Fast and Optimal Algorithms for Weighted Graph Prediction

Nicolò Cesa-Bianchi
Università di Milano, Italy
cesa-bianchi@dsi.unimi.it

Claudio Gentile
Università dell'Insubria, Italy
claudio.gentile@uninsubria.it

Fabio Vitale
Università di Milano, Italy
fabio.vitale@unimi.it

Giovanni Zappella
Università di Milano, Italy
giovanni.zappella@studenti.unimi.it

Abstract

We show that the mistake bound for predicting the nodes of an arbitrary weighted graph is characterized (up to logarithmic factors) by the weighted cutsize of a random spanning tree of the graph. The cutsize is induced by the unknown adversarial labeling of the graph nodes. In deriving our characterization, we obtain a simple randomized algorithm achieving the optimal mistake bound on any graph. Our algorithm draws a random spanning tree of the original graph and then predicts the nodes of this tree in constant amortized time and linear space. Preliminary experiments on real-world datasets show that our method outperforms both global (Perceptron) and local (majority voting) methods.

1 Introduction

A widespread approach to the solution of classification problems is representing the data through a weighted graph in which edge weights quantify the similarity between data points. This technique for coding input data has been applied to several domains, including Web spam detection [12], classification of genomic data [17], recognition of faces [5], and text categorization [8]. In most applications, edge weights are computed through a complex data-modelling process and convey crucially important information for classifying nodes.

This paper focuses on the online version of the graph classification problem: the entire graph is known in advance and, at each step, the algorithm is required to predict the label of a new arbitrarily chosen node. In the special case of unweighted graphs (where all edges have unit weight) a key parameter for controlling the number of prediction mistakes is the size of the cut induced by the unknown adversarial labeling of the graph. Although in the unweighted case previous studies use the cutsize to prove several interesting upper bounds [11, 10, 12], no general lower bounds on the number of prediction mistakes are known, leaving fully open the question of characterizing the complexity of learning a labeled graph. In a recent paper [4] the expected number of mistakes is bounded by the cutsize of a random spanning tree of the graph, a quantity stricly smaller than the cutsize of the whole graph. In this paper we show that this quantity captures the hardness of the graph learning problem, even in the general weighted case (where the expectation suitably depends on the edge weights). Given any weighted graph, we prove that any prediction algorithm must err on a number of nodes which is at least as big as the weighted cutsize of the graph's random spanning tree. Moreover, if the ratio of the largest to the smallest weight is polynomial in the number of nodes, we exhibit a simple algorithm achieving (to within logarithmic factors) the optimal mistake bound.

Following [4], our algorithm first extracts a random spanning tree of the original graph, and then predicts all nodes of this tree using a generalized variant of the method proposed in [11]. Our tree prediction procedure is extremely efficient: it only requires *constant* amortized time per prediction and space *linear in the number of nodes*. Note that computational efficiency is a central issue in practical applications where the involved datasets can be very large. Indeed, learning algorithms whose time complexity scales, say, more than quadratically with the number of data points should be considered impractical.

A significant contribution of this work is the experimental evaluation of our method, as compared to methods recently proposed in the literature on graph prediction. In particular, we compare our algorithm to the Perceptron algorithm with Laplacian kernel [10, 12], and to simple majority vote predictors. The experiments have been carried out on two medium-size biological datasets from [6]. The two tree-based algorithms (ours and the Perceptron) have been tested using spanning trees generated in various ways. Though preliminary in nature, our experimental comparison shows that, in terms of the online mistake count, our algorithm *always* outperforms the tested competitors while using the least amount of time and memory resources.

2 Preliminaries and basic notation

Let $G = (V, E, W)$ be an undirected, connected, and weighted graph with n nodes and positive edge weights $w_{i,j} > 0$ for $(i, j) \in E$. A labeling of G is any assignment $\boldsymbol{y} = (y_1, \dots, y_n) \in \{-1, +1\}^n$ of binary labels to its nodes. We use $(G, \boldsymbol{y})$ to denote the resulting labeled weighted graph. The online learning protocol for predicting $(G, \boldsymbol{y})$ is defined as follows. The learner is given G while $\boldsymbol{y}$ is kept hidden. The nodes of G are presented to the learner one by one, according to an unknown and arbitrary permutation $i_1, \dots, i_n$ of V. At each time step $t = 1, \dots, n$ node i_t is presented and the learner must predict its label y_{i_t}. Then y_{i_t} is revealed and the learner knows whether a mistake occurred. The learner's goal is to minimize the total number of prediction mistakes.

