Learning Unknown Graphs

Nicolò Cesa-Bianchi[1], Claudio Gentile[2], and Fabio Vitale[3]

[1] Dipartimento di Scienze dell'Informazione
Università degli Studi di Milano, Italy
cesa-bianchi@dsi.unimi.it
[2] Dipartimento di Informatica e Comunicazione
Università dell'Insubria, Varese, Italy
claudio.gentile@uninsubria.it
[3] Dipartimento di Scienze dell'Informazione
Università degli Studi di Milano, Italy
fabio.vitale@unimi.it

Abstract. Motivated by a problem of targeted advertising in social networks, we introduce and study a new model of online learning on labeled graphs where the graph is initially unknown, and the algorithm is free to choose the next vertex to predict. After observing that natural non-adaptive exploration/prediction strategies (like depth-first with majority vote) badly fail on simple binary labeled graphs, we introduce an adaptive strategy that performs well under the hypothesis that the vertices of the unknown graph (i.e., the members of the social network) can be partitioned into a few well-separated clusters within which labels are roughly constant (i.e., members in the same cluster tend to prefer the same products). Our algorithm is efficiently implementable and provably competitive against the best of these partitions.

Key words: online learning, graph prediction, unknown graph, clustering.

1 Introduction

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. Unlike previous approaches to this problem —see, e.g., [20]— we consider the more interesting scenario where the network and the preferences of network members for the products in a given set are initially unknown, apart from those of a single "seed member". We assume there exists a mechanism to explore the social network by discovering new members connected (i.e., with similar interests) to members that are already known. This mechanism can be implemented in different ways, e.g., by providing incentives or rewards to members with undiscovered 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 member is presumably costly, the goal of the marketing strategy is to minimize the number of new members not being offered their preferred product. In this respect the task may then be formulated

as the following sequential problem: At each step t find the member q_t, among those whose preferred product we already know, who is most likely to have undiscovered connections that have the same preferred product as q_t. Once this member q_t is identified, we obtain (through the above-mentioned mechanism) a connection i_t to whom we may advertise q_t's preferred product. In order to make the problem easier for the advertising agent, we make the simplifying assumption that once a product is advertised to a member the agent may observe the member's true preference, and thus know whether the decision made was optimal.

This social network advertising task can be naturally cast as a graph prediction problem where an agent sequentially explores the vertices and edges of an unknown graph with unknown labels (i.e., product preferences) assigned to its vertices. The online exploration proceeds as follows: At each time step t, the agent 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. Thus, in some sense, the agent is learning to explore the graph along directions that, given past observations, look easier to predict. Our basic measure of performance is the agent's cumulative loss $\ell(\widehat{y}_1, y_1) + \cdots + \ell(\widehat{y}_n, y_n)$ over a sequence of n predictions.

In order to leverage on the assumption that connected members tend to prefer the same products [20], we design agent strategies that perform well to the extent that the underlying graph labeling $\boldsymbol{y} = (y_1, \ldots, y_n)$ is regular. That is, 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 that, in the special case of binary labels, 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 be even quadratic in the number of nodes, our measure of label regularity is never vacuous (i.e., it is never larger than n). In the paper we also show that the algorithm achieving this bound is suitable to large scale applications because of its small time and memory requirements.

1.1 Related Work

Online prediction of labeled graphs has been also studied in a "transductive" learning model, different from the one studied here. In this 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 [16, 19], 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 [2, 3, 6, 9, 11, 12, 13, 14, 15, 17].

Our exploration/prediction model also 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 Sect. 2), whereas depth-first traversal is optimal in the setting of [8] for any undirected graph —see [1]. Finally, as we explain in Sect. 3, our exploration/prediction algorithm incrementally builds 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 [18], although only for graphs with random edge costs.

