Predicting the Labels of an Unknown Graph via Adaptive Exploration

Nicolò Cesa-Bianchi

Dipartimento di Scienze dell'Informazione
Università degli Studi di Milano, Italy

Claudio Gentile

Dipartimento di Informatica e Comunicazione
Università dell'Insubria, Varese, Italy

Fabio Vitale*

Dipartimento di Scienze dell'Informazione
Università degli Studi di Milano, Italy

Abstract

Motivated by a problem of targeted advertising in social networks, 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. In general, the merging degree is small when the vertices of the unknown graph can be partitioned into a few well-separated clusters within which labels are roughly constant. For the special case of binary labeled graphs, the merging degree is a more refined measure than the cutsize. 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. A matching lower bound shows that in the

*Corresponding author

Email addresses: nicolo.cesa-bianchi@unimi.it (Nicolò Cesa-Bianchi), claudio.gentile@uninsubria.it (Claudio Gentile), fabio.vitale@unimi.it (Fabio Vitale)

case of binary labels our analysis cannot be improved.

Keywords: online learning, graph prediction, graph clustering

1. Introduction

The study of online (game-theoretic) pattern recognition has traditionally focused on algorithms for learning linear functions defined on vector spaces. Many algorithms have been devised in this context, perhaps the most popular being the classical Perceptron algorithm. In recent years, there has been a growing interest in online learning problems where instances come from domains that are not easily structured as a linear space. The papers [6, 10, 11, 12, 13, 14, 15] investigate the problem of predicting in an online fashion the binary labels of vertices of an undirected graph. The learning model these authors formulate 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 paper 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 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 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 [22]), we design a learning/exploration strategy that perform well to the extent that the underlying graph labeling $\boldsymbol{y} = (y_1, \ldots, y_n)$ 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.

In the case of binary labels and zero-one loss, a common measure of label regularity for an n-vertex graph G with labels $\boldsymbol{y} = (y_1, \ldots, y_n) \in \{-1, +1\}^n$ is the *cutsize* $\Phi_G(\boldsymbol{y})$. This is the number of edges (i, j) in G whose endpoints vertices have disagreeing labels, $y_i \neq y_j$. The cumulative loss bound we prove in this paper 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 $\Phi_G(\boldsymbol{y})$, and never larger than $2\Phi_G(\boldsymbol{y})$. Furthermore, unlike $\Phi_G(\boldsymbol{y})$, 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 paper 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 paper is organized as follows. In the next subsection we briefly overview some related work. The exploration/prediction protocol is introduced in Section 2. We define our measure of graph regularity in Section 3. In Section 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. In Section 6 we describe time and space efficient implementations of our algorithm. We conclude in Section 7 with some comments and a few open questions.

1.1. Related work

Online prediction of labeled graphs has often been studied in a "transductive" learning model different from the one studied here. In the transductive model the graph G (without labels) is known in advance, and the task is to sequentially predict the unknown labels of an adversarially chosen permutation of G's vertices. A technique proposed in [10] for transductive binary prediction is to embed the graph into a linear space using the kernel defined by the Laplacian pseudoinverse —see [17, 21], and then run the standard (kernel) Perceptron algorithm for predicting the vertex labels. This approach guarantees that the number of mistakes is bounded by a quantity that depends linearly on the cutsize $\Phi_G(\boldsymbol{y})$. Further results involving the prediction of node labels in graphs with known structure include [3, 4, 6, 9, 11, 12, 13, 14, 16, 18]. Since all these papers assume knowledge of the entire graph in advance, the techniques proposed for transductive binary prediction 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 work.

On the other hand, our exploration/prediction model bears some similarities to the graph exploration problem introduced in [8], 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 2), whereas depth-first traversal is optimal in the setting of [8] for any undirected graph —see [1].

As explained in Section 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 online is also considered in [20], although only for graphs with random edge costs.

2. The exploration/prediction protocol

Let $G = (V, E)$ be an unknown undirected and connected graph with vertex set $V = \{1, 2, \dots, n\}$ and edge set $E \subseteq V \times V$. We use $\boldsymbol{y} = (y_1, \dots, y_n) \in \mathcal{Y}^n$ 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 $t = 1, \dots, n-1$, 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 $t = 1, \dots, n-1$, 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 $t = 1, \dots, n-1$, form a spanning tree for G. This is key to understanding the way our algorithm works —see Section 4.

