ELSEVIER Theoretical Computer Science 288 (2002) 217–235

Theoretical Computer Science

www.elsevier.com/locate/tcs

Predicting nearly as well as the best pruning of a planar decision graph

Eiji Takimoto[a,*,1], Manfred K. Warmuth[b,2]

[a]*Graduate School of Information Sciences, Tohoku University, Sendai, 980-8579, Japan*
[b]*Computer Science Department, University of California, Santa Cruz, CA 95064, USA*

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. © 2002 Elsevier Science B.V. All rights reserved.

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. Usually one pruning is produced from the large tree and then this pruning is used for prediction. In this paper, the prediction is based on a weighted combination of all the prunings and the weights may vary whenever a new example is processed.

Although there are exponentially many prunings, a recent method developed in coding theory [23] and machine learning [1] makes it possible to (implicitly) maintain one weight per pruning and predict using a weighted combination of the predictions of all the prunings. In particular, Helmbold and Schapire [6] use this method to design an elegant algorithm that is guaranteed to predict nearly as well as the best pruning

* Corresponding author. Tel.: +81-22-217-7149; fax: +81-22-263-9414.
E-mail address: t2@ecei.tohoku.ac.jp (E. Takimoto).
[1] This work was done while the author visited University of California, Santa Cruz.
[2] Supported by NSF grant CCR 9821087.

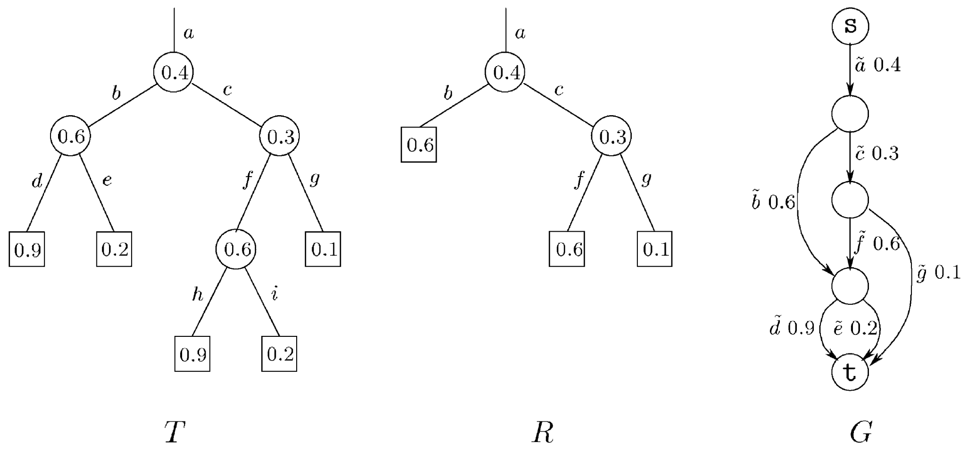

Fig. 1. An example of a decision tree, a pruning, and a decision dag.

of a decision tree. Note again that this algorithm does not settle on a particular pruning even though its predictive performance is compared to the predictive performance of the best pruning. In further work Pereira and Singer [16] modify the algorithm of Helmbold and Schapire to the case of edge-based prunings instead of the node-based prunings defined above.[3] 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 [16], 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). In the case of series-parallel dags our algorithm is more efficient. Such dags are built recursively by using series or parallel compositions. Trees and upside-down trees are special cases of series–parallel 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 R represents a node-based pruning of the decision tree T. Each node in the original tree T is assumed to have a prediction value in 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 T predicts the value 0.2 and the pruning R predicts 0.6.

[3] Takimoto et al. [19] develop another algorithm based on the fact that in the off-line setting the optimal pruning of a decision tree can be efficiently computed by dynamic programming (DP). This DP-based algorithm can be seen as an alternate method of implicitly maintaining one weight per pruning, although it is not explicitly designed to do so (see the appendix of [19]).

In what follows, we consider the prediction values to be associated with the edges. In tree T of 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, the entire tree T (which is also a pruning of T) and the pruning R of 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 $\mathtt{s}$–$\mathtt{t}$ 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 T if the two nodes labeled with 0.6 in T have the same (local) decision rule. The decision dag G has four prunings $\{\tilde{a}\}$, $\{\tilde{b}, \tilde{c}\}$, $\{\tilde{b}, \tilde{f}, \tilde{g}\}$ and $\{\tilde{d}, \tilde{e}, \tilde{g}\}$.

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 for 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, [2,9,13,14,21,22]). 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 [6,19]. Other applications that simulate algorithms with exponentially many weights are given in [5,15]. 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 [12].

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 et al. [3]. Now each edge is considered to be a specialist. In trial t only the edges on the path are “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. However, even in the case of decision trees the loss bound of their algorithm is quadratic in the size of the pruning.

The rest of the paper is organized as follows. In Section 2, we formally give our framework of the on-line prediction that is competitive with the best pruning or the best convex combination of prunings of a given decision dag. In Section 3, we give the dual prediction problem for a planar dag, where the performance of the algorithm is compared to the best path or the best convex combination of paths of the dual decision dag. In Section 4, we give the direct but inefficient implementation of general on-line prediction algorithms and present the performance bounds for the algorithms. In Section 5, we show how to efficiently simulate the direct algorithm for the dual problem. In each trial this indirect algorithm requires time proportional to the number of edges in the dag. In Section 6, we show that for series–parallel dags the time complexity per trial can be improved to be proportional to the size of the instance. Series–parallel dags are a subclass of planar dags that include forests of trees and upside-down trees.

2. On-line pruning of a decision dag

A decision dag is a directed acyclic graph $G = (V, E)$ with a designated start node $\mathtt{s}$ and a terminal node $\mathtt{t}$. We call $\mathtt{s}$ and $\mathtt{t}$ the source and the sink of G, respectively. An $\mathtt{s}$–$\mathtt{t}$ 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 ($\mathtt{s}$–$\mathtt{t}$ 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 $\xi(e)$ (instead of $\xi_t(e)$).

