Fast and Optimal Prediction on a Labeled Tree

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

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

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

Abstract

We characterize, up to constant factors, the number of mistakes necessary and sufficient for sequentially predicting a given tree with binary labeled nodes. We provide an efficient algorithm achieving this number of mistakes on any tree. Tree prediction algorithms can solve the general graph prediction problem by representing the graph via one of its spanning trees. In order to cope with adversarial assignments of labels over a general graph, we advocate the use of random spanning trees, which have the additional advantage of retaining relevant spectral information of the original graph.

1 Introduction

Several practically relevant classification tasks can be cast as the problem of predicting the labels associated with vertices of an undirected graph. Among them are, for example, the detection of "spam" sites in the Web graph [15], the classification of genomic data in functional classes where edges represent gene/protein interactions [13], the prediction of user interests in social networks [14]. In the online version of this problem, vertices are presented in an arbitrary order, and the learner must predict the label of each vertex before being able to observe its true value. As real-world applications typically involve large graphs, online learners play an important role because of their good scaling properties. An interesting special case of the online problem is the so-called transductive setting, where the learner is given prior access to the entire unlabeled graph. The transductive setting is interesting in that the learner has the possibily of "reconfiguring" the graph before learning starts, so as to make the problem look easier. This pre-processing of the domain might be seen as a kind of regularization method for the problem of graph prediction.

In this paper, we consider the simplest case of binary labels. In such a case, bounds on the number of prediction mistakes are naturally expressed in terms of the *cutsize*, i.e., the number of edges in the graph whose endpoints are assigned disagreeing labels. This immediately suggests a simple regularization technique: if the mistakes of a prediction algorithm are bounded in terms of the cutsize of the graph, then it should be beneficial to run the algorithm on a thinned version of the original graph where some of the edges have been dropped. Since dropping edges that cause the graph to disconnect is intuitively throwing away too much structural information, we are naturally led to the idea of running the learner on a spanning tree of the original graph.

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

Based on the above argument for using random spanning trees in graph prediction tasks, we mainly focus on the problem of designing a good algorithm for predicting an arbitrary tree. Our main contribution is the derivation of an algorithm that is both optimal (up to constant factors) and efficient. Optimality is meant in the following sense: Given any tree T, the worst-case (over labeling and node presentation order) number of mistakes made by our algorithm can only be improved by a factor which is constant with respect to the relevant parameters. As for efficiency, we show that the overall running time of our algorithm is of order $\min\{K, n_f\}K + n \log D_T$, where K is the cutsize of the (labeled) tree T, D_T is the diameter of T, n is the number of nodes in T, and $n_f < n/2$ is the number of nodes in T with degree bigger than two.

1.1 Related work

Online linear learners, such as the Perceptron algorithm, have been applied to the general graph prediction problem by embedding the n vertices of the graph in $\mathbb{R}^n$ through a map transforming node i to the i-th coordinate versor $\boldsymbol{e}_i \in \mathbb{R}^n$. For example, the graph Perceptron algorithm [8, 6] predicts the label of $\boldsymbol{e}_i$ using the linear kernel $K = L_G^+ + \mathbf{1}\,\mathbf{1}^\top$, where

L_G is the Laplacian of G, L_G^+ is its pseudoinverse, and $\mathbf{1} = (1,\ldots,1)^\top$. The resulting mistake bound is $8\Phi_G(\boldsymbol{y})R_G + 2$ where $\Phi_G(\boldsymbol{y})$ is the cutsize and $R_G = \max_{i,j} r_{i,j}$ is the resistance diameter of G (we write $r_{i,j}$ to denote the effective resistance between i and j). Note the interplay between the factors in the upper bound: if G is dense, then $R_G = \mathcal{O}(1)$ but $\Phi_G(\boldsymbol{y})$ can be of order n^2. If G is sparse, then $\Phi_G(\boldsymbol{y}) = \mathcal{O}(n)$ but then R_G may become of order n.

The idea of using a spanning tree to reduce the cutsize of G has been investigated in [7], where the graph Perceptron is applied to a spanning tree T of G. The resulting mistake bound is of the form $\Phi_T(\boldsymbol{y})D_T$, where D_T is the diameter of the chosen tree. Since $\Phi_T(\boldsymbol{y}) \le \Phi_G(\boldsymbol{y})$ this bound has a smaller cutsize than the previous one. On the other hand, $D_T = \Theta(D_G)$ where D_G is the diameter of G and, in general, D_G can be much larger than R_G. A different technique [5] attempts to control the cutsize by linearizing T via a depth-first visit. This gives a line graph S (the so-called *spine* of G) such that $\Phi_S(\boldsymbol{y}) \le 2\,\Phi_T(\boldsymbol{y})$. By running a Nearest Neighbor (NN) predictor on S, one can prove [5] the mistake bound $\Phi_S(\boldsymbol{y}) \log_2\big((n-1)/\Phi_S(\boldsymbol{y})\big) + \Phi_S(\boldsymbol{y})/\ln 2 + 1$. As observed in [9], similar techniques have been developed to solve low-congestion routing problems. In [7] it is suggested to pick T in order to minimize the diameter D_T. However, since the adversary may concentrate all ϕ-edges (i.e., edges connecting disagreeing labels) on the chosen tree T, there is no guarantee that $\Phi_T(\boldsymbol{y})$ will remain small. A further trick proposed in [5] to take advantage of both previous approaches (graph Perceptron and NN) involves building a binary tree on G. This "support tree" helps in keeping the diameter of G as small as possible. The resulting prediction algorithm is a combination of Perceptron and Nearest-Neighbor previously proposed in [4]. The corresponding mistake bound is $\min_{\rho>0}\big(\mathcal{N}(G,\rho) + 12\Phi_G(\boldsymbol{y})\rho\big) + 1$, where $\mathcal{N}(G,\rho)$ is the smallest number of balls of resistance diameter ρ it takes to cover G. Note that the graph Perceptron bound is recovered when $\rho = R_G$.