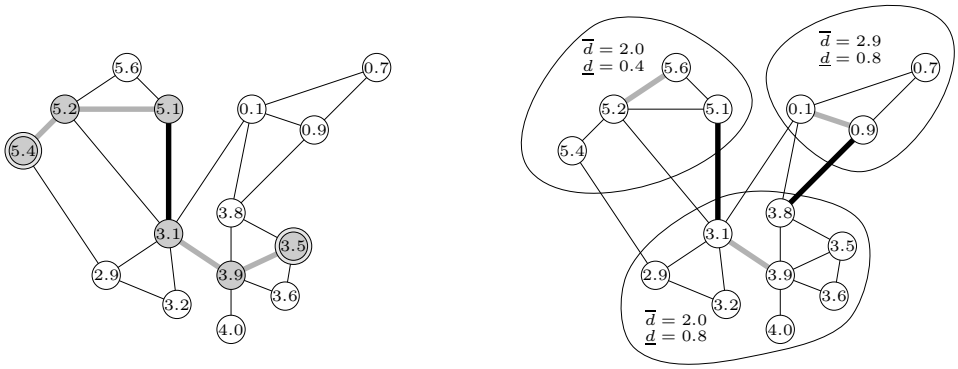

Figure 1: Two copies of a graph with real labels y_i associated with each vertex i. On the left, a shortest path connecting the two nodes enclosed in double circles is shown. The path length is $\max_t \ell(s_{k-1}, s_k)$, where $\ell(i,j) = |y_i - y_j|$. The thick black edge is incident to the nodes 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.

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 $\lambda(s_1, \ldots, s_d) = 0$, otherwise $\lambda(s_1, \ldots, s_d) = 1$.

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

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

for each path $s_1, \ldots, s_{d-1}, s_d$ in G. As we see in Section 6, condition (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 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 [5] and to the *strict separation property* of [2].

It is easy to see that, because of (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.

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 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** $\underline{\partial C}$ of C to be the set of all nodes $i \in C$ that are adjacent to any node $j \notin C$. The **outer 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 3 for an example.

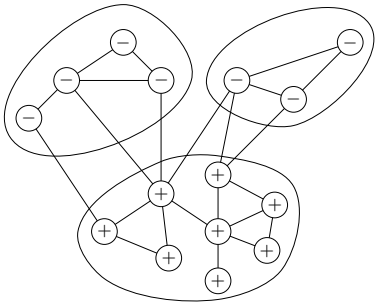

Figure 2: A (natural) regular partition for a graph with labels in $\{-1, +1\}$. The path length is measured as $\max_k \ell(s_{k-1}, s_k)$, where $\ell(i, j) = |y_i - y_j|$. The diameter of each cluster C (the maximum of the pairwise distances between nodes of C) is equal to 0, whereas the minimum of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

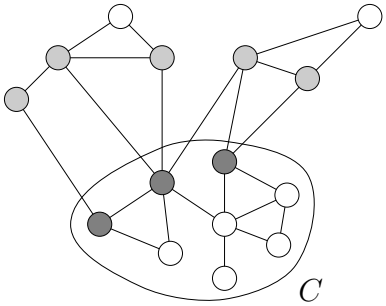

Figure 3: The inner border of the depicted subset C is the set of dark grey nodes, the outer border is made up of the light grey nodes, hence $|\underline{\partial C}| = 3$ and $|\overline{\partial C}| = 5$.

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

$$\delta(C) = \min\big\{|\underline{\partial C}|, |\overline{\partial C}|\big\} \ .$$