2 The Exploration/Prediction Model

Let $G = (V, E)$ be an unknown undirected and connected graph with vertex set $V = \{1, 2, \ldots, n\}$ and edge set $E \subseteq V \times V$. We use $\boldsymbol{y} = (y_1, \ldots, y_n) \in \mathcal{Y}^n$ 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 $V_0 = \{i_0\}$. Then:

1. The algorithm chooses node $q_t \in V_{t-1}$; at this time the algorithm knows that q_t has unexplored edges (i.e., edges connecting q_t to unseen nodes in $V \setminus V_{t-1}$), though the number and destination of such edges is currently unknown to the algorithm.
2. The adversary chooses a node $i_t \in V \setminus V_{t-1}$ that is 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 $\widehat{y}_t$;
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.

It is important to note that standard *nonadaptive* graph exploration strategies (combined with simple prediction rules) are suboptimal in this setting. For this purpose, 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, when $\mathcal{Y} = \{-1, +1\}$ and $\ell(\widehat{y}, y) = \mathbb{I}_{\{\widehat{y} \neq y\}}$ (zero-one loss), the graph cutsize $\Phi_G(\boldsymbol{y})$ is an obvious mistake bound achieved by such a strategy. Figure 1 shows an example where DEPTHFIRST makes $\Omega(|V|)$ 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. 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 cause $\Omega(|V|)$ mistakes.

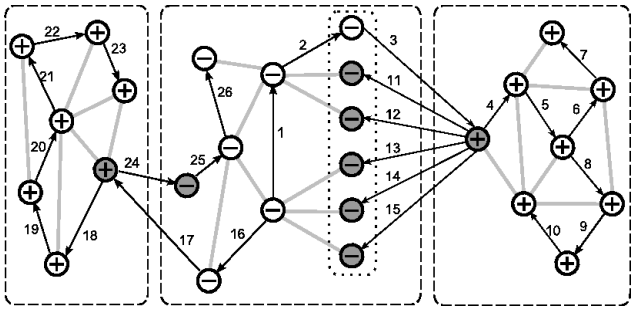


Fig. 1. A binary labeled graph with three clusters where DEPTHFIRST can make $\Omega(|V|)$ mistakes. Edges are either arrow edges or grey edges. Arrow edges indicate predictions, and numbers on such edges denote the adversarial order of presentation. For instance edge 3 (connecting a -1 node to a $+1$ node) says 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). As a matter of fact, in this example DEPTHFIRST could also predict y_t through a majority vote of the labels of previously observed nodes that are adjacent to i_t. Dark grey nodes are the mistaken nodes (for simplicity, ties are mistakes in this figure). Notice 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 $|V|$. These nodes would still be mistaken even if the majority vote were restricted to previously mistaken (and adjacent) nodes. This is because DEPTHFIRST is forced to err on the left-most node of the right-most cluster.

These algorithms fail mainly because their exploration strategy is oblivious to the sequence of revealed labels. Next, we show an *adaptive* exploration strategy that takes advantage of the revealed structure of the labeled graph in order to make a substantially smaller number of mistakes. Our algorithm CGA (Clustered Graph Algorithm) *learns* the next "good" node $q_t \in V_{t-1}$ to explore, and achieves a cumulative loss bound based on a notion of cluster/labeling regularity called *merging degree*. This notion arises naturally as a by-product of our analysis, and can be considered a natural measure of cluster similarity of independent interest.

3 Regular Partitions and the Clustered Graph Algorithm

We are interested in designing exploration/prediction strategies that work well to the extent 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, if the nodes $s_1, \ldots, s_d$ are either all positive or all negative, then $\lambda(s_1, \ldots, s_d) = 0$. 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 Sect. 5, 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, i.e., $d_t(i, j) = \min_{\pi \in P_t(i,j)} \lambda(\pi)$. A partition $\mathcal{P}$ of V in subsets (or clusters) 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 Fig. 2 for an example.

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 [4] and to the *strict separation property* of [5]. It is easy to see that according to (1) all subgraphs induced by the clusters on a regular partition are connected graphs.