There is a vast literature on the problem of drawing random spanning trees from a graph (see, e.g., the recent monograph [10]). For "most" graphs, a random spanning tree can be sampled with a random walk in time $\mathcal{O}(n \ln n)$ [2], or even $\mathcal{O}(n)$ [1, 16], although all known techniques take $\Theta(n^3)$ in the worst case. As a matter of fact, this cubic worst-case bound is a theoretical limitation only, since the bound is hardly met in practice. The space complexity for generating a random spanning tree is always linear in the graph size. Finally, although we exploit random spanning trees to reduce the cutsize, similar approaches can also be used to approximate the cutsize of a weighted graph (see, e.g., [12]).

1.2 Preliminaries

Let T be a tree with n nodes indexed by $1,\ldots,n$. A labeling of T is any assignment $\boldsymbol{y} = (y_1,\ldots,y_n) \in \{-1,+1\}^n$ of binary labels to its nodes. We use $(T,\boldsymbol{y})$ to denote the resulting labeled tree. The online learning protocol for predicting a labeled tree $(T,\boldsymbol{y})$ is defined as follows: The learner is initially given T, but not $\boldsymbol{y}$. At time $t = 1$ an arbitrary node i_1 in T is presented, and the learner must predict its label $y_{i_1} \in \{-1,+1\}$. Then y_{i_1} is revealed and a new node $i_2 \neq i_1$ of T is presented. This process goes on for $t = 1,2,\ldots,n$ until all nodes of T have been selected. The learner's goal is to minimize the number of prediction mistakes.

A ϕ-edge of a labeled tree $(T,\boldsymbol{y})$ is any edge (i,j) such that $y_i \neq y_j$. Let $\mathcal{Y}(T,k)$ be the set of all labelings of T with exactly k ϕ-edges. We will say that $\mathcal{Y}(T,k)$ has *cutsize* k.

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

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

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

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

where the minimum is over all deterministic prediction algorithms.

2 Lower bounds

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

The lower bound is based on the following fact. Given a line graph ℓ (i.e., a "list") with $n+1$ nodes $1,\ldots,n+1$ and $|\ell| = n$ edges, a simple dichotomic adversarial strategy can always force $\lfloor \log_2(n+1) \rfloor$ mistakes using a cutsize of at most 1. In order to achieve this, the adversary initially assigns an arbitrary label to one of the two terminal nodes of ℓ, say node 1. Now let i_1 be the node of ℓ such that there are exactly $\lceil n/2 \rceil$ edges between 1 and i_1. The adversary chooses node i_1 first, and forces a mistake by picking y_{i_1} to be different from the algorithm's prediction. Now let $\ell_1 \subseteq \ell$ be the (sub-)line having as terminal nodes 1 and i_1 if $y_{i_1} \neq y_1$, or nodes i_1 and $n+1$, otherwise. Let i_2 be the node of ℓ_1 such that there are $\lceil |\ell_1|/2 \rceil$ edges between i_1 and i_2. The adversary then chooses node i_2 and another mistake is forced as in the previous step. The adversary proceeds recursively in this way until the chosen sub-line contains a single edge. Then, irrespective to the algorithm's predictions, all the remaining nodes are labeled in such a way that the cutsize does not increase. It is then easy to check that $\lfloor \log_2(n+1) \rfloor$ mistakes are forced. Moreover, it is important to observe that the above adversarial strategy works even if y_1 is already known to the algorithm. On the other hand, this strategy cannot be applied if the known label is on an internal node of ℓ. This fact is used in the proof of Theorem 1 below.

The above adversarial strategy is extended to trees in the following way. The adversary looks for a certain set L of K edge-disjoint line graphs contained in T, and then applies the dichotomic strategy *independently* on each line. To this end, it suffices the set L is a *blanket*, a notion which we now define. Given a set L of edge-disjoint lines contained in a tree

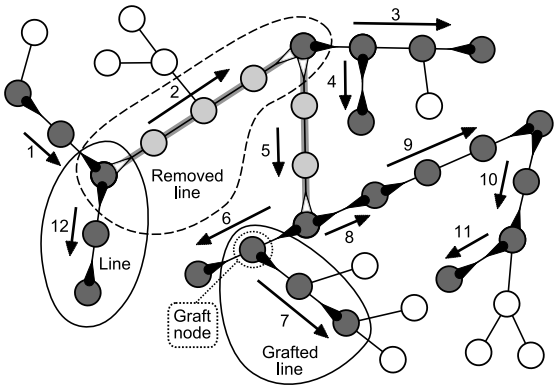

Figure 1: A tree T whose nodes have been divided into three types: dark shaded, light shaded, and white. A connected blanket is shown including dark shaded and light shaded nodes only, along with their connecting edges. Another blanket L, of size 9, is formed by 3 connected blankets (made up of 2, 2 and 6 lines, respectively), and is obtained from the connected blanket by removing the two lines indicated by the light shaded edges and nodes. The terminal nodes of each line are indicated by a bulbous endpoint of the incoming edge. The edges directly connected to white nodes are not part of the underlying connected blanket. The numbers denote a possible depth-first presentation order followed by an adversary that starts from, say, the dark shaded node on the top-left. This adversary assigns to the white nodes the same label as their closest (dark or light) shaded nodes. Similarly, the light shaded nodes belonging to the two removed lines are labeled as the corresponding line terminal nodes.

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

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

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

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

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

Theorem 1 *For any tree T with n nodes and any cutsize $K = 1, \ldots, n-1$, we have* $\text{OPT}(T, K) \geq \text{LB}(T, K)$.

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

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

If $\ell \in L_0$ is a grafted line, the depth-first order insures that ℓ will be processed only after the (unique) line ℓ is grafted onto (in Figure 1, Line 7 is guaranteed to be processed after Line 6). Note that, again, this is key to enabling the application of the lower bounding strategy for line graphs independently on each line in L. Finally, the parts of T not in L_0 (indicated by white nodes in Figure 1) are labeled at the very end. The adversary does not employ any further ϕ-edge by assigning to each such node the same label as the closest labeled node (for instance, the three white nodes on the bottom-right of Figure 1 are assigned the same label as the upper terminal node of Line 11). ■

3 The optimal tree algorithm

In this section we describe a tree prediction algorithm that achieves, up to constant factors, the lower bound proven in the previous section even without knowing the cutsize budget K. Our algorithm, TREEOPT, predicts a node with the label minimizing the cutsize consistent with all labels seen so far. If this label is not unique, then the algorithm predicts using a nearest neighbor method. As we show in Section 4.1, TREEOPT can be viewed as an approximate and efficient implementation of the Halving algorithm for trees.

