Efficient Prediction for Tree Markov Random Fields in a Streaming Model

Mark Herbster **Stephen Pasteris**
Department of Computer Science
University College London
London WC1E 6BT, England, UK
{m.herbster, s.pasteris}@cs.ucl.ac.uk

Fabio Vitale
Department of Computer Science
University of Milan
20135 Milan, Italy
fabio.vitale@unimi.it

Abstract

We consider streaming prediction model for tree Markov Random fields. Given the random field, at any point in time we may perform one of three actions: i) predict a label at a vertex on the tree ii) update by associating a label with a vertex or iii) delete the label at a vertex. Using the standard methodology of belief propagation each such action requires time linear in the size of the tree. We give a method based on an optimal *decomposition tree* that even in the worst case is an exponential speed-up over belief propagation.

1 Introduction

We consider a model for streaming prediction and updating in a Markov Random Field whose topology is identical to a tree T. In our streaming model we receive a potentially unbounded online sequence of vertex-label pairs $\langle (v_1, y_1), \ldots, (v_t, y_t), \ldots \rangle$. Our label set is binary $\{-1, 1\}$, but we also have a quasi-label "$\emptyset$" which when received in a pair $(v_{t'}, \emptyset)$ the action is to delete the association of any previous label in the sequence to the paired vertex $v_{t'}$. This deletion action makes our model suitable for describing a large class of real world classification problems, for example, when label information expires after a certain number of time steps. In the full paper we will treat the case where we combine multiple observations at a vertex, but in this preliminary version each successive observation at a vertex replaces the previous observation at that vertex. We denote at time t the *reduced data sequence* with conflicts and deletions resolved as $\mathcal{D}_t$. Thus, for example, given the sequence $\langle (v_2, 1), (v_3, -1), (v_1, 1), (v_2, -1), (v_3, \emptyset), (v_1, -1) \rangle$, we then have that $\mathcal{D}_5 = \langle (v_1, 1), (v_2, -1) \rangle$ and $\mathcal{D}_6 = \langle (v_1, -1), (v_2, -1) \rangle$. We also define the reduced data sequence at time t when restricted to a subset of vertices $V' \subseteq V$ as $\mathcal{D}_t[V']$; thus in the previous example $\mathcal{D}_4[\{v_2, v_3\}] = \langle (v_2, -1) \rangle$.

When we receive a pair (v_t, y_t), at first only the vertex v_t is revealed. We then perform the prediction action: the marginal at v_t is computed given reduced data sequence, i.e., $P(y_t = y \mid \mathcal{D}_{t-1})$. We then receive y_t or $\emptyset$ and we update $\mathcal{D}_{t-1}$ to $\mathcal{D}_t$. The novelty and power of our methods is that update and prediction is fast.

Recall that via belief-propagation the marginal distribution at each vertex on a tree may be simultaneously computed in time linear in the size of the tree [1]. In the streaming setting linear time is too slow. For example, when T is a path graph (a linear sequence of n vertices $\langle v^{(1)}, \ldots, v^{(n)} \rangle$) it is easy to show that, in the streaming setting, belief propagation requires $\Theta(n)$ time per prediction. Our algorithm, instead, always requires $\mathcal{O}(\log n)$ time per prediction.

The upper bound on the time complexity of our algorithm will depend linearly on the *decomposition potential* $\chi(T)$. In Section 2 we will explain that this quantity is equivalent to the minimal height complete hierarchical clustering of a set of vertices such that every cluster is a subtree of T that shares a single joint vertex with other clusters (see Definition 1).

Related work. In [2] an algorithm was given for this model, in which each prediction required $\min\{\Delta_T, \log n\}$ time where Δ_t is the diameter of the tree. We significantly improve on this result we observe that $\log \Delta_t \leq \chi(T) \leq \min\{\Delta_T, \log n\}$ and the lower bound is tight. For example, consider the tree T formed by $n/\log n$ path graphs having length $\log n$ that overlap at same central vertex. In this case it is not difficult to show that the total time required by our algorithm is $\chi(T) = \mathcal{O}(\log \log n)$ an exponential improvement over the result in [2]. The prediction algorithm was inspired by ideas for predicting efficiently on a path graph in [3].