It is reasonable to expect that prediction performance should degrade with the increase of "randomness" in the labeling. For this reason, our analysis of graph prediction algorithms bounds from above the number of prediction mistakes in terms of appropriate notions of graph label *regularity*. A standard notion of label regularity is the cutsize of a labeled graph, defined as follows. A ϕ-edge of a labeled graph $(G, \boldsymbol{y})$ is any edge (i, j) such that $y_i \neq y_j$. Similarly, an edge (i, j) is ϕ-free if $y_i = y_j$. Let $E^\phi \subseteq E$ be the set of ϕ-edges in $(G, \boldsymbol{y})$. The *cutsize* $\Phi_G(\boldsymbol{y})$ of $(G, \boldsymbol{y})$ is the number of ϕ-edges in $\Phi_G(\boldsymbol{y})$, i.e., $\Phi_G(\boldsymbol{y}) = \left|E^\phi\right|$ (independent of the edge weights). The *weighted* cutsize $\Phi^W_G(\boldsymbol{y})$ of $(G, \boldsymbol{y})$ is $\Phi^W_G(\boldsymbol{y}) = \sum_{(i,j)\in E^\phi} w_{i,j}$.

Fix $(G, \boldsymbol{y})$. Let $r^W_{i,j}$ be the effective resistance (see, e.g., [15]) between nodes i and j of G. For $(i, j) \in E$, let also $p_{i,j} = w_{i,j} r^W_{i,j} = w_{i,j}/\bigl(w_{i,j} + 1/\widetilde{r}^W_{i,j}\bigr)$ be the probability that (i, j) belongs to a random spanning tree T [15]. Here $\widetilde{r}^W_{i,j}$ denotes the effective resistance between i and j when edge (i, j) is eliminated —if (i, j) is a bridge, whose elimination disconnects G, we set $1/\widetilde{r}^W_{i,j} = 0$. Then we have

$$\mathbb{E}\,\Phi_T(\boldsymbol{y}) = \sum_{(i,j)\in E^\phi} p_{i,j} = \sum_{(i,j)\in E^\phi} \frac{w_{i,j}}{w_{i,j} + 1/\widetilde{r}^W_{i,j}} \, . \tag{1}$$

Since $\sum_{(i,j)\in E} p_{i,j}$ is equal to $n - 1$, irrespective of the edge weighting, the ratio $\frac{1}{n-1}\mathbb{E}\,\Phi_T(\boldsymbol{y}) \in [0, 1]$ provides an *edge density-independent* measure of the cutsize in G. and allows one even to compare labelings on different graphs. It is also important to note that $\mathbb{E}\,\Phi_T(\boldsymbol{y})$ can be much smaller than $\Phi^W_G(\boldsymbol{y})$ when there are strongly connected regions in G contributing prominently to the weighted cutsize. To see this, consider the following scenario: If $(i, j) \in E^\phi$ and $w_{i,j}$ is large, then (i, j) gives a big contribution to $\Phi^W_G(\boldsymbol{y})$. However, this might not happen in $\mathbb{E}\,\Phi_T(\boldsymbol{y})$. In fact, if i and j are strongly connected (i.e., if there are many disjoint paths connecting them), then $\widetilde{r}^W_{i,j}$ is very small, thus the terms $w_{i,j}/(w_{i,j} + 1/\widetilde{r}^W_{i,j})$ in (1) are small too. Therefore, the effect of the large weight $w_{i,j}$ may often be compensated by the small probability of including (i, j) in the random spanning tree.

3 A lower bound for any weighted graph

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

Theorem 1 *Let $G = (V, E, W)$ be a weighted undirected graph with n nodes and weights $w_{i,j} > 0$ for $(i, j) \in E$. Then for all $K \le n$ there exists a randomized labeling $\boldsymbol{y}$ of G such that for all (deterministic or randomized) algorithms A, the expected number of prediction mistakes made by A is at least $K/2$, while $\mathbb{E}\,\Phi_T(\boldsymbol{y}) < K$.*