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

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

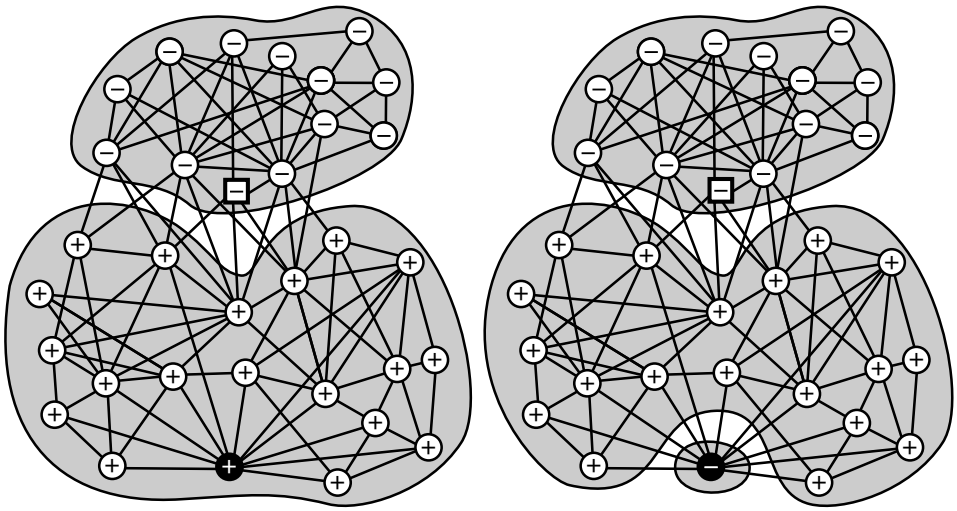

Figure 4: A relatively dense graph G (repeated twice) with two clusters C_1 and C_2 (left-hand side, from top to bottom), or three clusters C_1, C_2, and C_3 (right-hand side, from top to bottom), depending on the label of the black node at the bottom. If negative, this label might naturally be viewed 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 $\Phi_G(\boldsymbol{y}) = 14$ and $\delta(\mathcal{P}) = \delta(C_1) + \delta(C_2) = 5 + 5 = 10$, while for the graph on the right $\Phi_G(\boldsymbol{y}) = 25$ 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, $\Phi_G(\boldsymbol{y})$ would remain unchanged.

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 $\Phi_G(\boldsymbol{y})$ must be incident to each node in an inner or outer border of a cluster, $\delta(\mathcal{P})$ is never larger than $2\Phi_G(\boldsymbol{y})$. On the other hand, as suggested for example by Figure 4, $\delta(\mathcal{P})$ is often much smaller $\Phi_G(\boldsymbol{y})$. This is directly 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 $\Phi_G(\boldsymbol{y})$ 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 $\Phi_G(\boldsymbol{y})$ 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 4, $\delta(\mathcal{P})$ is typically more robust to label noise than $\Phi_G(\boldsymbol{y})$. 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 $\delta(\mathcal{P}, \boldsymbol{y})$ to emphasize the dependence of the merging degree on the labeling $\boldsymbol{y} \in \{-1, +1\}^n$. Let $\boldsymbol{y}_{\text{old}}$ be the labeling before the flip of y_i and $\boldsymbol{y}_{\text{new}}$ be the one after the flip. The following statement is easily verified. It provides *sufficient* conditions to insure that, after the label flip, $\delta(\mathcal{P}, \boldsymbol{y})$ cannot change by more than 2.

Fact 1. *Given a graph $G = (V, E)$ with labeling $\boldsymbol{y} \in \{-1, +1\}^n$ 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 cluster containing node i). If i is neither a border node of the cluster nor an articulation node[1] of G_i, then $\left|\delta(\mathcal{P}, \boldsymbol{y}_{\text{new}}) - \delta(\mathcal{P}, \boldsymbol{y}_{\text{old}})\right| \leq 2$ while $\left|\Phi_G(\boldsymbol{y}_{\text{new}}) - \Phi_G(\boldsymbol{y}_{\text{old}})\right|$ is always equal to the degree of i.*

See again Figure 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 shown in Figure 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 $\delta(\mathcal{P}, \boldsymbol{y})$ as compared to $\Phi_G(\boldsymbol{y})$. 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

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

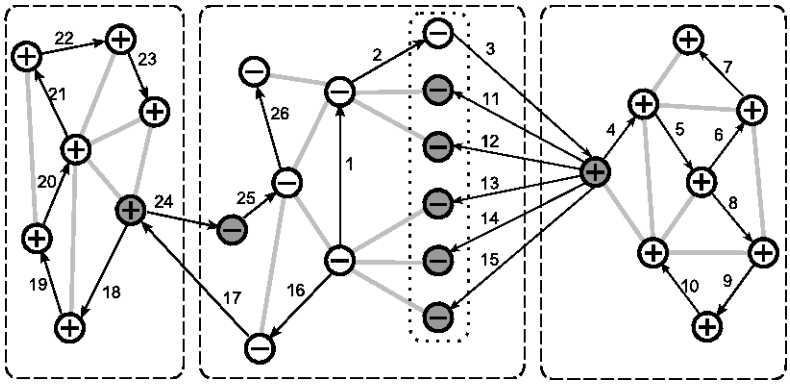

Figure 5: A binary labeled graph with three clusters such that $\delta(\mathcal{P}) = 4$ and $\Phi_G(\boldsymbol{y}) = 8$. We show that DEPTHFIRST makes order of $\Phi_G(\boldsymbol{y})$ mistakes. Arrow edges indicate predictions, and numbers on such edges denote the adversarial order of presentation. 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 $\Phi_G(\boldsymbol{y})$ 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.

cases, it is straightforward to verify that

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

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.

