See the Tree Through the Lines: The Shazoo Algorithm*

Fabio Vitale
DSI, University of Milan, Italy
fabio.vitale@unimi.it

Nicolò Cesa-Bianchi
DSI, University of Milan, Italy
nicolo.cesa-bianchi@unimi.it

Claudio Gentile
DICOM, University of Insubria, Italy
claudio.gentile@uninsubria.it

Giovanni Zappella
Dept. of Mathematics, Univ. of Milan, Italy
giovanni.zappella@unimi.it

Abstract

Predicting the nodes of a given graph is a fascinating theoretical problem with applications in several domains. Since graph sparsification via spanning trees retains enough information while making the task much easier, trees are an important special case of this problem. Although it is known how to predict the nodes of an unweighted tree in a nearly optimal way, in the weighted case a fully satisfactory algorithm is not available yet. We fill this hole and introduce an efficient node predictor, SHAZOO, which is nearly optimal on any weighted tree. Moreover, we show that SHAZOO can be viewed as a common nontrivial generalization of both previous approaches for unweighted trees and weighted lines. Experiments on real-world datasets confirm that SHAZOO performs well in that it fully exploits the structure of the input tree, and gets very close to (and sometimes better than) less scalable energy minimization methods.

1 Introduction

Predictive analysis of networked data is a fast-growing research area whose application domains include document networks, online social networks, and biological networks. In this work we view networked data as weighted graphs, and focus on the task of node classification in the transductive setting, i.e., when the unlabeled graph is available beforehand. Standard transductive classification methods, such as label propagation [2, 3, 18], work by optimizing a cost or energy function defined on the graph, which includes the training information as labels assigned to training nodes. Although these methods perform well in practice, they are often computationally expensive, and have performance guarantees that require statistical assumptions on the selection of the training nodes.

A general approach to sidestep the above computational issues is to sparsify the graph to the largest possible extent, while retaining much of its spectral properties —see, e.g., [5, 6, 12, 16]. Inspired by [5, 6], this paper reduces the problem of node classification from graphs to trees by extracting suitable *spanning trees* of the graph, which can be done quickly in many cases. The advantage of performing this reduction is that node prediction is much easier on trees than on graphs. This fact has recently led to the design of very scalable algorithms with nearly optimal performance guarantees in the online transductive model, which comes with no statistical assumptions. Yet, the current results in node classification on trees are not satisfactory. The TREEOPT strategy of [5] is optimal to within constant factors, but only on *unweighted* trees. No equivalent optimality results are available for general weighted trees. To the best of our knowledge, the only other comparable result is WTA by [6], which is optimal (within log factors) only on weighted lines. In fact, WTA can still be applied to weighted trees by exploiting an idea contained in [9]. This is based on linearizing the tree via a depth-first visit. Since linearization loses most of the structural information of the tree,

*This work was supported in part by Google Inc. through a Google Research Award, and by the PASCAL2 Network of Excellence under EC grant 216886. This publication only reflects the authors' views.

this approach yields suboptimal mistake bounds. This theoretical drawback is also confirmed by empirical performance: throwing away the tree structure negatively affects the practical behavior of the algorithm on real-world weighted graphs.

The importance of weighted graphs, as opposed to unweighted ones, is suggested by many practical scenarios where the nodes carry more information than just labels, e.g., vectors of feature values. A natural way of leveraging this side information is to set the weight on the edge linking two nodes to be some function of the similariy between the vectors associated with these nodes. In this work, we bridge the gap between the weighted and unweighted cases by proposing a new prediction strategy, called SHAZOO, achieving a mistake bound that depends on the detailed structure of the weighted tree. We carry out the analysis using a notion of learning bias different from the one used in [6] and more appropriate for weighted graphs. More precisely, we measure the regularity of the unknown node labeling via the weighted cutsize induced by the labeling on the tree (see Section 3 for a precise definition). This replaces the unweighted cutsize that was used in the analysis of WTA. When the weighted cutsize is used, a cut edge violates this inductive bias in proportion to its weight. This modified bias does not prevent a fair comparison between the old algorithms and the new one: SHAZOO specializes to TREEOPT in the unweighted case, and to WTA when the input tree is a weighted line. By specializing SHAZOO's analysis to the unweighted case we recover TREEOPT's optimal mistake bound. When the input tree is a weighted line, we recover WTA's mistake bound expressed through the weighted cutsize instead of the unweighted one. The effectiveness of SHAZOO on any tree is guaranteed by a corresponding lower bound (see Section 3).

SHAZOO can be viewed as a common nontrivial generalization of both TREEOPT and WTA. Obtaining this generalization while retaining and extending the optimality properties of the two algorithms is far from being trivial from a conceptual and technical standpoint. Since SHAZOO works in the online transductive model, it can easily be applied to the more standard train/test (or "batch") transductive setting: one simply runs the algorithm on an arbitrary permutation of the training nodes, and obtains a predictive model for all test nodes. However, the implementation might take advantage of knowing the set of training nodes beforehand. For this reason, we present two implementations of SHAZOO, one for the online and one for the batch setting. Both implementations result in fast algorithms. In particular, the batch one is linear in $|V|$. This is achieved by a fast algorithm for weighted cut minimization on trees, a procedure which lies at the heart of SHAZOO.