We say that a label (or node) is **revealed** at time t if the adversary already selected that node (thus causing its label

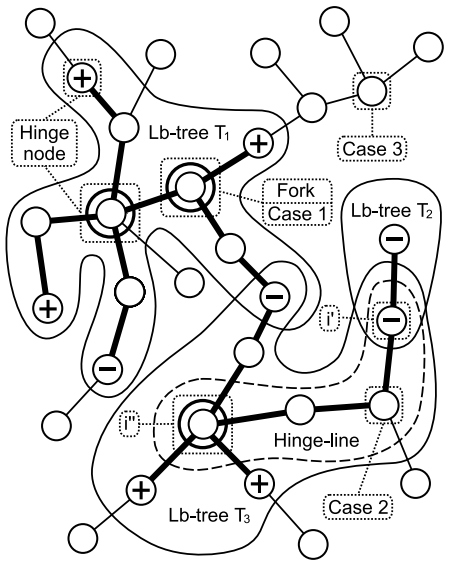

Figure 2: A tree T with 9 revealed labels inducing 3 lb-trees T_1, T_2, and T_3. Fork nodes are denoted by double circles. T_1 has two forks, T_2 has none, T_3 has one. The outer white nodes do not belong to any lb-tree. This figure also explains the behavior of TREEOPT by illustrating examples of the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

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

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

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

2. **Else if** i_t is contained in a lb-tree T' but it is not a fork then:
   - Let i' be the closest hinge node to i_t in T';
   - Let i'' be the second closest hinge node to i_t in T' such that the paths connecting i' and i'' to i have no edges in common (i'' always exists);

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

3. **Else** (i_t is not contained in any lb-tree)
   - Let s be the closest node to i_t in an lb-tree

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

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

to be observed by the algorithm). At any time step, the set of revealed labels defines a collection of edge-disjoint subtrees of T, which we call label-bordered trees (or lb-trees, for short). Formally, given a labeled tree $(T, \boldsymbol{y})$ with revealed labels $y_{i_1}, \ldots, y_{i_t}$, an **lb-tree** is any maximal subtree of T whose leaves are all revealed and no internal node is. Clearly, a non-revealed node can belong to at most one lb-tree. A **fork** node is any node of an lb-tree T' having degree greater than two in T'. Figure 2 gives an example. Note that the set of lb-trees, together with their fork nodes, depends on the set of revealed labels, and is therefore changing with time. For brevity, call a node that is either a fork or a revealed node a **hinge node**. Also, call **hinge line** any line whose terminals are hinge nodes, and such that no internal node is a hinge node. Given a hinge node i, we compute its *estimated label* in such a way that the cutsize of T given the past revealed labels is minimized. The procedure for computing this estimate, called *Fork Label Estimation Procedure* (FLEP), is the core of our algorithm. When there is no unique minimizing label, the procedure assigns the fork a value of 0 ("undecided"), rather than $+1$ or -1. Let $\widetilde{y}_i(t)$ be the label of i estimated by FLEP at time t. If i is revealed at time t then $\widetilde{y}_i(t) = y_i$. Otherwise, $\widetilde{y}_i(t)$ is computed as follows: Let T' be the (unique) lb-tree i belongs to. FLEP performs a depth-first visit of T' rooted at i. The visit starts at i and, when backtracking to a node j after all the children of j have been visited, FLEP assigns a *temporary label* to j given by the majority vote among the temporary or revealed labels of its children. Note that temporary labels set to 0 do not influence this vote. If the vote is a tie, i.e., the sum over all involved labels is 0, then the temporary label of j is set to 0, too. Once all nodes of T' have been visited (and the visit is back to node i) FLEP returns the temporary label $\widetilde{y}_i(t)$ assigned to i. Figure 3 gives an example.

In the box is the pseudocode of our algorithm. This algorithm takes in input a tree T, a set $y_{i_1}, \ldots, y_{i_{t-1}}$ of revealed labels, and a node i_t to be predicted. The algorithm then returns its prediction $\widehat{y}_{i_t}$ for the label of i_t. In particular, if i_t is a fork node inside some lb-tree (Case 1), then TREEOPT just outputs the label $\widetilde{y}_{i_t}(t)$ returned by FLEP, unless FLEP returns 0. In this latter case TREEOPT outputs the default value -1. On the other hand, if i_t is not a fork, but it is still contained in some lb-tree (Case 2), then the algorithm determines the opposite hinge nodes (i' and i'') closest to i_t, computes (again through FLEP) estimated values $\widetilde{y}_{i'}(t)$ and $\widetilde{y}_{i''}(t)$, and uses these values to compute its prediction. If i_t lies between nodes with estimated (or revealed) labels $+1$ and -1 (Case 2.d) then TREEOPT returns the label of the closer node. Finally, if i_t is not contained in any lb-tree (Case 3), the algorithm determines the closest node s inside some lb-tree, and then either predicts through the label of s (if y_s is revealed) or acts as if s were the label to be predicted at time t (i.e., TREEOPT recursively[2] invokes itself with $i_t = s$).

Figure 2 contains examples of the algorithm functioning.

[2]Note that after the recursive call, TREEOPT will not recur any more in that time step, since rule 3.b will subsequently rely on rules 1 or 2 only.

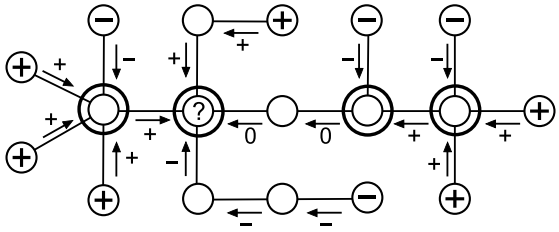

Figure 3: Fork label estimation procedure (FLEP) within the displayed lb-tree. The question mark indicates the fork node whose label has to be estimated. The arrows indicate the backtracking steps of the depth-first visit, where the majority vote (the arrow tags) among the temporary (or revealed) child labels is calculated. For instance, the right-most fork node receives two $+1$ and one -1 from its three incoming neighbors, and thus sends $+1$ to its left. The second fork node from the right receives one $+1$ and one -1, thereby sending out 0. The fork node tagged with "?" is estimated $+1$ (note that 0 is immaterial for the majority vote).