A *pruning* R of G is a set of edges such that for any $\mathtt{s}$–$\mathtt{t}$ 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 $\mathscr{P}(G)$ and $\mathscr{R}(G)$ 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., $\mathscr{R}(G) = \{\{\tilde{a}\}, \{\tilde{b}, \tilde{c}\}, \{\tilde{b}, \tilde{f}, \tilde{g}\}, \{\tilde{d}, \tilde{e}, \tilde{g}\}\}$. Assume that we are given an instance $P = \{\tilde{a}, \tilde{b}, \tilde{e}\}$. Then, the pruning $R = \{\tilde{b}, \tilde{f}, \tilde{g}\}$ predicts 0.6 for this instance P, which is the prediction of the predictor at edge $\tilde{b} = e_{R\cap P}$.

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 now state our learning model more precisely. A learning problem is specified by the prediction space $\hat{Y}$, an *outcome space* Y and a *loss function* $L : Y \times \hat{Y} \to [0, \infty]$. (Typically the outcome space Y would be the same as $\hat{Y}$.) Examples of loss functions are the square loss $L(y, \hat{y}) = (y - \hat{y})^2$ and the relative-entropy loss $L(y, \hat{y}) = y \ln(y/\hat{y}) + (1 - y)\ln((1 - y)/(1 - \hat{y}))$. In our (on-line) learning model a prediction algorithm A proceeds as follows:

```
Input: a decision dag $G$
For each trial $t = 1, 2, \ldots$,
    Receive an instance/path $P_t \in \mathscr{P}(G)$
    Generate a prediction $\hat{y}_t \in \hat{Y}$
    Observe an outcome $y_t \in Y$
    Suffer loss $L(y_t, \hat{y}_t)$
```

For any instance-outcome sequence $S = ((P_1, y_1), \ldots, (P_T, y_T)) \in (\mathscr{P}(G) \times Y)^*$, 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 pruning. To develop a more refined analysis, we define a mixture of losses of prunings. Let $\boldsymbol{u}$ be a probability vector indexed by R so that $u_R \geqslant 0$ for $R \in \mathscr{R}(G)$ and $\sum_R u_R = 1$. Then the goal of A is to make predictions so that its loss $L_A(S)$ is close to $\min_{\boldsymbol{u}} L^{\mathrm{I}}_{\boldsymbol{u}}(S)$, where

$$L^{\mathrm{I}}_{\boldsymbol{u}}(S) = \sum_{R \in \mathscr{R}(G)} u_R L_R(S) = \sum_{t=1}^{T} \sum_{R \in \mathscr{R}(G)} u_R L(y_t, \xi(e_{R \cap P_t})).$$

We call $L^{\mathrm{I}}_{\boldsymbol{u}}(S)$ the $\boldsymbol{u}$-mixture of losses on S. It is clear that $\min_{\boldsymbol{u}} L^{\mathrm{I}}_{\boldsymbol{u}}(S) = \min_{R \in \mathscr{R}(G)} L_R(S)$. So our first goal is to achieve a loss $L_A(S)$ close to the loss of the best R. In the bound given later the mixture vector $\boldsymbol{u}$ will minimize $L^{\mathrm{I}}_{\boldsymbol{u}}(S)$ plus a second term that also depends on $\boldsymbol{u}$. In that case the optimum $\boldsymbol{u}$ will usually not have all its weight concentrated on one pruning R.

The second goal is to achieve the loss of A close to the loss of the best mixture of prunings. That is, the algorithm A tries to make predictions so that its loss $L_A(S)$ is as small as $\min_{\boldsymbol{u}} L^{\mathrm{II}}_{\boldsymbol{u}}(S)$, where

$$L^{\mathrm{II}}_{\boldsymbol{u}}(S) = \sum_{t=1}^{T} L\left(y_t, \sum_{R \in \mathscr{R}(G)} u_R \xi(e_{R \cap P_t}) \right).$$

We call $L^{\mathrm{II}}_{\boldsymbol{u}}(S)$ the loss of the $\boldsymbol{u}$-mixture on S. Assume that the loss function L is convex w.r.t. the second argument. Many typical loss functions such as the square loss and the log-loss are convex. Then since $L^{\mathrm{II}}_{\boldsymbol{u}}(S) \leqslant L^{\mathrm{I}}_{\boldsymbol{u}}(S)$ holds for any $\boldsymbol{u}$, the second goal is harder to achieve than the first goal.

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 $\mathtt{s}$–$\mathtt{t}$ cut of G is a minimal set of edges of G such that its removal from G results in a graph where $\mathtt{s}$ and $\mathtt{t}$ are disconnected. First we point out that a pruning of G is an $\mathtt{s}$–$\mathtt{t}$ cut of G as well. The converse is not necessarily true. For instance, the set $\{a, e, f\}$ is an $\mathtt{s}$–$\mathtt{t}$ cut of G in Fig. 2 but it is not a pruning because a path $\{a, d, e\}$ intersects the cut with more than one edge. So, the set of prunings $\mathscr{R}(G)$ is a subset of all $\mathtt{s}$–$\mathtt{t}$ cuts of G, and our problem can be seen as an on-line min-cut problem where cuts are restricted in $\mathscr{R}(G)$. 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 $\mathscr{R}(G)$.

It is known in the literature that the (unrestricted) min-cut problem for an $\mathtt{s}$–$\mathtt{t}$ planar graph can be reduced to the shortest path problem for its dual graph (see, e.g. [4,7,11]). A slight modification of the reduction gives us a dual problem for the best pruning

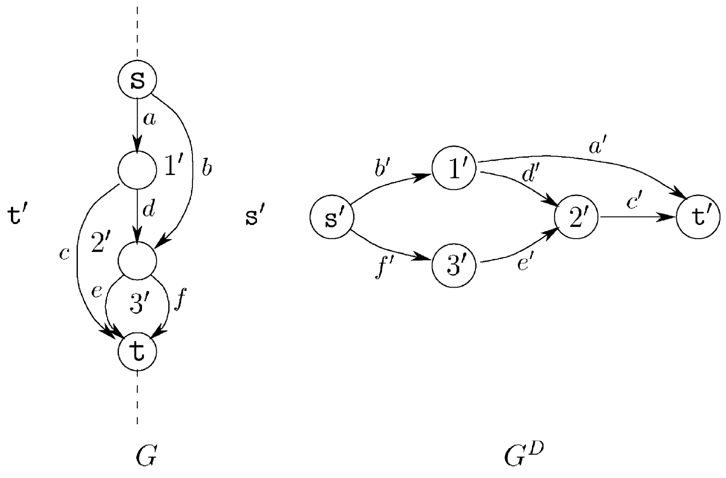

Fig. 2. A decision dag G and its dual dag G^{D}.