Finally, we test SHAZOO against WTA, label propagation, and other competitors on real-world weighted graphs. In *almost all* cases (as expected), we report improvements over WTA due to the better sensitivity to the graph structure. In some cases, we see that SHAZOO even outperforms standard label propagation methods. Recall that label propagation has a running time per prediction which is proportional to $|E|$, where E is the graph edge set. On the contrary, SHAZOO can typically be run in *constant* amortized time per prediction by using Wilson's algorithm for sampling random spanning trees [17]. By disregarding edge weights in the initial sampling phase, this algorithm is able to draw a random (unweighted) spanning tree in time proportional to $|V|$ on most graphs. Our experiments reveal that using the edge weights only in the subsequent prediction phase causes in practice only a minor performance degradation.

2 Preliminaries and basic notation

Let $T = (V, E, W)$ be an undirected and weighted tree with $|V| = n$ nodes, positive edge weights $W_{i,j} > 0$ for $(i, j) \in E$, and $W_{i,j} = 0$ for $(i, j) \notin E$. A binary 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 weighted tree. The online learning protocol for predicting $(T, \boldsymbol{y})$ is defined as follows. The learner is given T while $\boldsymbol{y}$ is kept hidden. The nodes of T are presented to the learner one by one, according to an unknown and arbitrary permutation $i_1, \ldots, i_n$ of V. At each time step $t = 1, \ldots, n$ node i_t is presented and the learner must issue a prediction $\widehat{y}_{i_t} \in \{-1, +1\}$ for the 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.

Following previous works [10, 9, 5, 6], we measure the regularity of a labeling $\boldsymbol{y}$ of T in terms of ϕ-edges, where a ϕ-edge for $(T, \boldsymbol{y})$ is any $(i, j) \in E$ such that $y_i \neq y_j$. The overall amount of irregularity in a labeled tree $(T, \boldsymbol{y})$ is the **weighted cutsize** $\Phi^W = \sum_{(i,j) \in E^\phi} W_{i,j}$, where $E^\phi \subseteq E$ is the subset of ϕ-edges in the tree. We use the weighted cutsize as our learning bias, that is, we want to design algorithms whose predictive performance scales with Φ^W. Unlike the ϕ-edge count $\Phi = |E^\phi|$, which is a good measure of regularity for unweighted graphs, the weighted cutsize takes

the edge weight $W_{i,j}$ into account when measuring the irregularity of a ϕ-edge (i,j). In the sequel, when we measure the distance between any pair of nodes i and j on the input tree T we always use the resistance distance metric d, that is, $d(i,j) = \sum_{(r,s)\in\pi(i,j)} \frac{1}{W_{r,s}}$, where $\pi(i,j)$ is the unique path connecting i to j.

3 A lower bound for weighted trees

In this section we show that the weighted cutsize can be used as a lower bound on the number of online mistakes made by any algorithm on any tree. In order to do so (and unlike previous papers on this specific subject —see, e.g., [6]), we need to introduce a more refined notion of adversarial "budget". Given $T = (V, E, W)$, let $\xi(M)$ be the maximum number of edges of T such that the sum of their weights does not exceed M, $\xi(M) = \max\left\{|E'| : E' \subseteq E,\ \sum_{(i,j)\in E'} w_{i,j} \le M\right\}$. We have the following simple lower bound (all proofs are omitted from this extended abstract).

Theorem 1 *For any weighted tree $T = (V, E, W)$ there exists a randomized label assignment to V such that any algorithm can be forced to make at least $\xi(M)/2$ online mistakes in expectation, while $\Phi^W \le M$.*

Specializing [6, Theorem 1] to trees gives the lower bound $K/2$ under the constraint $\Phi \le K \le |V|$. The main difference between the two bounds is the measure of label regularity being used: Whereas Theorem 1 uses Φ^W, which depends on the weights, [6, Theorem 1] uses the weight-independent quantity Φ. This dependence of the lower bound on the edge weights is consistent with our learning bias, stating that a heavy ϕ-edge violates the bias more than a light one. Since ξ is nondecreasing, the lower bound implies a number of mistakes of at least $\xi(\Phi^W)/2$. Note that $\xi(\Phi^W) \ge \Phi$ for any labeled tree $(T, \boldsymbol{y})$. Hence, whereas a constraint K on Φ implies forcing at least $K/2$ mistakes, a constraint M on Φ^W allows the adversary to force a potentially larger number of mistakes.

In the next section we describe an algorithm whose mistake bound nearly matches the above lower bound on any weighted tree when using $\xi(\Phi^W)$ as the measure of label regularity.