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

4 Mistake bound analysis

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

The following simple property of the function LB is of primary importance. The proof is given in the appendix.

Lemma 2 *For any tree T with n nodes and for any $1 \leq K \leq K' < n$, we have $\textsc{lb}(T, K') \leq \frac{K'}{K}\textsc{lb}(T, K)$.*

At a high level, the proof of optimality hinges on showing $m(\textsc{treeOpt}, T, \leq K) = \mathcal{O}\big(\textsc{ub}(T, K) + K\big)$, where UB is a function that bounds the number of mistakes made by TREEOPT in terms of a size-$\mathcal{O}(K)$ blanket $L(T, \boldsymbol{y})$ over T. This blanket is obtained by dividing T into subparts, and then by mapping each subpart to a set of lines. The union of these lines forms the blanket. Then we show that $\textsc{ub}(T, K) \leq K + 1 + \textsc{lb}(T, \mathcal{O}(K))$. Since by Lemma 2 we have $\textsc{lb}(T, \mathcal{O}(K)) = \mathcal{O}\big(\textsc{lb}(T, K)\big)$, and $K \leq \textsc{lb}(T, K)$ holds by definition of LB, we immediately get $m(\textsc{treeOpt}, T, \leq K) = \mathcal{O}\big(\textsc{lb}(T, K)\big)$. Combined with Theorem 1, this implies the optimality condition $m(\textsc{treeOpt}, T, \leq K) = \mathcal{O}\big(\textsc{opt}(T, K)\big)$.

The proof is a bit involved and requires us to step through several auxiliary definitions and intermediate results. We first introduce the notion of cluster, along with its (inner and outer) structure.

A **cluster** C of a labeled tree $(T, \boldsymbol{y})$ is a maximal subtree of T containing no ϕ-edges. The blanket L over T used in the proof is the union of sets L_C of edge-disjoint lines from each cluster C. The sets L_C will be defined later on.

Let C be a (non-degenerate) cluster containing at least two nodes. We will define a covering $\mathcal{P}(C)$ of the nodes of cluster C —i.e., each node of C belongs to at least one subset in $\mathcal{P}(C)$. Then, we will construct a mapping f that, for each C, bijectively associates elements of $\mathcal{P}(C)$ with elements of subsets of lines in L_C, in such a way that for any non-degenerate C the number of mistakes TREEOPT makes on each element $Q \in \mathcal{P}(C)$ is $\mathcal{O}\big(\sum_{\ell \in f(Q)} m_\ell\big)$. If C is a degenerate cluster (i.e., C contains only one node), then f will not be defined.

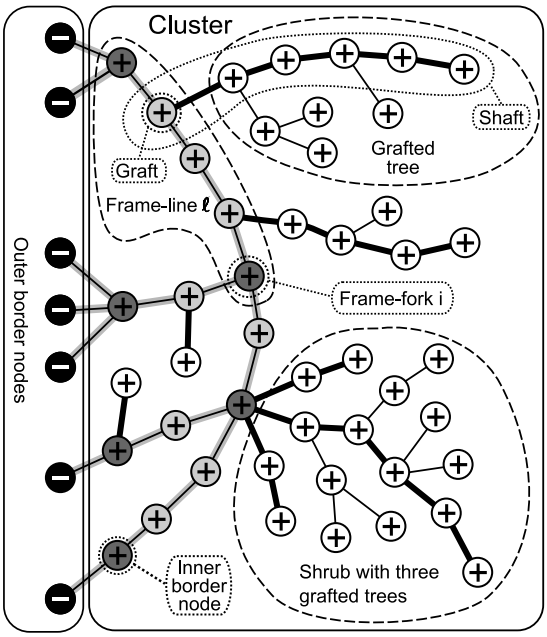

Figure 4: Cluster structure. All nodes within the cluster are labeled $+1$. The displayed cluster C has 7 outer border nodes (hence $\Phi_C = 7$) and 4 inner border nodes. The frame F_C is made up of all dark or light shaded (and $+1$-labeled) nodes. Dark shaded nodes are either frame-forks or inner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

In order to define f over non-degenerate clusters, we need to introduce a specific cluster structure terminology. See Figure 4 for reference.

Definition 3 *Let a cluster C of a labeled tree $(T, \boldsymbol{y})$ be given.*

- *The **outer border nodes** are all nodes of T not in C that are adjacent to (exactly) one node of C. We denote by Φ_C the number of outer border nodes of cluster C, i.e., the number of ϕ-edges connecting nodes in C to the outside.*
- *The **inner border nodes** are all nodes of C that are adjacent to at least an outer border node of C.*
- *The **frame** F_C is the subtree of C whose nodes are on a path connecting any two outer border nodes. We denote by d_i the maximum number of edge-disjoint paths connecting i with outer border nodes.*

- *A **frame-fork** is a node i of F_C such that $d_i \geq 3$.*
- *A **frame-line** is a line $\ell \subseteq F_C$ where each terminal node is either a frame-fork or an inner border node, and such that no internal node of ℓ is a frame-fork. Notice that $d_i = 2$ for all internal nodes i of ℓ.*
- *A tree **grafted on a frame** is*[3] *any connected component of C that remains after deleting all nodes of the frame F_C and all edges incident to them. Notice that $d_i = 1$ for all nodes i of such trees. One can define, more generally, a tree grafted on a subtree T' in a similar way.*
- *A **graft** node i of T is any node of F_C adjacent to a node of a grafted tree T'; we will say that T' is grafted on i;*
- *A **grafted shrub** is a set of one or more trees grafted on the same node;*
- *The **shaft** of a grafted tree T', denoted by $\sigma(T')$, is the line connecting the graft node i of T' to the farthest node in T'. We define $\sigma(\ell)$ to be the shaft of maximal length among all trees grafted on internal nodes of a frame-line ℓ. Moreover, for any shrub S grafted on a node i, we define $\sigma(S)$ to be the set containing the*[4] *$d_i + 1$ longest shafts of trees in S.*

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

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

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

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

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

$$L(T, \boldsymbol{y}) = \bigcup_{C \in \mathcal{C} \setminus \mathcal{C}_1} L_C = \bigcup_{C \in \mathcal{C} \setminus \mathcal{C}_1} \bigcup_{Q \in \mathcal{P}(C)} f(Q)\,. \quad (1)$$

Note that $L = L(T, \boldsymbol{y})$ is a union of lines that do not contain ϕ-edges. If we add to L all ϕ-edges of T we obtain a set of edge-disjoint lines whose only grafted lines are the shafts. Those, in turn, share with the other lines the graft nodes only. Hence this augmented set of lines is a connected blanket, implying that the original L is indeed a blanket over T. We show below (proof of Theorem 12) that $|L| = \mathcal{O}(K)$, being K the maximum cutsize of $(T, \boldsymbol{y})$.