(a 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 $\mathtt{s}$ and sink $\mathtt{t}$. Since the graph G is acyclic, we have a planar representation of G so that all edges are directed downward (see e.g. [8]). In this downward representation, the source $\mathtt{s}$ and the sink $\mathtt{t}$ are placed on the top and the bottom in the plane, respectively (see Fig. 2). The vertical line (the dotted line) going through $\mathtt{s}$ and $\mathtt{t}$ bisects the plane and the outer face (the unbounded region in the planar representation) of G is split in two, denoted $\mathtt{s}'$ and $\mathtt{t}'$. Let $\mathtt{s}'$ be the right face. The dual dag $G^{\mathrm{D}}=(V^{\mathrm{D}},E^{\mathrm{D}})$ is constructed as follows. The set of vertices V^{D} consists of all faces (with new outer faces $\mathtt{s}'$ and $\mathtt{t}'$) of G. Let $e\in E$ be an edge of G which is common to the boundaries of the two faces $\mathtt{f}_r$ and $\mathtt{f}_l$ in G. By virtue of the downward representation, we can let $\mathtt{f}_r$ be the "right" face on e and $\mathtt{f}_l$ be the "left" face on e. Then, let E^{D} include the edge $e'=(\mathtt{f}_r,\mathtt{f}_l)$ directed from $\mathtt{f}_r$ to $\mathtt{f}_l$.[4] We will sometimes call e' the edge going across e. For a subset of edges $A\subseteq E$, the dual of A, denoted $A^{\mathrm{D}}\subseteq E^{\mathrm{D}}$, is defined as $A^{\mathrm{D}}=\{e' \text{ going across } e \mid e\in A\}$. It is clear that the dual dag G^{D} is a planar dag with source $\mathtt{s}'$ and sink $\mathtt{t}'$ 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 $\mathtt{s}$–$\mathtt{t}$ paths $\mathscr{P}(G)$ in G and prunings $\mathscr{R}(G^{\mathrm{D}})$ in G^{D}, and there is also a one-to-one correspondence between prunings $\mathscr{R}(G)$ in G and $\mathtt{s}'$–$\mathtt{t}'$ paths $\mathscr{P}(G^{\mathrm{D}})$ in G^{D}.*

[4] In the usual construction the backward edge $(\mathtt{f}_l,\mathtt{f}_r)$ (with 0 capacity) is also included in E^{D}, so that there exists a one-to-one correspondence between cuts in G and paths in G^{D} (see e.g. [11]). In our construction we get rid of the backward edges to eliminate in G the paths that correspond to cuts that are not prunings in G.

Proof. Since the dual of G^{D} is G, it suffices to give a one-to-one correspondence between prunings in G and $\mathtt{s}'$–$\mathtt{t}'$ paths in G^{D}.

First we show that any $\mathtt{s}'$–$\mathtt{t}'$ path P' in G^{D} is the dual of a pruning of G. Let R be such that $R^{\mathrm{D}} = P'$. It suffices to show that R is a pruning of G. Fix an $\mathtt{s}$–$\mathtt{t}$ path P in G arbitrarily. Since the faces $\mathtt{s}'$ and $\mathtt{t}'$ are separated by the path P, P' must go across at least one edge of P. Moreover, since all edges in G^{D} going across P are directed from right to left, P' cannot go across P more than once. These imply that $|R \cap P| = 1$ and so R is a pruning of G.

Next we show that the dual of any pruning R of G is an $\mathtt{s}'$–$\mathtt{t}'$ path in G^{D}. Let $P' = R^{\mathrm{D}}$. Below we will construct an $\mathtt{s}'$–$\mathtt{t}'$ path that uses only the edges of P'. If the constructed path P'' is not P' (that is, $P'' \subsetneq P'$), then by the above argument we would have a pruning R' such that its dual $(R')^{\mathrm{D}}$ is P'' and thus $R' \subsetneq R$, a contradiction. Therefore $P' = R^{\mathrm{D}}$ is the constructed $\mathtt{s}'$–$\mathtt{t}'$ path.

Note that any inner face $\mathtt{f}$ in G is surrounded by two paths with common endpoints. Let Right($\mathtt{f}$) and Left($\mathtt{f}$) denote the right and the left path of them, respectively. Both paths start from the same node, denoted $\mathtt{s}(\mathtt{f})$, and end up to the other same node, denoted $\mathtt{t}(\mathtt{f})$. The outer face $\mathtt{s}'$ has only the left path Left($\mathtt{s}'$). For example, for face $\mathtt{1}'$ in Fig. 2, Right$(\mathtt{1}') = \{b\}$ and Left$(\mathtt{1}') = \{a, d\}$, and for the outer face $\mathtt{s}'$, Left$(\mathtt{s}') = \{b, f\}$. Since Left($\mathtt{s}'$) is an $\mathtt{s}$–$\mathtt{t}$ path in G, $|R \cap \mathrm{Left}(\mathtt{s}')| = 1$. Let e be the edge at which R and Left($\mathtt{s}'$) intersect, and $\mathtt{f}$ be the face such that $e \in \mathrm{Right}(\mathtt{f})$, i.e., the edge e is common to the boundaries of $\mathtt{s}'$ and $\mathtt{f}$. By definition P' contains the edge $e' = (\mathtt{s}', \mathtt{f})$ that goes across e.

