this figure shows a case in which the mistake bound of our algorithm is tight (see Section 5.5). Note that the middle cluster has merging degree 2 no matter how we increase the number of negatively labeled nodes in the dotted area (together with the corresponding outbound edges).

5.5 Analysis

This section contains the analysis of CGA's predictive performance. The computational complexity analysis is contained in Section 5.6. For the sake of presentation, we treat the binary classification case first, since it is an important special case of our setting.

Fix an undirected and connected graph $G = (V, E)$. The following lemma is a key property of our algorithm.

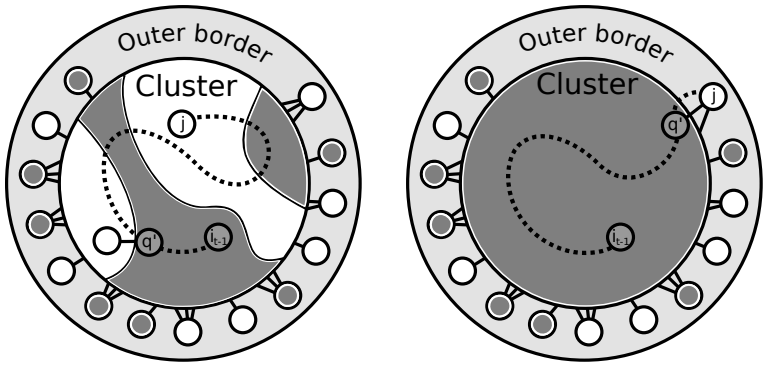

Figure 5.7: Two clusters corresponding to the two cases mentioned in the proof of Lemma 1. In both clusters the dark shaded area is $C \cap V_{t-1}$ (i.e., the set of nodes in cluster C that have already been explored) and the white area is $C \setminus V_{t-1}$. Case 1 (left cluster): A node j in C exists which has not been explored yet. Then there is a node q' on the inner border of V_{t-1}, along a path connecting i_{t-1} to j so as the path from i_{t-1} to q' is all contained in $C \cap V_{t-1}$. Case 2 (right cluster): A node j in the outer border of C exists which has not been explored yet. Then there is a node in the inner border of C which is connected to i_{t-1} so that we can single out a further node q' with the same properties as in Case 1.

Lemma 1 *Assume* CGA *is run on a graph* G *with labeling* $\mathbf{y} \in \mathcal{Y}^n$*, and pick any time step* $t > 0$*. Let* $\mathcal{P}$ *be a regular partition and assume* $i_{t-1} \in C$*, where* C *is any cluster in* $\mathcal{P}$*. Then* C *is exhausted at time* $t-1$ *if and only if* $q_t \notin C$*.*

PROOF. First, assume C is exhausted at time $t-1$, i.e., $C \cup \overline{\partial C} \subseteq V_{t-1}$. Then all nodes in C have been visited, and no node in C has unexplored edges. This implies $C \cap \partial V_{t-1} \equiv \emptyset$ and that the selection rule (5.2) makes the algorithm pick q_t outside of C. Assume now $q_t \notin C$. Since each cluster is a connected subgraph, if the labels are binary the prediction rule ensures that cluster C is exhausted. In the general case (when labels are not binary) we can prove by contradiction that C is exhausted by analyzing the following two cases (see Figure 5.7).

Case 1. There exists $j \in C \setminus V_{t-1}$. Since the subgraph in cluster C is connected, there is a path in C connecting i_{t-1} to j such that at least one node $q' \in C$ on this path: (a) has unexplored edges, and (b) belongs to V_{t-1}, (i.e., $q' \in \partial V_{t-1}$), and (c) is connected to i_{t-1} by a path all contained in

$C \cap V_{t-1}$. Since the partition is regular, q' is closer to i_{t-1} than to any node outside of C. Hence, by construction —see (5.2), the algorithm would choose this q' instead of q_t (due to (c) above), thereby leading to a contradiction.
Case 2. There exists $j \in \overline{\partial C} \setminus V_{t-1}$. Again, since the subgraph in cluster C is connected, there is a path in C connecting i_{t-1} to a node in ∂C adjacent to j. Then we fall back into the previous case since at least one node q' on this path: (a) has unexplored edges, and (b) belongs to V_{t-1}, and (c) is connected to i_{t-1} by a path all contained in $C \cap V_{t-1}$. □

We begin to analyze the special case of binary labels and zero-one loss.

Theorem 1 *If* CGA *is run on an undirected and connected graph* G *with binary labels then the total number* m *of mistakes satisfies*

$$m \leq \delta(\mathcal{P})$$

where $\mathcal{P}$ *is the smallest partition* $\mathcal{P}$ *of* V *whose each cluster only includes nodes having the same label.* [2]

The key idea to the proof of this theorem is the following. Fix a cluster $C \in \mathcal{P}$. In each time step t when both q_t and i_t belong to C a mistake never occurs. The remaining time steps are of two kinds only: (1) Incoming lossy steps, where node i_t belongs to the inner border of C; (2) outgoing lossy steps, where i_t belongs to the outer border of C. With each such step we can thus uniquely associate a node i_t in either (inner or outer) border of C. The overall loss involving C, however, is typically much smaller than the *sum* of border cardinalities. Consider all the incoming and outgoing lossy steps concerning cluster C. The first lossy step after an incoming lossy step must be outgoing and, viceversa, the first lossy step after an outgoing lossy step must be incoming. In other words, for each given cluster C, incoming and outgoing steps are interleaved. Since during any incoming lossy step t a new node of C must be visited, before the subsequent incoming lossy step $t' > t$ the algorithm must visit a new node of $V \setminus C$. Visiting the first node of $V \setminus C$ after time t will necessarily lead to a new outgoing lossy step. Hence, incoming and outgoing steps must occur the same number of times, and their sum must be at most twice the *minimum* of the size of borders

[2] Recall that such a $\mathcal{P}$ is a regular partition of V. Moreover, one can show that for this partition the bound in the theorem is never vacuous.

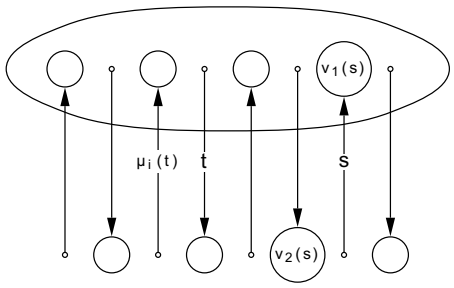

Figure 5.8: Sequence (starting from the left) of incoming and regular outgoing lossy steps involving a given cluster C_i. We only show the border nodes contributing to lossy steps. We map injectively each regular outgoing lossy step t to the previous (incoming) lossy step $\mu_i(t)$. We also map injectively each incoming lossy step s to the node $v_1(s)$ in the inner border, whose label was predicted at time s. Finally, we map injectively s also to the node $v_2(s)$ in the outer border that caused the previous (outgoing) lossy step for the same cluster.

(what we called merging degree of the cluster), since each node is visited only once. The only exception to this interleaving pattern occurs when a cluster gets exhausted. In this case, an incoming step is not followed by any outgoing step for the exhausted cluster.