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

We now introduce an algorithm, CGA, that takes advantage of regular partitions. As we show in Sect. 4, the cumulative loss of CGA can be expressed in terms of the *best* regular partition of G with respect to the unknown labeling $\boldsymbol{y} \in \mathbb{R}^n$.

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

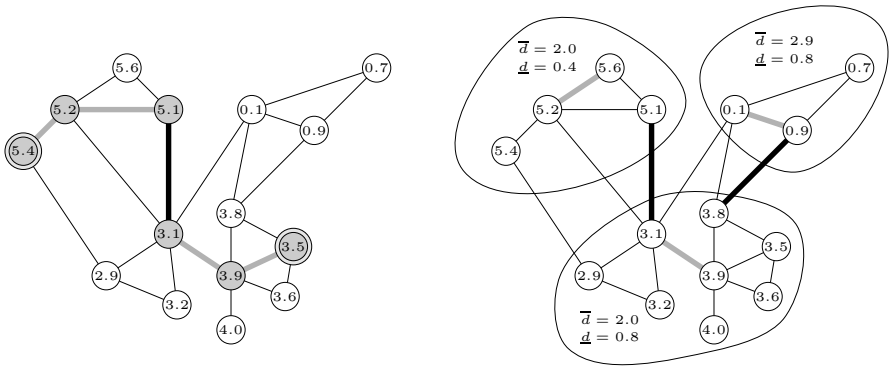
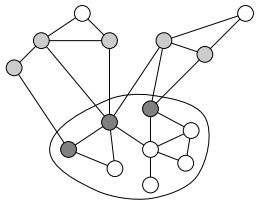


Fig. 2. Two copies of a graph with real labels y_i associated with each vertex i. On the left, a shortest path connecting the two nodes enclosed in double circles is shown. The path length is $\max_t \ell(s_{k-1}, s_k)$, where $\ell(i,j) = |y_i - y_j|$. The thick black edge is incident to the nodes achieving 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 $\overline{d}$ is determined by one of the thick black edges connecting C with 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.

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. This is based on the following notions.

Fix an arbitrary subset $C \subseteq V$. The **inner border** $\underline{\partial C}$ of C is the set of all nodes $i \in C$ that are adjacent to a node $j \notin C$ (the dark grey nodes in the picture at the side). The **outer border** $\overline{\partial C}$ of C is the set of all the nodes $j \notin C$ that are adjacent to at least one node in the inner border of C (the light grey nodes).



We are now ready to define the exploration/prediction rule of our algorithm. 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

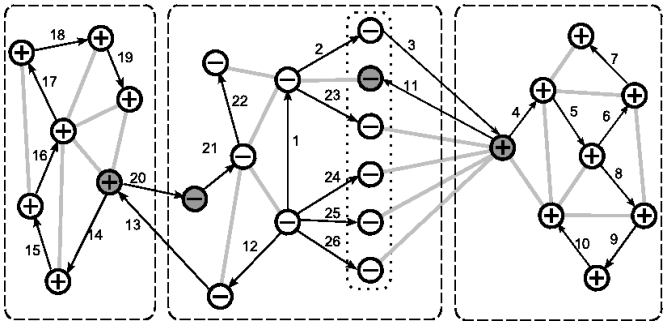


Fig. 3. The behavior of CGA displayed on the binary labeled graph of Fig. 1. The length of a path $s_1, \ldots, s_d$ is measured by $\max_k \ell(s_{k-1}, s_k)$ and the loss is the zero-one loss. The pictorial conventions are as in Fig. 1. As in that figure, the cutsize $\Phi_G(\boldsymbol{y})$ of this graph can be made as close to $|V|$ as we like, still CGA makes 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.