Proof. The adversary uses the weighting P induced by W and defined by $p_{i,j} = w_{i,j} r^W_{i,j}$. Note that $p_{i,j}$ is the probability that edge (i,j) belongs to a random spanning tree T of G. Hence $\sum_{(i,j)\in E} p_{i,j} = n-1$ and $\Phi^P_G(\boldsymbol{y}) = \mathbb{E}\,\Phi_T(\boldsymbol{y})$ for any given labeling $\boldsymbol{y}$ of G. Let $P_i = \sum_j p_{i,j}$ be the sum over the induced weights of all edges incident to node i. We call P_i the *weight* of node i. Let $S \subseteq V$ be the set of K nodes i in G having the smallest weight P_i. The adversary assigns a random label to each node $i \in S$. This guarantees that, no matter what, the algorithm A will make on average $K/2$ mistakes on the nodes in S. The labels of the remaining nodes in $V \setminus S$ are set either all $+1$ or all -1, depending on which one of the two choices yields the smaller $\Phi^P_G(\boldsymbol{y})$. We now show that the weighted cutsize $\Phi^W_P(\boldsymbol{y})$ of this labeling $\boldsymbol{y}$ is less than K, *independent of* the labels of the nodes in S. Since the nodes in $V \setminus S$ have all the same label, the ϕ-edges induced by this labeling can only connect either two nodes in S or one node in S and one node in $V \setminus S$. Hence $\Phi^W_P(\boldsymbol{y}) = \Phi^{P,int}_G(\boldsymbol{y}) + \Phi^{P,ext}_G(\boldsymbol{y})$, where $\Phi^{P,int}_G(\boldsymbol{y})$ is the cutsize contribution within S, and $\Phi^{P,ext}_G(\boldsymbol{y})$ is the one from edges between S and $V \setminus S$. Let $P^{int}_S = \sum_{(i,j)\in E\,:\,i,j\in S} p_{i,j}$ and $P^{ext}_S = \sum_{(i,j)\in E\,:\,i\in S, j\in V\setminus S} p_{i,j}$. From the very definition of P^{int}_S and $\Phi^{P,int}_G(\boldsymbol{y})$ we have $\Phi^{P,int}_G(\boldsymbol{y}) \le P^{int}_S$. Moreover, from the way the labels of nodes in $V \setminus S$ are selected, it follows that $\Phi^{P,ext}_G(\boldsymbol{y}) \le P^{ext}_S/2$. Finally, $\sum_{i\in S} P_i = 2P^{int}_S + P^{ext}_S$ holds, since each edge connecting nodes in S is counted twice in the sum $\sum_{i\in S} P_i$. Putting everything together we obtain

$$2P^{int}_S + P^{ext}_S = \sum_{i\in S} P_i \le \frac{K}{n}\sum_{i\in V} P_i = \frac{2K}{n}\sum_{(i,j)\in E} p_{i,j} = \frac{2K(n-1)}{n}$$

the inequality following from the definition of S. Hence

$$\mathbb{E}\,\Phi_T(\boldsymbol{y}) = \Phi^P_G(\boldsymbol{y}) = \Phi^{P,int}_G(\boldsymbol{y}) + \Phi^{P,ext}_G(\boldsymbol{y}) \le P^{int}_S + \frac{P^{ext}_S}{2} \le \frac{K(n-1)}{n} < K\,.$$

$\square$

4 The Weighted Tree Algorithm for weighted trees

In this section, we describe the Weighted Tree Algorithm (WTA) for predicting the labels of a weighted tree. In Section 6 we show how to apply WTA to solve the more general weighted graph prediction problem. WTA first turns the tree into a line graph (i.e., a list), then runs a fast nearest neighbor method to predict the labels of each node in the line. Though this technique is similar to that one used in [11], the fact that the tree is weighted makes the analysis significantly more difficult.

Given a labeled weighted tree $(T, \boldsymbol{y})$, the algorithm initially creates a weighted line graph L' containing some duplicates of the nodes in T. Then, each duplicate node (together with its incident edges) is replaced by a single edge with a suitably chosen weight. This results in the final weighted line graph L which is then used for prediction. In order to create L from T, WTA performs the following *tree linearization* steps:

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

The analysis of Section 5 shows that this choice of $w_{j',j''}$ guarantees that the weighted cutsize of L is smaller than twice the weighted cutsize of T. Once L is created from T, the algorithm predicts the label of each node i_t using a nearest-neighbor rule operating on L with a *resistance distance* metric.

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

That is, the prediction on i_t is the label of i_{s^*}, being $s^* = \mathrm{argmin}_{s<t} d(i_s, i_t)$ the previously revealed node closest to i_t, and $d(i,j) = \sum_{s=1}^{k} 1/w_{v_s,v_{s+1}}$ is the sum of the resistors (i.e., reciprocals of edge weights) along the (unique) path $i = v_1 \to v_2 \to \cdots \to v_{k+1} = j$ connecting node i to node j.