PROOF OF THEOREM 1. Index by $1, \ldots, |\mathcal{P}|$ the clusters in $\mathcal{P}$. We abuse the notation and use $\mathcal{P}$ also to denote the set of cluster indices. Let $k(t)$ be the index of the cluster which i_t belongs to, i.e., $i_t \in C_{k(t)}$. We say that step t is a *lossy step* if $\hat{y}_t \neq y_t$, i.e. the label of q_t is different from the label of i_t. A step t in which a mistake occurs is *incoming for cluster* i (denoted by $* \rightarrow i$) if $q_t \notin C_i$ and $i_t \in C_i$, and it is *outgoing for cluster* i (denoted by $i \rightarrow *$) if $q_t \in C_i$ and $i_t \notin C_i$. An outgoing step for cluster C_i is *regular* if the previous step in which the algorithm made a mistake is incoming for C_i. All other outgoing steps are called *irregular*. Let $M_{\rightarrow i}$ ($M_{i\rightarrow}^{\text{reg}}$) be the set of all incoming (regular outgoing) lossy steps for cluster C_i. Also, let $M_{i\rightarrow}^{\text{irr}}$ be the set of all irregular outgoing lossy steps for C_i.

For each $i \in \mathcal{P}$, define an injective mapping $\mu_i : M_{i\rightarrow}^{\text{reg}} \rightarrow M_{\rightarrow i}$ as follows (see Figure 5.8 for reference): Each lossy step t in $M_{i\rightarrow}^{\text{reg}}$ is mapped to the previous step $t' = \mu_i(t)$ when a mistake occurred. Lemma 1 insures that such step must be incoming for i since t is a regular outgoing step. This shows that $|M_{i\rightarrow}^{\text{reg}}| \leq |M_{\rightarrow i}|$. Now, let t be any irregular outgoing step for some cluster, t' be the last lossy step occurred before time t, and set $j = k(t')$. The

very definition of an irregular lossy step, combined with Lemma 1, allows us to conclude that t' is the last lossy step involving cluster C_j. This implies that t' cannot be followed by an outgoing lossy step $j \to *$. Hence t' is not in the image of μ_j, and the previous inequality for $|M^{\text{reg}}_{i\to}|$ can be refined as $|M^{\text{reg}}_{i\to}| \leq |M_{\to i}| - \mathbb{I}_i$. Here $\mathbb{I}_i$ is the indicator function of the following event: "The very last lossy step t' such that either q'_t or i'_t belong to C_i is incoming for C_i". We now claim that

$$\sum_{i\in\mathcal{P}} \mathbb{I}_i \geq \sum_{i\in\mathcal{P}} |M^{\text{irr}}_{i\to}| \,.$$

In fact, if we let t be an irregular lossy step and i be the index of the cluster for which the previous lossy step t' is incoming, the fact that t is irregular implies that C_i must be exhausted between time t' and time t, which in turn implies that $\mathbb{I}_i = 1$, since t' must be the very last lossy step involving cluster C_i. This allows us to write

$$m = \sum_{i\in\mathcal{P}} |M^{\text{reg}}_{i\to} \cup M^{\text{irr}}_{i\to}| \leq \sum_{i\in\mathcal{P}} \Big(|M_{\to i}| - \mathbb{I}_i + |M^{\text{irr}}_{i\to}| \Big) \leq \sum_{i\in\mathcal{P}} |M_{\to i}| \,. \tag{5.3}$$

Next, for each $i \in \mathcal{P}$ we define two further injective mappings that associate with each incoming lossy step $* \to i$ a vertex in the inner border of C_i and a vertex in the outer border of C_i. This shows that

$$|M_{\to i}| \leq \min\big\{ |\partial C_i|, |\overline{\partial C_i}| \big\} = \delta(C_i)$$

for each $i \in \mathcal{P}$. Together with (5.3), which we prove next, this completes the proof (see again Figure 5.8 for a pictorial explanation).

The first injective mapping $\nu_1 : M_{\to i} \to \partial C_i$ is easily defined: $\nu_1(t) = i_t \in C_i$. This is an injection because the algorithm can incur loss on a vertex at most once. The second injective mapping $\nu_2 : M_{\to i} \to \overline{\partial C_i}$ is defined in the following way. Let $M_{\to i}$ be equal to $\{t_1, \ldots, t_k\}$, with $t_1 < \cdots < t_k$. If $t = t_1$ then $\nu_2(t)$ is simply $q_t \in \overline{\partial C_i}$. If instead $t = t_j$ with $j \geq 2$, then $\nu_2(t) = i_{t'} \in \overline{\partial C_i}$, where t' is an outgoing lossy step $i \to *$, lying between t_{j-1} and t_j. Note that cluster C_i cannot be exhausted after step t_{j-1} since another incoming lossy step $* \to i$ occurs at time $t_j > t_{j-1}$. Combined with Lemma 1 this guarantees the existence of such a t'. Moreover, no subsequent outgoing lossy steps $i \to *$ can mispredict the same label $y_{i_{t'}}$. Hence ν_2 is an injection and the proof is concluded. □

Next, we turn to considering *lower* bounds on the prediction performance. First, as we already observed, the edges (q_t, i_t) produced during the on-line functioning of the algorithm form a spanning tree T for G. Therefore, in the case of binary labels, CGA's number m of mistakes is always equal to the cutsize $\Phi_T(\mathbf{y})$ of this spanning tree. This shows that an obvious lower bound on m is the cost of the minimum spanning tree for G or, equivalently, the size of the smallest regular partition $\mathcal{P}$ of V where each cluster includes only nodes having the same label.

This argument can be strenghtened to show that an adaptive adversary can always force *any* learner to make order of $\delta(\mathcal{P})$ mistakes in the binary case, thus matching the upper bound of Theorem 1. For simplicity of exposition, the following theorem is stated for deterministic algorithms, though it can be trivially seen to hold (with a different leading constant) for randomized algorithms as well.

Theorem 2 *For any undirected and connected graph $G = (V, E)$, for all $K < |V|$, and for any learning strategy, there exists a labeling $\mathbf{y}$ of V such that the strategy makes at least K mistakes while $\delta(\mathcal{P}) \leq 2K$. Here $\mathcal{P}$ is the smallest regular partition $\mathcal{P}$ of V where each cluster only includes nodes having the same label.*

PROOF. Let $G_0 = (V_0, E_0)$ be any connected component of G with $|V| - K$ nodes, and let $V' = V \setminus V_0$ be the set of the remaining K nodes. The adversarial strategy forces a mistake on each node in V', and uses a common arbitrary label for all the nodes in V_0.

To finish the proof, we must now show that $\delta(\mathcal{P}) \leq 2K$. In order to do so, observe that since G_0 is a connected component in G, and all nodes of V_0 have the same label, V_0 must be included in a cluster $C_0 \in \mathcal{P}$. Since $|C_0| \geq |V_0| = |V| - K$, we have that $\delta(C_0) \leq |\overline{\partial C_0}| \leq |V'| = K$. Consequently, for the remaining clusters we have

$$\sum_{C \in \mathcal{P} \setminus \{C_0\}} \delta(C) \leq \sum_{C \in \mathcal{P} \setminus \{C_0\}} |\partial C| \leq |V'| = K \ .$$

Hence, $\delta(\mathcal{P}) \leq 2K$, and the proof is concluded. □

It is important to observe that the adversarial strategy described in the above proof works against a broad class of learning algorithms. In particular, it

works against learners that are given the graph structure beforehand and have full control on the sequence $i_1, \ldots, i_n$ of nodes to be predicted. In this respect, Theorem 1 shows that our less informed protocol is actually sufficient to match the performance level dictated by the lower bound.