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 3 contains a pictorial explaination of the behavior of CGA, as compared to DEPTHFIRST on the same binary labeled graph as in Fig. 1. As we argue in the next section (Lemma 1 in Sect. 4), 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 Fig. 1 to Fig. 3. CGA is seen to make a *constant* number of binary prediction mistakes on simple graphs where DEPTHFIRST makes order of $|V|$ mistakes.

The next definition provides our main measure of graph label regularity, which we relate to CGA's predictive ability. 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 $\delta(C) = \min\{|\underline{\partial C}|, |\overline{\partial C}|\}$. 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 $\mathcal{P}$. For instance, in Fig. 3 the left-most cluster has merging degree 1, the middle one has merging degree 2, and the right-most one has merging degree 1. Hence this figure shows a case in which the mistake bound of our algorithm is tight. 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).

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 Fig. 3, $\delta(\mathcal{P})$ is often much smaller $\Phi_G(\boldsymbol{y})$ (observe that $\delta(\mathcal{P})$ is never larger than n, while $\Phi_G(\boldsymbol{y})$ can even be quadratic on dense graphs). Finally, as hinted again by Fig. 3, $\delta(\mathcal{P})$ is typically more robust to noise as compared to $\Phi_G(\boldsymbol{y})$. For instance, if we flip the label of the left-most node of the cluster on the right, the merging degree of the depicted partition gets affected only by a small amount, whereas the cutsize can decrease significantly.

4 Analysis

This section contains the analysis of CGA's predictive performance. The computational complexity analysis is contained in Sect. 5. For the sake of presentation, we single out the binary classification case 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:

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 $C \cap V_{t-1}$. 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.
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 partition of V made up of the* smallest *number of clusters, each including only nodes with the same label.*[4]

[4] Note 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. This because, in general, in each given cluster incoming and outgoing steps alternate, since the algorithm first enters and then leaves the cluster. Hence, incoming and outgoing steps must occur the same number of times, and their sum must then be at most twice the *minimum* of the size of borders (what we called merging degree of the cluster). The only exception to this alternating 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 $* \to i$) if $q_t \notin C_i$ and $i_t \in C_i$, and it is *outgoing for cluster* i (denoted by $i \to *$) 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 $M_{\to i}$ ($M_{i\to}^{\text{reg}}$) be the set of all incoming (regular outgoing) lossy steps for cluster C_i. Also, let $M_{i\to}^{\text{irr}}$ be the set of all irregular outgoing lossy steps for C_i.

For each $i \in \mathcal{P}$, define an injective mapping $\mu_i : M_{i\to}^{\text{reg}} \to M_{\to i}$ as follows (see Fig. 4 for reference): Each lossy step t in $M_{i\to}^{\text{reg}}$ 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 $|M_{i\to}^{\text{reg}}| \leq |M_{\to i}|$. 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 $|M_{i\to}^{\text{reg}}|$ can be refined as $|M_{i\to}^{\text{reg}}| \leq |M_{\to i}| - \mathbb{I}_i$. 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 $\sum_{i\in\mathcal{P}} \mathbb{I}_i \geq \sum_{i\in\mathcal{P}} |M_{i\to}^{\text{irr}}|$. 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. Hence

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

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 $|M_{\to i}| \leq \min\{|\underline{\partial C_i}|, |\overline{\partial C_i}|\} = \delta(C_i)$ for each $i \in \mathcal{P}$. Together with (3) this would complete the proof (see again Fig. 4 for a pictorial explanation).

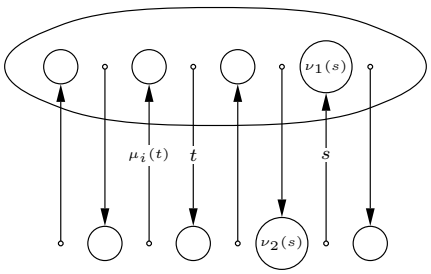


Fig. 4. Sequence (starting from the left) of incoming and regular outgoing lossy steps involving a given cluster C_i. We only show the border nodes contributing to lossy steps. We map injectively each regular outgoing lossy step t to the previous (incoming) lossy step $\mu_i(t)$. We also map injectively 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.

