Predicting Nearly as Well as the Best Pruning of a Planar Decision Graph

Eiji Takimoto[1]⋆ and Manfred K. Warmuth[2]⋆⋆

[1] Graduate School of Information Sciences, Tohoku University
Sendai, 980-8579, Japan.
[2] Computer Science Department, University of California, Santa Cruz
Santa Cruz, CA 95064, U.S.A.

Abstract. We design efficient on-line algorithms that predict nearly as well as the best pruning of a planar decision graph. We assume that the graph has no cycles. As in the previous work on decision trees, we implicitly maintain one weight for each of the prunings (exponentially many). The method works for a large class of algorithms that update its weights multiplicatively. It can also be used to design algorithms that predict nearly as well as the best convex combination of prunings.

1 Introduction

Decision trees are widely used in Machine Learning. Frequently a large tree is produced initially and then this tree is pruned for the purpose of obtaining a better predictor. A pruning is produced by deleting some nodes and with them all their successors. Although there are exponentially many prunings, a recent method developed in coding theory [WST95] and machine learning [Bun92] makes it possible to (implicitly) maintain one weight per pruning. In particular Helmbold and Schapire [HS97] use this method to design an elegant algorithm that is guaranteed to predict nearly as well as the best pruning of a decision tree. Pereira and Singer [PS97] modify this algorithm to the case of edge-based prunings instead of the node-based prunings defined above. Edge-based prunings are produced by cutting some edges of the original decision tree and then removing all nodes below the cuts. Both definitions are closely related. Edge-based prunings have been applied to statistical language modeling [PS97], where the out-degree of nodes in the tree may be very large.

In this paper we generalize the methods from decision trees to planar directed acyclic graphs (dags). Trees, upside-down trees and series-parallel dags are all special cases of planar dags. We define a notion of edge-based prunings of a planar dag. Again we find a way to efficiently maintain one weight for each of the exponentially many prunings.

In Fig. 1, the tree $\mathcal{T}'$ represents a node-based pruning of the decision tree $\mathcal{T}$. Each node in the original tree $\mathcal{T}$ is assumed to have a prediction value in

⋆ This work was done while the author visited University of California, Santa Cruz.
⋆⋆ Supported by NSF grant CCR 9700201

O. Watanabe, T. Yokomori (Eds.): ALT'99, LNAI 1720, pp. 335–346, 1999.

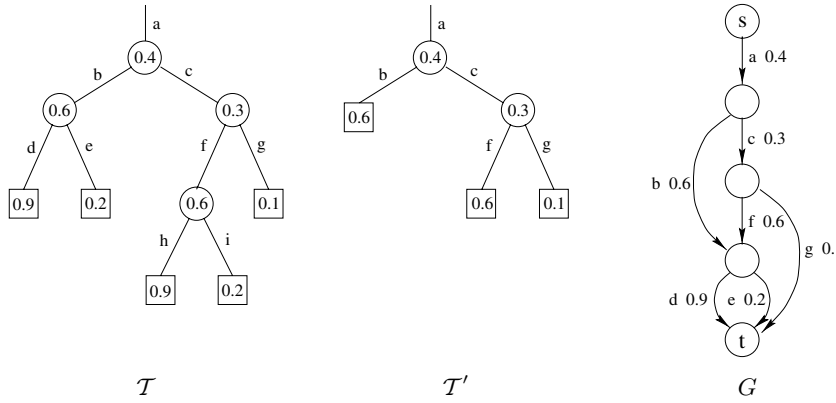

Fig. 1. An example of a decision tree, a pruning, and a decision dag

some prediction space $\hat{Y}$. Here, we assume $\hat{Y} = [0, 1]$. In the usual setting each instance (from some instance space) induces a path in the decision tree from the root to a leaf. The path is based on decisions done at the internal nodes. Thus w.l.o.g. our instances are paths in the original decision tree. For a given path, a tree predicts the value at the leaf at which the path ends. For example, for the path $\{a, c, f, i\}$, the original tree $\mathcal{T}$ predicts the value 0.2 and the pruning $\mathcal{T}'$ predicts 0.6.

In what follows, we consider the prediction values to be associated with the edges. In Fig. 1 the prediction value of each edge is given at its lower endpoint. For example, the edges a, b and c have prediction values 0.4, 0.6 and 0.3, respectively. Moreover, we think of a pruning as the set of edges that are incident to the leaves of the pruning. So, $\mathcal{T}$ and $\mathcal{T}'$ are represented by $\{d, e, g, h, i\}$ and $\{b, f, g\}$, respectively. Note that for any pruning R and any path P, R intersects P at exactly one edge. That is, a pruning "cuts" each path at an edge. The pruning R predicts on path P with the prediction value of the edge that is cut.

The notion of pruning can easily be generalized to directed acyclic graphs. We define *decision dags* as dags with a special source and sink node where each edge is assumed to have a prediction value. A pruning R of the decision dag is defined as a set of edges such that for any `s-t` path P, R intersects P with exactly one edge. Again the pruning R predicts on the instance/path P with the value of the edge that is cut. It is easily seen that the rightmost graph G in Fig. 1 is a decision dag that is equivalent to $\mathcal{T}$.