We now turn to the analysis of upper bounds for CGA in the general case of *nonbinary* labels. The following definitions are useful for espressing the cumulative loss bound of our algorithm: Let $\mathcal{P}$ be a regular partition of the vertex set V and fix a cluster $C \in \mathcal{P}$. We say that edge (q_t, i_t) causes an **inter-cluster loss** at time t if one of the two nodes of this edge lies in ∂C and the other lies in $\overline{\partial C}$. Edge (q_t, i_t) causes an **intra-cluster loss** when both q_t and i_t are in C. We denote by $\ell(C)$ the largest inter-cluster loss in C, i.e.,

$$\ell(C) = \max_{i \in \partial C,\, j \notin \overline{\partial C},\, (i,j) \in E} \ell(y_i, y_j) \ .$$

Also $\ell_{\mathcal{P}}^{\max}$ is the maximum inter-cluster loss in the whole graph G, i.e., $\ell_{\mathcal{P}}^{\max} = \max_{C \in \mathcal{P}} \ell(C)$. We also set for brevity $\bar{\ell}_{\mathcal{P}} = |\mathcal{P}|^{-1} \sum_{C \in \mathcal{P}} \ell(C)$. Finally, we define

$$\varepsilon(C) = \max_{T_C} \sum_{(i,j) \in E(T_C)} \ell(y_i, y_j)$$

where the maximum is over all spanning trees T_C of C and $E(T_C)$ is the edge set of T_C. Note that $\varepsilon(C)$ bounds from above the total loss incurred in all steps t where q_t and i_t both belong to C. As a matter of fact, CGA's cumulative loss is actually $\sum_{t=1}^{|V|} \ell(q_t, i_t)$, where, as we said in Section 5.2, the edges (q_t, i_t), $t = 1, \ldots, |V| - 1$ form a spanning tree for G; hence the subset of such edges which are also incident to nodes in C form a spanning forest for C. Our definition of $\varepsilon(C)$ takes into account that the total loss associated with the edge set of a spanning tree T_C for C is at least as large as the total loss associated with the edge set $E(\mathcal{F})$ of any spanning forest $\mathcal{F}$ for C such that $E(\mathcal{F}) \subseteq E(T_C)$.

In the above definition, $\ell(C)$ is a measure of connectedness between C and the remaining clusters, $\varepsilon(C)$ is a measure of "internal cohesion" of C, while $\ell_{\mathcal{P}}^{\max}$ and $\bar{\ell}_{\mathcal{P}}$ give global distance measures among the clusters within $\mathcal{P}$.

The following theorem shows that CGA's cumulative loss can be bounded in terms of the regular partition $\mathcal{P}$ that best trades off total intra-cluster loss, which is expressed by $\varepsilon(C)$), against total inter-cluster loss, which is

expressed by $\delta(C)$ times the largest inter-cluster loss $\ell(C)$. It is important to stress that CGA never explicitely computes this optimal partition: it is the selection rule for q_t in (5.2) that guarantees this optimal behavior.

Theorem 3 *If* CGA *is run on an undirected and connected graph* G *with arbitrary real labels, then the cumulative loss can be bounded as*

$$\sum_{t=1}^{n} \ell(\widehat{y}_t, y_t) \le \min_{\mathcal{P}} \left(|\mathcal{P}| \Big(\ell_{\mathcal{P}}^{\max} - \bar{\ell}_{\mathcal{P}} \Big) + \sum_{C \in \mathcal{P}} \Big(\varepsilon(C) + \ell(C)\delta(C) \Big) \right) \tag{5.4}$$

where the minimum is over all regular partitions $\mathcal{P}$ *of* V*.*

Remark 3 *If* ℓ *is the zero-one loss, then the bound in (5.4) reduces to*

$$\sum_{t=1}^{n} \ell(\widehat{y}_t, y_t) \le \min_{\mathcal{P}} \sum_{C \in \mathcal{P}} \Big(\varepsilon(C) + \delta(C) \Big) \ . \tag{5.5}$$

This shows that in the binary case the total number of mistakes can also be bounded by the maximum number of edges connecting different clusters that can be part of a spanning tree for G*. In the binary case (5.5) achieves its minimum either on the trivial partition* $\mathcal{P} = \{V\}$ *or on the partition made up of the smallest number of clusters* C*, each one including only nodes with the same label (this is what in Section 5.3 was called the natural regular partition — see Theorem 1). In most cases, the natural regular partition is the minimizer of (5.5), so that the intra-cluster term* $\varepsilon(C)$ *disappears. Then the bound only includes the sum of merging degrees (w.r.t. that partition), thereby recovering the bound in Theorem 1. However, in certain degenerate cases, the trivial partition* $\mathcal{P} = \{V\}$ *turns out to be the best one. In such a case, the right-hand side of (5.5) becomes* $\varepsilon(V)$ *which, in turn, is bounded by* $\Phi_G(\mathbf{y})$*.*

The proof of Theorem 3 is similar to the one for the binary case, hence we only emphasize the main differences. Let $\mathcal{P}$ be a regular partition of V. Clearly, no matter how each $C \in \mathcal{P}$ is explored, if $q_t, i_t \in C$ then the contribution of $\ell(q_t, i_t)$ to the total loss is bounded by $\varepsilon(C)$. The remaining losses contributed by any cluster C are of two kinds only: losses on incoming steps, where the node i_t belongs to the inner border of C, and losses on

outgoing steps, where i_t belongs to the outer border of C. As for the binary case, with each such step we can thus associate a node in the inner and the outer border of C, since incoming and outgoing step alternate for each cluster. The exception is when a cluster is exhausted which, at first glance, seems to requires adding an extra term as big as $\ell_{\mathcal{P}}^{\max}$ times the size $|\mathcal{P}|$ of the partition (this term could have a significant impact for certain graphs). However, as explained in the proof below, $\ell_{\mathcal{P}}^{\max}$ can be replaced by the potentially much smaller term $\ell_{\mathcal{P}}^{\max} - \bar{\ell}_{\mathcal{P}}$. In fact, in certain cases this extra term disappears, and the final bound we obtain is just (5.5).

PROOF OF THEOREM 3. Fix an arbitrary regular partition $\mathcal{P}$ of V and index by $1, \dots, |\mathcal{P}|$ the clusters in it. We abuse the notation and use $\mathcal{P}$ also to denote the set of cluster indices. We say that step t is a *lossy step* if $\ell(q_t, i_t) > 0$, and we distinguish between intra-cluster lossy steps (when q_t and i_t belong to the same cluster) and inter-cluster lossy steps (when q_t and i_t belong to different clusters). We crudely upper bound the total loss incurred during intra-cluster lossy steps by $\sum_{C \in \mathcal{P}} \varepsilon(C)$. Hence, in the rest of the proof we focus on bounding the total loss incurred during inter-cluster lossy steps only. We define incoming and outgoing (regular and irregular) inter-cluster lossy steps for a given cluster C_i (and the relative sets $M_{\to i}$, $M_{i\to}^{\text{reg}}$ and $M_{i\to}^{\text{irr}}$) as in the binary case proof, as well as the injective mapping μ_i. In the binary case we bounded $|M_{i\to}^{\text{reg}}|$ by $|M_{\to i}| - \mathbb{I}_i$. In a similar fashion, we now bound $\sum_{t \in M_{i\to}^{\text{reg}}} \ell_t$ by $\ell(C_i)\Big(|M_{\to i}| - \mathbb{I}_i\Big)$, where we set for brevity $\ell_t = \ell(q_t, i_t)$. We can write