5 Analysis

In this section we derive an upper bound on the number of mistakes made by WTA on any weighted tree $T = (V, E, W)$ in terms of the number of ϕ-edges, the weighted cutsize, and the sum of resistors of ϕ-free edges, $R_T^W = \sum_{(i,j)\in E\setminus E^\phi} 1/w_{i,j}$. The following lemma establishes some simple but important relationships between the tree T and its linearized version L. Theorem 3 below exploits this lemma to bound the total number of mistakes on any tree.

From the construction in Section 4 we see that when we transform L' into L the pairs of edges (j', j) and (j, j'') of L' which are incident to a repeated node j get replaced in L (together with j) by a single edge (j', j'') —step 3 in Section 4. We call these edges *spurious* edges. Assume that (j', j'') is spurious in L. When $y_{j'} \neq y_{j''}$ we have created a spurious ϕ-edge by eliminating a ϕ-edge and a ϕ-free edge from L'. When $y_{j'} = y_{j''} \neq y_j$, we have created a spurious ϕ-free edge by eliminating two ϕ-edges from L'. Let R_0^W be the sum of resistors of all spurious ϕ-free edges created during the elimination of pairs of ϕ-edges in L'.

Lemma 2 *Let $(T, \boldsymbol{y})$ be a labeled tree, $(L, \boldsymbol{y})$ be a linearized version of it, and L' be the line graph with duplicates (as described in Section 4). Then the following holds: $R_L^W \le R_{L'}^W + R_0^W \le 2R_T^W + R_0^W$, $\Phi_L^W(\boldsymbol{y}) \le \Phi_{L'}^W(\boldsymbol{y}) \le 2\Phi_T^W(\boldsymbol{y})$, and $\Phi_L(\boldsymbol{y}) \le \Phi_{L'}(\boldsymbol{y}) \le 2\Phi_T(\boldsymbol{y})$.*

Proof. Note that each edge of T occurs in L' at least once and at most twice. This proves $\Phi_{L'}^W(\boldsymbol{y}) \le 2\Phi_T^W(\boldsymbol{y})$ and $\Phi_{L'}(\boldsymbol{y}) \le 2\Phi_T(\boldsymbol{y})$. Note further that L contains some non-spurious edges from L' plus a number of spurious edges. Each spurious ϕ-free edge (j', j'') can be created (by eliminating a node j) when either (i) $y_{j'} = y_{j''} = y_j$, which implies that $w_{j',j''}$ corresponds to the weight of a ϕ-free edge eliminated in L' together with node j, and thus $w_{j',j''}$ is not included in R_0^W; or (ii) $y_{j'} = y_{j''} \neq y_j$, which implies that $w_{j',j''}$ is included in R_0^W. This proves the first inequality. To prove the remaining inequalities, first note that a spurious edge (j', j'') cannot be a ϕ-edge in L unless either (j, j') or (j, j'') is a ϕ-edge in L'. Moreover, if (j', j'') is a ϕ-edge in L, then its weight is not larger than the weight of the associated ϕ-edge in L' —Step 3 in Section 4. □

Theorem 3 *If* WTA *is run on a weighted and labeled tree $(T, \boldsymbol{y})$, then the total number m_T of mistakes satisfies*

$$m_T = \mathcal{O}\left(\Phi_T(\boldsymbol{y})\left(1 + \log\left(1 + \frac{R_T^W \Phi_T^W(\boldsymbol{y})}{\Phi_T(\boldsymbol{y})}\right)\right)\right) .$$

The mistake bound in Theorem 3 shows, in the logarithmic factors, that the algorithm takes advantage of labelings such that the weights of ϕ-edges are small (thus making $\Phi_T^W(\boldsymbol{y})$ small) and the weights of ϕ-free edges are high (thus making R_T^W small). This somehow matches the intuition behind WTA's nearest-neighbor rule according to which nodes that are close to each other are expected to have the same label. In particular, observe that the way the above quantities are combined makes the bound independent of rescaling of the edge weights. Again, this has to be expected, since WTA's prediction is scale insensitive. On the other hand, it may appear less natural that the mistake bound also depends linearly on the cutsize $\Phi_T(\boldsymbol{y})$, *independent of the edge weights*. As a matter of fact, this linear dependence on the unweighted cutsize cannot be eliminated (this is a consequence of Theorem 1 in Section 3).

The following lemma (proof omitted due to space limitations) proves a mistake bound for any weighted line graph. It also shows that, for any $K \ge 0$, one can drop from the bound the contribution of any set of K resistors in R_T^L at the cost of adding K extra mistakes.