Now we have found a (partial) path in G^{D} that is a subset of P' from $\mathtt{s}'$ to $\mathtt{f}$ with e' being the terminal edge. If $\mathtt{f} = \mathtt{t}'$ then we are done. Otherwise since $\mathtt{f}$ is an inner face, we have in G two paths Right($\mathtt{f}$) and Left($\mathtt{f}$) with $e \in \mathrm{Right}(\mathtt{f})$. Fix a path P_1 from $\mathtt{s}$ to $\mathtt{s}(\mathtt{f})$ and a path P_2 from $\mathtt{t}(\mathtt{f})$ to $\mathtt{t}$. By concatenating Right($\mathtt{f}$) with P_1 and P_2, we have an $\mathtt{s}$–$\mathtt{t}$ path. Since R is a pruning of G, we must have $P_1 \cap R = P_2 \cap R = \emptyset$ and Right$(\mathtt{f}) \cap R = \{e\}$. Then consider another $\mathtt{s}$–$\mathtt{t}$ path obtained by concatenating Left($\mathtt{f}$) with P_1 and P_2. Since $P_1 \cap R = P_2 \cap R = \emptyset$, there must exist an edge e_L such that Left$(\mathtt{f}) \cap R = \{e_L\}$. Let e'_L be the edge in G^{D} that goes across e_L and $\mathtt{f}_L$ be the face such that $e'_L = (\mathtt{f}, \mathtt{f}_L)$. Since P' contains e'_L, we extend the partial path in G^{D} to $\mathtt{f}_L$. Repeating the above procedure with $\mathtt{f} = \mathtt{f}_L$ and $e = e_L$ until $\mathtt{f}_L = \mathtt{t}'$, we can find a path from $\mathtt{s}'$ to $\mathtt{t}'$ that consists of the edges of P'. □

Thus there is a natural dual problem associated with the on-line pruning problem. We now describe this dual on-line shortest path problem as follows:

```
Input: a decision dag $G$
For each trial $t = 1, 2, \ldots$,
 Receive an instance/pruning $R_t \in \mathcal{R}(G)$
 Generate a prediction $\hat{y}_t \in \hat{Y}$
 Observe an outcome $y_t \in Y$
 Suffer loss $L(y_t, \hat{y}_t)$
```

The loss of algorithm 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 to which the performance of A is now compared 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 $u_P\geqslant 0$ for $P\in\mathscr{P}(G)$ and $\sum_P u_P=1$, the $\boldsymbol{u}$-mixture of losses of paths and the loss of the $\boldsymbol{u}$-mixture of paths are defined as

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

and

$$L_{\boldsymbol{u}}^{\mathrm{II}}(S)=\sum_{t=1}^{T} L\left(y_t,\sum_{P\in\mathscr{P}(G)} u_P\xi(e_{R_t\cap P})\right),$$

respectively. The goal of A is to make the loss as small as the best $\boldsymbol{u}$-mixture of losses of paths, i.e. $\min_{\boldsymbol{u}} L_{\boldsymbol{u}}^{\mathrm{I}}(S)$ or the loss of the best $\boldsymbol{u}$-mixture of paths, i.e. $\min_{\boldsymbol{u}} L_{\boldsymbol{u}}^{\mathrm{II}}(S)$. 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 $P\in\mathscr{P}(G)$; 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 $P\in\mathscr{P}(G)$, respectively. Let N be the number of experts, i.e., the cardinality of $\mathscr{P}(G)$. More precisely, the generic algorithm consists of two parts:

- a prediction function `pred`: $\bigcup_N([0,1]^N\times\hat{Y}^N)\rightarrow\hat{Y}$ 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 `update`: $\bigcup_N([0,1]^N\times\hat{Y}^N\times Y)\rightarrow[0,1]^N$ 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 shown in Fig. 3.

```
Initialize $\boldsymbol{w}_1$
For each trial $t = 1, 2, \ldots$,
    Observe the predictions $\boldsymbol{x}_t$ from the experts/paths
    Predict $\hat{y}_t = \texttt{pred}(\boldsymbol{w}_t, \boldsymbol{x}_t)$
    Observe the outcome $y_t$ and suffer loss $L(y_t, \hat{y}_t)$
    Calculate the new weight vector as $\boldsymbol{w}_{t+1} = \texttt{update}(\boldsymbol{w}_t, \boldsymbol{x}_t, y_t)$
```

Fig. 3. The direct algorithm of the generic form.

Vovk's aggregating algorithm (AA) [21] is a seminal on-line algorithm of the 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

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

for any $P \in \mathscr{P}(G)$. 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) [10]. 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:

$$\hat{y}_t = \texttt{pred}(\boldsymbol{w}_t, \boldsymbol{x}_t) = \sum_{P \in \mathscr{P}(G)} \bar{w}_{t,P} x_{t,P}, \tag{2}$$

where

$$\bar{w}_{t,P} = \frac{w_{t,P}}{\sum_{P' \in \mathscr{P}(G)} w_{t,P'}}.$$

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

Theorem 2 (Kivinen and Warmuth [9]). *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 $S \in (\mathscr{R}(G) \times Y)^*$,*

$$L_{\mathrm{WAA}}(S) \leqslant \min_{\boldsymbol{u}} \left\{ L^{\mathrm{I}}_{\boldsymbol{u}}(S) + c_L \mathbf{RE}(\boldsymbol{u} \| \bar{\boldsymbol{w}}_1) \right\},$$

where $\boldsymbol{u}$ is a probability vector indexed by $P \in \mathscr{P}(G)$ and

$$\mathbf{RE}(\boldsymbol{u} \| \bar{\boldsymbol{w}}_1) = \sum_{P \in \mathscr{P}(G)} u_P \ln(u_P / \bar{w}_{1,P})$$

is the relative entropy between $\boldsymbol{u}$ and the initial normalized weight vector $\bar{\boldsymbol{w}}_1$.

Restricting the mixture vector $\boldsymbol{u}$ to unit vectors ($u_P = 1$ for some P), we have by Theorem 2 that

$$L_{\mathrm{WAA}}(S) \leqslant \min_{P \in \mathscr{P}(G)} \{L_P(S) + c_L \ln(1/\bar{w}_{1,P})\}.$$

Namely, the loss of the WAA is not much worse than the loss of the best path.

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 [9]. 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 $P \in \mathscr{P}(G)$,

$$w_{t+1,P} = w_{t,P} \exp\left(-\eta x_{t,P} \left. \frac{\partial L(y_t, z)}{\partial z} \right|_{z=\hat{y}_t} \right). \tag{3}$$

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 3 (Kivinen and Warmuth [9]). *Assume $Y = \hat{Y} = [0,1]$. Let L be the square loss function. Then, for any instance-outcome sequence $S \in (\mathscr{R}(G) \times Y)^*$,*

$$L_{\mathrm{EG}}(S) \leqslant \min_{\boldsymbol{u}} \left\{ \frac{2}{2-\eta} L^{\mathrm{II}}_{\boldsymbol{u}}(S) + \frac{1}{\eta} \mathbf{RE}(\boldsymbol{u} \| \bar{\boldsymbol{w}}_1) \right\}.$$

5. Efficient implementation of the direct algorithm

First we give the two conditions on the direct algorithm of Fig. 3 that are required for our efficient implementation.