$$\begin{aligned}
\sum_{i \in \mathcal{P}} \sum_{t \in M_{i\to}^{\text{reg}} \cup M_{i\to}^{\text{irr}}} \ell_t &\le \sum_{i \in \mathcal{P}} \Big(\ell(C_i)\Big(|M_{\to i}| - \mathbb{I}_i\Big) + \ell_{\mathcal{P}}^{\max} |M_{i\to}^{\text{irr}}| \Big) \\
&\le \sum_{i \in \mathcal{P}} \ell(C_i)|M_{\to i}| + \sum_{j \in \mathcal{P} \,:\, \mathbb{I}_j = 1} \Big(\ell_{\mathcal{P}}^{\max} - \ell(C_j) \Big) \\
&\le \sum_{i \in \mathcal{P}} \ell(C_i)|M_{\to i}| + \sum_{i \in \mathcal{P}} \Big(\ell_{\mathcal{P}}^{\max} - \ell(C_i) \Big) \\
&= \sum_{i \in \mathcal{P}} \ell(C_i)|M_{\to i}| + |\mathcal{P}|\Big(\ell_{\mathcal{P}}^{\max} - \bar{\ell}_{\mathcal{P}} \Big)
\end{aligned}$$

where the second inequality follows from $\sum_{i \in \mathcal{P}} \mathbb{I}_i \ge \sum_{i \in \mathcal{P}} |M_{i\to}^{\text{irr}}|$ (as for the natural regular partition considered in the binary case).

The proof is concluded after defining the two injective mapping ν_1 and ν_2 as in the binary case, and bounding again $|M_{\to i}|$ through $\delta(C_i)$. □

5.6 Computational complexity

In this section we describe an efficient implementation of CGA and discuss some improvements for the special case of binary labels. This implementation shows that CGA is especially useful when dealing with large scale applications.

Recall that the path length assignment λ is a parameter of the algorithm and satisfies (5.1). In order to develop a consistent argument about CGA's time and space requirements, we need to make assumptions on the time it takes to compute this function. When given the distance between any pair of nodes i and j, and the loss $\ell(j, j')$ for any j' adjacent to j, we assume the length of the shortest path $i, \dots, j, j'$ can be computed in *constant* time . This assumption is easily seen to hold for many natural path length assignments λ over graphs, for instance $\lambda(s_1, \dots, s_d) = \max_k \ell(s_{k-1}, s_k)$ and $\lambda(s_1, \dots, s_d) = \sum_k \ell(s_{k-1}, s_k)$ —note that both these assignments fulfill (5.1).

Because of the above assumption on the path length λ, in the general case of real labels CGA can be implemented using the well-known Dijkstra's algorithm for single-source shortest path (see, e.g., [29, Ch. 21]). After all nodes in V_{t-1} and all edges incident to i_t have been revealed (step 3 of the protocol in Section 5.2), CGA computes the distance between i_t and any other node in V_{t-1} by invoking Dijkstra's algorithm on the sub-graph G_t, so that CGA can easily find node q_{t+1}. If Dijkstra's algorithm is implemented with Fibonacci heaps [29, Ch. 25], the total time required for predicting all $|V|$ labels is[3] $\mathcal{O}\Big(|V||E| + |V|^2 \log |V|\Big)$. In practice, the actual running time is often lower, since at each time step t Dijkstra's algorithm can be stopped as soon as the node of ∂V_{t-1} nearest to i_t in G_t has been found.

On the other hand, the space complexity is always linear in the size of G.

[3] In practice, the actual running time is often far less than $\mathcal{O}\big(|V||E| + |V|^2 \log |V|\big)$, since at each time step t Dijkstra's algorithm can be stopped as soon as the node of ∂V_{t-1} nearest to i_t in G_t has been found.

5.6.1 An improved analysis for the binary case

We now describe a special implementation for the case of binary labels. The additional assumption $\lambda(s_1, \ldots, s_d) = \max_k \ell(s_{k-1}, s_k)$ allows us to exploit the simple structure of regular partitions. Coarsely speaking, we maintain information about the current inner border and clusters, and organize this information in a balanced tree, connecting the nodes lying in the same cluster through special linked lists.

In order to describe this implementation, it is important to observe that, since the graph is revelead incrementally, it might be the case that a single cluster C in G at time t happens to be split into several disconnected parts in G_t. In other words, the algorithm always knows that each group of nodes being part of the same uniformly labeled and connected subgraph of G_t is a subset of the same cluster C in G, but need not know if there are other groups of nodes of V_t with the same label, that are actually part of C.

We call **sub-cluster** each maximal set of nodes that are part of the same uniformly labeled and connected subgraph of G_t. The data structures we use for organizing the nodes observed so far by the algorithm combine the following substructures:

- A self-balancing binary search tree T containing the labeled nodes in ∂V_t. Each node of T corresponds to a node in ∂V_t and contains the associated label. We will refer to nodes in ∂V_t and to nodes in T interchangeably.

- Given a sub-cluster C, all nodes in $C \cap \partial V_t$ are connected via a special list $L(C)$ called **border sub-cluster list**.

- All nodes in each border sub-cluster list $L(C)$ are linked to a special time-varying set $r(C)$ called **sub-cluster record**. This record enables access to the first and last element of $L(C)$ and stores the size of $L(C)$.

Let $C(i_t)$ be the sub-cluster containing i_t at time t. The above data structures are intended to support the following operations, which are executed in the following order at each time step t, just after the algorithm has selected q_t.

1. **Insertion** of i_t. When i_t is chosen by the adversary, CGA receives the list $N(i_t)$ of all nodes in V_{t-1} adjacent to i_t. Note that all nodes

in $N(i_t)$ belong to ∂V_{t-1} and are therefore in T at the current time step. In order to perform the insertion, the algorithm inserts i_t into T and temporarily considers i_t as the unique node of a new sub-cluster $C(i_t)$. Hence, the algorithm creates a border sub-cluster list $L(C(i_t))$ containing only i_t and a sub-cluster record $r(C(i_t))$ linked solely to i_t. In this step i_t is (provisionally) inserted into T and in $L(C(i_t))$ even if i_t has no unexplored neighbors at time t. The insertion of i_t requires $\mathcal{O}(\log t)$ time, since $|\partial V_t| \leq t$.

2. **Union** of sub-clusters. After prediction, the label y_t of i_t is revealed. Since all nodes in $N(i_t)$ having the same label as i_t belong now to the same sub-cluster, we need to execute a sequence of merging operations on each node $j \in N(i_t)$. This essentially involves concatenating border sub-cluster lists and updating the links from nodes in T to sub-cluster records. The merging operation can be implemented as a union-by-rank in standard union-find data structures (e.g., [29, Ch. 22]).

3. **Elimination** of nodes. All nodes in $\{i_t\} \cup N(i_t)$ that are not part of ∂V_t are deleted from T, and the linked sub-cluster records are updated (or eliminated). Once a node gets deleted from T, it will never become again part of T. Hence, executing this step over the whole graph requires $\mathcal{O}(|V| \log |V|)$ time.