2 Decomposing a tree

Given a tree, we may define on its vertices a *decomposition tree*. We show in section 2.2 that this decomposition tree determines a collection of covering of subtrees that hierarchically cover the original tree. Our algorithm (section 3) for predicting and updating requires time proportional to the height of the decomposition tree. We then show in section 2.3 that we can find a decomposition tree of minimal height in cubic time.

2.1 Preliminaries

A graph G is a pair of sets (V, E) such that E is a set of unordered pairs of distinct elements from V. The elements of V are called vertices and those of E are called edges. In order to avoid ambiguities deriving from dealing with different graphs, in some cases we will highlight the membership to graph G denoting these sets as $V(G)$ and $E(G)$ respectively. With slight abuse of notation, by writing $i \in G$, we mean $i \in V(G)$. S is a subgraph G (we write $S \subseteq G$) iff $V(S) \subseteq V(G)$ and $E(S) = \{(i,j) : i,j \in V(S), (i,j) \in E(G)\}$. Given any subgraph $S \subseteq G$, we define its *boundary* (or inner border) $B_G(S)$ and its *neighbourhood* (or outer border) $N_G(S)$ as: $B_G(S) := \{i : i \in S, j \notin S, (i,j) \in E(G)\}$, and $N_G(S) := \{j : i \in S, j \notin S, (i,j) \in E(G)\}$. With slight abuse of notation, $N_G(v) := N_G(\{v\})$, and thus the degree of a vertex v is $|N_G(v)|$. Given any graph G, we define the set of its leaves and its interior respectively as $\text{leaves}(G) := \{i \in G : |N_G(i)| = 1\}$, and $\text{int}(G) := \{i \in G : |N_G(i)| \neq 1\}$ A path (or line-graph) P is a graph with vertex set $V(P) = v_1, v_2, ..., v_n$ and edge set $E(P) = \{(v_m, v_{m+1}) : m \leq n-1\}$. The length of a path P is equal to $|E(P)| = |V(P)| - 1$. We say that v_1 and v_n are connected by P.

A tree, T is a graph in which for all $v, w \in T$ there exists a unique path connecting v with w. Such a path is denoted by $\text{path}_T(v, w)$. The distance $d_T(v, w)$ between $v, w \in T$ is the path length $|E(P)|$. We denote a rooted tree T with pair (T, r), where r is the T's root. Given a rooted tree (T, r) and any vertex $i \in V$, the descendants of i are all vertices that can be connected with r via paths $P \subseteq T$ containing i. Analogously the ancestors of i are all vertices that lie on the path $P \subseteq T$ connecting i with r. Observe that each vertex $i \in V$ is ancestor and descendent of its self. We denote the set of all descendants (resp. all ancestors) of i by $\Downarrow^r_T(i)$ (resp. $\Uparrow^r_T(i)$). We shall omit the root r when it is clear from the context. Vertex i is the parent (resp. child) of j is denoted by $\uparrow^r_T(j)$ (resp. $i \in \downarrow^r_T(j)$) if $(i,j) \in E(T)$ and $i \in \Uparrow^r_T(j)$ (resp. $i \in \Downarrow^r_T(j)$). The height of a rooted tree (T, r) is maximum length of a path $P \subseteq T$ connecting the root to any vertex, $h_T(r) := \max_{v \in T} d_T(v, r)$. The diameter of any tree T is the maximum length of a path $P \subseteq T$ connecting any pair of vertices of T. If T is a tree we only use the notation $S \subseteq T$ if S is a tree and subgraph of T.

2.2 The $*$-decomposition potential

In this section we describe a splitting process that recursively decomposes a tree T. A (decomposition) tree D identifies the splitting process that generates a hierarchical covering collection of sets $\mathcal{S}$.