Definition 4. Let $\boldsymbol{w} \in [0,1]^N$ and $\boldsymbol{x} \in \hat{Y}^N$ be a weight and a prediction vector. Let $\mathscr{P}_1 \cup \cdots \cup \mathscr{P}_k = \mathscr{P}(G)$ be a partition of $\mathscr{P}(G)$ such that all paths in the same class have the same prediction. That is, for each class $\mathscr{P}_i$, there exists $x'_i \in \hat{Y}$ such that $x_P = x'_i$ for any $P \in \mathscr{P}_i$. In other words, $\boldsymbol{x}' = (x'_1, \ldots, x'_k)$ and $\boldsymbol{w}' = (w'_1, \ldots, w'_k)$, where $w'_i = \sum_{P \in \mathscr{P}_i} w_P$, can be seen as a projection of the original prediction vector $\boldsymbol{x}$ and weight vector $\boldsymbol{w}$ onto the partition $\{\mathscr{P}_1, \ldots, \mathscr{P}_k\}$. The prediction function pred is *projection-preserving* if $\texttt{pred}(\boldsymbol{w}, \boldsymbol{x}) = \texttt{pred}(\boldsymbol{w}', \boldsymbol{x}')$ for any $\boldsymbol{w}$, $\boldsymbol{x}$ and any partition $\{\mathscr{P}_1, \ldots, \mathscr{P}_k\}$.

Definition 5. The update function update is *multiplicative* if there exists a function $f : \hat{Y} \times \hat{Y} \times Y \to [0,1]$ such that for any $\boldsymbol{w} \in [0,1]^N$, $\boldsymbol{x} \in \hat{Y}^N$ and $y \in Y$, the new weight $\tilde{\boldsymbol{w}} = \texttt{update}(\boldsymbol{w}, \boldsymbol{x}, y)$ is given by $\tilde{w}_P = w_P f(x_P, \hat{y}, y)$ for any P, where $\hat{y} = \texttt{pred}(\boldsymbol{w}, \boldsymbol{x})$.

These conditions are natural and actually satisfied by the prediction and update functions used in many families of algorithms such as AA [21], WAA [10], EG and EGU [9]. In fact, the prediction function given by (2) is clearly projection-preserving, and the update functions given by (1) and (3) are both multiplicative. Note that the projection used may change from trial to trial.

Now we give an efficient implementation of the direct algorithm of Fig. 3 that consists of a projection-preserving prediction function pred and a multiplicative update function update. The latter implies that $\boldsymbol{w}_{t+1} = \mathtt{update}(\boldsymbol{w}_t, \boldsymbol{x}_t, y_t)$, where

$$w_{t+1,P} = w_{t,P} f(x_{t,P}, \hat{y}_t, y_t) \tag{4}$$

for some function f. 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 of G. The weights $v_{t,e}$ for edges implicitly represent the weights $w_{t,P}$ for all paths P as follows:

$$w_{t,P} = \prod_{e \in P} v_{t,e}. \tag{5}$$

We first give an efficient algorithm for updating the weights $v_{t,e}$ at the end of trial t. For any edge $e \in E$, the weight of e is updated according to

$$v_{t+1,e} = \begin{cases} v_{t,e} f(\xi(e), \hat{y}_t, y_t) & \text{if } e \in R_t, \\ v_{t,e} & \text{if } e \notin R_t. \end{cases} \tag{6}$$

Note that only the weights of the edges in the instance R_t are updated. So this indirect update algorithm can be computed in time linear in the size of R_t.

Lemma 6. *Relation* (5) *is preserved by update* (6).

Proof. Fix a path $P \in \mathscr{P}(G)$ arbitrarily. Let $e' = e_{R_t \cap P}$. Then

$$\begin{aligned} w_{t+1,P} &= w_{t,P} f(x_{t,P}, \hat{y}_t, y_t) \qquad \text{by (4)} \\ &= \left(\prod_{e \in P} v_{t,e}\right) f(\xi(e'), \hat{y}_t, y_t) \qquad \text{by (5) and } x_{t,P} = \xi(e') \\ &= \left(\prod_{e \in P \setminus \{e'\}} v_{t,e}\right) (v_{t,e'} f(\xi(e'), \hat{y}_t, y_t)) \\ &= \prod_{e \in P} v_{t+1,e} \qquad \text{by (6)}, \end{aligned}$$

as required. □

Next, we show an indirect algorithm for computing the prediction $\hat{y}_t$. Let the given pruning be $R_t = \{e_1, \ldots, e_l\}$. For $1 \leqslant i \leqslant l$, let $\mathscr{P}_i = \{P \in \mathscr{P}(G) \mid e_i \in P\}$. Since $|R_t \cap P| = 1$ for any P, $\mathscr{P}_1 \cup \cdots \cup \mathscr{P}_l = \mathscr{P}(G)$ forms a partition of $\mathscr{P}(G)$ and clearly for any path $P \in \mathscr{P}_i$, we have $x_{t,P} = \xi(e_i)$. So,

$$\boldsymbol{x}'_t = (\xi(e_1), \ldots, \xi(e_l)) \tag{7}$$

is a projected prediction vector of $\boldsymbol{x}_t$. Therefore, if we have the corresponding projected weight vector $\boldsymbol{w}'_t$, then by the projection-preserving property of pred we can obtain

$\hat{y}_t$ by $\texttt{pred}(\boldsymbol{w}'_t, \boldsymbol{x}'_t)$, which equals $\hat{y}_t = \texttt{pred}(\boldsymbol{w}_t, \boldsymbol{x}_t)$. Now what we have to do is to efficiently compute the projected weights for $1 \leqslant i \leqslant l$:

$$w'_{t,i} = \sum_{P \in \mathscr{P}_i} w_{t,P} = \sum_{P: e_i \in P} w_{t,P} = \sum_{P: e_i \in P} \prod_{e \in P} v_{t,e}. \tag{8}$$