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

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

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

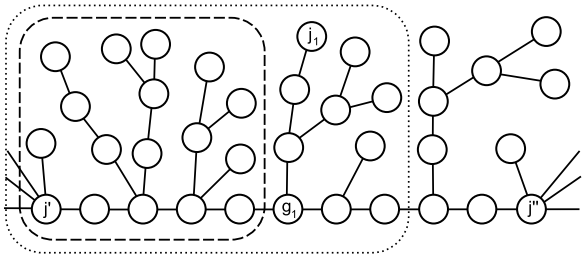

Figure 5: A line ℓ with terminal nodes j' and j'' and grafted trees above. Dichotomic behavior: after j' is revealed, the algorithm makes no more mistakes on the nodes in the dashed rectangle (ties are broken as in prediction rule 1.b). After j_1 is revealed, the algorithm makes no more mistakes on the nodes in the dotted rectangle, etc. Node g_1 is the graft node involved in Case 3.b of TREEOPT when predicting the label of j_1.

Lemma 4 *Let C be a cluster and $\ell \subseteq \ell'$ be a sub-line of some frame-line $\ell' \in F_C$. Assume at time t one of the two terminal nodes of ℓ are revealed. Then after time t the total number of mistakes made by* TREEOPT *on either internal nodes of ℓ or trees grafted on ℓ is bounded by $\lfloor \log_2 |\ell| \rfloor \leq m_\ell$ (see Figure 5 for reference).*

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

Lemma 5 *The total number of mistakes* TREEOPT *makes on internal nodes of ℓ is at most $2m_\ell$.*

Lemma 6 *(Proof omitted.) The total number of trees grafted on ℓ on which* TREEOPT *makes mistakes is at most $2m_\ell + 1$.*

Lemma 7 *(Proof omitted.) There exists at most one tree grafted on ℓ where* TREEOPT *makes more than one mistake.*

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

Lemma 8 *(Proof omitted.) The number of mistakes made by* TREEOPT *on a tree T_0 grafted on the frame F_C of C is at most $m_{\sigma(T_0)} + 1$.*

[3] The reader might expect a grafted line, as defined in Section 2, be a special case of a grafted tree. In fact, the two definitions are slightly divergent, in the sense that the former includes the graft node, while the latter does not. For the sake of presentation, we find it more convenient here to keep the graft node out of the grafted tree.

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

Lemma 9 *The number of grafted trees of a shrub S grafted on i on which* TREEOPT *can make mistakes is at most $d_i + 1$.*

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

Lemma 10 *Let C be a non-degenerate cluster, $\mathcal{P}(C)$ be the corresponding covering. Let f be the bijective mapping defined above. Then the number of mistakes made by* TREEOPT *on any $Q \in \mathcal{P}(C)$ is bounded by $\mathcal{O}\big(\sum_{\ell \in f(Q)} m_\ell\big)$.*

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

Lemma 11 *For all $K = 1, \ldots, n-1$, the function* UB *satisfies* $\textsc{ub}(T, K-1) \leq \textsc{ub}(T, K) + 1$.

Theorem 12 *For any tree T with n nodes and any cutsize budget K, $m(\textsc{treeOpt}, T, \leq K) = \mathcal{O}\big(\textsc{lb}(T, K)\big)$.*

Proof: Pick any labeled tree $(T, \boldsymbol{y})$ and let $k \leq K$ be its cutsize. Let $L = L(T, \boldsymbol{y})$ be the blanket (1). Pick a non-degenerate cluster $C \in \mathcal{C}(T, \boldsymbol{y})$. Let T_C be the tree obtained by augmenting the frame F_C with the Φ_C outer border nodes of C as leaves (referring to Figure 4, the resulting T_C is the tree including all non-white nodes). Then observe that the number of inner border nodes is $\mathcal{O}(\Phi_C)$. Since in any tree the number of nodes of degree larger than 2 cannot be greater than the number of leaves, the total number of frame-forks in C is also $\mathcal{O}(\Phi_C)$. Finally, since a frame-line in F_C is terminated by either a frame-fork or an inner border node, the total number of frame-lines is also $\mathcal{O}(\Phi_C)$. This can be seen by noting that collapsing each frame-line to a single edge turns F_C into a tree with a number of nodes linear in Φ_C. This tree then has $\mathcal{O}(\Phi_C)$ edges, which implies that the frame-lines in F_C are also $\mathcal{O}(\Phi_C)$. Now, by definition of f: (i) the number of lines in L_C deriving from subsets Q of type 1 is linear in the number of frame-lines in C; (ii) since the number of shafts having as terminal node an inner border node (i.e., a leaf of F_C) is linear in Φ_C, and the total number of remaining shafts (i.e., those grafted on frame-forks that are internal nodes of F_C) is linear in the number of frame-lines, the number of lines in L_C deriving from subsets Q of type 2 is $\mathcal{O}(\Phi_C)$. Therefore,

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

To finish the proof observe that, by Lemma 10 and by definition of f,

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

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

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

using Lemma 11 in the last step. Now, UB is defined in terms of a specific blanket L, with $|L| = \mathcal{O}(K)$, and $\big|\mathcal{C}_1(T, \boldsymbol{y})\big| \leq K + 1$ when $\boldsymbol{y}$ has cutsize bounded by K. These facts imply $\textsc{ub}(T, K) \leq K + 1 + \textsc{lb}\big(T, \mathcal{O}(K)\big)$. Finally, using Lemma 2 and $\textsc{lb}(T, K) \geq K$, we obtain $m(\textsc{treeOpt}, T, \leq K) = \mathcal{O}\Big(K + \textsc{lb}\big(T, \mathcal{O}(K)\big)\Big) = \mathcal{O}\big(\textsc{lb}(T, K)\big)$. ■