This process recursively splits at each step a (component) subtree resulting from some previous splits. More precisely a subtree $S \subseteq T$ is split into two or more subcomponents and the decomposition of S depends only on the choice of a vertex $v \in \text{int}(S)$, which we call *splitting vertex*, in the following way. The splitting vertex $v \in \text{int}(S)$ of S induces the *split* $\Omega(S, v) = \{S_1, \ldots, S_{|N_S(v)|}\}$ which is the unique set of S's subtrees overlapping at vertex v solely and representing a cover for S, i.e. it satisfies (i) $\cup_{S' \in \Omega(S,v)} S' = S$ and (ii) $\{v\} = S_i \cap S_j$ for all $1 \leq i < j \leq |N_S(v)|$. Thus the split may be visualized by considering the forest F resulting from removing a vertex from a S, but afterwards each component $S_1, \ldots, S_{|N_S(v)|}$ of F has the "removed vertex" v added back to it.

A component having only two vertices is called *atomic*, since it cannot be split further. We indicate with $S^v \subseteq T$ the component subtree whose splitting vertex is $v \in \mathrm{int}(S^v)$ and we denote atomic components by $S^{(i,j)}$, where $E(S^{(i,j)}) = \{(i,j)\}$. We finally denote by $\mathcal{S}$ the set of all component subtrees obtained by this splitting process. Since the method is recursive, we can associate a rooted tree (D,r), with T's decomposition into a hierarchical cover, whose internal vertices are the splitting vertices of the splitting process, its leaves are the single edges of each atomic component, and a vertex "parent-child" relation $c \in \downarrow_D^r(p)$ corresponds to the "splits-into" relation $S^c \in \Omega(S,p)$.

We will now formalize the splitting process by defining the *hierarchical cover* $\Gamma(T) = (\mathcal{S},(D,r))$ of any tree T, which is key concept used by our algorithm.

Definition 1. *The* ***hierarchical cover*** $\Gamma(T) = (\mathcal{S},(D,r))$ *of a tree* T *consists of a collection of covering subtrees* $\mathcal{S} = \{S^{(i,j)} : (i,j) \in E(T)\} \cup \{S^i : i \in \mathrm{int}(T)\}$ *of* T *and a* ***decomposition tree*** (D,r) *such that* $\mathrm{int}(D) = \mathrm{int}(T)$, $\mathrm{leaves}(D) = E(T)$*, and the following three properties hold,*

1. $S^r = T$,
2. *for all* $c,p \in \mathrm{int}(D) \setminus \{r\}$, $c \in \downarrow_D^r(p)$ *iff* $S^c \in \Omega(S^p,p)$,
3. *for all* $(i,j) \in \mathrm{leaves}(D)$, $\uparrow_D^r((i,j))$ *is either* i*, iff* $S^{(i,j)} \in \Omega(S^i,i)$*, or* j*, iff* $S^{(i,j)} \in \Omega(S^j,j)$.

In the first step T splits into $|N_T(r)|$ components. This process continues until we obtain all the atomic components $S^{(i,j)}$ corresponding to the leaves of D. The choice of the splitting vertices for each component corresponding to one of the internal vertices of D unequivocally defines the splitting process. The height of a hierarchical cover $\Gamma(T) = (\mathcal{S},(D,r))$ is the height of the decomposition tree D.

Note that the height of a decomposition tree D may be exponentially smaller than the height of T, since, for example, it is not difficult to show that there exists a decomposition tree isomorphic to a binary tree when the input tree T is a line-graph. With slight abuse of terminology we will indicate the height of the hierarchical cover Γ as the height of D.

The following two definitions are central for the definition of our algorithm. Given any subtree $S \subseteq T$ and any hierarchical cover $\Gamma(S) = (\mathcal{S},(D,r))$, we define the **exposure** of S (with respect to tree T) as $\max_{S' \in \mathcal{S}} B_T(S')$.