We study learning in the on-line prediction model where the decision dag is given to the learner. At each trial $t = 1, 2, \ldots$, the learner receives a path P_t and must produce a prediction $\hat{y}_t \in \hat{Y}$. Then an outcome y_t in the outcome space Y is observed (which can be thought of as the correct value of P_t). Finally, at the end of the trial the learner suffers loss $L(y_t, \hat{y}_t)$, where $L : Y \times \hat{Y} \to [0, \infty]$ is a fixed loss function. Since each pruning R has a prediction value for P_t, the loss of R at this trial is defined analogously. The goal of the learner is to make predictions so that its total loss $\sum_t L(y_t, \hat{y}_t)$ is not much worse than the total

loss of the best pruning or that of the best mixture (convex combination) of prunings.

It is straightforward to apply any of a large family of on-line prediction algorithms to our problem. To this end, we just consider each pruning R of G as an *expert* that predicts as the pruning R does on all paths. Then, we can make use of any on-line algorithm that maintains one weight per expert and forms its own prediction $\hat{y}_t$ by combining the predictions of the experts using the current weights (See for example: [Lit88, LW94, Vov90, Vov95, CBFH$^+$97, KW97]). Many relative loss bounds have been proven for this setting bounding the additional total loss of the algorithm over the total loss of the best expert or best weighted combination of experts. However, this "direct" implementation of the on-line algorithms is inefficient because one weight/expert would be used for each pruning of the decision dag and the number of prunings is usually exponentially large. So the goal is to find efficient implementations of the direct algorithms so that the exponentially many weights are implicitly maintained. In the case of trees this is possible [HS97]. Other applications that simulate algorithms with exponentially many weights are given in [HPW99, MW98]. We now sketch how this can be done when the decision dag is planar.

Recall that each pruning and path intersect at one edge. Therefore in each trial the edges on the path determine the predictions of all the prunings as well as their losses. So in each trial the edges on the path also incur a loss and the total loss of a pruning is always the sum of the total losses of all of its edges. Under a very general setting the weight of a pruning is then a function of the weights of its edges. Thus the exponentially many weights of the prunings collapse to one weight per edge. It is obvious that if we can efficiently update the edge weights and compute the prediction of the direct algorithm from the edge weights, then we have an efficient algorithm that behaves exactly as the direct algorithm.

One of the most important family of on-line learning algorithms is the one that does multiplicative updates of its weights. For this family the weight w_R of a pruning R is always the product of the weights of its edges, i.e. $w_R = \prod_{e \in R} v_e$. The most important computation in determining the prediction and updating the weights is summing the current weights of all the prunings, i.e. $\sum_R \prod_{e \in R} v_e$. We do not know how to efficiently compute this sum for arbitrary decision dags. However, for planar decision dags, computing this sum is reduced to computing another sum for the dual planar dag. The prunings in the primal graph correspond to paths in the dual, and paths in the primal to prunings in the dual. Therefore the above sum is equivalent to $\sum_P \prod_{e \in P} v_e$, where P ranges over all paths in the dual dag. Curiously enough, the same formula appears as the likelihood of a sequence of symbols in a Hidden Markov Model where the edge weights are the transition probabilities. So we can use the well known forward-backward algorithm for computing the above formula efficiently [LRS83].

The overall time per trial is linear in the number of edges of the decision dag. For the case where the dag is series-parallel, we can improve the time per trial to grow linearly in the size of the instance (a path in the dag).

Another approach for solving the on-line pruning problem is to use the *specialist* framework developed by Freund, Schapire, Singer and Warmuth [FSSW97]. Now each edge is considered to be a specialist. In trial t only the edges on the path "awake" and all others are "asleep". The predictions of the awake edges are combined to form the prediction of the algorithm. The redeeming feature of their algorithm is that it works for arbitrary sets of prunings and paths over some set of edges with the property that any pruning and any path intersect at exactly one edge. They can show that their algorithm performs nearly as well as any mixture of specialists, that is, essentially as well as the best single pruning.

However, even in the case of decision trees the loss bound of their algorithm is quadratic in the size of the pruning. In contrast, the loss bound for the direct algorithm grows only linearly in the size of the pruning. Also when we use for example the EG algorithm [KW97] as our direct algorithm, then the direct algorithm (as well as its efficient simulation) predicts nearly as well as the best convex combination of prunings.

2 On-Line Pruning of a Decision Dag

A decision dag is a directed acyclic graph $G = (V, E)$ with a designated start node `s` and a terminal node `t`. We call `s` and `t` the source and the sink of G, respectively. An `s-t` path is a set of edges of G that forms a path from the source to the sink. In the decision dag G, each edge $e \in E$ is assumed to have a predictor that, when given an instance (`s-t` path) that includes the edge e, makes a prediction from the *prediction space* $\hat{Y}$. In a typical setting, the predictions would be real numbers from $\hat{Y} = [0, 1]$. Although the predictor at edge e may make different predictions whenever the path passes through e, we write its prediction as $\xi(e) \in \hat{Y}$.