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 $y_{i_{t'}}$. □

As we already noted, the edges (q_t, i_t) produced during the online functioning of the algorithm form a spanning tree T for G. Therefore CGA's number of mistakes m is always equal to $\Phi_T(\boldsymbol{y})$. This shows that an obvious lower bound on m is the total number of clusters $|\mathcal{P}|$, i.e., the cost of the minimum spanning tree for G. In fact, it is not difficult to prove that an adaptive adversary can always force any algorithm working within our learning protocol to make $\Omega(|\mathcal{P}|)$ mistakes. This simple observation can be strengthened so as to match the upper bound in Theorem 1.

Theorem 2. *For all undirected and connected graphs G with n nodes and degree bounded by a constant, for all $K < n$, and for any (randomized) exploration/prediction strategy, there exists a labeling $\boldsymbol{y}$ of G's vertices such that the strategy makes at least $K/2$ mistakes (in expectation) with respect to the algorithm's internal randomization, while $\delta(\mathcal{P}) = \mathcal{O}(K)$.*

The above lower bound, whose proof is omitted due to space limitations, can actually be shown to hold even in cases when G does not have bounded degree nodes, like cliques or general trees.

We now turn to 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 max 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[5] the total loss incurred in all steps t where q_t and i_t both belong to C.

In the above definition, $\ell(C)$ is a measure of connectivity of C to 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 (expressed by $\varepsilon(C)$), against total inter-cluster loss (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*

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

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

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

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

[5] CGA's cumulative loss is $\sum_{t=1}^{|V|} \ell(q_t, i_t)$, where 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 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 $E(\mathcal{F}) \subseteq E(T_C)$.

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 (as in Theorem 1). In most cases, the nontrivial 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, the contribution to the total loss of $\ell(q_t, i_t)$ for $q_t, i_t \in C$ 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 $1, \dots, |\mathcal{P}|$ the clusters in it. We abuse the notation and use $\mathcal{P}$ also to denote the set of cluster indices. 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 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 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

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

where the second inequality follows from $\sum_{i\in\mathcal{P}} \mathbb{I}_i \geq \sum_{i\in\mathcal{P}} |M^{\text{irr}}_{i\rightarrow}|$ (as for the 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 $\delta(C_i)$. □

5 Computational Complexity

In this section we briefly 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. If we are 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 to be able to compute in *constant* time the length of the shortest path $i, \ldots, j, j'$. 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 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, 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[6] $\mathcal{O}\big(|V||E| + |V|^2 \log |V|\big)$. On the other hand, the space complexity is always linear in the size of G.

We now sketch the binary case. 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 specially designed 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. We call *sub-cluster* each maximal set of nodes that are part of the same uniformly labeled and connected subgraph of G_t. The main data structures we use (further details are omitted due to space limitations) for organizing the nodes observed so far by the algorithm combine the following:

– A self-balancing binary search tree T containing the labeled nodes in V_t. We will refer to nodes in V_t and to nodes in T interchangeably.

[6] 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.

– Given a sub-cluster C, all nodes in $C \cap \underline{\partial V}_t$ are connected via a special list called *border sub-cluster list*. The remaining nodes in C are connected through a list called *internal sub-cluster list*.
– All nodes in each sub-cluster $C \subseteq V_t$ are linked to a special time-varying set called *sub-cluster record*. This record enables access to the first and last element of both the border and the internal sub-cluster list of C. The sub-cluster record also contains the size of C.

The above data structures are intended to support the following main 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 also receives the list $N(i_t)$ of all nodes in V_{t-1} adjacent to i_t; (2) merging of subclusters required after the disclosure of y_t; (3) update of border and internal sub-cluster lists (since some nodes in $\underline{\partial V}_{t-1}$ are not in $\underline{\partial V}_t$); (4) choice of q_{t+1}.