Definition 2. *A hierarchical cover with exposure at most* k *is called* k***-decomposition***. *Given any subtree* $S \subseteq T$*, the* k***-decomposition potential*** $\chi_T^k(S)$ *of* S *is the minimum height of all hierarchical covers of* S *with exposure (with respect to* T*) not larger than* k*. The* $*$***-decomposition potential*** $\chi^*(S)$ *is the minimum height of all hierarchical covers of* S*. If* $B_T(S) > k$ *then* $\chi_T^k(S) := +\infty$.

In Section 3 we will give an algorithm to compute the marginal at vertex in a MRF on T, the time required will be proportional to the height of the decomposition. We are specifically interested in hierarchical covers with small exposure, as the governing principles of our algorithm require us to maintain, for every $S \in \mathcal{S}$, a separate quantity for each possible labeling of $B_T(S)$, thus we require time and memory exponential in the exposure. Therefore we are interested in decompositions with small exposure. In the following lemma we provide justification for restricting our algorithm to hierarchical covers with an exposure at most two as we show that, given an arbitrary hierarchical cover, we can always use it to construct one of exposure two which has no more than twice the height:

Lemma 3. *The* 2*-decomposition potential of a tree* T *is bounded by twice its* $*$*-decomposition potential,*

$$\chi_T^2(T) \le 2\chi^*(T)\,. \tag{1}$$

We provide a proof in the full version of this paper.

2.3 Computing an optimal 2-decomposition

In this section we describe a recursive algorithm able to find an optimal 2-decomposition of T, i.e., a 2-decomposition with minimum height.

Given any subtree $S \subseteq T$, now let $\Gamma^*(S)$ be an optimal 2-decomposition (with respect to T) of S. The splitting vertex of S must be one of the vertices $v \in \mathrm{int}(S)$ such that we have $|B_T(R)| \le 2$ for any $R \in \Omega(S, v)$. Then, for all $S \subseteq T$ we have:

$$\chi_T^2(S) = \min_{v \in \mathrm{int}(S)} \left[\max_{R \in \Omega(S,v)} \chi_T^2(R) \right] + 1 \ . \tag{2}$$

Observe now that, for any component $S \subseteq T$ of a 2-decomposition, we have $B_T(S) \subseteq \mathrm{leaves}(S)$. Define now $\mathbb{T}(T)$ as the set of all subtrees S of T such that $B_T(S) \subseteq \mathrm{leaves}(S)$ and $|B_T(S)| \leq 2$:

$$\mathbb{T}(T) := \{S \subseteq T : B_T(S) \subseteq \mathrm{leaves}(S), |B_T(S)| \leq 2\} \ .$$

From this definition it is clear that any subtree S that is a component of an optimal 2-decomposition is contained in $\mathbb{T}(T)$.

The basic idea of our algorithm is calculate $\chi_T^2(S)$ for all $S \in \mathbb{T}(T)$ proceeding in order of ascendending cardinality exploiting equation (2), in such a way to keep track of all splitting vertices of the optimal 2-decompositions. In order to use equation (2), this calculation is accomplished in a number of steps equal to $\chi_T^2(T)$ in the following way: at the t-th step the algorithm calculate $\chi_T^2(S)$ for all S such that $|S| = t$ using the results of the calculation made in the previous steps. For each $S \in \mathbb{T}(T)$, according to equation (2), the algorithm stores the splitting vertex v which minimizes $\max_{R \in \Omega(S,v)} \chi_T^2(R)$. In order to access to the result of the calculation we accomplished in the previous steps, we need to index each $S \in \mathbb{T}(T)$ as follows. Note that, for any pair of different vertices $\{i, j\}$, there is at most one subtree, that we denote by $S_{i,j}$, such that $|B_T(S)| = \{i, j\}$. Moreover, for each subtree $R \in \Omega(T', v)$ for some $T' \subseteq T$, if we have $|B_T(R)| = 1$ then $B_T(R) = \{v\}$ and $|N_R(v)| = 1$. Thus if $R \in \mathbb{T}(T)$ and $\{v\} = B_T(R)$ then letting $\{w\} = N_R(v)$ we have that $R = \Downarrow_T^v(w) \cup \{v\}$. We denote such a subtree $R_{v \to w}$. Thus, for each ordered pair $v \to w$ of adjacent vertices of T, we cannot have more than one subtree $R \in \mathbb{T}(T)$ such that $B_T(R) = \{v\}$ and $w \in R$. Hence, we can store all the values of decomposition potential calculated step by step by our algorithm in two matrices: $M^{(2)}$ and $M^{(1)}$, whose elements are respectively $m_{i,j}^{(2)} = \chi_T^2(S_{i,j})$, and $m_{u,v}^{(1)} = \chi_T^2(R_{u \to v})$. Our algorithm operates in three steps as follows:

1. Compute each cardinality of every tree in $\mathbb{T}(T)$.
2. Sort each element of $\mathbb{T}(T)$ by its cardinality .
3. For each $2 \leq t \leq \chi_T^2(T)$, compute $\chi_T^2(S)$ for all $S \in \mathbb{T}(T)$ such that $|S| = t$ using Equation (2) .

Since there are $|\mathbb{T}(T)| = \mathcal{O}(n^2)$ trees and for each $S \in \mathbb{T}$ we can compute $\chi_T^2(S)$ in $\mathcal{O}(|\mathrm{int}(S)|) = \mathcal{O}(n)$ time because we have constant time access to the elements of $M^{(1)}$ and $M^{(2)}$, the total time required by our method for obtaining an optimal 2-decomposition is equal to $\mathcal{O}(n^3)$.

3 Streaming prediction and update via a decomposition tree

In belief propagation the "messages passing" follows the topology of the input tree. In the following algorithm information is propagated via the topology of the decomposition tree D.

3.1 Probabilistic Preliminaries

Definition 4. *Given a graph, G, a Markov Random Field (MRF) on G is a probability measure on the set of all possible labelings of G which satisfies the Markov Property: Given a vertex, v, the label of v is conditionally independent of the labels on $V(T) \setminus \{v\}$ given the labels on $N_T(v)$.*

We consider a *two-level* MRF. The inner level (hidden variables) corresponds to the vertices of a given tree T_0. We construct outer level (observable variables) by constructing a tree $T \supseteq T_0$. For each vertex in $V(T_0) = \{v_1, \ldots, v_n\}$, in T_0 we will have a paired vertex so that $V(T) := \{v_1, \ldots, v_n\} \cup \{\tilde{v}_1, \ldots, \tilde{v}_n\}$ and the edge set is $E(T) := E(T_0) \cup \{(v_i, \tilde{v}_i) : 1 \leq i \leq n\}$. We also use the following notation so that if $v \in T_0$ then $\tilde{v} := w$ such that $w \in V(T) \setminus V(T_0)$ and $(v, w) \in E(T)$. We observe that given an optimal hierarchical cover $\Gamma(T_0) = (\mathcal{S}_0, (D_0, r))$ with exposure k, there exists an optimal hierarchical cover $\Gamma(T) = (\mathcal{S}, (D, r))$ with exposure k with $\chi_k(T) = \chi_k(T_0) + 1$ if $|V(T_0)| > 2$.

3.2 The prediction, update and initialisation algorithm

We now describe the algorithms for initialising and maintaining the data structure that we use for predicting the label of any vertex. We introduce the following notations. Given a tree, T, a labeling, u, of T, is a map from $V(G)$ to $\{-1, 1\}$ and we abbreviate $u_i := u(i)$. We assume we are given a two-level tree. When we predict at time t we compute the marginal at the $P(y = u_{v_t} \mid \mathcal{D}_{t-1})$. The received label y_t however is associated with the observable vertex $\tilde{v_t}$.