4 The Shazoo algorithm

In this section we introduce the SHAZOO algorithm, and relate it to previously proposed methods for online prediction on unweighted trees (TREEOPT from [5]) and weighted line graphs (WTA from [6]). In fact, SHAZOO is optimal on any weighted tree, and reduces to TREEOPT on unweighted trees and to WTA on weighted line graphs. Since TREEOPT and WTA are optimal on *any* unweighted tree and *any* weighted line graph, respectively, SHAZOO necessarily contains elements of both of these algorithms.

In order to understand our algorithm, we now define some relevant structures of the input tree T. See Figure 1 (left) for an example. These structures evolve over time according to the set of observed labels. First, we call **revealed** a node whose label has already been observed by the online learner; otherwise, a node is **unrevealed**. A **fork** is any unrevealed node connected to at least three different revealed nodes by edge-disjoint paths. A **hinge node** is either a revealed node or a fork. A **hinge tree** is any component of the forest obtained by removing from T all *edges* incident to hinge nodes; hence any fork or labeled node forms a 1-node hinge tree. When a hinge tree H contains only one hinge node, a **connection node** for H is the node contained in H. In all other cases, we call a connection node for H any node outside H which is adjacent to a node in H. A **connection fork** is a connection node which is also a fork. Finally, a **hinge line** is any path connecting two hinge nodes such that no internal node is a hinge node.

Given an unrevealed node i and a label value $y \in \{-1, +1\}$, the **cut function** $\mathrm{cut}(i, y)$ is the value of the minimum weighted cutsize of T over all labelings $\boldsymbol{y} \in \{-1, +1\}^n$ consistent with the labels seen so far and such that $y_i = y$. Define $\Delta(i) = \mathrm{cut}(i, -1) - \mathrm{cut}(i, +1)$ if i is unrevealed, and $\Delta(i) = y_i$, otherwise. The algorithm's pseudocode is given in Algorithm 1. At time t, in order to predict the label y_{i_t} of node i_t, SHAZOO calculates $\Delta(i)$ for all connection nodes i of $H(i_t)$, where $H(i_t)$ is the hinge tree containing i_t. Then the algorithm predicts y_{i_t} using the label of the connection node i of $H(i_t)$ which is closest to i_t and such that $\Delta(i) \neq 0$ (recall from Section 2 that all distances/lengths are measured using the resistance metric). Ties are broken arbitrarily. If $\Delta(i) = 0$ for all connection nodes i in $H(i_t)$ then SHAZOO predicts a default value (-1 in the

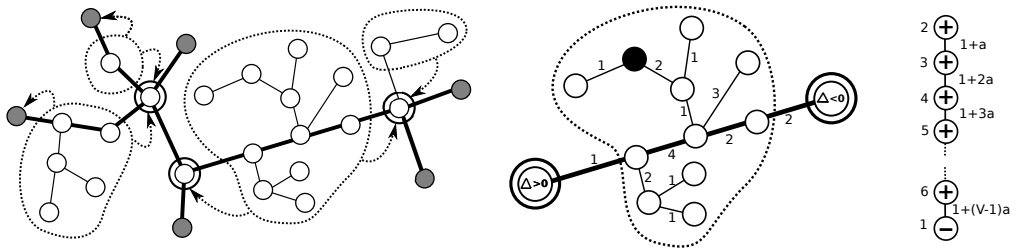


Figure 1: **Left:** An input tree. Revealed nodes are dark grey, forks are doubly circled, and hinge lines have thick black edges. The hinge trees not containing hinge nodes (i.e., the ones that are not singletons) are enclosed by dotted lines. The dotted arrows point to the connection node(s) of such hinge trees. **Middle:** The predictions of SHAZOO on the nodes of a hinge tree. The numbers on the edges denote edge weights. At a given time t, SHAZOO uses the value of Δ on the two hinge nodes (the doubly circled ones, which are also forks in this case), and is required to issue a prediction on node i_t (the black node in this figure). Since i_t is between a positive Δ hinge node and a negative Δ hinge node, SHAZOO goes with the one which is closer in resistance distance, hence predicting $\widehat{y}_{i_t} = -1$. **Right:** A simple example where the mincut prediction strategy does not work well in the weighted case. In this example, mincut mispredicts all labels, yet $\Phi = 1$, and the ratio of Φ^W to the total weight of all edges is about $1/|V|$. The labels to be predicted are presented according to the numbers on the left of each node. Edge weights are also displayed, where a is a very small constant.

pseudocode). If i_t is a fork (which is also a hinge node), then $H(i_t) = \{i_t\}$. In this case, i_t is a connection node of $H(i_t)$, and obviously the one closest to itself. Hence, in this case SHAZOO predicts y_t simply by $\widehat{y}_{i_t} = \operatorname{sgn}\big(\Delta(i_t)\big)$. See Figure 1 (middle) for an example. On unweighted