4. Adaptive vs. nonadaptive strategies and 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 $\Phi_G(\boldsymbol{y})$ is an obvious mistake bound achieved by such a strategy. Figure 5 shows an example where $\delta(\mathcal{P}) = \mathcal{O}(1)$ while DEPTHFIRST makes $\Phi_G(\boldsymbol{y}) = \Omega(|V|)$ 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 [19] for binary classification. This algorithm predicts at time t just with the majority vote of the labels of previously mistaken nodes that are adjacent to i_t. The number of mistakes satisfies $|E_{\mathcal{M}}| \leq 2\,\Phi_G(\boldsymbol{y})$, 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 shows that, even in our easier protocol, this algorithm makes order of $\Phi_G(\boldsymbol{y})$ 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, the cumulative loss of CGA can be expressed in terms of the *best* regular partition of G with respect to the unknown labeling $\boldsymbol{y} \in \mathbb{R}^n$, 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 $\big\{(q_t, i_t) \,:\, t = 1, \ldots, |V| - 1\big\}$ 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,

$$\widehat{y}_t = y_{q_t} \quad \text{where} \quad q_t = \operatorname*{argmin}_{q \in \underline{\partial V}_{t-1}} d_{t-1}(i_{t-1}, q) \ . \tag{2}$$

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 (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 6 contains a pictorial explaination of the behavior of CGA, as compared to DEPTHFIRST on the same binary labeled graph as in Figure 5. As we argue in the next section (Lemma 1 in Section 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. This suggests a fundamental difference between CGA and simple algorithms like DEPTHFIRST. Evidence of that is provided by comparing Figure 5 to Figure 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). 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. Analysis

This section contains the analysis of CGA's predictive performance. The computational complexity analysis is contained in Section 6. For the sake

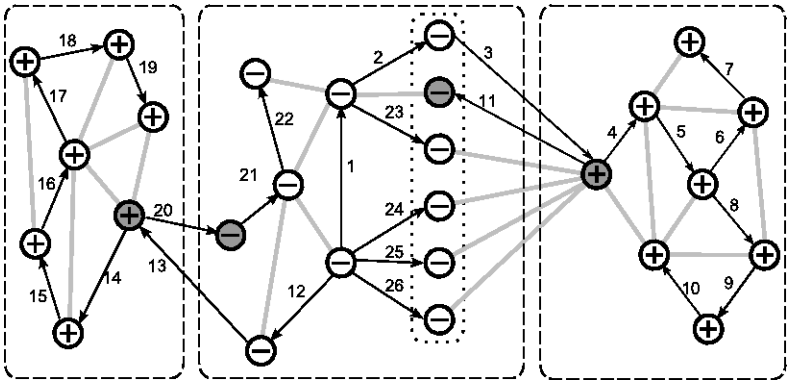

Figure 6: The behavior of CGA displayed on the binary labeled graph of Figure 5. The length of a path $s_1, \ldots, s_d$ is measured by $\max_k \ell(s_{k-1}, s_k)$ and the loss is the zero-one loss. The pictorial conventions are as in Figure 5. As in that figure, the cutsize $\Phi_G(\boldsymbol{y})$ 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 $\Phi_G(\boldsymbol{y})$ times on this graph.

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.