Lemma 4 *If* WTA *is run on a weighted line graph $(L, \boldsymbol{y})$, then the total number m_L of mistakes satisfies*

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

where $\widetilde{R}_L^W$ is the sum of of the resistors of any set formed by all but K ϕ-free edges of L.

Proof of Theorem 3 [sketch]. Recall that R_0^W is the sum of resistors on all spurious ϕ-free edges obtained by eliminating pairs of ϕ-edges in L'. Hence, we can injectively associate with each such edge two distinct ϕ-edges in L', and therefore the total number of spurious edges giving contribution to R_0^W is bounded by $\Phi_{L'}(\boldsymbol{y})/2$, which in turn can be bounded by $\Phi_T(\boldsymbol{y})$ via Lemma 2. Applying Lemma 4 (setting $\widetilde{R}_L^W$ to $R_L^W - R_0^W$) along with Lemma 2 concludes the proof. $\square$

6 The Weighted Tree Algorithm on weighted graphs

In order to solve the more general problem of predicting the labels of a weighted graph G, one can first generate a spanning tree T of G and then run WTA directly on T. In this case it is possible to rephrase Theorem 3 in terms of properties of G. Note that for each spanning tree T of G, $\Phi_T^W(\boldsymbol{y}) \le \Phi_G^W(\boldsymbol{y})$ and $\Phi_T(\boldsymbol{y}) \le \Phi_G(\boldsymbol{y})$. Specific choices of the spanning tree T control in different ways the quantities in the mistake bound of Theorem 3. For example, a minimum spanning tree tends to reduce the value of R_T^W, betting on the fact that ϕ-edges are light. Adapting the proof of Theorem 3, we can prove the following result.

Theorem 5 *If* WTA *is run on a random spanning tree T of a labeled weighted graph $(G, \boldsymbol{y})$ with n nodes, then the total number m_G of mistakes statisfies*

$$\mathbb{E}\, m_G = \mathcal{O}\left(\mathbb{E}\big[\Phi_T(\boldsymbol{y})\big]\left(1 + \log\left(1 + n\frac{w_{\max}^{\phi}}{w_{\min}^{\neg\phi}}\right)\right)\right)$$

where $w_{\min}^{\neg\phi} = \min_{(i,j)\in E\setminus E^{\phi}} w_{i,j}$ and $w_{\max}^{\phi} = \max_{(i,j)\in E^{\phi}} w_{i,j}$. In particular, if the ratio $\max_{(i,j),(k,\ell)\in E} w_{i,j}\big/w_{k,\ell}$ is bounded by a polynomial in n, then $\mathbb{E}\, m_G = \mathcal{O}\big(\mathbb{E}[\Phi_T(\boldsymbol{y})]\log n\big)$.

Note that having K ϕ-free edges with exponentially small (in n) weights does not necessarily lead to a vacuous bound in Theorem 5 when K is small enough. Indeed, one can use Lemma 4 also to replace the factor $w_{\min}^{\neg\phi}$ by the $(K+1)$-th smallest ϕ-free weight at the cost of adding just K more mistakes. On the other hand, if G has exponentially large ϕ-edge weights, then the bound can indeed become vacuous. This is not surprising, though, since the algorithm's inductive bias is to bet on graphs having small weighted cutsize.

7 Computational complexity

A direct implementation of WTA operating on a tree T with n nodes would require running time $\mathcal{O}(n \log n)$ over the n prediction trials, and linear memory space. We now sketch how to implement WTA in $\mathcal{O}(n)$ time, i.e., in *constant* amortized time per trial.

Once the given tree T is linearized into an n-node line L, we initially traverse L from left to right. Call j_0 the left-most terminal node of L. During this traversal, the resistance distance $d(j_0, i)$ is incrementally computed for each node i in L. This makes it possible to calculate in constant time $d(i,j)$ for any pair of nodes, since $d(i,j) = |d(j_0,i) - d(j_0,j)|$ $\forall i,j \in L$. On top of line L a complete binary tree T' is constructed having $2^{\lceil \log_2 n \rceil}$ leaves.[2] The k-th leftmost leaf (in the usual tree representation) of T' is the k-th node in L (numbering the nodes of L from left to right). The algorithm maintains this data-structure in such a way that at time t: (i) the subsequence of leaves whose labels are revealed at time t are connected through a (bidirectional) list B, and (ii) all the ancestors in T' of the leaves of B are marked. See Figure 1 for an example.