The essence of the algorithm is to not to traverse the topology of the original tree T but the decomposition tree D, thus for the following we will abbreviate both $\downarrow_D, \uparrow_D$ as both $\downarrow, \uparrow$ respectively as convenient. We define $C_i := \{z \in D : |B_T(S^z)| = i\}$ for $i \in \{1, 2\}$. As $\downarrow_D (v)$ is a set of children, we define the following functions to select specific children,

$$\triangleleft(v) := w \text{ if } w \in \downarrow(v), \uparrow(v) \in B_T(S^{(w)}) \qquad (v \in \operatorname{int}(D) \cap (C_1 \cup C_2))$$
$$\triangleright(v) := w \text{ if } w \in \downarrow(v), w \neq \triangleleft(v), w \in C_2 \qquad (v \in \operatorname{int}(D) \cap (C_2))$$

The algorithms work by caching certain quantities (derived from the MRF and the observed labels) associated with the subtrees in $\mathcal{S}$. We shall call these quantities weights. In practice rather than explicitly maintaining the weights, one may maintain normalised versions of these weight to prevent underflow. Out of the following weights, only α, β and γ needed to be cached between time t to $t+1$ - the others are computed "on-the-fly" during prediction. *In the following weights we introduce, there is an implicit time index we have dropped this index for brevity in notation.*

First, we define the weights that represents the influence that a tree in $\mathcal{S}$ has on the rest of T :

$$\alpha_a(v) = \frac{P(u_x = a, \mathcal{D}_t[S^v])}{P(u_x = a)} \qquad v \in C_1; x = \uparrow(v)$$
$$\beta_{ab}(v) = \frac{P(u_x = a, u_y = b, \mathcal{D}_t[S^v])}{P(u_x = a)P(u_y = b)} \qquad v \in C_2; \{x, y\} \in B_T(S^v); x = \uparrow(v)$$

We shall often need to take the product of many of the α-weights of the children of a vertex in D that are in C_1. Computing this product directly each time needed is inefficient so we henceforth store the product of the α-weights (multiplied by $P(u_v)$) of all such children:

$$\gamma_a(v) = P(u_v = a) \prod_{w \in \downarrow(v) \cap C_1} \alpha_a(w) \qquad v \in T_0$$

We now define the weights that represent how a tree in $\mathcal{S}$ is influenced by the rest of T:

$$\delta^{\triangleleft}_a(v) = \frac{P(u_x = a, \mathcal{D}_t[\bigcup\{Q \in \Omega(T, x) | v \notin Q\}])}{P(u_x = a)} \qquad v \in V(T_0) \setminus \{r\}; x = \uparrow(v)$$
$$\delta^{\triangleright}_a(v) = \frac{P(u_y = a, \mathcal{D}_t[\bigcup\{Q \in \Omega(T, y) | v \notin Q\}])}{P(u_y = a)} \qquad v \in C_2; \{x, y\} = B_T(S^v); x = \uparrow(v)$$

The following weights represent the information about T that comes into a vertex, $v \in T_0$, through one of its children in C_2:

$$\epsilon^{\triangleleft}_a(v) = \frac{P(u_y = a, \mathcal{D}_t[Q : Q \in \Omega(T, v), x \in Q])}{P(u_y = a)} \qquad v \in T_0 \setminus \{r\}; x = \uparrow(v)$$
$$\epsilon^{\triangleright}_a(v) = \frac{P(u_y = a, \mathcal{D}_t[Q : Q \in \Omega(T, v), y \in Q])}{P(u_y = a)} \qquad v \in C_2 \cap T_0; \{x, y\} = B_T(S^v); x = \uparrow(v)$$

Finally, we define the weight, ρ, to be, when normalised, the marginal probability of individual hidden label conditioned on the observed labels:

$$\rho_a(v) = P(u_v = a, \mathcal{D}_t[T]) \qquad v \in T_0$$

Theorem 5. *The prediction algorithm in Figure 1, on trial t correctly computes probability $\rho_a(v_t) = P(u_v = a, \mathcal{D}_t[T])$ in $\mathcal{O}(\chi^*(T))$ time, and thus $P(u_v = a|\mathcal{D}_t[T]) = \rho_a(v_t)/\sum_b \rho_b(v_t)$.*