A *pruning* R of G is a set of edges such that for any `s-t` path P, R intersects P with exactly one edge, i.e., $|R \cap P| = 1$. Let $e_{R \cap P}$ denote the edge at which R and P intersect. Because of the intersection property, a pruning R can be thought of as a well-defined function from any instance P to a prediction $\xi(e_{R \cap P}) \in \hat{Y}$. Let $\mathcal{P}(G)$ and $\mathcal{R}(G)$ denote the set of all paths and all prunings of G, respectively. For example, the decision dag G in Fig. 1 has four prunings, i.e., $\mathcal{R}(G) = \{\{a\}, \{b, c\}, \{b, f, g\}, \{d, e, g\}\}$. Assume that we are given an instance $P = \{a, b, e\}$. Then, the pruning $R = \{b, f, g\}$ predicts 0.6 for this instance P, which is the prediction of the predictor at edge $b = e_{R \cap P}$.

We study learning in the on-line prediction model, where an algorithm is required not to actually produce prunings but to make predictions for a given instance sequence based on a given decision dag G. The goal is to make predictions that are competitive with those made by the best pruning of G or with those by the best mixture of prunings of G. We will now state our learning model more precisely. A prediction algorithm A is given a decision dag G as its input. At each trial $t = 1, 2, \ldots$, algorithm A receives an instance/path $P_t \in \mathcal{P}(G)$ and generates a prediction $\hat{y}_t \in \hat{Y}$. After that, an outcome $y_t \in Y$ is observed. Y is a set called the *outcome space*. Typically, the outcome space Y would be the same

as $\hat{Y}$. At this trial, the algorithm A suffers loss $L(y_t, \hat{y}_t)$, where $L : Y \times \hat{Y} \to [0, \infty]$ is a fixed *loss function*. For example the square loss is $L(y, \hat{y}) = (y - \hat{y})^2$ and the relative-entropic loss is given by $L(y, \hat{y}) = y \ln(y/\hat{y}) + (1 - y) \ln((1 - y)/(1 - \hat{y}))$. For any instance-outcome sequence $S = ((P_1, y_1), \ldots, (P_T, y_T)) \in (\mathcal{P}(G) \times Y)^*$, the cumulative loss of A is defined as $L_A(S) = \sum_{t=1}^{T} L(y_t, \hat{y}_t)$. In what follows, the cumulative loss of A is simply called the loss of A. Similarly, for a pruning R of G, the loss of R for S is defined as

$$L_R(S) = \sum_{t=1}^{T} L(y_t, \xi(e_{R \cap P_t})) \ .$$

The performance of A is measured in two ways. The first one is to compare the loss of A to the loss of the best R. In other words, the goal of algorithm A is to make predictions so that its loss $L_A(S)$ is close to $\min_{R \in \mathcal{R}(G)} L_R(S)$. The other goal (that is harder to achieve) is to compare the loss of A to the loss of the best mixture of prunings. To be more precise, we introduce a mixture vector $\boldsymbol{u}$ indexed by R so that $u_R \geq 0$ for $R \in \mathcal{R}(G)$ and $\sum_R u_R = 1$. Then the goal of A is to achieve a loss $L_A(S)$ that is close to $\min_{\boldsymbol{u}} L_{\boldsymbol{u}}(S)$, where

$$L_{\boldsymbol{u}}(S) = \sum_{t=1}^{T} L(y_t, \sum_{R \in \mathcal{R}(G)} u_R \xi(e_{R \cap P_t})) \ .$$

Note that the former goal can be seen as the special case of the latter one where the mixture vector $\boldsymbol{u}$ is restricted to unit vectors (i.e., $u_R = 1$ for some particular R).

3 Dual Problem for a Planar Decision Dag

In this section, we show that our problem of on-line pruning has an equivalent dual problem provided that the underlying graph G is planar. The duality will be used to make our algorithms efficient. An `s-t` cut of G is a minimal set of edges of G such that its removal from G results in a graph where `s` and `t` are disconnected. First we point out that a pruning of G is an `s-t` cut of G as well. The converse is not necessarily true. For instance, the set $\{a, e, f\}$ is an `s-t` cut of G in Fig. 2 but it is not a pruning because a path $\{a, d, e\}$ intersects the cut with more than 1 edge. So, the set of prunings $\mathcal{R}(G)$ is a subset of all `s-t` cuts of G, and our problem can be seen as an on-line min-cut problem where cuts are restricted in $\mathcal{R}(G)$. To see this, let us consider the cumulative loss $\ell_e = \sum_{t: e \in P_t} L(y_t, \xi(e))$ at edge e as the capacity of e. Then, the loss of a pruning R, $L_R(S) = \sum_t L(y_t, \xi(e_{R \cap P_t})) = \sum_{e \in R} \ell_e$, can be interpreted as the total capacity of the cut R. This implies that a pruning of minimum loss is a minimum capacity cut from $\mathcal{R}(G)$.