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) [12]. Thus we can compute (8) with the forward–backward algorithm. For node $u \in V$, let $\mathscr{P}_{\mathtt{s} \to u}$ and $\mathscr{P}_{u \to \mathtt{t}}$ be the set of paths from $\mathtt{s}$ to the node u and the set of paths from the node u to $\mathtt{t}$, respectively. Define

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

Suppose that $e_i = (u_1, u_2)$. Then, the set of all paths in $\mathscr{P}(G)$ through e_i is represented as $\{P_1 \cup \{e_i\} \cup P_2 \mid P_1 \in \mathscr{P}_{\mathtt{s} \to u_1}, P_2 \in \mathscr{P}_{u_2 \to \mathtt{t}}\}$, and therefore formula (8) is given by

$$w'_{t,i} = \alpha(u_1) v_{t,e_i} \beta(u_2). \tag{9}$$

We summarize this result as the following lemma.

Lemma 7. *Let $\boldsymbol{x}'_t$ and $\boldsymbol{w}'_t$ be given by* (7) *and* (9), *respectively. Then $(\boldsymbol{x}'_t, \boldsymbol{w}'_t)$ is a projection of $(\boldsymbol{x}_t, \boldsymbol{w}_t)$, and thus $\texttt{pred}(\boldsymbol{w}'_t, \boldsymbol{x}'_t) = \texttt{pred}(\boldsymbol{w}_t, \boldsymbol{x}_t)$.*

The forward–backward algorithm [12] is an algorithm that efficiently computes α and β by dynamic programming as follows:

$$\alpha(u) = \begin{cases} 1 & \text{if } u = \mathtt{s}, \\ \displaystyle\sum_{\substack{u' \in V \\ (u',u) \in E}} \alpha(u') v_{t,(u',u)} & \text{otherwise} \end{cases}$$

and

$$\beta(u) = \begin{cases} 1 & \text{if } u = \mathtt{t}, \\ \displaystyle\sum_{\substack{u' \in V \\ (u,u') \in E}} \beta(u') v_{t,(u,u')} & \text{otherwise.} \end{cases}$$

It is clear that both α and β can be computed in time $\mathrm{O}(|E|)$.

The indirect algorithm is summarized in Fig 4.

For completeness we state the main result in this section as the next theorem.

Theorem 8. *The direct algorithm of Fig.* 3 *is simulated by the indirect algorithm of Fig.* 4 *in* $\mathrm{O}(|E|)$ *time at each trial.*

```
Initialize $v_{1,e}$, for $e \in E$
For each trial $t = 1, 2, \ldots,$
    Receive a pruning $R_t = \{e_1, \ldots, e_l\}$
    Compute $\boldsymbol{x}'_t$ and $\boldsymbol{w}'_t$ as defined in (7) and (9), respectively
    Predict $\hat{y}_t = \mathtt{pred}(\boldsymbol{w}'_t, \boldsymbol{x}'_t)$
    Observe outcome $y_t$ and suffer loss $L(y_t, \hat{y}_t)$
    Update $v_{t,e}$ to $v_{t+1,e}$, for $e \in R_t$ as (6)
```

Fig. 4. The indirect algorithm for planar decision dags.

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) [6,19]. In this section we give an algorithm with the same improved time per trial for series–parallel dags, which are much more general than decision trees. Note that we consider the dual problem, where instances are prunings and the experts are paths in the dual dag of the primal series–parallel dag. Our algorithm works on this dual dag, which turns out to be a series–parallel dag as well.

A *series–parallel dag* $G(\mathtt{s},\mathtt{t})$ with source $\mathtt{s}$ and sink $\mathtt{t}$ is defined recursively as follows:[5] 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 $1 \leqslant i \leqslant k-1$ 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. In Fig. 5, we give an example of a series–parallel dag and its parse tree.

For an edge e of G we define $\mathscr{P}(G,e)$ as the set of all paths in $\mathscr{P}(G)$ that pass through the particular edge e. For a pruning $R = \{e_1,\ldots,e_l\} \in \mathscr{R}(G)$, let $W(G,R)$ denote the projected weight vector induced by R. That is,

$$W(G,R) = (w'_{t,1},\ldots,w'_{t,l}),$$

where

$$w'_{t,i} = \sum_{P \in \mathscr{P}(G,e_i)} w_{t,P} = \sum_{P \in \mathscr{P}(G,e_i)} \prod_{e \in P} v_{t,e}.$$

Here $v_{t,e}$'s are the edge weights which are maintained by the algorithm and updated according to (6) as before. Recall that the forward–backward algorithm computes the projected weight vector $W(G,R)$ in $\mathrm{O}(|E|)$ time. Now we will show that, for a

[5] The definition is based on edge series–parallel multidigraphs (sometimes called two-terminal series–parallel multidigraphs). This is closely related but not the same as vertex series–parallel multidigraphs. See e.g. [20].

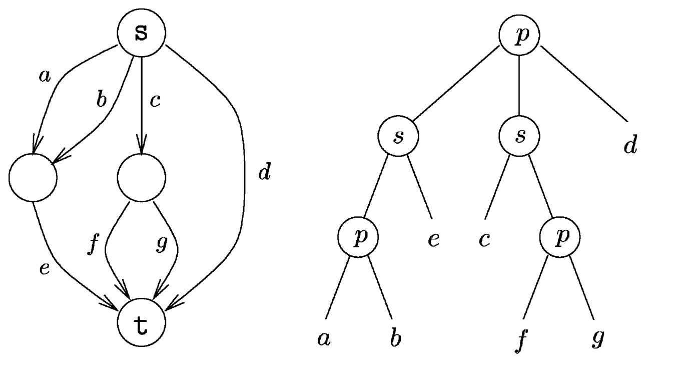

Fig. 5. An example of a series parallel dag and its parse tree.

series–parallel dag G, $W(G,R)$ can be computed in $O(|R|)$ time. To do so the algorithm maintains one weight per node of the parse tree. Note that each node, denoted H, of the parse tree can be identified with a sub-series–parallel dag of G and so we can define $\mathscr{P}(H)$ and $W(H,R)$ as well. The entire dag G is represented by the root of the parse tree. Specifically, the weight of the node H at trial t is defined as

$$v_{t,H} = \sum_{P \in \mathscr{P}(H)} \prod_{e \in P} v_{t,e}. \tag{10}$$

We have to update the weights $v_{t,H}$ for all internal nodes H of the parse tree.