Proof. We give a sketch of the proof. We have the following identities that all from the application of the Markov property. If $v = r$ then $\rho_a(r) = \gamma_a(r)$. If $v \in C_1$, we have:

$\alpha_a(v) = \sum_c \beta_{ca}(\lhd(v))\gamma_c(v)$, $\delta^{\lhd}_a(v) = \frac{\rho_a(\uparrow(v))}{\alpha_a(v)}$, $\epsilon^{\lhd}_a(v) = \sum_b \delta^{\lhd}_b(v)\beta_{ab}(\lhd(v))$, and $\rho_a(v) = \epsilon^{\lhd}_a(v)\gamma_a(v)$. If $v \in C_2$ then $\beta_{ab}(v) = \sum_c \beta_{ca}(\lhd(v))\beta_{cb}(\rhd(v))\gamma_c(v)$, $\epsilon^{\lhd}_a(v) = \sum_b \delta^{\lhd}_b(v)\beta_{ab}(\lhd(v))$, $\epsilon^{\rhd}_a(v) = \sum_b \delta^{\rhd}_b(v)\beta_{ab}(\rhd(v))$, and $\rho_a(v) = \epsilon^{\lhd}_a(v)\epsilon^{\rhd}_a(v)\gamma_a(v)$, furthermore if, in addition, $v = \lhd(\uparrow(v))$: $\delta^{\lhd}_a(v) = \epsilon^{\rhd}_a(\uparrow(v))\gamma_a(\uparrow(v))$,$\delta^{\rhd}_a(v) = \delta^{\lhd}_a(\uparrow(v))$ and if $v = \rhd(\uparrow(v))$ we have: $\delta^{\lhd}_a(v) = \epsilon^{\lhd}_a(\uparrow(v))\gamma_a(\uparrow(v))$, and $\delta^{\rhd}_a(v) = \delta^{\rhd}_a(\uparrow(v))$.

Update: Suppose we are given a new label, $\tilde{u}_{\tilde{v}}$ for a vertex $\tilde{v}$. We now describe how to update the α, β and γ weights appropriately. By the previous identities, we only need update the weights of ancestors of v.

Firstly, we set:

$$\gamma_a(v) \leftarrow \frac{\gamma_a(v)P(u_v = a, u_{\tilde{v}} = \tilde{u}_{\tilde{v}})}{\alpha_a((v,\tilde{v}))P(u_v = a)}; \text{ then set } \alpha_a((v,\tilde{v})) \leftarrow \frac{P(u_v = a, u_{\tilde{v}} = \tilde{u}_{\tilde{v}})}{P(u_v = a)} \tag{3}$$

Observing, that if we receive the quasi-label $\emptyset$ then $\frac{P(u_v=a,u_{\tilde{v}}=\tilde{u}_{\tilde{v}})}{P(u_v=a)} = 1$. We then proceed to the update algorithm in Figure 1, which is based on the identities above.

Prediction: We now describe how to perform the exact Bayes' prediction of the label of a vertex, v. Note: we only need to use the weights of the ancestors of v.

We first climb D from v to r in order to create, on the set of ancestors of v, the map $\uparrow^v$, which maps a vertex, x, to the vertex, y, for which, if D was re-rooted at v, y would be the parent of x. We then use the map to descend D from r to v, calculating, via the weights δ, ϵ and ρ for each vertex we encounter. □