Algorithm 1: SHAZOO

for $t = 1 \dots n$
  Let $C\big(H(i_t)\big)$ be the set of the connection nodes i of $H(i_t)$ for which $\Delta(i) \neq 0$
  if $C\big(H(i_t)\big) \not\equiv \emptyset$
    Let j be the node of $C\big(H(i_t)\big)$ closest to i_t
    Set $\widehat{y}_{i_t} = \operatorname{sgn}\big(\Delta(j)\big)$
  else Set $\widehat{y}_{i_t} = -1$ (default value)

trees, computing $\Delta(i)$ for a connection node i reduces to the Fork Label Estimation Procedure in [5, Lemma 13]. On the other hand, predicting with the label of the connection node closest to i_t in resistance distance is reminiscent of the nearest-neighbor prediction of WTA on weighted line graphs [6]. In fact, as in WTA, this enables to take advantage of labelings whose ϕ-edges are light weighted. An important limitation of WTA is that this algorithm linearizes the input tree. On the one hand, this greatly simplifies the analysis of nearest-neighbor prediction; on the other hand, this prevents exploiting the structure of T, thereby causing logaritmic slacks in the upper bound of WTA. The TREEOPT algorithm, instead, performs better when the unweighted input tree is very different from a line graph (more precisely, when the input tree cannot be decomposed into long edge-disjoint paths, e.g., a star graph). Indeed, TREEOPT's upper bound does not suffer from logaritmic slacks, and is tight up to constant factors on any unweighted tree. Similar to TREEOPT, SHAZOO does not linearize the input tree and extends to the weighted case TREEOPT's superior performance, also confirmed by the experimental comparison reported in Section 6.

In Figure 1 (right) we show an example that highlights the importance of using the Δ function to compute the fork labels. Since Δ predicts a fork i_t with the label that minimizes the weighted cutsize of T consistent with the revealed labels, one may wonder whether computing Δ through mincut based on the number of ϕ-edges (rather than their weighted sum) could be an effective prediction strategy. Figure 1 (right) illustrates an example of a simple tree where such a Δ mispredicts the labels of all nodes, when both Φ^W and Φ are small.

Remark 1 *We would like to stress that* SHAZOO *can also be used to predict the nodes of an arbitrary* graph *by first drawing a random spanning tree T of the graph, and then predicting optimally on T —see, e.g., [5, 6]. The resulting mistake bound is simply the expected value of* SHAZOO*'s mistake bound over the random draw of T. By using a fast spanning tree sampler [17], the involved computational overhead amounts to constant amortized time per node prediction on "most" graphs.*

Remark 2 *In certain real-world input graphs, the presence of an edge linking two nodes may also carry information about the extent to which the two nodes are* dissimilar*, rather than similar. This information can be encoded by the sign of the weight, and the resulting network is called a* signed graph*. The regularity measure is naturally extended to signed graphs by counting the weight of* frustrated edges *(e.g.,[7]), where (i, j) is frustrated if $y_i y_j \neq \operatorname{sgn}(w_{i,j})$. Many of the existing algorithms for node classification [18, 9, 10, 5, 8, 6] can in principle be run on signed graphs. However, the computational cost may not always be preserved. For example, mincut [4] is in general NP-hard when the graph is signed [13]. Since our algorithm sparsifies the graph using trees, it can be run efficiently even in the signed case. We just need to re-define the Δ function as $\Delta(i) = \operatorname{fcut}(i,-1) - \operatorname{fcut}(i,+1)$, where* fcut *is the minimum total weight of frustrated edges consistent with the labels seen so far. The argument contained in Section 5 for the positive edge weights (see, e.g., Eq. (1) therein) allows us to show that also this version of Δ can be computed efficiently. The prediction rule has to be re-defined as well: We count the parity of the number z of negative-weighted edges along the path connecting i_t to the closest node $j \in C\big(H(i_t)\big)$, i.e., $\widehat{y}_{i_t} = (-1)^z \operatorname{sgn}\big(\Delta(j)\big)$.*

Remark 3 *In [5] the authors note that* TREEOPT *approximates a version space (Halving) algorithm on the set of tree labelings. Interestingly,* SHAZOO *is also an approximation to a more general Halving algorithm for weighted trees. This generalized Halving gives a weight to each labeling consistent with the labels seen so far and with the sign of $\Delta(f)$ for each fork f. These weighted labelings, which depend on the weights of the ϕ-edges generated by each labeling, are used for computing the predictions. One can show (details omitted due to space limitations) that this generalized Halving algorithm has a mistake bound within a constant factor of* SHAZOO*'s.*

5 Mistake bound analysis and implementation

We now show that SHAZOO is nearly optimal on every weighted tree T. We obtain an upper bound in terms of Φ^W and the structure of T, nearly matching the lower bound of Theorem 1. We now give some auxiliary notation that is strictly needed for stating the mistake bound.

Given a labeled tree $(T, \boldsymbol{y})$, a **cluster** is any maximal subtree whose nodes have the same label. An **in-cluster line graph** is any line graph that is entirely contained in a single cluster. Finally, given a line graph L, we set $R_L^W = \sum_{(i,j)\in L} \frac{1}{W_{i,j}}$, i.e., the (resistance) distance between its terminal nodes.