When WTA is required to predict the label y_{i_t}, the algorithm looks for the two closest leaves i' and i'' oppositely located in L with respect to i_t. The above data-structure supports this operation as follows. WTA starts from i_t and goes upwards in T' until the first marked ancestor $\mathrm{anc}(i_t)$ of i_t is reached. During this upward traversal, the algorithm marks each internal node of T' on the path connecting i_t to $\mathrm{anc}(i_t)$. Then, WTA starts from $\mathrm{anc}(i_t)$ and goes downwards in order to find the leaf $i' \in B$ closest to i_t. Notice how the algorithm uses node marks for finding its way down: For instance, in Figure 1 the algorithm goes left since $\mathrm{anc}(i_t)$ was reached from below through the right child node, and then keeps right all the way down to i'. Node i'' (if present) is then identified via the links in B. The two distances $d(i_t, i')$ and $d(i_t, i'')$ are compared, and the closest node to i_t within B is then determined. Finally, WTA updates the links of B by inserting i_t between i' and i''.

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

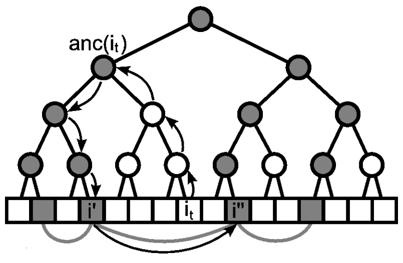

Figure 1: Constant amortized-time implementation of WTA. The line L is made up of $n = 16$ nodes (the adjacent boxes at the bottom). Shaded boxes are the revealed nodes, connected through a dark grey doubly-linked list B. The depicted tree T' has both unmarked (white) and marked (shaded) nodes. The arrows indicate the traversal operations performed by WTA when predicting the label of node i_t: The upwards traversal stops as soon as a marked ancestor anc(i_t) is found, and then a downward traversal begins. Notice that WTA first descends to the left, and then keeps going right all the way down. Once i' is determined, a single step within B suffices to determine i''.

In order to quantify the amortized time per trial, the key observation is that each internal node k of T' gets visited only twice during *upward* traversals over the n trials: The first visit takes place when k gets marked for the first time, the second visit of k occurs when a subsequent upwards visit also marks the other (unmarked) child of k. Once both of k's children are marked, we are guaranteed that no further upwards visits to k will be performed. Since the preprocessing operations take $\mathcal{O}(n)$, the above shows that the total running time over the n trials is linear in n, as anticipated.[3]

8 Preliminary experiments

We now present the results of a preliminary experimental comparison on two real-world weighted graph datasets. Our goal is to compare the prediction accuracy of WTA to the one achieved by fast algorithms for weighted graphs (and for which accuracy performance guarantees are available in the literature). We compare our algorithm to the following two online prediction methods, intended as representatives of two different ways of facing the graph prediction problem, a global approach and a local approach.

The **Perceptron algorithm with graph Laplacian kernel** [10] (abbreviated as GPA, Graph Perceptron Algorithm). This algorithm predicts the nodes of a weighted graph $G = (V, E)$ after mapping V via the linear kernel based on $L_G^+ + \mathbf{1}\,\mathbf{1}^\top$, where L_G is the laplacian matrix of G. As recently shown in [12], computing the pseudoinverse L_G^+ when G is a tree takes quadratic time in the number of nodes n. This can be exploited by generating a spanning tree T of G, and then invoking GPA on T. Both time and space are quadratic in n (rather than linear, as for WTA). The mistake bound on T has the form $m_T \le \Phi_T^W(\boldsymbol{y}) D_T^W$, where D_T^W is the spanning tree diameter. GPA is a global approach in the sense that the graph topology affects, via the inverse Laplacian, the prediction on all nodes.

The **Online Majority Vote algorithm** (abbreviated as OMV). As the common underlying assumption to graph prediction algorithms is that nearby nodes are labeled similarly, a very intuitive and fast algorithm for predicting the label of a node i_t is via a weighted majority vote on all labels of the adjacent nodes seen so far, i.e., SGN$(\sum_{j<t\,:\,(i_t,i_j)\in E} y_{i_j} w_{i_t,i_j})$. The total time required, as well as the memory space, is $\Theta(|E|)$, since we need to read (at least once) the weights of all edges. OMV-like algorithms are local approaches, in the sense that prediction at one node is affected only by adjacent nodes.