In order to compare the optimal bound achieved by our algorithm to the bounds mentioned in Section 1.1, we note that, for any given labeled tree $(T, \boldsymbol{y})$, our algorithm makes a number of prediction mistakes whose upper bound can be re-written as

$$\mathcal{O}\big(\Phi_T(\boldsymbol{y})\, \overline{m}_\ell\big) \tag{2}$$

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

Note that $m_\ell < \log_2 D_T + \mathcal{O}(1)$ for all ℓ. Moreover, for many classes of trees T, if the cutsize is not too small then it is not even possible to find a blanket of size $\Phi_T(\boldsymbol{y})$ whose lines have average length linear in D_T. In these cases $\overline{m}_\ell$ can be much smaller than $\log D_T$. As for the time complexity, since $m(\textsc{treeOpt}, T, \leq K) \geq K$ and $m = m(A, T, \leq K) = \Omega\big(m(\textsc{treeOpt}, T, \leq K)\big)$ for any deterministic algorithm A, if the cutsize is $\Omega(\log D_T)$ our algorithm is faster than the one in [4], which predicts all labels in time $\Theta(n\, m)$. Note also that TREEOPT does not require any explicit (and costly) pre-computation. Moreover, unlike Perceptron-like algorithms which use $n \times n$ matrices, the space required by TREEOPT is always linear in n.

4.1 Comparison to the Halving algorithm

We now compare TREEOPT to the so-called HALVING algorithm (applied to trees). This is a standard version space algorithm defined as follows. Let $\mathcal{Y}_t$ be the set of labelings $\boldsymbol{y} \in \{-1, +1\}^n$ consistent with all labels revelead up to time t. Define now $\mathcal{Y}_t^{\min} \subseteq \mathcal{Y}_t$ as those labelings with minimum cutsize in $\mathcal{Y}_t$. HALVING predicts the label of i_t with the value $y \in \{-1, +1\}$ that maximizes $\big|\{\boldsymbol{u} \in \mathcal{Y}_{t-1}^{\min} : u_{i_t} = y\}\big|$. For example, if assigning a certain label to i_t increases the current cutsize (irrespective to the value of the remaining non-revealed labels), then HALVING always predicts the opposite label, i.e., the cut-minimizing label.

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

The following lemma (whose proof is sketched in the appendix) shows that when the fork label estimation procedure

(FLEP) of TREEOPT returns a nondefault value (as in prediction rule 1.a), then this value is the same cut-minimizing label predicted by HALVING.

Lemma 13 *Let $\widetilde{y}_r(t)$ be the value returned by* FLEP *run by* TREEOPT *at time t to evaluate the label of node r. If $\widetilde{y}_r(t) \neq 0$ then $u_r = \widetilde{y}_r(t)$ for each $\boldsymbol{u} \in \mathcal{Y}_t^{\min}$.*

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

Theorem 14 *For any labeled tree $(T, \boldsymbol{y})$ with cutsize K, and any presentation $i_1, \ldots, i_n$ of the nodes of T, the number of times when* TREEOPT *and* HALVING *output a disagreeing prediction is bounded by $\mathcal{O}(\text{LB}(T, K))$.*

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

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

Theorem 14 implies that TREEOPT approximates HALVING, the two algorithms making the same number of mistakes up to constant factors. A close examination of the two algorithms reveals that when TREEOPT predicts a default value, HALVING apparently needs to perform a certain amount of computation. In this respect, we can view TREEOPT as a "lighter" implementation of HALVING. In fact, in the next section we show that TREEOPT can be implemented in a quite efficient manner.

5 Efficient implementation

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

Theorem 15 *The total time* TREEOPT *requires to predict all labels of a labeld tree $(T, \boldsymbol{y})$ with n nodes is*

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

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

Note that whenever $K = \mathcal{O}\big(\sqrt{n}\big)$, the amortized time per step is at most logarithmic in the diameter[5] of T. In order to achieve this speed up, we maintain the following data structures (see Figure 6).

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

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

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

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

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

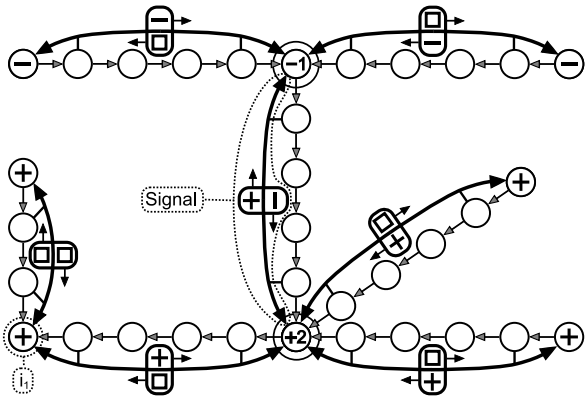

Figure 6: Two lb-trees with the main auxiliary data-structures for the efficient implementation. The numbers inside the fork nodes (the two doubly-circled nodes) indicate the fork values v_i. Node i_1 is located at the bottom-left. The gray arrows, directed towards i_1, are aimed at supporting a quick implementation of prediction rule 3 of TREEOPT when finding the nearest node contained in an lb-tree. The bidirectional black arrows denote signals exchanged between pairs of terminal nodes of hinge lines.

increase (decrease) of signal value $v([i \to j])$ will not propagate if, before the change, $v_j \geq 2$ ($v_j \leq -2$) (all values of outgoing non-empty signal will remain equal to $\text{SGN}(v_j)$).

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

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

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

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

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

Lemma 16 *Let t belong to a phase and let $v_i(t)$ be the value of a fork node i at the beginning of time t. If y_i is not revelead at time t, then $v_i(t+1) \geq v_i(t)$ if $v_i(t) > 0$, and $v_i(t+1) \leq v_i(t)$ if $v_i(t) < 0$.*

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

Proof: [Theorem 15, sketch] Each internal node i of a hinge line ℓ can be visited only $\mathcal{O}\big(\log |\ell|\big) = \mathcal{O}(\log D_T)$ times through prediction rule 2. As a matter of fact, for each of the two traversing directions, the distance between i and the node from which the breadth-first visit over ℓ starts is at least halved each time i gets visited. This fact accounts for the $\mathcal{O}(n \log D_T)$ term in the bound.