Prediction:

```
$w \leftarrow r$
$\rho_a(w) \leftarrow \gamma_a(r)$
**while**$(w \neq v)$
    $w \leftarrow \uparrow^v(w)$
    **if**$(w \in C_1)$
        $\delta^{\lhd}_a(w) \leftarrow \rho_a(\uparrow(w))/\alpha_a(w)$
        $\epsilon^{\lhd}_a(w) \leftarrow \sum_b \beta_{ca}(\lhd(w))\gamma_c(w)$
        $\rho_a(w) = \gamma_a(w)\epsilon^{\lhd}_a(w)$
    **else**
        **if**$(j = \lhd(\uparrow(j)))$
            $\delta^{\lhd}_a(w) \leftarrow \epsilon^{\rhd}_a(\uparrow(w))\gamma_a(\uparrow(w))$
            $\delta^{\rhd}_a(w) \leftarrow \delta^{\lhd}_a(\uparrow(w))$
        **else**
            $\delta^{\lhd}_a(w) \leftarrow \epsilon^{\lhd}_a(\uparrow(w))\gamma_a(\uparrow(w))$
            $\delta^{\rhd}_a(w) \leftarrow \delta^{\rhd}_a(\uparrow(w))$
        $\epsilon^{\lhd}_a(w) \leftarrow \sum_b \delta^{\lhd}_b(w)\beta_{ab}(\lhd(w))$
        $\epsilon^{\rhd}_a(w) \leftarrow \sum_b \delta^{\rhd}_b(w)\beta_{ab}(\rhd(w))$
        $\rho_a(w) \leftarrow \epsilon^{\lhd}_a(w)\epsilon^{\rhd}_a(w)\gamma_a(w)$

**Output:** $\rho_a(v)$
```

Initialisation: The α, β and γ weights are initialised in a bottom-up fashion on the decomposition tree - we initialise the weights of a vertex after we have initialised the weights of all its children. Specifically, we first do a depth-first search of D starting from r: When we reach an edge $(v,\tilde{v}) \in E(T)$ we set: $\alpha_a((v,\tilde{v})) \leftarrow 1$ and when we reach an edge $(v,w) \in E(T_0)$, we set: $\alpha_a((v,w)) \leftarrow \frac{P(u_v=a,u_w=b)}{P(u_v=a)P(u_w=b)}$ When we reach a vertex, $v \in V(T)$, for the last time (i.e. just before we backtrack from v) then set: $\gamma_a(v) \leftarrow P(u_v = a)\prod_{w\in\downarrow(v)\cap C_1} \alpha_a(w)$, and if $v \in C_2$ then $\beta_{ab}(v) \leftarrow \sum_c \beta_{ca}(\lhd(v))\beta_{cb}(\rhd(v))\gamma_c(v)$, or if $v \in C_1$ then $\alpha_a(v) \leftarrow \sum_c \beta_{ca}(\lhd(v))\gamma_c(v)$.

Update:

```
$w \leftarrow v$
**while**$(w \neq r)$
    **if**$(w \in C_1)$
        $\alpha^{\text{old}}_a \leftarrow \alpha_a(w)$
        $\alpha_a(w) \leftarrow \sum_c \beta_{ca}(\lhd(w))\gamma_c(w)$
        $\gamma_a(\uparrow(w)) \leftarrow \gamma_a(\uparrow(w))\alpha_a(w)/\alpha^{\text{old}}_a$
    **else**
        $\beta_{ab}(w) \leftarrow \sum_c \beta_{ca}(\lhd(w))\beta_{cb}(\rhd(v))\gamma_c(v)$
    $i \leftarrow \uparrow(j)$
```

Figure 1: Algorithm: Initialisation, prediction and update

Acknowledgements. We would like to thank David Barber and Guy Lever for valuable discussions.

References

[1] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[2] A. L. Delcher, A. J. Grove, S. Kasif, and J. Pearl. Logarithmic-time updates and queries in probabilistic networks. *J. Artif. Int. Res.*, 4:37–59, February 1996.

[3] M. Herbster, G. Lever, and M. Pontil. Online prediction on large diameter graphs. In *NIPS*, pages 649–656. MIT Press, 2008. Note, referenced prediction algorithm is in an extended version in preparation, 2011.