4. **Choice** of q_{t+1}. If the border sub-cluster list of $C(i_t)$ is not empty, q_{t+1} is chosen arbitrarily among the nodes of this list. Otherwise q_{t+1} is set to any node in T.

Observe that checking all neighbors of i_t in T for deciding whether it is necessary to run a merging operation in step 2 takes time $\mathcal{O}(|N(i_t)| \log t)$. Moreover, we now show that the running time for actually executing the merging operation on $|V|$ nodes is $\mathcal{O}\big(|V| \log |V|\big)$. In fact, for each node j in $N(i_t)$ CGA repeats the following substeps:

1. We reach node j in T in time $\mathcal{O}(\log t)$.

2. If the label of j is equal to y_t, the algorithm *merges* the sub-cluster $C(j)$ with the current sub-cluster $C(i_t)$ as follows: The algorithm concatenates the two associated border sub-cluster lists $L(C(i_t))$ and $L(C(j))$. Let now L_{min} be the smaller border sub-cluster list and L_{max} be the

larger one. CGA makes all nodes of L_{min} to point to the sub-cluster record r_{max} of L_{max}. The sub-cluster record associated with L_{min} is eliminated and the size of the concatenated border sub-cluster list of r_{max} is updated, along with its initial and final nodes. This operation requires $\mathcal{O}(|C_{min}|)$ time. Note that, after the update of the link between a node $s \in V$ and its sub-cluster record, the size of the new sub-cluster of s must be at least doubled. This implies that the total time needed for this operation over all nodes in V is $\mathcal{O}(|V| \log |V|)$.

3. If instead the label of j is different from y_t the algorithm does nothing.

Figure 5.9 depitcs the first three steps (Insertion, Union, and Elimination) of the above sequence at time $t = 15$.

The dominating cost in the time complexity is the total cost for reaching at each time t the nodes of V_{t-1} adjacent to i_t. (hence $\sum_{t=1}^{|V|-1} |N(i_t)| = |E|$). Therefore the overall running time for predicting $|V|$ labels is $\mathcal{O}\Big(|E| \log |V| + |V| \log |V|\Big) = \mathcal{O}\Big(|E| \log |V|\Big)$, which is the best one can hope for (an obvious lower bound is $|E|$) up to a logarithmic factor.[4]

As for space complexity, it is important to stress that on every step t the algorithm first stores and then throws away the received node list $N(i_t)$ —in the worst case the length of $N(i_t)$ is linear in $|V|$. The space complexity is therefore $\mathcal{O}(|V|)$. This optimal use of space is one of the most important practical strengths of CGA since the algorithm never needs to store the whole graph seen so far.

5.7 Conclusions and open questions

We have presented a first step towards the study of problems related to learning labeled graph exploration strategies. As we said in Subsection 5.1.1, this is a significant departure from more standard approaches assuming prior knowledge of the underlying graph structure.

Our exploration/prediction model could be extended in several directions. For example, in order to take into account information related to the

[4]Of course, if the algorithm knew beforehand the total number of nodes in V and each node were numbered with an integer from 1 to $|V|$ then, instead of T, we could use a static data structure with constant time access to each element. In this case, the total time complexity would be $\mathcal{O}(|E| + |V| \log |V|)$.

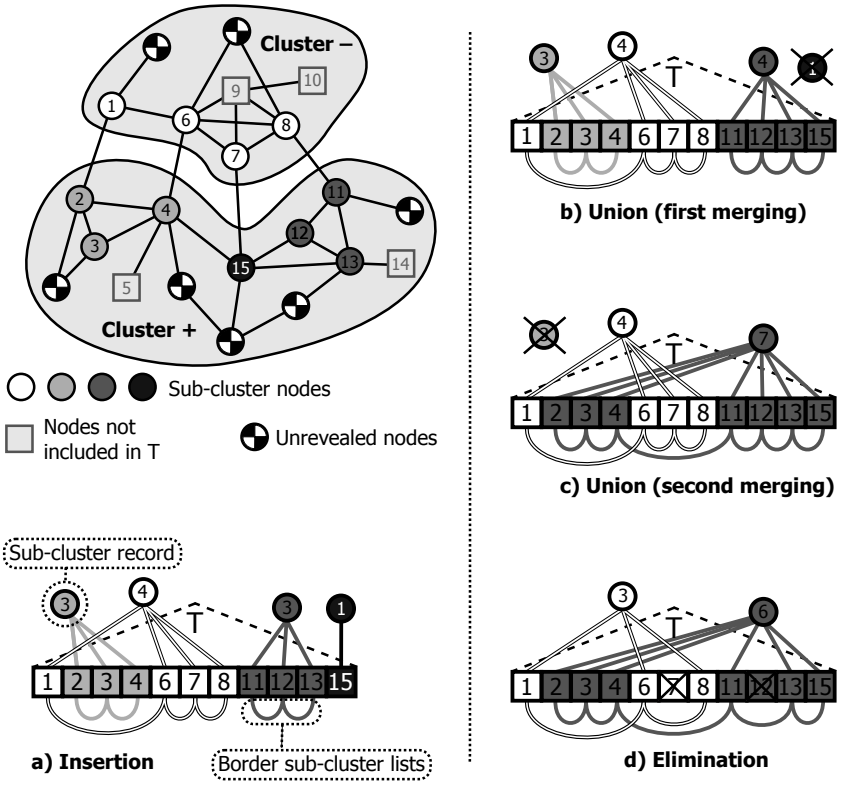

Figure 5.9: A snapshot of CGA and related data structures at time $t = 15$ on a labeled graph made up of two clusters (cluster "+" and cluster "−"). For simplicity in this figure $i_j = j$, for $j = 1, \ldots, 15$. The top left part shows the graph with revealed and unrevealed nodes. Just *before* node 15 gets revealed, the graph contains three subclusters: $C_1 = \{2, 3, 4, 5\}$, $C_2 = \{1, 6, 7, 8, 9, 10\}$, and $C_3 = \{11, 12, 13, 14\}$. The associated border sub-cluster lists $L(C_1) = \{2, 3, 4\}$, $L(C_2) = \{1, 6, 7, 8\}$, and $L(C_3) = \{11, 12, 13\}$ are organized into a balanced tree T. The corresponding sub-cluster records contain the cardinality of each list as well as a pointer to the first and the last element of the lists. Bottom left (a): When node 15 is revealed, a new (provisional) cluster $C_4 = \{15\}$ is created along with the associated list and record. On the right-hand side a sequence of merging operations between sub-clusters is shown. In particular: (b) shows how C_4 is merged with C_3; in (c) the result of the previous merger is merged with C_1. In (d) the elimination of all nodes which are no longer in the inner border of V_{15} (nodes 7 and 12 in this case) is shown.

presence of edge *weights*, our protocol of Section 5.2 could be modified to let CGA observe the weights of all edges incident to the current node. When-

ever the weights of intra-cluster edges are heavier than those of inter-cluster ones, our algorithm could take advantage of the additional weight information. This calls for an analysis able to capture the interaction between node labels and edge weights.

We may also consider scenarios where the optimal prediction on a node i is some (possibly stochastic) function of an unknown node parameter $\mathbf{u}_i \in \mathbb{R}^d$ and some time-dependent side information $\mathbf{x}_{i,t} \in \mathbb{R}^d$. In this model the advertising agent can potentially suffer loss upon each visit of the same node i, until $\mathbf{u}_i$ is learned sufficiently well. This creates a trade-off between exploration of new regions and exploitation of nodes that have been visited often enough in the past. In this context, a regular labelling of the graph is an assignment of vectors $\mathbf{u}_i$ such that nodes can be partitioned in a way that $\|\mathbf{u}_i - \mathbf{u}_j\|$ tends to be small whenever i and j belong to the same partition element.