Now observe that a node with degree smaller than 3 can never become a fork. Moreover, the number of forks involved in a signal propagation process in each tree grafted on a cluster frame is constant. The number of trees grafted on a frame-line ℓ on which a signal change can propagate is again constant. For each shrub S grafted on a node i, the number of trees of S involved in the signal propagation is $\mathcal{O}(d_i)$. Lemma 16 applied to each fork j, together with these observations, allows us to deduce that in a single phase the signal propagation process takes time $\mathcal{O}\big(\min\{n_f, K\}\big)$. This is also the time required by a signal propagation in each step where the minimal cutsize gets increased. Finally, the number of phases is equal to $\mathcal{O}(K)$.

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

6 Application to labeled graph prediction

We now discuss the application of our tree prediction algorithm to the general problem of predicting the labels of an undirected graph, and compare our results to the existing literature. As mentioned in the introduction, when given a graph $G = (V, E)$ with n nodes and arbitrary binary labels $\boldsymbol{y}$, we suggest running TREEOPT on a (uniformly generated) random spanning tree of G. By exploiting Kirchoff's equivalence between the effective resistance $r_{i,j}$ of $(i, j) \in E$ and the probability that (i, j) belongs to a random spanning tree T, we immediately obtain that the expected cutsize of T is the resistance-weighted cutsize of G, $\Phi_R(\boldsymbol{y}) = \frac{1}{4}\sum_{(i,j)\in E} r_{i,j}(y_i - y_j)^2$. This is significantly better than G's cutsize $\Phi_G(\boldsymbol{y})$ in most cases. In fact, on an unweighted graph with n nodes, the effective resistance $r_{i,j}$ of an edge (i, j) always lies in $[2/n, 1]$. In particular, $r_{i,j}$ is very small when (i, j) is located in a densely connected area of the graph, while $r_{i,j} = 1$ when (i, j) is a bridge edge. For instance, in a dense graph where $r_{i,j} = \mathcal{O}(1/n)$ for all $(i, j) \in E$, the adversary may choose $\boldsymbol{y}$ so as to concentrate $\Theta(n)$ ϕ-edges on any specific tree, and yet $\Phi_R(\boldsymbol{y}) = \mathcal{O}(1)$.

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

Corollary 17 *For any undirected labeled graph $(G, \boldsymbol{y})$, and for any presentation order $i_1, \ldots, i_n$ of the nodes of G, the expected (over the random choice of the spanning tree T) number of mistakes* TREEOPT+ *makes on $(G, \boldsymbol{y})$ is bounded as $\mathbb{E}\big[m(\text{TREEOPT+}, G, \boldsymbol{y})\big] = \mathcal{O}\big(\Phi_R(\boldsymbol{y}) \log n\big)$.*

Proof: We have

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

where the second equality is (2), and the last one follows after (crudely) upper bounding $\overline{m}_\ell$ by $\log n$. ■

Similar bounds could also be shown to hold with high probability, rather than in expectation, by exploiting known concentration properties of random spanning trees. See, e.g., [3] and references therein.

The best mistake bound we know of for the general graph prediction problem has the form $\min_\rho\big(\mathcal{N}(G, \rho) + \Phi_G(\boldsymbol{y})\rho\big)$, where $\mathcal{N}(G, \rho)$ is the covering number of G in the resistance metric [5]. It is easy to see that this bound gets large when the diameter D_G is large. Moreover, real-world graphs G (such as parts of the web graph) have dense regions that can cause a large cutsize. In some of these cases, (take the lollipop graph as an extreme situation), it is just impossible to find a small-sized covering using balls of small radius. A uniformly generated random spanning tree T of G guarantees, instead, that the presence of dense parts of G will not dramatically increase the cutsize of T. Hence the use of TREEOPT on a random tree ensures an appealing (expected) mistake bound where the cutsize factor cannot get too large, except for degenerate and very irregular labelings.

7 Ongoing research

We close by mentioning a few research directions we are currently investigating. First, we are exploring to what extent our tree prediction strategy could be applied to weighted graphs. We would like to prove mistake bounds where the adversary is measured by a weighted version of the effective resistance over ϕ-edges. This seems to require us to somehow generalize TREEOPT to weighted trees.

Second, we are studying the case when the learning algorithm has at its disposal side information about the binary labels to be predicted. A standard way of doing so is to assume each node i is associated with an unknown linear-threshold function $\boldsymbol{u}_i \in \mathbb{R}^d$, and at the beginning of time t the algorithm observes $\boldsymbol{x}_t \in \mathbb{R}^d$ such that $y_{i_t} = \text{SGN}(\boldsymbol{u}_i^\top \boldsymbol{x}_t)$.

Third, we are planning to collect experimental evidence of the performance of our algorithm. Our analysis suggests that TREEOPT is very efficient in both time and space, making it suitable to large-scale practical applications. In particular, we expect a committee of spanning trees drawn at random from a low diameter graph to be a very accurate and efficient compound predictor.

Fourth, as pointed out by one of the reviewers, there seem to be connections between the line of research pursued here and the results presented in [11] about tree searching. It might be possible, although apparently not straightforward, that a different interpretation of our results could be obtained by extending that work. Conversely, it might be possible that the results presented here could contribute to the literature on tree searching. We are planning to explore these connections.

Acknowledgments. Thanks to Mark Herbster for useful discussions. We would also like to thank the COLT 2009 reviewers for their comments which greatly improved the presentation of this paper, and also for pointing out reference [11]. This work was supported in part by the PASCAL2 Network of Excellence under EC grant no. 216886. This publication only reflects the authors' views.

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*, pages 119–128. ACM Press, 2008.

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

[3] N. Goyal, L. Rademacher, and S. Vempala. Expanders via random spanning trees. In *Proc. 19th SODA*, pages 576–585. ACM/SIAM, 2009.

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

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

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

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

[8] M. Herbster, M. Pontil, and L. Wainer. Online learning over graphs. In *Proc. 22nd ICML*, pages 305–132. ACM Press, 2005.