It is fair to stress that many other algorithms have been proposed which are able to deal with weighted graphs, including the label-consistent mincut approach of [3] and the energy minimization methods in [2, 20]. We do not carry out a direct experimental comparison to them either because (seemingly) they do not have good scaling properties or because they do not have online prediction performance guarantees. We combine WTA and[4] GPA with spanning trees generated in different ways.

Random Spanning Tree (abbreviated as RST). Each spanning tree is taken with probability proportional to the product of its edge weights (e.g., [15, Ch. 4]).

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

[4]Note that OMV-like algorithms do not operate on spanning trees.

Depth first spanning tree (DFST). The spanning tree is created with a randomized depth-first visit in the following way: A root is randomly selected; then each newly visited node is chosen with probability proportional to the weights of the edges connecting the current vertex with the adjacent nodes that have not been visited yet. This spanning tree generation is intended to approximate the standard RST generation which in practice might be more time-consuming.

Minimum Spanning Tree (MST), i.e., the spanning tree minimizing the sum of the resistors of all edges. MST is the tree whose Laplacian best approximates the Laplacian of G according to the trace norm criterion (see, e.g., [12]).

Shortest Path Spanning Tree (SPST). In [12], the shortest path tree is used for its small diameter, which is always at most twice the diameter of G. A short diameter tree allows for a better control over the (theoretical) performance of GPA. By varying the root node, we generated n shortest path spanning tree, and then took the one having minimal diameter among them.

We ran our experiments on two medium size biological datasets: (1) Krogan et al.'s dataset [14, 6] (abbreviated as **Krogan**); (2) A second dataset (abbreviated as **Comb**) resulting from a combination [6] of three datasets from [7, 13, 18].

Both Krogan and Comb represent high-throughput protein-protein interaction networks of budding yeast taken from [6]. In particular, Krogan is a weighted graph based on a large high-throughput and reliable dataset reported in [14]; Comb is the combination of three high-throughput yeast interaction sets from [7, 18, 13]. We only consider the biggest connected components of both datasets, obtaining 2,169 nodes and 6,102 edges for Krogan, and 2,871 nodes and 6,407 edges for Comb. In these graphs, each node belongs to one or more classes, each class representing a protein function. We selected the set of functional labels at depth one in the *FunCat* classification scheme of the MIPS database [16], resulting in 17 classes per dataset. We finally binarized the problems via a standard one-vs-rest scheme, obtaining $17 + 17 = 34$ binary classification problems.

The experimental setup on the 34 binary classification problems is the following: (i) We first generated 50 random permutations of the node indices for each dataset; (ii) we computed MST and SPST for each graph and made (for both WTA and GPA) one run per permutation on each binary problem, averaging results over permutations; (iii) we generated 50 RST and 50 DFST for each graph, and operated as in (ii) with a further averaging over the randomness in the tree generation; (iv) we ran 50 experiments (one per permutation) for each binary problem with OMV, again averaging over permutations.

In order to analyze the labeling properties of each binary problem, we calculated the percentage of ϕ-edges in the graphs, as well as in each type of spanning tree used in our experiments. These statistics are reported in the table below. Figures for RST and DFST are averaged over random generation of spanning trees. In particular, those concerning RST estimate the edge density-independent measure $\mathbb{E}\,\Phi_T(\boldsymbol{y})/(n-1)$ mentioned in Section 2.

	Original	RST	DFST	SPST	MST
Percentage ϕ-edges Krogan	17.62	18.73	18.57	19.33	18.08
Percentage ϕ-edges Comb	19.14	31.58	31.66	20.25	19.58

This table shows that the average fraction of ϕ-edges in MST is always smaller than those of the other spanning trees. Since MST is made up of edges with large weight, this suggests that in the considered datesets the heaviest edges are likely to be ϕ-free, as one is expecting.

In the next table we give the fraction of prediction mistakes achieved by the various algorithms on the two datasets. The results for OMV are omitted, since they tend to perform poorly on our biological graphs which are rather sparse. Hence, we selected the best three spanning tree performers for WTA, and the best three spanning tree performers for GPA. For simplicity of presentation, the results are averaged over the 17 binary classification problems, In bold are the best accuracy on each dataset. Standard deviations are in braces (these are further averaged over the 17 classes).