It is known in the literature that the (unrestricted) min-cut problem for an `s-t` planar graph can be reduced to the shortest path problem for its dual graph (see, e.g., [Hu69, Law70, Has81]). A slight modification of the reduction gives

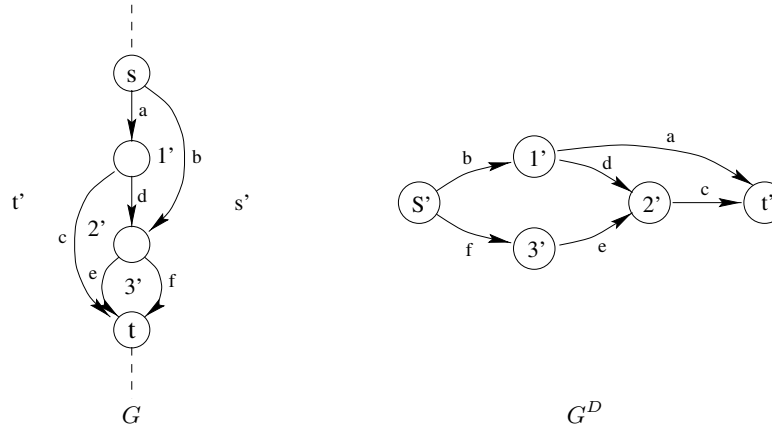

Fig. 2. A decision dag G and its dual dag G^D.

us a dual problem for the best pruning (restricted min-cut) problem. Below we show how to construct the dual dag G^D from a planar decision dag G that is suitable for our purpose.

Assume we have a planar decision dag $G = (V, E)$ with source `s` and sink `t`. Since the graph G is acyclic, we have a planar representation of G so that the vertices in V are placed on a vertical line with all edges downward. In this linear representation, the source `s` and the sink `t` are placed on the top and the bottom on the line, respectively (See Fig. 2). The vertical line (the dotted line) bisects the plane and defines two outer faces `s'` and `t'` of G. Let `s'` be the right face. The dual dag $G^D = (V^D, E^D)$ is constructed as follows. The set of vertices V^D consists of all faces of G. Let $e \in E$ be an edge of G which is common to the boundaries of two faces f_r and f_l in G. By virtue of the linear representation, we can let f_r be the "right" face on e and f_l be the "left" face on e. Then, let E^D include the edge $e' = (f_r, f_l)$ directed from f_r to f_l. It is clear that the dual dag G^D is a planar directed acyclic graph with source `s'` and sink `t'`, and the dual of G^D is G. The following proposition is crucial in this paper.

Proposition 1. *Let G be a planar decision dag and G^D be its dual dag. Then, there is a one-to-one correspondence between* `s-t` *paths $\mathcal{P}(G)$ in G and prunings $\mathcal{R}(G^D)$ in G^D, and there is also a one-to-one correspondence between prunings $\mathcal{R}(G)$ in G and* `s'-t'` *paths $\mathcal{P}(G^D)$ in G^D.*

Thus there is a natural dual problem associated with the on-line pruning problem. We now describe this dual on-line shortest path problem. An algorithm A is given as input a decision dag G. At each trial $t = 1, 2, \ldots$, algorithm A receives a pruning $R_t \in \mathcal{R}(G)$ as the instance and generates a prediction $\hat{y}_t \in \hat{Y}$. The loss of A, denoted $L_A(S)$, for an instance-outcome sequence $S = ((R_1, y_1), \ldots, (R_T, y_T)) \in (\mathcal{R}(G) \times Y)^*$ is defined as $L_A(S) = \sum_{t=1}^{T} L(y_t, \hat{y}_t)$. The class of predictors which the performance of A is now compared to consists of all paths. For a path P of G, the loss of P for S is defined as

$$L_P(S) = \sum_{t=1}^{T} L(y_t, \xi(e_{R_t \cap P})) \ .$$

Similarly, for a mixture vector $\boldsymbol{u}$ indexed by P so that $u_P \geq 0$ for $P \in \mathcal{P}(G)$ and $\sum_P u_P = 1$, the loss of the $\boldsymbol{u}$-mixture of paths is defined as

$$L_{\boldsymbol{u}}(S) = \sum_{t=1}^{T} L(y_t, \sum_{P \in \mathcal{P}(G)} u_P \xi(e_{R_t \cap P})) \ .$$

The objective of A is to make the loss as small as the loss of the best path P, i.e., $\min_P L_P(S)$, or the best mixture of paths, i.e., $\min_{\boldsymbol{u}} L_{\boldsymbol{u}}(S)$. It is natural to call this the on-line shortest path problem because if we consider the cumulative loss $\ell_e = \sum_{t:e \in R_t} L(y_t, \xi(e))$ at edge e as the length of e, then the loss of P, $L_P(S) = \sum_{e \in P} \ell_e$, can be interpreted as the total length of P. It is clear from the duality that the on-line pruning problem for a decision dag G is equivalent to the on-line shortest path problem for its dual dag G^D. In what follows, we consider only the on-line shortest path problem.

4 Inefficient Direct Algorithm

In this section, we show the direct implementation of the algorithms for the on-line shortest path problem. Namely, the algorithm considers each path P of G as an expert that makes a prediction $x_{t,P} = \xi(e_{R_t \cap P})$ for a given pruning R_t. Note that this direct implementation would be inefficient because the number of experts (the number of paths in this case) can be exponentially large.