Theorem 2 *For any labeled and weighted tree $(T, \boldsymbol{y})$, there exists a set $\mathcal{L}_T$ of $\mathcal{O}\big(\xi(\Phi^W)\big)$ edge-disjoint in-cluster line graphs such that the number of mistakes made by* SHAZOO *is at most of the order of*

$$\sum_{L\in\mathcal{L}_T} \min\Big\{|L|, 1 + \big\lfloor \log\big(1 + \Phi^W R_L^W\big)\big\rfloor\Big\} .$$

The above mistake bound depends on the tree structure through $\mathcal{L}_T$. The sum contains $\mathcal{O}\big(\xi(\Phi^W)\big)$ terms, each one being at most logarithmic in the scale-free products $\Phi^W R_L^W$. The bound is governed by the same key quantity $\xi\big(\Phi^W\big)$ occurring in the lower bound of Theorem 1. However, Theorem 2 also shows that SHAZOO can take advantage of trees that cannot be covered by long line graphs. For example, if the input tree T is a weighted line graph, then it is likely to contain long in-cluster lines. Hence, the factor multiplying $\xi\big(\Phi^W\big)$ may be of the order of $\log\big(1 + \Phi^W R_L^W\big)$. If, instead, T has constant diameter (e.g., a star graph), then the in-cluster lines can only contain a constant number of nodes, and the number of mistakes can never exceed $\mathcal{O}\big(\xi(\Phi^W)\big)$. This is a log factor improvement over WTA which, by its very nature, cannot exploit the structure of the tree it operates on.[1]

As for the implementation, we start by describing a method for calculating $\operatorname{cut}(v, y)$ for any unlabeled node v and label value y. Let T^v be the maximal subtree of T rooted at v, such that no internal node is revealed. For any node i of T^v, let T_i^v be the subtree of T^v rooted at i. Let $\Phi_i^v(y)$ be the minimum weighted cutsize of T_i^v consistent with the revealed nodes and such that $y_i = y$. Since

[1] One might wonder whether an arbitrarily large gap between upper (Theorem 2) and lower (Theorem 1) bounds exists due to the extra factors depending on $\Phi^W R_L^W$. One way to get around this is to follow the analysis of WTA in [6]. Specifically, we can adapt here the more general analysis from that paper (see Lemma 2 therein) that allows us to drop, for any integer K, the resistance contribution of K arbitrary non-ϕ edges of the line graphs in $\mathcal{L}_T$ (thereby reducing R_L^W for any L containing any of these edges) at the cost of increasing the mistake bound by K. The details will be given in the full version of this paper.

$\Delta(v) = \mathrm{cut}(v,-1) - \mathrm{cut}(v,+1) = \Phi_v^v(-1) - \Phi_v^v(+1)$, our goal is to compute $\Phi_v^v(y)$. It is easy to see by induction that the quantity $\Phi_i^v(y)$ can be recursively defined as follows, where C_i^v is the set of all children of i in T^v, and $Y_j \equiv \{y_j\}$ if y_j is revealed, and $Y_j \equiv \{-1,+1\}$, otherwise:[2]

$$\Phi_i^v(y) = \begin{cases} \displaystyle\sum_{j \in C_i^v} \min_{y' \in Y_j} \Big(\Phi_j^v(y') + \mathbb{I}\{y' \neq y\}\, w_{i,j} \Big) & \text{if } i \text{ is an internal node of } T^v \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

Now, $\Phi_v^v(y)$ can be computed through a simple depth-first visit of T^v. In all backtracking steps of this visit the algorithm uses (1) to compute $\Phi_i^v(y)$ for each node i, the values $\Phi_j^v(y)$ for all children j of i being calculated during the previous backtracking steps. The total running time is therefore linear in the number of nodes of T^v.

Next, we describe the basic implementation of SHAZOO for the on-line setting. A batch learning implementation will be given at the end of this section. The online implementation is made up of three steps.

1. Find the hinge nodes of subtree T^{i_t}. Recall that a hinge-node is either a fork or a revealed node. Observe that a fork is incident to at least three nodes lying on different hinge lines. Hence, in this step we perform a depth-first visit of T^{i_t}, marking each node lying on a hinge line. In order to accomplish this task, it suffices to single out all forks marking each labeled node and, recursively, each parent of a marked node of T^{i_t}. At the end of this process we are able to single out the forks by counting the number of edges (i,j) of each marked node i such that j has been marked, too. The remaining hinge nodes are the leaves of T^{i_t} whose labels have currently been revealed.

2. Compute $\mathrm{sgn}(\Delta(i))$ for all connection forks of $H(i_t)$. From the previous step we can easily find the connection node(s) of $H(i_t)$. Then, we simply exploit the above-described technique for computing the cut function, obtaining $\mathrm{sgn}(\Delta(i))$ for all connection forks i of $H(i_t)$.