Lemma 1. *Assume* CGA *is run on a graph G with labeling $\boldsymbol{y} \in \mathcal{Y}^n$, 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 $C \cap \underline{\partial V}_{t-1} \equiv \emptyset$ and that the selection rule (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 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., $q' \in \underline{\partial V}_{t-1}$), and (c) is connected to i_{t-1} by a path all contained in

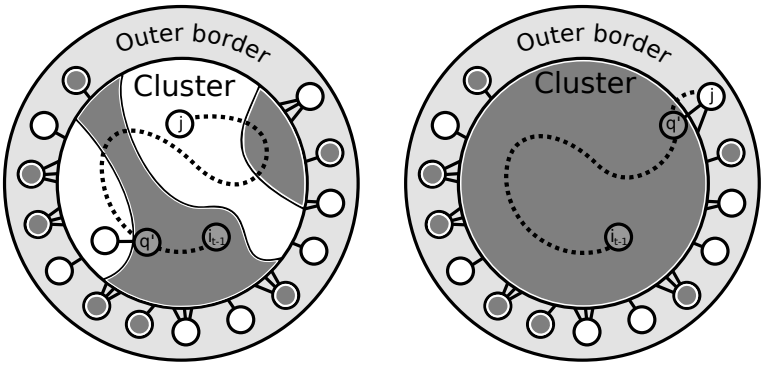

Figure 7: Two clusters corresponding to the two cases mentioned in the proof of Lemma 1. In both clusters the dark shaded area is $C \cap V_{t-1}$ (i.e., the set of nodes in cluster C that have already been explored) and the white area is $C \setminus V_{t-1}$. Case 1 (left cluster): A node j in C exists 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.

$C \cap V_{t-1}$. Since the partition is regular, q' is closer to i_{t-1} than to any node outside of C. Hence, by construction —see (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 $\underline{\partial C}$ 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]

[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.

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 (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 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

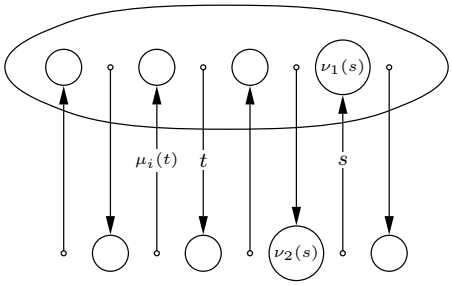

Figure 8: Sequence (starting from the left) of incoming and regular outgoing lossy steps involving a given cluster C_i. We only show the border nodes contributing to lossy steps. We map injectively each regular outgoing lossy step t to the previous (incoming) lossy step $\mu_i(t)$. We also map injectively each incoming lossy step s to the node $\nu_1(s)$ in the inner border, whose label was predicted at time s. Finally, we map injectively s also to the node $\nu_2(s)$ in the outer border that caused the previous (outgoing) lossy step for the same cluster.

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

$$m = \sum_{i\in\mathcal{P}} |M^{\text{reg}}_{i\to} \cup M^{\text{irr}}_{i\to}| \leq \sum_{i\in\mathcal{P}} \big(|M_{\to i}| - \mathbb{I}_i + |M^{\text{irr}}_{i\to}|\big) \leq \sum_{i\in\mathcal{P}} |M_{\to i}| \ . \qquad (3)$$

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

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

for each $i \in \mathcal{P}$. Together with (3), which we prove next, this completes the proof (see again Figure 8 for a pictorial explanation).

The first injective mapping $\nu_1 : M_{\to i} \to \underline{\partial C_i}$ 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 online 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 $\Phi_T(\boldsymbol{y})$ 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 $\boldsymbol{y}$ 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 OF THEOREM 2. 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

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

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 $\underline{\partial C}$ 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.,

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

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 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 (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*

$$\sum_{t=1}^{n} \ell(\widehat{y}_t, y_t) \leq \min_{\mathcal{P}} \left(|\mathcal{P}| \big(\ell_{\mathcal{P}}^{\max} - \bar{\ell}_{\mathcal{P}} \big) + \sum_{C \in \mathcal{P}} \Big(\varepsilon(C) + \ell(C)\delta(C) \Big) \right) \tag{4}$$

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

Remark 1. If ℓ is the zero-one loss, then the bound in (4) reduces to

$$\sum_{t=1}^{n} \ell(\widehat{y}_t, y_t) \leq \min_{\mathcal{P}} \sum_{C \in \mathcal{P}} \Big(\varepsilon(C) + \delta(C) \Big) \ . \tag{5}$$

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) 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 3 was called the natural regular

partition — see Theorem 1). In most cases, the natural regular partition is the minimizer of (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) becomes $\varepsilon(V)$ which, in turn, is bounded by $\Phi_G(\boldsymbol{y})$.

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).