In general, such direct algorithms have the following generic form: They maintain a weight $w_{t,P} \in [0,1]$ for each path $P \in \mathcal{P}(G)$; when given the predictions $x_{t,P} (= \xi(e_{R_t \cap P}))$ of all paths P, they combine these predictions based on the weights to make their own prediction $\hat{y}_t$, and then update the weights after the outcome y_t is observed. In what follows, let $\boldsymbol{w}_t$ and $\boldsymbol{x}_t$ denote the weight and prediction vectors indexed by $P \in \mathcal{P}(G)$, respectively. Let N be the number of experts, i.e., the cardinality of $\mathcal{P}(G)$. More precisely, the generic algorithm consists of two parts:

- a prediction function $\texttt{pred} : \bigcup_N \left([0,1]^N \times \hat{Y}^N \right) \to \hat{Y}$ which maps the current weight and prediction vectors $(\boldsymbol{w}_t, \boldsymbol{x}_t)$ of experts to a prediction $\hat{y}_t$; and
- an update function $\texttt{update} : \bigcup_N \left([0,1]^N \times \hat{Y}^N \times Y \right) \to [0,1]^N$ which maps $(\boldsymbol{w}_t, \boldsymbol{x}_t)$ and outcome y_t to a new weight vector $\boldsymbol{w}_{t+1}$.

Using these two functions, the generic on-line algorithm behaves as follows: For each trial $t = 1, 2, \ldots,$

1. Observe predictions $\boldsymbol{x}_t$ from the experts.
2. Predict $\hat{y}_t = \texttt{pred}(\boldsymbol{w}_t, \boldsymbol{x}_t)$.

3. Observe outcome y_t and suffer loss $L(y_t, \hat{y}_t)$.
4. Calculate the new weight vector according to $\boldsymbol{w}_{t+1} = \texttt{update}(\boldsymbol{w}_t, \boldsymbol{x}_t, y_t)$.

Vovk's Aggregating Algorithm (AA) [Vov90] is a seminal on-line algorithm of generic form and has the best possible loss bound for a very wide class of loss functions. It updates its weights as $\boldsymbol{w}_{t+1} = \texttt{update}(\boldsymbol{w}_t, \boldsymbol{x}_t, y_t)$, where

$$w_{t+1,P} = w_{t,P} \exp(-L(y_t, x_{t,P})/c_L)$$

for any $P \in \mathcal{P}(G)$. Here c_L is a constant that depends on the loss function L. Since AA uses a complicated prediction function, we only discuss a simplified algorithm called the Weighted Average Algorithm (WAA) [KW99]. The latter algorithm uses the same updates with a slightly worse constant c_L and predicts with the weighted average based on the normalized weights:

$$\hat{y}_t = \texttt{pred}(\boldsymbol{w}_t, \boldsymbol{x}_t) = \sum_{P \in \mathcal{P}(G)} \bar{w}_{t,P} x_{t,P}, \text{ where } \bar{w}_{t,P} = w_{t,P} / \sum_{P'} w_{t,P'}.$$

The following theorem gives an upper bound on the loss of the WAA in terms of the loss of the best path.

Theorem 1 ([KW99]). *Assume $Y = \hat{Y} = [0,1]$. Let the loss function L be monotone convex and twice differentiable with respect to the second argument. Then, for any instance-outcome sequence $S \in (\mathcal{R}(G) \times Y)^*$,*

$$L_{\mathrm{WAA}}(S) \leq \min_{P \in \mathcal{P}(G)} \left\{ L_P(S) + c_L \ln(1/\bar{w}_{1,P}) \right\},$$

where $\bar{w}_{1,P}$ is the normalized initial weight of P.

We can obtain a more powerful bound in terms of the loss of the best mixture of paths using the exponentiated gradient (EG) algorithm due to Kivinen and Warmuth [KW97]. The EG algorithm uses the same prediction function `pred` as the WAA and uses the update function $\boldsymbol{w}_{t+1} = \texttt{update}(\boldsymbol{w}_t, \boldsymbol{x}_t, y_t)$ so that for any $P \in \mathcal{P}(G)$,

$$w_{t+1,P} = w_{t,P} \exp\left(-\eta x_{t,P} \frac{\partial L(y_t, z)}{\partial z} \bigg|_{z=\hat{y}_t} \right) .$$

Here η is a positive learning rate. Kivinen and Warmuth show the following loss bound of the EG algorithm for the square loss function $L(y, \hat{y}) = (y - \hat{y})^2$. Note that, for the square loss, the update above becomes $w_{t+1,P} = w_{t,P} \exp(-2\eta(\hat{y}_t - y_t)x_{t,P})$.

Theorem 2 ([KW97]). *Assume $Y = \hat{Y} = [0,1]$. Let L be the square loss function. Then, for any instance-outcome sequence $S \in (\mathcal{R}(G) \times Y)^*$ and for any probability vector $\boldsymbol{u} \in [0,1]^N$ indexed by P,*

$$L_{\mathrm{EG}}(S) \leq \frac{2}{2-\eta} L_{\boldsymbol{u}}(S) + \frac{1}{\eta} \mathbf{RE}(\boldsymbol{u} || \bar{\boldsymbol{w}}_1),$$

where $\mathbf{RE}(\boldsymbol{u}||\bar{\boldsymbol{w}}_1) = \sum_P u_P \ln(u_P/\bar{w}_{1,P})$ is the relative entropy between $\boldsymbol{u}$ and the initial normalized weight vector $\bar{\boldsymbol{w}}_1$.