3. Propagate the labels of the nodes of $C(H(i_t))$ (only if i_t is not a fork). We perform a visit of $H(i_t)$ starting from every node $r \in C(H(i_t))$. During these visits, we mark each node j of $H(i_t)$ with the label of r computed in the previous step, together with the length of $\pi(r,j)$, which is what we need for predicting any label of $H(i_t)$ at the current time step.

The overall running time is dominated by the first step and the calculation of $\Delta(i)$. Hence the worst case running time is proportional to $\sum_{t \le |V|} |V(T^{i_t})|$. This quantity can be quadratic in $|V|$, though this is rarely encountered in practice if the node presentation order is not adversarial. For example, it is easy to show that in a line graph, if the node presentation order is random, then the total time is of the order of $|V| \log |V|$. For a star graph the total time complexity is always linear in $|V|$, even on adversarial orders.

In many real-world scenarios, one is interested in the more standard problem of predicting the labels of a given subset of *test* nodes based on the available labels of another subset of *training* nodes. Building on the above on-line implementation, we now derive an implementation of SHAZOO for this train/test (or "batch learning") setting. We first show that computing $|\Phi_i^i(+1)|$ and $|\Phi_i^i(-1)|$ for all unlabeled nodes i in T takes $\mathcal{O}(|V|)$ time. This allows us to compute $\mathrm{sgn}(\Delta(v))$ for all forks v in $\mathcal{O}(|V|)$ time, and then use the first and the third steps of the on-line implementation. Overall, we show that predicting *all* labels in the test set takes $\mathcal{O}(|V|)$ time.

Consider tree T^i as rooted at i. Given any unlabeled node i, we perform a visit of T^i starting at i. During the backtracking steps of this visit we use (1) to calculate $\Phi_j^i(y)$ for each node j in T^i and label $y \in \{-1,+1\}$. Observe now that for any pair i,j of adjacent unlabeled nodes and any label $y \in \{-1,+1\}$, once we have obtained $\Phi_i^i(y)$, $\Phi_j^i(+1)$ and $\Phi_j^i(-1)$, we can compute $\Phi_i^j(y)$ in constant time, as $\Phi_i^j(y) = \Phi_i^i(y) - \min_{y' \in \{-1,+1\}} \big(\Phi_j^i(y') + \mathbb{I}\{y' \neq y\}\, w_{i,j} \big)$. In fact, all children of j in T^i are descendants of i, while the children of i in T^i (but j) are descendants of j in T^j. SHAZOO computes $\Phi_i^i(y)$, we can compute in constant time $\Phi_i^j(y)$ for all child nodes j of i in T^i, and use this value for computing $\Phi_j^j(y)$. Generalizing this argument, it is easy to see that in the next phase we can compute $\Phi_k^k(y)$ in constant time for all nodes k of T^i such that for all ancestors u of k and all $y \in \{-1,+1\}$, the values of $\Phi_u^u(y)$ have previously been computed.

[2] The recursive computations contained in this section are reminiscent of the sum-product algorithm [11].

The time for computing $\Phi_s^s(y)$ for all nodes s of T^i and any label y is therefore linear in the time of performing a breadth-first (or depth-first) visit of T^i, i.e., linear in the number of nodes of T^i. Since each labeled node with degree d is part of at most d trees T^i for some i, we have that the total number of nodes of all distinct (edge-disjoint) trees T^i across $i \in V$ is linear in $|V|$.

Finally, we need to propagate the connection node labels of each hinge tree as in the third step of the online implementation. Since also this last step takes linear time, we conclude that the total time for predicting all labels is linear in $|V|$.

6 Experiments

We tested our algorithm on a number of real-world weighted graphs from different domains (character recognition, text categorization, bioinformatics, Web spam detection) against the following baselines:

Online Majority Vote (OMV). This is an intuitive and fast algorithm for sequentially predicting the node labels is via a weighted majority vote over the labels of the adjacent nodes seen so far. Namely, OMV predicts y_{i_t} through the sign of $\sum_s y_{i_s} w_{i_s,i_t}$, where s ranges over $s < t$ such that $(i_s, i_t) \in E$. Both the total time and space required by OMV are $\Theta(|E|)$.

Label Propagation (LABPROP). LABPROP [18, 2, 3] is a batch transductive learning method computed by solving a system of linear equations which requires total time of the order of $|E| \times |V|$. This relatively high computational cost should be taken into account when comparing LABPROP to faster online algorithms. Recall that OMV can be viewed as a fast "online approximation" to LABPROP.

Weighted Tree Algorithm (WTA). As explained in the introductory section, WTA can be viewed as a special case of SHAZOO. When the input graph is not a line, WTA turns it into a line by first extracting a spanning tree of the graph, and then linearizing it. The implementation described in [6] runs in constant amortized time per prediction whenever the spanning tree sampler runs in time $\Theta(|V|)$.