Moreover, it would be interesting to see whether algorithms as efficient as CGA could be made competitive with respect to clustering of the nodes which are more general than regular partitions. Some examples of these weaker notions of valid clustering are mentioned in [7].

Finally, recalling that the lower bound of Theorem 2 holds for the transductive learning protocol as well —see Subsection 5.1.1, it would be interesting to further investigate the connections between on-line transductive learning protocols studied in Chapter 2 and 3 and semi-active learning protocols, like the one studied in this chapter.

Chapter 6

Conclusive remarks

We presented an in-depth investigation on the problem of classifying the nodes of a graph, whose edges represent the interaction between pairs of data entities. We focus on solving the problem by relying on the graph topology and the homophilic bias only. This is a significant departure from more standard approaches, which assume that the data set contains also information associated with each individual node. However, the scope of this study is not only restricted to this specific bias, since the validation of the power of the homiphilic principle can be seen as a first step toward the development of more complex methodologies, able to take into account both the relational components of the input data set and the information associated with each individual data element.

In developing our algorithms, we focus on providing theoretical guarantees on prediction accuracy and, at the same time, on computational efficiency. The development of methods that simultaneously guarantee optimal accuracy and computational efficiency is a very challenging goal. In fact, the accuracy of most methods in the literature is not rigorously analyzed from a theoretical point of view. Likewise, tight time and space complexity bounds are not generally provided. This contrasts with the need to manage extremely large relational datasets like, e.g., snapshots of the World Wide Web.

The systematic study of the node classification that we carried out poses several open problems and suggests new research directions to be investigated.

One of the most important aspect of our methodology, which is perhaps

the main key for obtaining very good scalability properties, is the use of spanning trees of the original graph. Spanning trees (especially uniformly generated) enable the use of fast combinatorial techniques and, at the same time, condense in a suitable way the structural information of the given graph. However, in the active setting (see Chapter 4), it is not easy to derive a theoretical guarantee on the performance of algorithms operating only with spanning trees of the input network. This suggests that, in some cases, the information contained in the networked data could perhaps be synthesized in a structure richer than a tree. A natural research direction for addressing this problematic issue might be represented by the study of a suitable combination of different spanning trees, which could shed light on the use of new sparsification techniques even for other node classification settings.

Another interesting research direction of our works is represented by extending our methods for handling datasets with several relations of different arities and different entity types. Assuming that the labeling are binary, if these kind of datasets are mapped into an undirected graph, for example with an incidence graph ([17] Chapter 1 Section 3), then one could use the algorithms described in Chapter 2 and 3, independently of the fact that prediction are not required for all vertex types. The mistake bound could be related to the complete labeling (involving also the nodes that are not originally associated to any label) which is consistent with the labels present in the dataset and minimizes the considered regularity measure. However, it is not clear how to interpret graph connectivity measures, like the effective resistance, relative to the incidence graph. Hence, it would be interesting to develop some analytical tool and new methodologies for handling this kind of datasets, providing performance guarantees that can be easily interpreted.

Some of the tools and methods we developed could be also be used for addressing link prediction problems, in which the learner must predict the existence of new links as the input network evolves in time. The study of the graph topology performed for classifying the nodes of the input graph could turn out to be useful even for solving this problem, since the research of new high-throughput sparsification techniques, similar to the uniformly generated random spanning tree, could reasonably be the key for developing scalable algorithms even in this context. For similar reasons, it is worthwhile to mention also the link classification problem (also known as edge sign

prediction problem —see [61]), in which the learner is required to predict the semantic binary label associated with each edge. In the link classification model each edge can be in fact positive, representing friendship or similarity, or negative, meaning antagonism or dissimilarity. As highlighted in [61], an input graph for this problem could be separated in different clusters such that the intra-cluster edges are mostly positive links, whereas the inter-cluster ones are mostly negative. A possible bridge between node and link classification problem is that a solution for the node classification problem entails a "collateral" partitioning in cluster for the input network, so that the two problems could be somehow cast in a common clustering framework.

Finally, besides the research directions mentioned above, the study we carried out on the node classification problem based on network topology and homophily could be combined both from a theoretical and practical viewpoint with classification methodologies relying on side information associated to each individual node. In fact, we believe that modularity is one of the main feature of our classification procedures, and the enhancement of our techniques to handle more structured relational data and more complex biases may be one of the most promising directions for future work.

Bibliography

[1] P. Afshani, E. Chiniforooshan, R. Dorrigiv, A. Farzan, M. Mirzazadeh, N. Simjour, H. Zarrabi-Zadeh. On the complexity of finding an unknown cut via vertex queries. COCOON 2007, pages 459–469.

[2] S. Albers, M. Henzinger. Exploring unknown environments, SIAM Journal on Computing 29 (4) 2000, pages 1164–1188.

[3] N. Alon, C. Avin, M. Koucky, G. Kozma, Z. Lotker and M.R. Tuttle. Many random walks are faster than one. In Proc. 20th SPAA, pages 119–128. ACM Press, 2008.

[4] J.R. Anderson. The Architecture of Cognition. Cambridge, MA: Harvard Univ. Press, 1983.

[5] N.T.J. Bailey. The Mathematical Theory of Infectious Diseases and Its Applications, Hafner Press, New York, 1975.

[6] M.F. Balcan, A. Blum, P.P. Choi, J. Lafferty, B. Pantano, M.R. Rwebangira, X. Zhu. Person identification in webcam images: An application of semi-supervised learning. ICML 2005 Workshop on Learning with Partially Classified Training Data.

[7] N. Balcan, A. Blum, S. Vempala. A discriminative framework for clustering via similarity functions, in: Proc. of the 40th ACM Symposium on the Theory of Computing, ACM Press, 2008.

[8] M. Belkin , I. Matveeva, P. Niyogi. Regularization and semi-supervised learning on large graphs. Proc. of COLT 2004, pages 624–638.

[9] M. Belkin and P. Niyogi. Towards a theoretical foundation for Laplacian-based manifold methods. Proc. of COLT, 2005.

[10] Y. Bengio , O. Delalleau, N. Le Roux. Label propagation and quadratic criterion. In Semi-Supervised Learning, pages 193–216. MIT Press, 2006.

[11] J. Besag. On the statistical analysis of dirty pictures, J. R. Statist. Soc. B-48, pages 259–302, 1986.

[12] J. Besag. Spatial interaction and the statis- tical analysis of lattice systems. J. Roy. Statist. SOC. B 36, pages 192–236, 1974.

[13] P. M. Blau. Inequality and Heterogeneity: A Primitive Theory of Social Structure. NewYork: Free Press, 1977.

[14] A. Blum, and S. Chawla. Learning from labeled and unlabeled data using graph mincuts. Proc. of ICML 2001, pages 19–26.

[15] A. Blum, J. Lafferty, R. Reddy, M.R. Rwebangira. Semi-supervised learning using randomized mincuts. ICML 2004.

[16] A. Bonato. A survey of properties and models of on-line social networks; In Proc. of the 5th International Conference on Mathematical and Computational Models, ICMCM 2009.