Now we give the two conditions on the direct algorithms that are required for our efficient implementation.

Definition 1. *Let* $\boldsymbol{w} \in [0,1]^N$ *and* $\boldsymbol{x} \in \hat{Y}^N$ *be a weight and a prediction vector. Let* $\mathcal{P}_1 \cup \cdots \cup \mathcal{P}_k = \mathcal{P}(G)$ *be a partition of* $\mathcal{P}(G)$ *such that* $\mathcal{P}_i \cap \mathcal{P}_j = \emptyset$ *for any* $i \neq j$ *and all paths in the same class have the same prediction. That is, for each class* $\mathcal{P}_i$*, there exists* $x'_i \in \hat{Y}$ *such that* $x_P = x'_i$ *for any* $P \in \mathcal{P}_i$*. In other words,* $\boldsymbol{x}' = (x'_1, \ldots, x'_k)$ *and* $\boldsymbol{w}' = (w'_1, \ldots, w'_k)$*, where* $w'_i = \sum_{P \in \mathcal{P}_i} w_P$*, can be seen as a projection of the original prediction vector* $\boldsymbol{x}$ *and weight vector* $\boldsymbol{w}$ *onto the partition* $\{\mathcal{P}_1, \ldots, \mathcal{P}_k\}$*. The prediction function* `pred` *is* projection-preserving *if* $\texttt{pred}(\boldsymbol{w}, \boldsymbol{x}) = \texttt{pred}(\boldsymbol{w}', \boldsymbol{x}')$ *for any* $\boldsymbol{w}$ *and* $\boldsymbol{x}$*.*

Definition 2. *The update function* `update` *is* multiplicative *if there exists a function* $f : \hat{Y} \times \hat{Y} \times Y$ *such that for any* $\boldsymbol{w} \in [0,1]^N$*,* $\boldsymbol{x} \in \hat{Y}^N$ *and* $y \in Y$*, the new weight* $\boldsymbol{w}' = \texttt{update}(\boldsymbol{w}, \boldsymbol{x}, y)$ *is given by* $w'_P = w_P f(x_P, \hat{y}, y)$ *for any* P*, where* $\hat{y} = \texttt{pred}(\boldsymbol{w}, \boldsymbol{x})$*.*

These conditions are natural. In fact, they are actually satisfied by the prediction and update functions used in many families of algorithms such as AA [Vov90], WAA [KW99], EG and EGU [KW97]. Note that the projection used may change from trial to trial.

5 Efficient Implementation of the Direct Algorithm

Now we give an efficient implementation of a direct algorithm that consists of a projection-preserving prediction function `pred` and a multiplicative update function `update`. Clearly, it is sufficient to show how to efficiently compute the functions `pred` and `update`. Obviously, we cannot explicitly maintain all of the weights $w_{t,P}$ as the direct algorithm does since there may be exponentially many paths P in G. Instead, we maintain a weight $v_{t,e}$ for each edge e, which requires only a linear space. We will give indirect algorithms below for computing `pred` and `update` so that, the weights $v_{t,e}$ for edges implicitly represent the weights $w_{t,P}$ for all paths P as follows:

$$w_{t,P} = \prod_{e \in P} v_{t,e} \ . \tag{1}$$

First we show an indirect algorithm for `update` which is simpler. Suppose that, given a pruning R_t as instance, we have already calculated a prediction $\hat{y}_t$ and observe an outcome y_t. Then, the following update for the weight of the edges is equivalent to the update $\boldsymbol{w}_{t+1} = \texttt{update}(\boldsymbol{w}_t, \boldsymbol{x}_t, y_t)$ of the weights of the paths. Recall that $\boldsymbol{w}_t$ is a weight vector indexed by P given by (1) and $\boldsymbol{x}_t$ is a prediction vector given by $x_{t,P} = \xi(e_{R_t \cap P})$. Let f be the function associated with our multiplicative update (see Definition 2). For any edge $e \in E$, the weight of e is updated according to

$$v_{t+1,e} = \begin{cases} v_{t,e} f(\xi(e), \hat{y}_t, y_t) & \text{if } e \in R_t, \\ v_{t,e} & \text{otherwise.} \end{cases} \tag{2}$$

Lemma 1. *The update rule for edges given by (2) is equivalent to* $\texttt{update}(\boldsymbol{w}_t, \boldsymbol{x}_t, y_t)$.