Proof of Theorem 3. Fix an arbitrary regular partition $\mathcal{P}$ of V and index by $1, \ldots, |\mathcal{P}|$ 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 $M_{\to i}$, $M_{i\to}^{\mathrm{reg}}$ and $M_{i\to}^{\mathrm{irr}}$) as in the binary case proof, as well as the injective mapping μ_i. In the binary case we bounded $|M_{i\to}^{\mathrm{reg}}|$ by $|M_{\to i}| - \mathbb{I}_i$. In a similar fashion, we now bound $\sum_{t \in M_{i\to}^{\mathrm{reg}}} \ell_t$ 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

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

where the second inequality follows from $\sum_{i\in\mathcal{P}}\mathbb{I}_i \geq \sum_{i\in\mathcal{P}}|M_{i\rightarrow}^{\mathrm{irr}}|$ (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 $|M_{\rightarrow i}|$ through $\delta(C_i)$. □

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 (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 $i,\ldots,j,j'$ can be computed in *constant* time . This assumption is easily seen to hold for many natural path length assignments λ over graphs, for instance $\lambda(s_1,\ldots,s_d)=\max_k \ell(s_{k-1},s_k)$ and $\lambda(s_1,\ldots,s_d)=\sum_k \ell(s_{k-1},s_k)$ —note that both these assignments fulfill (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., [7, 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 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 [7, 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 $\underline{\partial V}_{t-1}$ 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.

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 $\underline{\partial V}_t$. Each node of T corresponds to a node in $\underline{\partial V}_t$ and contains the associated label. We will refer to nodes in $\underline{\partial V}_t$ and to nodes in T interchangeably.

- Given a sub-cluster C, all nodes in $C \cap \underline{\partial V}_t$ 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

[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 $\underline{\partial V}_{t-1}$ nearest to i_t in G_t has been found.

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 $\underline{\partial V}_{t-1}$ 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 $|\underline{\partial V_t}| \leq t$.
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., [7, Ch. 22]).
3. **Elimination** of nodes. All nodes in $\{i_t\} \cup N(i_t)$ that are not part of $\underline{\partial V}_t$ 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 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. Each of these i_t's neighbors can be bijectively associated with an edge of E (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.

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 1.1,

[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 $\mathcal{O}(|E| + |V| \log |V|)$.

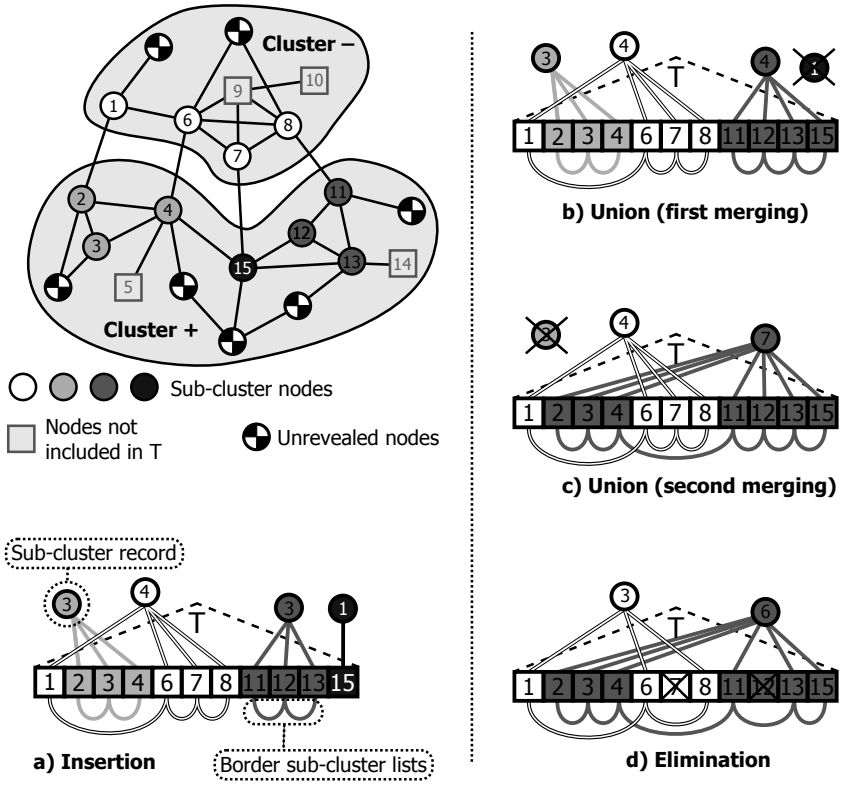

Figure 9: A snapshopt of CGA and related data structures at time $t = 15$ on a labeled graph made up of two clusters (cluster "+" and cluster "−"). For simplicity in this figure $i_j = j$, for $j = 1, \ldots, 15$. The top left part shows the graph with revealed and unrevealed nodes. Just *before* node 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.