The Graph Perceptron algorithm [10] is another readily available baseline. This algorithm has been excluded from our comparison because it does not seem to be very competitive in terms of performance (see, e.g., [6]), and is also computationally expensive.

In our experiments, we combined SHAZOO and WTA with spanning trees generated in different ways (note that OMV and LABPROP do not need to extract spanning trees from the input graph).

Random Spanning Tree (RST). Following Ch. 4 of [12], we draw a weighted spanning tree with probability proportional to the product of its edge weights. We also tested our algorithms combined with random spanning trees generated uniformly at random ignoring the edge weights (i.e., the weights were only used to compute predictions on the randomly generated tree) —we call these spanning trees NWRST (no-weight RST). On most graphs, this procedure can be run in time linear in the number of nodes [17]. Hence, the combinations SHAZOO+NWRST and WTA+NWRST run in $\mathcal{O}(|V|)$ time on most graphs.

Minimum Spanning Tree (MST). This is the spanning tree minimizing the sum of the resistors on its edges. This tree best approximates the original graph in terms of the trace norm distance of the corresponding Laplacian matrices.

Following [10, 6], we also ran SHAZOO and WTA using committees of spanning trees, and then aggregating predictions via a majority vote. The resulting algorithms are denoted by k*SHAZOO and k*WTA, where k is the number of spanning trees in the aggregation. We used either $k = 7, 11$ or $k = 3, 7$, depending on the dataset size.

For our experiments, we used five datasets: RCV1, USPS, KROGAN, COMBINED, and WEBSPAM. WEBSPAM is a big dataset (110,900 nodes and 1,836,136 edges) of inter-host links created for the Web Spam Challenge 2008 [15].[3] KROGAN (2,169 nodes and 6,102 edges) and COMBINED (2,871 nodes and 6,407 edges) are high-throughput protein-protein interaction networks of budding yeast taken from [14] —see [6] for a more complete description. Finally, USPS and RCV1 are graphs obtained from the USPS handwritten characters dataset (all ten categories) and the first 10,000 documents in chronological order of Reuters Corpus Vol. 1 (the four most frequent categories), respectively. In both cases, we used Euclidean 10-Nearest Neighbor to create edges, each

[3] We do not compare our results to those obtained within the challenge since we are only exploiting the graph (weighted) topology here, disregarding content features.

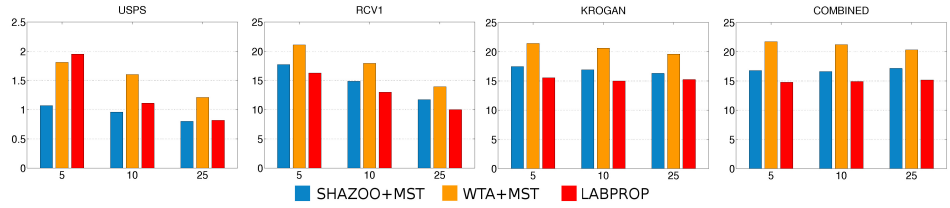
weight $w_{i,j}$ being equal to $e^{-\|x_i - x_j\|^2/\sigma_{i,j}^2}$. We set $\sigma_{i,j}^2 = \frac{1}{2}(\sigma_i^2 + \sigma_j^2)$, where σ_i^2 is the average squared distance between i and its 10 nearest neighbours.

Following previous experimental settings [6], we associate binary classification tasks with the five datasets/graphs via a standard one-vs-all reduction. Each error rate is obtained by averaging over ten randomly chosen training sets (and ten different trees in the case of RST and NWRST). WEBSPAM is natively a binary classification problem, and we used the same train/test split provided with the dataset: 3,897 training nodes and 1,993 test nodes (the remaining nodes being unlabeled).

In the below table, we show the macro-averaged classification error rates (percentages) achieved by the various algorithms on the first four datasets mentioned in the main text. For each dataset we trained ten times over a random subset of 5%, 10% and 25% of the total number of nodes and tested on the remaining ones. In boldface are the lowest error rates on each column, excluding LABPROP which is used as a "yardstick" comparison. Standard deviations averaged over the binary problems are small: most of the times less than 0.5%.