Proof. It suffices to show that the relation (1) is preserved after updating. That is, $w_{t+1,P} = \prod_{e \in P} v_{t+1,e}$ for any $P \in \mathcal{P}(G)$. Let $e' = e_{R_t \cap P}$. Since `update` is multiplicative, we have

$$w_{t+1,P} = w_{t,P} f(x_{t,P}, \hat{y}_t, y_t) = \prod_{e \in P} v_{t,e} f(\xi(e'), \hat{y}_t, y_t)$$
$$= \left(\prod_{e \in P \setminus \{e'\}} v_{t,e} \right) \left(v_{t,e'} f(\xi(e'), \hat{y}_t, y_t) \right) = \prod_{e \in P} v_{t+1,e},$$

as required. □

Next we show an indirect algorithm for `pred`. Let the given pruning be $R_t = \{e_1, \ldots, e_k\}$. For $1 \le i \le k$, let $\mathcal{P}_i = \{P \in \mathcal{P}(G) \mid e_i \in P\}$. Since $|R_t \cap P| = 1$ for any P, $\mathcal{P}_1 \cup \cdots \cup \mathcal{P}_k = \mathcal{P}(G)$ forms a partition of $\mathcal{P}(G)$ and clearly for any path $P \in \mathcal{P}_i$, we have $x_{t,P} = \xi(e_i)$. So,

$$\boldsymbol{x}' = (\xi(e_1), \ldots, \xi(e_k)) \tag{3}$$

is a projected prediction vector of $\boldsymbol{x}_t$. Therefore, if we have the corresponding projected weight vector $\boldsymbol{w}'$, then by the projection-preserving property of `pred` we can obtain $\hat{y}_t$ by $\texttt{pred}(\boldsymbol{w}', \boldsymbol{x}')$, which equals $\hat{y}_t = \texttt{pred}(\boldsymbol{w}, \boldsymbol{x})$. Now what we have to do is to efficiently compute the projected weights for $1 \le i \le k$:

$$w'_{t,i} = \sum_{P \in \mathcal{P}_i} w_{t,P} = \sum_{P : e_i \in P} w_{t,P} = \sum_{P : e_i \in P} \prod_{e \in P} v_{t,e} \ . \tag{4}$$

Surprisingly, the $\sum\prod$-form formula above is similar to the formula of the likelihood of a sequence of symbols in a Hidden Markov Model (HMM) with a particular state transition (e_i) [LRS83]. Thus we can compute (4) with the forward-backward algorithm. For node $u \in V$, let $\mathcal{P}_{\mathtt{s} \to u}$ and $\mathcal{P}_{u \to \mathtt{t}}$ be the set of paths from `s` to the node u and the set of paths from the node u to `t`, respectively. Define

$$\alpha(u) = \sum_{P \in \mathcal{P}_{\mathtt{s} \to u}} \prod_{e \in P} v_{t,e} \qquad \text{and} \qquad \beta(u) = \sum_{P \in \mathcal{P}_{u \to \mathtt{t}}} \prod_{e \in P} v_{t,e} \ .$$

Suppose that $e_i = (u_1, u_2)$. Then, the set of all paths in $\mathcal{P}(G)$ through e_i is represented as $\{P_1 \cup \{e_i\} \cup P_2 \mid P_1 \in \mathcal{P}_{\mathtt{s} \to u_1}, P_2 \in \mathcal{P}_{u_2 \to \mathtt{t}}\}$, and therefore the formula (4) is given by

$$w'_{t,i} = \alpha(u_1) v_{t,e_i} \beta(u_2) \ . \tag{5}$$

We summarize this result as the following lemma.

Lemma 2. *Let* $\boldsymbol{x}'$ *and* $\boldsymbol{w}'$ *be given by (3) and (5), respectively. Then* $\texttt{pred}(\boldsymbol{w}', \boldsymbol{x}') = \texttt{pred}(\boldsymbol{w}, \boldsymbol{x})$.

The forward-backward algorithm [LRS83] is an algorithm that efficiently computes α and β by dynamic programming as follows: $\alpha(u) = 1$ if $u = \mathtt{s}$ and $\alpha(u) = \sum_{u' \in V:(u',u)\in E} \alpha(u') v_{t,(u',u)}$, otherwise. Similarly, $\beta(u) = 1$ if $u = \mathtt{t}$ and $\beta(u) = \sum_{u' \in V:(u,u')\in E} \beta(u') v_{t,(u,u')}$, otherwise. It is clear that both α and β can be computed in time $O(|E|)$.

6 A More Efficient Algorithm for Series-Parallel Dags

In the case of decision trees, there is a very efficient algorithm with per trial time linear in the size of the instance (a path in the decision tree) [HS97]. We now give an algorithm with the same improved time per trial for series-parallel dags, which include decision trees.

A series-parallel dag $G(\mathtt{s},\mathtt{t})$ with source $\mathtt{s}$ and sink $\mathtt{t}$ is defined recursively as follows: An edge $(\mathtt{s},\mathtt{t})$ is a series-parallel dag; If $G_1(\mathtt{s}_1,\mathtt{t}_1),\ldots,G_k(\mathtt{s}_k,\mathtt{t}_k)$ are disjoint series-parallel dags, then the series connection $G(\mathtt{s},\mathtt{t}) = s(G_1,\ldots,G_k)$ of these dags, where $\mathtt{s} = \mathtt{s}_1$, $\mathtt{t}_i = \mathtt{s}_{i+1}$ for $1 \le i \le k-1$ and $\mathtt{t} = \mathtt{t}_k$, or the parallel connection $G(\mathtt{s},\mathtt{t}) = p(G_1,\ldots,G_k)$ of these dags, where $\mathtt{s} = \mathtt{s}_1 = \cdots = \mathtt{s}_k$ and $\mathtt{t} = \mathtt{t}_1 = \cdots = \mathtt{t}_k$, is a series-parallel dag. Note that a series-parallel dag has a parse tree, where each internal node represents a series or a parallel connection of the dags represented by its child nodes.