[17] J. A. Bondy and U. S. R. Murty. Graph Theory (Graduate Texts in Mathematics), Springer, 2008.

[18] A. Broder. Generating random spanning trees. In Proc. 30th FOCS, pages 442–447. IEEE Press, 1989.

[19] D. Bryant, V. Berry, A structured family of clustering and tree construction methods, Advances in Applied Mathematics 27, pages 705–732, 2001.

[20] M.A. Carreira-Perpinan, R.S. Zemel. Proximity graphs for clustering and manifold learning. In L. K. Saul, Y. Weiss and L. Bottou (Eds.), Advances in neural information processing systems 17. Cambridge, MA: MIT Press, 2005.

[21] C. Catutto, C. Schmitz, A. Baldassarri, V. D. P. Servedio, V. Loreto, A. Hotho, M. Grahl, G. Stumme. Network properties of folksonomies. AI

Communications Journal, Special Issue on Network Analysis in Natural Sciences and Engineering, 2007.

[22] N. Cesa-Bianchi, C. Gentile, F. Vitale. Fast and optimal prediction of a labeled tree. Proc. of COLT 2009.

[23] N. Cesa-Bianchi, C. Gentile, F. Vitale. Learning unknown Graphs. Accepted for publication in Theoretical Computer Science, special issue on Algorithmic Learning Theory.

[24] N. Cesa-Bianchi, C. Gentile, F. Vitale, G. Zappella. Active learning on trees and graphs. Proc. of COLT 2010.

[25] N. Cesa-Bianchi, C. Gentile, F. Vitale, G. Zappella. Random spanning trees and the prediction of weighted graphs. Proc. of ICML 2010.

[26] S. Chakrabarti, B. Dom, P. Indyk. Enhanced hypertext categorization using hyperlinks. In Proc. of ACM SIGMOD International Conference on Management of Data, pages 307–318, Seattle, Washington, 1998.

[27] H. Chang. and D.Y. Yeung. Graph Laplacian kernels for object classification from a single example. In IEEE CVPR, pages 2011–2016. IEEE Press, 2006.

[28] C. Chaomei. Mining the Web: Discovering knowledge from hypertext data. Journal of the American Society for Information Science and Technology, Volume 55, Issue 3, pages 275–276, 2004.

[29] T. Cormen, C. Leiserson, R. Rivest, Introduction to Algorithms, McGraw Hill, 1990.

[30] C. Cortes, D. Pregibon, C. Volinsky. Communities of Interest. Proc. of the Fourth Interna- tional Symposium on Intelligent Data Analysis, 2001.

[31] G. C. Cross and A. K. Jain. Markov random field texture models, IEEE Trans. Patt. Recog. and Mach. Intell. 5, pages 25–39, 1983.

[32] X. Deng, C. Papadimitriou, Exploring an unknown graph, in: Proc. of the 31st Annual Symposium on the Foundations of Computer Science, IEEE Press, pages 355–361, 1990.

[33] J. Fakcharoenphol and B. Kijsirikul. Low congestion online routing and an improved mistake bound for online prediction of graph labeling. Manuscript, 2008.

[34] N. Friedman, L. Getoor, D. Koller, A. Pfeffer. Learning probabilistic relational models. In Proc. of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI), pages 1300–1309, 1999.

[35] A. Ganesh, L. Massoulie, D. Towsley. The effect of network topology on the spread of epidemics, Proc. 24th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), vol. 2, pages 1455-1466, 2005.

[36] A.C. Gavin, et al. Functional organization of the yeast proteome by systematic analysis of protein complexes. Nature, 415 (6868): pages 141–147, 2002.

[37] S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. IEEE Trans. Pattern Analysis and Machine Intelligence, 6(6): pages 721–741, 1984.

[38] Lise Getoor, Nir Friedman, Daphne Koller, Benjamin Taskar Journal of Machine Learning Research, Volume 3, page 679- -707, 2002.

[39] A. Goldberg, and X. Zhu. Seeing stars when there aren't many stars: Graph-based semi-supervised learning for sentiment categorization. In HLT-NAACL 2006 Workshop on Textgraphs: Graph-based algorithms for Natural Language Processing, 2004.

[40] N. Goyal, L. Rademacher, S. Vempala. Expanders via random spanning trees. In Proc. SODA 2009, pages 576–585.

[41] A. Guillory and J. Bilmes. Label Selection on Graphs. NIPS 2009.

[42] S. Hanneke, An analysis of graph cut size for transductive learning, in: Proc. of ICML 2006, pages 393–399.

[43] M. Hassner and J. Sklansky. The use of Markov random fields as models of texture, Computer Graphics and Image Processing, vol. 12, pages 357–370, 1980.

[44] M. Herbster, Exploiting cluster-structure to predict the labeling of a graph, in: Proc. of the 19th International Conference on Algorithmic Learning Theory, Springer, 2008.

[45] M. Herbster, and G. Lever. Predicting the labelling of a graph via minimum p-seminorm interpolation. In Proc. COLT 2009.

[46] M. Herbster, G. Lever, M. Pontil, Online prediction on large diameter graphs, in: Advances in Neural Information Processing Systems 22, MIT Press, 2009.

[47] M. Hebster, M. Pontil, Prediction on a graph with the Perceptron, in: Advances in Neural Information Processing Systems 21, MIT Press, pages 577–584, 2007

[48] M. Herbster, M. Pontil, S. Rojas-Galeano, Fast prediction on a tree, in: Advances in Neural Information Processing Systems 22, MIT Press, 2009.

[49] M. Hebster, M. Pontil, L. Wainer, Online learning over graphs, in Proc. ICML 2005, pages 305–132.

[50] E. Ising. Beitrag zur theorie der ferromagnetismus. Zeitschrift 31 253-258, 1925.

[51] T. Ito, T. Chiba, R. Ozawa, M. Yoshida, M. Hattori, Y. Sakaki. A comprehensive two-hybrid analysis to explore the yeast protein interactome. PNAS, 8(98):4569–4574, 2001.

[52] T. Jebara, J. Wang, and S. F. Chang. Graph construction and b-matching for semi-supervised learning. In Proc. of ICML 2009.

[53] T. Joachims, Transductive learning via spectral graph partitioning, in: Proc. of ICML 2003, pages 305–132.

[54] J. Kandola, J. Shawe-Taylor, N. Cristianini. Learning semantic similarity. In Proc. of NIPS 2002.

[55] J. Katajainen, F. Vitale. Navigation piles with applications to sorting, priority queues, and priority deques. Nordic Journal of Computing, 10(3): pages 238–262, 2003.

[56] J. Kleinberg. Authoritative sources in a hyperlinked environment, Proc. 9th ACM-SIAM SODA, 1998.

[57] I. Kondor, J. Lafferty, Diffusion kernels on graphs and other discrete input spaces, in: Proc. of the 19th International Conference on Machine Learning, Morgan Kaufmann, pages 315–322, 2002.

[58] Krogan, N.J. et al. Global landscape of protein complexes in the yeast Saccharomyces cerevisiae. Nature, 440:pages 637–643, 2006.

[59] R. Kumar, J. Novak, A. Tomkins. Structure and evolution of online social networks. In 12th KDD, pages 611–617, 2006.

[60] N. Lavrac and S. Dzeroski. Inductive Logic Programming: Techniques and Applications. Ellis Horwood, 1994.