It is easy to see that if H consists of a single edge e, then $\mathscr{P}(H)$ consists of the single path $\{e\}$; if $H = s(H_1,\dots,H_k)$, then $\mathscr{P}(H) = \{P_1 \cup \cdots \cup P_k \mid P_i \in \mathscr{P}(H_i), 1 \leqslant i \leqslant k\}$; if $H = p(H_1,\dots,H_k)$, then $\mathscr{P}(H) = \mathscr{P}(H_1) \cup \cdots \cup \mathscr{P}(H_k)$. From these it follows that

$$v_{t,H} = \begin{cases} v_{t,e} & \text{if } H \text{ consists of a single edge } e, \\ \prod_{i=1}^{k} v_{t,H_i} & \text{if } H = s(H_1,\dots,H_k), \\ \sum_{i=1}^{k} v_{t,H_i} & \text{if } H = p(H_1,\dots,H_k). \end{cases} \tag{11}$$

Moreover, for any pruning $R \in \mathscr{R}(H)$, if H consists of a single edge e, then $R = \{e\}$; if $H = s(H_1,\dots,H_k)$, then $R \in \mathscr{R}(H_i)$ for some $1 \leqslant i \leqslant k$; if $H = p(H_1,\dots,H_k)$, then there exist disjoint sub-prunings $R_1 \in \mathscr{R}(H_1)$, $R_2 \in \mathscr{R}(H_2),\dots,R_k \in \mathscr{R}(H_k)$ such that $R = R_1 \cup R_2 \cup \cdots \cup R_k$.

Now we give an algorithm called `project` that computes $W(H,R)$ (Fig 6.).

Lemma 9. *The algorithm* `project`(H,R) *computes the projected weight vector* $W(H,R)$.

Proof. First we show by an induction on the depth of the parse tree that `project`(H,R) $= W(H,R)$.

```
project($H, R$)
    if $H$ consists of a single edge $e$, then
        return $v_{t,e}$
    if $H = s(H_1, \ldots, H_k)$ and $R \in \mathcal{R}(H_i)$, then
        return project($H_i, R$) $* (v_{t,H}/v_{t,H_i})$
            with the multiplication $*$ computed for every component.
    if $H = p(H_1, \ldots, H_k)$ and $R = R_1 \cup \cdots \cup R_k$
            with $R_i \in \mathcal{R}(H_i)$ for $1 \leqslant i \leqslant k$, then
        return (project($H_1, R_1$), ..., project($H_k, R_k$))
```

Fig. 6. Algorithm project that computes $W(H, R)$.

In the case that H consists of a single edge e (i.e., $R = \{e\}$) it is trivial that project$(H, R) = W(H, R) = v_{t,e}$.

Consider the case that $H = s(H_1, \ldots, H_k)$ and $R = (e_1, \ldots, e_l) \in \mathcal{R}(H_i)$ for some i. By the inductive hypothesis we have

$$\begin{aligned}
\texttt{project}(H,R) &= \texttt{project}(H_i,R) * (v_{t,H}/v_{t,H_i}) \\
&= W(H_i,R) * (v_{t,H}/v_{t,H_i}) \\
&= W(H_i,R) * \left(\prod_{j \neq i} v_{t,H_j}\right) \qquad \text{by (11)} \\
&= (W(H_i,e_1), \ldots, W(H_i,e_l)) * \left(\prod_{j \neq i} v_{t,H_j}\right).
\end{aligned}$$

Since for each $1 \leqslant m \leqslant l$

$$\mathcal{P}(H, e_m) = \{P_1 \cup \cdots \cup P_k \mid P_i \in \mathcal{P}(H_i, e_m),\ P_j \in \mathcal{P}(H_j) \text{ for } j \neq i\},$$

the mth component of project(H, R) is

$$\begin{aligned}
W(H_i, e_m) \prod_{j \neq i} v_{t,H_j} &= \sum_{P \in \mathcal{P}(H_i, e_m)} \prod_{e \in P} v_{t,e} \prod_{j \neq i} \left(\sum_{P \in \mathcal{P}(H_j)} \prod_{e \in P} v_{t,e} \right) \\
&= \sum_{P \in \mathcal{P}(H, e_m)} \prod_{e \in P} v_{t,e} \\
&= W(H, e_m),
\end{aligned}$$

as required.

Finally, consider the case that $H = p(H_1, \ldots, H_k)$ and $R = R_1 \cup \cdots \cup R_k$ with $R_i \in \mathcal{R}(H_i)$ for $1 \leqslant i \leqslant k$. Using the inductive hypothesis we have

$$\begin{aligned}
\texttt{project}(H,R) &= (\texttt{project}(H_1,R_1), \ldots, \texttt{project}(H_k,R_k)) \\
&= (W(H_1,R_1), \ldots, W(H_k,R_k)).
\end{aligned}$$

On the other hand, since $W(H,e)=W(H_i,e)$ for any edge $e\in R_i$, we have

$$W(H,R)=(W(H_1,R_1),\dots,W(H_k,R_k)).$$

Therefore $\texttt{project}(H,R)=W(H,R)$. □

Clearly, when R_t is given at trial t, we get $W(G,R_t)$ by just calling $\texttt{project}(G,R_t)$. However, we need to build a data structure for the parse tree to make $\texttt{project}(G,R_t)$ run in time linear in $|R_t|$. In particular, the data structure helps the $\texttt{project}(H,R)$ find i such that $R_i\in\mathscr{R}(H_i)$ in O(1) time when the node H is labeled with s (i.e., the series composition). The edges of R_t are a subset of the leaves of the parse tree. Clearly these leaves induce a subtree of size $O(|R_t|)$. We extract this induced subtree by marking all nodes in the parse tree that have at least one leaf in common with R_t.

```
mark(H)
 if H is not marked, then
  mark H
  if H is not the root, then
   H' = the parent node of H
   remember that H is a child of H'
   mark(H')
```

Calling `mark(e)` for all $e\in R_t$, we get the subtree induced by the marked nodes. Note that if a node of series composition is marked, then exactly one of its children is marked. Also if a node of parallel composition is marked, then all of its children are marked. It is easy to see that the number of the marked nodes is $O(|R_t|)$. The algorithm `project` now proceeds by traversing only the subtree induced by the marked nodes and spending O(1) time per node. So the overall time to get the projected weight vector $W(G,R_t)$ is $O(|R_t|)$.