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

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

[11] K. Onak and P. Parys. Generalization of binary search: searching in trees and forest-like partial orders. In *Proc. 47th FOCS*, pages 379-388. IEEE Press, 2006.

[12] D.A. Spielman and N. Srivastava. Graph sparsification by effective resistances. In *Proc. 40th STOC*. ACM Press, 2008.

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

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

[15] Web Spam Challenge, `webspam.lip6.fr`.

[16] D.B. Wilson. Generating random spanning trees more quickly than the cover time. In *Proc. 28th STOC*, pages 206–303. ACM Press, 1996.

A Proofs

Throughout the appendix $\pi(i,j)$ denotes the (unique) path connecting node i to node j, and $d(i,j)$ denotes the number of edges in $\pi(i,j)$. Moreover, without loss of generality, when focusing on the nodes of a given cluster C, we assume they are all given label $+1$.

Proof of Lemma 2: Let L' be a blanket of size K' over T achieving the maximum in the definition of LB. Let L be the subset of L' obtained by keeping only the K longest lines in L'. Since L' is a blanket, so is L. By definition of LB,

$$\sum_{\ell \in L} m_\ell \leq \max_{L \in \mathcal{L}(T,K)} \sum_{\ell \in L} m_\ell = \textsc{lb}(T,K). \tag{3}$$

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

$$m_{\ell'} \leq \frac{1}{K} \sum_{\ell \in L} m_\ell \leq \frac{\textsc{lb}(T,K)}{K} \,. \tag{4}$$

Hence

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

the inequality following from (3) and (4).

Proof of Lemma 4: Let j' be the terminal node of ℓ whose label is revealed, and j'' be the other terminal node. After time t, as soon as the algorithm makes the first mistake on a tree T' grafted on an internal node g of ℓ, the majority vote in the Fork Label Estimation Procedure (FLEP) ensures that the algorithm's estimation on y_g will be correct. Moreover the prediction rules 2.a, 3.a, and 3.b of TREEOPT ensure that no other mistake will be made in the whole shrub grafted on g. In both cases we have used the hypothesis that ℓ contains no frame-forks which could change the outcome of the majority vote. Since the four prediction rules 2.a–2.d of TREEOPT make no distinction between estimated fork labels and true (revealed) labels, for the purpose of this analysis a mistake made on g in ℓ is equivalent to a mistake made on a tree grafted on that node. These observations, combined with prediction rules 2.d and 2.a, imply the two following facts. Given a node r of ℓ, denote by $n(r)$ the closest hinge node to r on $\pi(r,j')$. Then:

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

From the above, it is then easy to see that the number of internal nodes of ℓ on which the algorithm can make mistakes is at least halved after every new mistake. Since correct predictions on nodes of ℓ imply correct predictions on the whole shrub grafted on those nodes (see Figure 5), this halving process implies that the total number of mistakes made after time t on internal nodes of ℓ, or on trees grafted on ℓ, is at most $\lfloor \log_2 |\ell| \rfloor \leq m_\ell$.

Proof of Lemma 5: As soon as the first node i gets revealed, line ℓ is split into the two sub-lines ℓ_1 and ℓ_2 sharing i as terminal node. By Lemma 4 the number of mistakes made on the internal nodes of ℓ is therefore bounded by $1 + \lfloor \log_2 |\ell_1| \rfloor + \lfloor \log_2 |\ell_2| \rfloor \leq 1 + \lfloor \log_2 |\ell_1||\ell_2| \rfloor \leq 1 + \lfloor 2 \log_2 |\ell| - 2 \rfloor \leq 2m_\ell$.

Proof of Lemma 9: If $|S| \leq d_i + 1$ the claim is trivial. Hence, we continue by assuming $|S| > d_i + 1$. Suppose that at least one mistake has been made on $d_i + 1$ trees grafted on i. If y_i is revelead at time t, then the prediction rules 3.a and 2.a ensure that no more mistakes will be made on S. On the contrary, if y_i is not revelead, the majority vote in FLEP guarantees that y_i will always be correctly estimated in the future (i.e., $\widetilde{y}_i(s) = y_i$ for any $s > t$), and the prediction rules 2.a, 3.a, and 3.b guarantee no more mistakes.

Proof of Lemma 10: We first consider the case when Q is of type 1. In this case, the total number of mistakes made on Q can be simply bounded by summing: (i) the mistakes on ℓ (Lemma 5); (ii) the mistakes on the trees grafted on ℓ on which the algorithm can make more than one mistake (Lemma 7 and Lemma 8); (iii) the number of the remaining trees grafted on ℓ where the algorithm can make at most one mistake (Lemma 6 and Lemma 7). Putting together, the total number of mistakes made on Q can be bounded by $\mathcal{O}\big(\sum_{\ell \in f(Q)} m_\ell\big)$.

Let us now consider a subset Q of type 2, and let S be the shrub referred to in the definition of such Q. By Lemma 8, Lemma 9, and the definition of $\sigma(S)$, the total number of mistakes made on Q can be bounded as

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

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

Proof sketch of Lemma 11: Fix T. Any labeling $\boldsymbol{y}$ of T with cutsize $K-1$ can always be obtained from a labeling $\boldsymbol{y}'$ with cutsize K by merging two clusters C_1 and C_2. After this merge, $L(T,\boldsymbol{y})$ contains at most a new line that was not already in $L(T,\boldsymbol{y}')$. This new line ℓ is the ϕ-edge deleted in the merge. Since $|\ell| = 1$, $\textsc{ub}(T,K-1) \leq \textsc{ub}(T,K) + m_\ell = \textsc{ub}(T,K) + 1$.

Proof sketch of Lemma 13: Let T_r be the lb-tree rooted at r at time t. Recall that FLEP works by assigning temporary labels while backtracking in the depth-first visit of T_r. We prove the following claim: each temporary label $y'_i(t) \neq 0$ assigned to node i of T_r is such that the cutsize is at least as small as the cutsize when i is assigned the opposite label $-y'_i(t)$. The proof is by induction on the maximum distance between i and its descendants in T_r. When $y'_i(t) = 0$ we show that the cutsize-minimizing label for i is the same as i's parent. Finally, by applying the claim to the children of r, we obtain that the cutsize-minimizing label of r is the majority vote over the children's temporary (or revealed) labels.