The merging operation can be implemented as union-by-rank in standard union-find data structures (e.g., [7, Ch. 22]). The overall running time for $|V|$ nodes is smaller than $\mathcal{O}\big(|V| \log |V|\big)$. In fact, the dominating cost in the time complexity is the 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, the height of tree T being at most logarithmic in V. Hence 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.

As for space complexity, it is important to stress that on every step t the algorithm first stores and then "throws way" 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.

6 Conclusions and Ongoing Research

We have presented a first step towards the study of problems related to learning (labeled) graph exploration strategies. This is a significant departure from more standard approaches assuming prior knowledge of the underlying graph structure (e.g., [2, 3, 6, 9, 10, 11, 12, 13, 14, 17] and references therein).

We are currently investigating to what extent our approach can be extended to weighted graphs. In order to exploit the benefits of edge weights, our protocol in Sect. 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 can take advantage of the additional weight information. This calls for an analysis being able to capture the interaction between node labels and edge weights.

Acknowledgments. We would like to thank the ALT 2009 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 and M. Henzinger. Exploring unknown environments. *SIAM Journal on Computing*, 29(4):1164–1188, 2000.

[2] A. Blum and S. Chawla. Learning from labeled and unlabeled data using graph mincuts. In *Proc. 18th ICML*. Morgan Kaufmann, 2001.

[3] A. Blum, J. Lafferty, M. Rwebangira, and R. Reddy. Semi-supervised learning using randomized mincuts. In *Proc. 21st ICML*. ACM Press, 2004.

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

[5] N. Balcan, A. Blum, and S. Vempala. A discriminative framework for clustering via similarity functions. In *Proc. 40th STOC*. ACM Press, 2008.

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

[7] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. MIT Press, 1990.

[8] X. Deng and C.H. Papadimitriou. Exploring an unknown graph. In *Proc. 31st FOCS*, pages 355–361. IEEE Press, 1990.

[9] S. Hanneke. An analysis of graph cut size for transductive learning. In *Proc. 23rd ICML*, pages 393–399. ACM Press, 2006.

[10] M. Hebster and M. Pontil. Prediction on a graph with the Perceptron. In *NIPS 19*, pages 577–584. MIT Press, 2007.

[11] M. Herbster. Exploiting cluster-structure to predict the labeling of a graph. In *Proc. 19th ALT*. Springer, 2008.

[12] M. Herbster, G. Lever, and M. Pontil. Online prediction on large diameter graphs. In *NIPS 22*. MIT Press, 2009.

[13] M. Herbster, M. Pontil, and S. Rojas-Galeano. Fast prediction on a tree. In *NIPS 22*. MIT Press, 2009.

[14] M. Herbster and G. Lever. Predicting the labelling of a graph via minimum p-seminorm interpolation. In *Proc. 22nd COLT*. Omnipress, 2009.

[15] T. Joachims. Transductive Learning via Spectral Graph Partitioning In *Proc. 20th ICML*, pages 305–132. AAAI Press, 2003.

[16] I. Kondor and J. Lafferty. Diffusion kernels on graphs and other discrete input spaces. In *Proc. 19th ICML*, pages 315–322. Morgan Kaufmann, 2002.

[17] J. Pelckmans, J. Shawe-Taylor, J. Suykens, and B. De Moor. Margin based transductive graph cuts using linear programming. In *Proc. 11th AISTAT*, pages 360–367. JMLR Proceedings Series, 2007.

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

[19] A. Smola and I. Kondor. Kernels and regularization on graphs. In *Proc. 16th COLT*, pages 144–158. Springer, 2003.

[20] W.S. Yang and J.B. Dia. Discovering cohesive subgroups from social networks for targeted advertising. In *Expert Systems with Applications*, 34:2029–2038. Elsevier, 2008.