Similarly, the weights $v_{t,H}$ can also be recursively updated in time $O(|R_t|)$. For unmarked nodes H, the weights do not need to be changed, i.e. $v_{t+1,H}=v_{t,H}$. For the marked nodes, the weights are updated as follows.

$$v_{t+1,H}=\begin{cases} v_{t+1,e} & \text{if } H \text{ consists of } e,\\ v_{t+1,H_i}v_{t,H}/v_{t,H_i} & \text{if } H=s(H_1,\dots,H_k) \text{ and } H_i \text{ is marked},\\ \sum_{i=1}^k v_{t+1,H_i} & \text{if } H=p(H_1,\dots,H_k),\end{cases}$$

where $v_{t+1,e}$ is given by (6). Thus we have the following theorem.

Theorem 10. *In the case when the decision dag is series–parallel, the indirect algorithm of Fig.* 4 *can be implemented in time* $O(|R_t|)$ *for each trial* t, *where* R_t *is the instance/pruning of trial* 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.

7. Concluding remarks

We gave algorithms that efficiently simulate a large family of on-line prediction algorithms that implicitly maintain one weight per pruning and predict nearly as well as the best pruning or the best convex combination of prunings of a given planar dag. The previous results were only for the case when the dag is a decision tree. The dual of the on-line pruning problem for planar dags is an on-line shortest path problem for planar dags. Our algorithms actually work for this dual problem. Interestingly, we do not need the planarity of the dags when applying our algorithms to the dual problem.

Our algorithms require a planar dag as input, which is supposed to be generated by another algorithm as the first phase. There are a number of algorithms for constructing decision trees such as the C4.5 algorithm [18]. However, we do not know an algorithm for producing good planar dags or good series–parallel dags. One solution to this would be to use a "hard-wired" planar dag that has a fixed structure (see e.g. [17]).

Acknowledgements

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

References

[1] W. Buntine, Learning classification trees, Statist. Comput. 2 (1992) 63–73.
[2] N. Cesa-Bianchi, Y. Freund, D. Haussler, D.P. Helmbold, R.E. Schapire, M.K. Warmuth, How to use expert advice, J. ACM 44 (3) (1997) 427–485.
[3] Y. Freund, R.E. Schapire, Y. Singer, M.K. Warmuth, Using and combining predictors that specialize, in: Proc. 29th Ann. ACM Symp. Theory of Computing, ACM, New York, 1997, pp. 334–343.
[4] R. Hassin, Maximum flow in (s, t) planar networks, Inform. Process. Lett. 13 (3) (1981) 107–107.
[5] D. Helmbold, S. Panizza, M. Warmuth, Direct and indirect algorithm for on-line learning of disjunctions, 4th EuroCOLT, 1999, pp. 138–152.
[6] D.P. Helmbold, R.E. Schapire, Predicting nearly as well as the best pruning of a decision tree, Mach. Learning 27 (1) (1997) 51–68.
[7] T. Hu, Integer Programming and Network Flows, Addison-Wesley, Reading, MA, 1969.
[8] M. Hutton, A. Lubiew, Upward planar drawing of single source acyclic digraphs, SIAM J. Comput. 25 (2) (1996) 291–311.
[9] J. Kivinen, M.K. Warmuth, Additive versus exponentiated gradient updates for linear prediction, Inform. Comput. 132 (1) (1997) 1–64.
[10] J. Kivinen, M. Warmuth, Averaging expert prediction, 4th EuroCOLT, 1999, pp. 153–167.
[11] E. Lawler, Combinatorial Optimization: Network and Matroids, Hold, Rinehart & Winston, New York, 1970.
[12] S.E. Levinson, L.R. Rabiner, M.M. Sondhi, An introduction to the application of the theory of probabilistic functions of a Markov process to automatic speech recognition, Bell System Tech. J. 62 (4) (1983) 1035–1074.
[13] N. Littlestone, Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm, Mach. Learning 2 (1988) 285–318.
[14] N. Littlestone, M.K. Warmuth, The weighted majority algorithm, Inform. Comput. 108 (2) (1994) 212–261.

[15] M. Maass, M.K. Warmuth, Efficient learning with virtual threshold gates, Inform. Comput. 141 (1) (1998) 66–83.
[16] F. Pereira, Y. Singer, An efficient extension to mixture techniques for prediction and decision trees, in: Proc. 10th Annu. Conf. on Comput. Learning Theory, ACM Press, New York, NY, 1997, pp. 114–121.
[17] J.C. Platt, N. Cristianini, J. Shawe-Taylor, Large margin DAGs for multiclass classification, Adv. Neural Inform. Process. Systems 12 (2000), in preparation.
[18] J.R. Quinlan, C4.5: Programs for Machine Learning, Morgan Kaufmann, Los Altos, CA, 1993.
[19] E. Takimoto, A. Maruoka, V. Vovk, Predicting nearly as well as the best pruning of a decision tree through dynamic programming scheme, Theoret. Comput. Sci., in preparation.
[20] J. Valdes, R.E. Tarjan, E.L. Lawler, The recognition of series parallel digraphs, SIAM J. Comput. 11 (2) (1982) 298–313.
[21] V. Vovk, Aggregating strategies, in: Proc. 3rd Annu. Workshop on Comput. Learning Theory, Morgan Kaufmann, Los Altos, CA, 1990, pp. 371–383.
[22] V.G. Vovk, A game of prediction with expert advice, in: Proc. 8th Annu. Conf. on Comput. Learning Theory, ACM Press, New York, NY, 1995, pp. 51–60.
[23] F. Willems, Y. Shtarkov, T. Tjalkens, The context tree weighting method: basic properties, IEEE Trans. Inform. Theory 41 (3) (1995) 653–664.