[61] J. Leskovec, D. P. Huttenlocher, J. M. Kleinberg. Predicting positive and negative links in online social networks. In M. Rappa, P. Jones, J. Freire, and S. Chakrabarti, editors, WWW, pages 641–650. ACM, 2010.

[62] D. Liben-Nowell, J. Kleinberg. The Link Prediction Problem for Social Networks. Proc. 12th International Conference on Information and Knowledge Management CIKM, 2003.

[63] R. Lyons and Y. Peres. Probability on Trees and Networks. Manuscript, 2009.

[64] S. A. Macskassy and F. Provost. A simple relational classifier. In Proc. of the Multi- Relational Data Mining Workshop (MRDM) at the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2003.

[65] M. McPherson, L. Smith-Lovin, J. M. Cook. Birds of a Feather: Homophily in Social Networks. Annual Review of Sociology, 27: pages 415–444, 2001.

[66] N. Memon , A. R. Qureshi. Investigative data mining and its application in counterterrorism, Proc. of the 5th WSEAS International Conference on Applied Informatics and Communications, pages 397–403, 2005.

[67] J. Neville and D. Jensen. Dependency networks for relational data. In Proceedings of the Fourth IEEE International Conference in Data Mining (ICDM), pages 170–177, 2004.

[68] M. E. J. Newman. The spread of epidemic disease on networks, Phys. Rev. E, 66:016128, 2002.

[69] L. Ngo and P. Haddawy. Probabilistic logic programming and bayesian networks. In Algorithms, Concurrency and Knowledge (Proc. ACSC95), volume 1023 of Lecture Notes in Computer Science, pages 286–300. Springer-Verlag, 1995.

[70] K. Onak and P. Parys. Generalization of binary search: searching in trees and forest-like partial orders. In Proc. 47th FOCS, pages 379–388. IEEE Press, 2006.

[71] G. Pandey, M. Steinbach, R. Gupta, T. Garg, V. Kumar. Association analysis-based transformations for protein interaction networks: a function prediction case study. In Proc. 13th ACM KDD, pp. 540–549. ACM Press, 2007.

[72] L- Page, S. Brin, R. Motwani, T. Winograd. The Pagerank citation ranking: Bringing order to the web, Technical report, Stanford University, 1998.

[73] J. Pearl. Probabilistic Reasoning in Intelligent Systems. Morgan Kaufmann, 1988.

[74] K. Pelckmans, An online algorithm for learning a labeling of a graph, in: 6th International Workshop on Mining and Learning with Graphs, 2008.

[75] J. Pelckmans, J. Shawe-Taylor, J. Suykens, B. D. Moor, Margin based transductive graph cuts using linear programming, in: Proc. of the 11th International Conference on Artificial Intelligence and Statistics, JMLR Proc. Series, pages 360–367, 2007.

[76] D. Poole. Probabilistic Horn abduction and Bayesian networks. Artificial Intelligence, 64:81–129,1993.

[77] A. Popescul, L. Ungar, S. Lawrence, D. Pennock. Towards structural logistic regression: Combining relational and statistical learning. In Proc. KDD-2002 Workshop on Multi-Relational Data Mining. ACM, 2002.

[78] J. Remy, A. Souza, A. Steger, On an online spanning tree problem in randomly weighted graphs, Combinatorics, Probability and Computing 16 pages 127–144, 2007.

[79] B.D. Ripley. Spatial Statistics, John Wiley and Sons Ltd, Chichester, UK. 1981.

[80] A. Ruepp. The FunCat, a functional annotation scheme for systematic classification of proteins from whole genomes. Nucleic Acids Research, 32(18):pages 5539–5545, 2004.

[81] H. Shin, K. Tsuda, B. Schölkopf. Protein functional class prediction with a combined graph. Expert Systems with Applications, 36:3284–3292. Elsevier, 2009.

[82] J. Shrager, T. Hogg, B. A. Huberman. Observation of phase transitions in spreading activation networks. Science, 236:1092–1094, 1987.

[83] S. Slattery and M. Craven. Combining statistical and relational methods for learning in hypertext domains. In David Page, editor, Proceedings of ILP-98, 8th International Conference on Inductive Logic Programming, pages 38–52, Madison, US, 1998. Springer Verlag, Heidelberg, DE.

[84] S. Slattery and T. Mitchell. Discovering test set regularities in relational domains. In Proc. of the Seventeenth International Conference on Machine Learning, pages 895–902, Stanford, California, 2000.

[85] A. Smola, I. Kondor, Kernels and regularization on graphs, in: Proc. of the 16th Annual Conference on Learning Theory, Springer, pages 144–158, 2003.

[86] D.A. Spielman and N. Srivastava. Graph sparsification by effective resistances. In Proc. 40th STOC. ACM Press, 2008.

[87] M. Szummer, T. Jaakkola. Partially labeled classification with Markov random walks. Advances in Neural Information Processing Systems, 14, 2001.

[88] B. Taskar, P. Abbeel, D. Koller. Discriminative probabilistic models for relational data. In Eighteenth Conference on Uncertainty in Artificial Intelligence, 2002.

[89] B. Taskar, E. Segal, D. Koller. Probabilistic classification and clustering in relational data. In Proc. IJCAI01, pages 870–876, Seattle, Wash., 2001

[90] H. Tsuda, K. Shin, B. Schölkopf. Protein functional class prediction with a combined graph. Expert Systems with Applications, 36:pages 3284–3292, 2009.

[91] P. Uetz et al. Pa comprehensive analysis of protein-protein interactions in Saccharomyces cerevisiae. Nature, 6770(403):pages 623–627, 2000.

[92] W. Yang, J. Dia, Discovering cohesive subgroups from social networks for targeted advertising, Expert Systems with Applications 34, pages 2029–2038, 2008.

[93] Web Spam Challenge, `webspam.lip6.fr`.

[94] D.B. Wilson. Generating random spanning trees more quickly than the cover time. In Proc. 28th STOC, pages 206–303. ACM Press, 1996.

[95] J. W. Woods. Markov image modeling, IEEE Transactions on Automatic Control, vol. 23, pages 846–850, 1978.

[96] Y. Liu and Z. Kou. Predicting who rated what in large-scale datasets. In Proc. of the KDD Cup 2007, pages 59–62. ACM Press, 2007.

[97] W.S. Yang, J.B. Dia, H.C. Cheng, H.T. Lin. Mining social networks for targeted advertising. Hawaii International Conference on System Sciences (HICSS), 6:137a, 2006.

[98] T. Yang, R. Jin, Y. Chi, S. Zhu. Combining link and content for community detection: a discriminative approach. In KDD, pages 927–936, 2009.

[99] D. Zhou, O. Bousquet, T. Lal, J. Weston, B. Schlkopf. Learning with local and global consistency. Advances in Neural Information Processing System 16, 2004.

[100] X. Zhu. Semi-supervised learning literature survey. Computer Sciences Technical Report 1530, University of Wisconsin-Madison, 2005.

[101] X. Zhu. and Z. Ghahramani. Learning from labeled and unlabeled data with label propagation. Technical Report CMU-CALD-02-107, 2002.

[102] X. Zhu, Z. Ghahramani, J. Lafferty. Semi-supervised learning using Gaussian fields and harmonic functions. In ICML Workshop on the Continuum from Labeled to Unlabeled Data in Machine Learning and Data Mining, 2003.