Dataset	WTA+MST	WTA+DFST	WTA+RST	GPA+MST	GPA+SPST	GPA+RST
Krogan	18.69 ($\pm$ 0.59)	**18.53 ($\pm$ 0.65)**	18.90 ($\pm$ 0.69)	20.94 ($\pm$ 0.48)	19.82 ($\pm$ 0.47)	21.53 ($\pm$ 0.55)
Comb	**19.82 ($\pm$ 0.56)**	**19.82 ($\pm$ 0.62)**	19.94 ($\pm$ 0.60)	20.90 ($\pm$ 0.40)	21.52 ($\pm$ 0.41)	22.11 ($\pm$ 0.49)

The experiments show that our algorithm outperforms GPA and OMV on both datasets. In particular, though we only reported aggregated results, the same relative performance pattern among the various algorithms repeats sistematically over all 17 binary problems. In addition, WTA runs significantly faster than its competitors, and is also fairly easy to implement. The combination WTA+MST tends to perform best. This might be explained by the fact that MST tends to select light ϕ-edges of the original graph. As a matter of fact, our results also show that WTA can achieve good accuracy results even when combined with DFST, though the use of this kind of spanning tree does not provide the same theoretical performance guarantees as RST. Hence, in practice, DFST might be viewed as a fast and practical way to generate spanning trees for WTA.

9 Work in progress

The above experiments have only been performed on medium-size sparse graphs, and should therefore be considered preliminary in nature. We are now running extensive experiments with further datasets with both sparse and dense graphs. We expect to be able to report them at the workshop. In addition, we are also running experiments with RST by disregarding the edge weights *at generation time*, and then re-assigning them at the end of the tree generation phase. As shown in [19, 1], it is possible to generate this kind of spanning tree in time linear in n for many and important classes of unweighted graphs. The preliminary experiments we conducted suggest that WTA is able to achieve very similar performances as the ones of standard RST. Observe that the resulting algorithm has a total time (including the generation of spanning tree) which is *linear* in the number n of nodes of (most) graphs.

References

[1] N. Alon, C. Avin, M. Koucky, G. Kozma, Z. Lotker and M.R. Tuttle. Many random walks are faster than one. In *Proc. 20th SPAA*, 2008.

[2] M. Belkin, I. Matveeva, and P. Niyogi. Regularization and semi-supervised learning on large graphs. In *Proc. 17th COLT*, 2004.

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

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

[5] H. Chang, and D.Y. Yeung. Graph laplacian kernels for object classification from a single example. *CVPR (2)*, 2011–2016, 2006.

[6] G. Pandey, M. Steinbach, R. Gupta, T. Garg, and V. Kumar. Association analysis-based transformations for protein interaction networks: a function prediction case study. In *Proc. 13th ACM SIGKDD*, 2007.

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

[8] A. Goldberg, and X. Zhu. Seeing stars when there arent many stars: Graph-based semi-supervised learning for sentiment categorization. HLT-NAACL 2006 Workshop on Textgraphs: Graph-based algorithms for Natural Language Processing, 2004.

[9] N. Goyal, L. Rademacher, and S. Vempala. Expanders via random spanning trees. In *Proc. 19th SODA*, 2009.

[10] M. Herbster and M. Pontil. Prediction on a graph with the Perceptron. In *NIPS 19*, 2007.

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

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

[13] T. Ito, T. Chiba, R. Ozawa, M. Yoshida, M. Hattori, and Y. Sakaki. A comprehensive two-hybrid analysis to explore the yeast protein interactome. *PNAS*, 98(8):4569-4574, 2001.

[14] N.J. Krogan *et al.* Global landscape of protein complexes in the yeast Saccharomyces cerevisiae. In *Nature*, 440:637-643, 2006.

[15] R. Lyons and Y. Peres. *Probability on Trees and Networks*. Manuscript, 2009.

[16] A. Ruepp *et al.* The FunCat, a functional annotation scheme for systematic classification of proteins from whole genomes. *Nucleic Acids Research*, 32(18):5539-5545, 2004.

[17] H. Shin, K. Tsuda, and B. Schölkopf. Protein functional class prediction with a combined graph. *Expert Systems with Applications*, 36:3284–3292, 2009.

[18] P. Uetz *et al.* A comprehensive analysis of protein-protein interactions in Saccharomyces cerevisiae. *Nature*, 403(6770):623-627, 2000.

[19] D.B. Wilson. Generating random spanning trees more quickly than the cover time. In *Proc. 28th STOC*, 1996.

[20] X. Zhu, Z. Ghahramani, and J. Lafferty. Semi-supervised learning using Gaussian fields and harmonic functions. In *ICML Workshop on the Continuum from Labeled to Unlabeled Data in Machine Learning and Data Mining*, 2003.