It suffices to show that the projected weights (4) can be calculated in time linear in the size of instance/pruning R_t. To do so the algorithm maintains one weight $v_{t,G}$ per one node G of the parse tree so that

$$v_{t,G} = \sum_{P\in\mathcal{P}(G)} \prod_{e\in P} v_{t,e}$$

holds. Note that if G consists of an single edge e, then $v_{t,G} = v_{t,e}$; if $G = s(G_1,\ldots,G_k)$, then $v_{t,G} = \prod_{i=1}^k v_{t,G_i}$; if $G = p(G_1,\ldots,G_k)$, then $v_{t,G} = \sum_{i=1}^k v_{t,G_i}$. Now (4), i.e., $W(G,e) = \sum_{P\in\mathcal{P}(G),e\in P} \prod_{e'\in P} v_{t,e'}$, is recursively computed as

$$W(G,e) = \begin{cases} v_{t,e} & \text{if } G \text{ consists of } e, \\ W(G_i,e) v_{t,G}/v_{t,G_i} & \text{if } G = s(G_1,\ldots,G_k) \text{ and } e \in G_i, \\ W(G_i,e) & \text{if } G = p(G_1,\ldots,G_k) \text{ and } e \in G_i. \end{cases}$$

The weights $v_{t,G}$ are also recursively updated as

$$v_{t+1,G} = \begin{cases} v_{t+1,e} & \text{if } G \text{ consists of } e, \\ v_{t+1,G_i} v_{t,G}/v_{t,G_i} & \text{if } G = s(G_1,\ldots,G_k) \text{ and } R_t \text{ is in } G_i, \\ \sum_{i=1}^k v_{t+1,G_i} & \text{if } G = p(G_1,\ldots,G_k). \end{cases}$$

It is not hard to see that the prediction and the update can be calculated in time linear in the size of R_t.

Note that the dual of a series-parallel dag is also a series-parallel dag that has the same parse tree with the series and the parallel connections exchanged. So we can solve the primal on-line pruning problem using the same parse tree.

Acknowledgments

We would like to thank Hiroshi Nagamochi for calling our attention to the duality of planar graphs.

References

[Bun92] W. Buntine. Learning classification trees. *Statistics and Computing*, 2:63–73, 1992.

[CBFH+97] N. Cesa-Bianchi, Y. Freund, D. Haussler, D. Helmbold, R. Schapire, and M. Warmuth. How to use expert advice. *Journal of the ACM*, 44(3):427–485, 1997.

[FSSW97] Y. Freund, R. Schapire, Y. Singer, and M. Warmuth. Using and combining predictors that specialize. *29th STOC*, 334–343, 1997.

[Has81] R. Hassin. Maximum flow in (s, t) planar networks. *Information Processing Letters*, 13(3):107–107, 1981.

[HPW99] D. Helmbold, S. Panizza, and M. Warmuth. Direct and indirect algorithm for on-line learning of disjunctions. *4th EuroCOLT*, 138–152, 1999.

[HS97] D. Helmbold and R. Schapire. Predicting nearly as well as the best pruning of a decision tree. *Machine Learning*, 27(1):51–68, 1997.

[Hu69] T. Hu. *Integer Programming and Network Flows*. Addison-Wesley, 1969.

[KW97] J. Kivinen and M. Warmuth. Additive versus exponentiated gradient updates for linear prediction. *Information and Computation*, 132(1):1–64, 1997.

[KW99] J. Kivinen and M. Warmuth. Averaging expert prediction. *4th EuroCOLT*, 153–167, 1999.

[Law70] E. Lawler. *Combinatorial Optimization: Network and Matroids*. Hold, Rinehart and Winston, New York, 1970.

[Lit88] N. Littlestone. Learning when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1988.

[LRS83] S. Levinson, L. Rabiner, and M. Sondhi. An introduction to the application of the theory of probabilistic functions of a markov process to automatic speech recognition. *Bell System Technical Journal*, 62(4):1035–1074, 1983.

[LW94] N. Littlestone and M. Warmuth. The weighted majority algorithm. *Information and Computation*, 108(2):212–261, 1994.

[MW98] M. Maass and M. Warmuth. Efficient learning with virtual threshold gates. *Information and Computation*, 141(1):66–83, 1998.

[PS97] F. Pereira and Y. Singer. An efficient extension to mixture techniques for prediction and decision trees. *10th COLT*, 114–121, 1997.

[Vov90] V. Vovk. Aggregating strategies. *3rd COLT*, 371–383, 1990.

[Vov95] V. Vovk. A game of prediction with expert advice. *8th COLT*, 51–60, 1995.

[WST95] F. Willems, Y. Shtarkov, and T. Tjalkens. The context tree weighting method: basic properties. *IEEE Transactions on Information Theory*, 41(3):653–664, 1995.