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 presence

of edge *weights*, our protocol of Section 2 could be modified to let CGA observe the weights of all edges incident to the current node. Whenever 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 $\boldsymbol{u}_i \in \mathbb{R}^d$ and some time-dependent side information $\boldsymbol{x}_{i,t} \in \mathbb{R}^d$. In this model the advertising agent can potentially suffer loss upon each visit of the same node i, until $\boldsymbol{u}_i$ 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 $\boldsymbol{u}_i$ such that nodes can be partitioned in a way that $\|\boldsymbol{u}_i - \boldsymbol{u}_j\|$ 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 [2].

Finally, recalling that the lower bound of Theorem 2 holds for the transductive learning protocol as well —see Subsection 1.1, it would be interesting to further investigate the connections between online transductive learning protocols and semi-active learning protocols, like the one studied in this paper.

Acknowledgments

We would like to thank the anonymous reviewers for their comments which greatly improved the presentation of this paper. This work was supported in part by the PASCAL2 Network of Excellence under EC grant 216886. This publication only reflects the authors' views.

References

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

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

[3] A. Blum, S. Chawla, Learning from labeled and unlabeled data using graph mincuts, in: Proceedings of the 18th International Conference on Machine learning, Morgan Kaufmann, 2001.

[4] A. Blum, J. Lafferty, M. Rwebangira, R. Reddy, Semi-supervised learning using randomized mincuts, in: Proceedings of the 21st International Conference on Machine learning, 2004.

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

[6] N. Cesa-Bianchi, C. Gentile, F. Vitale, Fast and optimal prediction of a labeled tree, in: Proceedings of the 22nd Annual Conference on Learning Theory, Omnipress, 2009.

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

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

[9] S. Hanneke, An analysis of graph cut size for transductive learning, in: Proceedings of the 23rd International Conference on Machine learning, ACM Press, 2006, pp. 393–399.

[10] M. Hebster, M. Pontil, Prediction on a graph with the Perceptron, in: Advances in Neural Information Processing Systems 21, MIT Press, 2007, pp. 577–584.

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

[12] M. Herbster, G. Lever, M. Pontil, Online prediction on large diameter graphs, in: Advances in Neural Information Processing Systems 22, MIT Press, 2009.

[13] M. Herbster, M. Pontil, S. Rojas-Galeano, Fast prediction on a tree, in: Advances in Neural Information Processing Systems 22, MIT Press, 2009.

[14] M. Herbster, G. Lever, Predicting the labelling of a graph via minimum p-seminorm interpolation, in: Proceedings of the 22nd Annual Conference on Learning Theory, Omnipress, 2009.

[15] M. Hebster, M. Pontil, and L. Wainer, Online learning over graphs, in Proc. 22nd ICML, ACM Press, 2005, pp 305–132.

[16] T. Joachims, Transductive learning via spectral graph partitioning, in: Proceedings of the 20th International Conference on Machine learning, AAAI Press, 2003, pp. 305–132.

[17] I. Kondor, J. Lafferty, Diffusion kernels on graphs and other discrete input spaces, in: Proceedings of the 19th International Conference on Machine Learning, Morgan Kaufmann, 2002, pp. 315–322.

[18] J. Pelckmans, J. Shawe-Taylor, J. Suykens, B. D. Moor, Margin based transductive graph cuts using linear programming, in: Proceedings of the 11th International Conference on Artificial Intelligence and Statistics, JMLR Proceedings Series, 2007, pp. 360–367.

[19] K. Pelckmans, An online algorithm for learning a labeling of a graph, in: 6th International Workshop on Mining and Learning with Graphs, 2008.

[20] J. Remy, A. Souza, A. Steger, On an online spanning tree problem in randomly weighted graphs, Combinatorics, Probability and Computing 16 (2007) 127–144.

[21] A. Smola, I. Kondor, Kernels and regularization on graphs, in: Proceedings of the 16th Annual Conference on Learning Theory, Springer, 2003, pp. 144–158.

[22] W. Yang, J. Dia, Discovering cohesive subgroups from social networks for targeted advertising, Expert Systems with Applications 34 (2008) 2029–2038.