Datasets	USPS			RCV1			KROGAN			COMBINED		
Predictors	5%	10%	25%	5%	10%	25%	5%	10%	25%	5%	10%	25%
SHAZOO+RST	3.62	2.82	2.02	21.72	18.70	15.68	18.11	17.68	17.10	17.77	17.24	17.34
SHAZOO+NWRST	3.88	3.03	2.18	21.97	19.21	15.95	18.11	18.14	17.32	17.22	17.21	17.53
SHAZOO+MST	**1.07**	**0.96**	**0.80**	17.71	14.87	11.73	17.46	16.92	16.30	16.79	16.64	17.15
WTA+RST	5.34	4.23	3.02	25.53	22.66	19.05	21.82	21.05	20.08	21.76	21.38	20.26
WTA+NWRST	5.74	4.45	3.26	25.50	22.70	19.24	21.90	21.28	20.18	21.58	21.42	20.64
WTA+MST	1.81	1.60	1.21	21.07	17.94	13.92	21.41	20.63	19.61	21.74	21.20	20.32
7*SHAZOO+RST	1.68	1.28	0.97	16.33	13.52	11.07	15.54	15.58	15.46	15.12	15.24	15.84
7*SHAZOO+NWRST	1.89	1.38	1.06	16.49	13.98	11.37	15.61	15.62	15.50	15.02	15.12	15.80
7*WTA+RST	2.10	1.56	1.14	17.44	14.74	12.15	16.75	16.64	15.88	16.42	16.09	15.72
7*WTA+NWRST	2.33	1.73	1.24	17.69	15.18	12.53	16.71	16.60	16.00	16.24	16.13	15.79
11*SHAZOO+RST	1.52	1.17	0.89	**15.82**	**13.04**	**10.59**	**15.36**	15.40	**15.29**	14.91	15.06	15.61
11*SHAZOO+NWRST	1.70	1.27	0.98	15.95	13.42	10.93	15.40	**15.33**	15.32	**14.87**	**14.99**	15.67
11*WTA+RST	1.84	1.36	1.01	16.40	13.95	11.42	16.20	16.15	15.53	15.90	15.58	**15.30**
11*WTA+NWRST	2.04	1.51	1.12	16.70	14.28	11.68	16.22	16.05	15.50	15.74	15.57	15.33
OMV	24.79	12.34	2.10	31.65	22.35	11.79	43.13	38.75	29.84	44.72	40.86	33.24
LABPROP	1.95	1.11	0.82	16.28	12.99	10.00	15.56	14.98	15.23	14.79	14.93	15.18

Next, we extract from the above table a specific comparison among SHAZOO, WTA, and LABPROP. SHAZOO and WTA use a single minimum spanning tree (the best performing tree type for both algorithms). Note that SHAZOO consistently outperforms WTA.



We then report the results on WEBSPAM. SHAZOO and WTA use only non-weighted random spanning trees (NWRST) to optimize scalability. Since this dataset is extremely unbalanced (5.4% positive labels) we use the average test set F-measure instead of the error rate.

SHAZOO	WTA	OMV	LABPROP	3*WTA	3*SHAZOO	7*WTA	7*SHAZOO
0.954	0.947	0.706	0.931	0.967	0.964	0.968	0.968

Our empirical results can be briefly summarized as follows:

1. Without using committees, SHAZOO outperforms WTA on all datasets, irrespective to the type of spanning tree being used. With committees, SHAZOO works better than WTA almost always, although the gap between the two reduces.

2. The predictive performance of SHAZOO+MST is comparable to, and sometimes better than, that of LABPROP, though the latter algorithm is slower.

3. k*SHAZOO, with $k = 11$ (or $k = 7$ on WEBSPAM) seems to be especially effective, outperforming LABPROP, with a small (e.g., 5%) training set size.

4. NWRST does not offer the same theoretical guarantees as RST, but it is extremely fast to generate (linear in $|V|$ on most graphs — e.g., [1]), and in our experiments is only slightly inferior to RST.

References

[1] N. Alon, C. Avin, M. Koucký, G. Kozma, Z. Lotker, and M.R. Tuttle. Many random walks are faster than one. In *Proc. 20th Symp. on Parallel Algo. and Architectures*, pages 119–128. Springer, 2008.

[2] M. Belkin, I. Matveeva, and P. Niyogi. Regularization and semi-supervised learning on large graphs. In *Proceedings of the 17th Annual Conference on Learning Theory*, pages 624–638. Springer, 2004.

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

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

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

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

[7] C. Altafini G. Iacono. Monotonicity, frustration, and ordered response: an analysis of the energy landscape of perturbed large-scale biological networks. *BMC Systems Biology*, 4(83), 2010.

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

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

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

[11] F. R. Kschischang, B. J. Frey, and H. A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47(2):498–519, 2001.

[12] R. Lyons and Y. Peres. Probability on trees and networks. Manuscript, 2008.

[13] S. T. McCormick, M. R. Rao, and G. Rinaldi. Easy and difficult objective functions for max cut. *Math. Program.*, 94(2-3):459–466, 2003.

[14] 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 *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 540–549. ACM Press, 2007.

[15] Yahoo! Research and Laboratory of Web Algorithmics University of Milan. Web spam collection. http://barcelona.research.yahoo.net/webspam/datasets/.

[16] D. A. Spielman and N. Srivastava. Graph sparsification by effective resistances. In *Proc. of the 40th annual ACM symposium on Theory of computing (STOC 2008)*. ACM Press, 2008.

[17] D.B. Wilson. Generating random spanning trees more quickly than the cover time. In *Proceedings of the 28th ACM Symposium on the Theory of Computing*, pages 296–303. ACM Press, 1996.

[18] X. Zhu, Z. Ghahramani, and J. Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. In *Proceedings of the 20th International Conference on Machine Learning*, 2003.