A New Efficient Probabilistic Model for Mining Labeled Ordered Trees

Kosuke Hashimoto, Kiyoko F. Aoki-Kinoshita, Nobuhisa Ueda,
Minoru Kanehisa, Hiroshi Mamitsuka

Bioinformatics Center, Institute for Chemical Research, Kyoto University
Gokasho Uji, 611-0011 Japan

{khashimo,kiyoko,ueda,kanehisa,mami} at kuicr.kyoto-u.ac.jp

ABSTRACT

Mining frequent patterns is a general and important issue in data mining. Complex and unstructured (or semi-structured) datasets have appeared in major data mining applications, including text mining, web mining and bioinformatics. Mining patterns from these datasets is the focus of many of the current data mining approaches. We focus on labeled ordered trees, typical datasets of semi-structured data in data mining, and propose a new probabilistic model and its efficient learning scheme for mining labeled ordered trees. The proposed approach significantly improves the time and space complexity of an existing probabilistic modeling for labeled ordered trees, while maintaining its expressive power. We evaluated the performance of the proposed model, comparing it with that of the existing model, using synthetic as well as real datasets from the field of glycobiology. Experimental results showed that the proposed model drastically reduced the computation time of the competing model, keeping the predictive power and avoiding overfitting to the training data. Finally, we assessed our results using the proposed model on real data from a variety of biological viewpoints, verifying known facts in glycobiology.

Categories and Subject Descriptors: I.2.6 [Artificial Intelligence]: Learning–*knowledge acquisition, parameter learning* ; I.5.1 [Pattern Recognition]: Models–*Statistical*

General Terms: Algorithms and Experimentation

Keywords: Labeled Ordered Trees, Probabilistic Models, Maximum Likelihood, Expectation-Maximization

1. INTRODUCTION

Mining frequent patterns is a general and important issue in data mining. The data in conventional data mining is well structured, and so relatively simple patterns, such as associations and sequences, are targeted. However, datasets of more complex and unstructured (or semi-structured) data such as trees and graphs have recently appeared in major data mining applications, such as text mining [21], web mining [4] and bioinformatics [2]. A typical example is a tree-structured text format called XML on the web [1]. Mining XML documents has become an important domain in the field of data mining [22]. XML documents are datasets of semi-structured data, or more specifically, labeled ordered trees, and methods based on kernel functions [11] and frequent pattern mining [23] have appeared in recent years.

Probabilistic modeling and learning is a noise-robust, powerful and efficient approach in machine learning and data mining. Bayesian networks [16], which can handle directed acyclic graphs, have been used quite frequently since its introduction. However, directed acyclic graphs are much more complex than labeled ordered trees, and so simpler probabilistic models and more efficient learning algorithms for trees have been proposed. For example, the hidden tree Markov model (HTMM) [6] is a probabilistic model for labeled trees that handles parent-child dependencies in a given tree. However, it does not handle sibling dependencies, meaning that HTMM is not applicable to labeled ordered trees. The probabilistic sibling-dependent Tree Markov Model (PSTMM) [18, 19] has been developed for labeled ordered trees, containing both parent-child and sibling dependencies. These models, which are subclasses of Bayesian networks, significantly reduce the high complexity of Bayesian networks by restricting the target to labeled trees or labeled ordered trees [19]. The extension of HTMM to PSTMM for trees is equivalent to that of the hidden Markov model (HMM) [3, 17] to the probabilistic context free grammars (PCFGs) [12, 14] for sequences (strings). Consequently, the time and space complexity of the learning algorithm of HTMM roughly increases by a factor of the number of states in PSTMM. This is a sizable difference, since labeled ordered trees in the real world, such as XML documents, are often large (or long), while natural language sentences dealt with by PCFG are rather short. Applying PSTMM to such large data would make it intractable due to its high time and space complexity. On the other hand, for small-sized data such as those found in glycobiology, PSTMM would overfit. Thus, it is necessary to reduce its complexity in order for it to be applicable to real-world problems involving labeled ordered trees.

In light of the above, we propose a new and efficient probabilistic model, which we call the ordered tree Markov

KDD'06, August 20–23, 2006, Philadelphia, Pennsylvania, USA.

model (OTMM). We present the time and space efficient algorithms for the three problems arising when OTMM is applied to real-world problems: likelihood computation, learning (estimating) the probabilistic parameters based on maximum likelihood, and parsing (finding the most likely path). OTMM has sibling dependencies as well as parent-child dependencies, but the parent-child dependencies are confined to those between just the eldest siblings and their parents. Consequently, our model reduces the complexity of PSTMM to the same level as that of HTMM. The maximum likelihood estimation of the parameters of OTMM is based on an EM (Expectation-Maximization) algorithm [5, 15]. This is an extension of the Baum-Welch (Forward-Backward) algorithm for HMMs, which updates the forward and backward probabilities based on dynamic programming. This extension is similar to that of HTMM and PSTMM, but we emphasize that these estimation algorithms are significantly different from each other. HTMM deals with parent-child dependencies only, and so the learning algorithm of HTMM is a straightforward modification of the forward and backward algorithm in a string to compute the parent-to-child (downward) and child-to-parent (upward) probabilities. On the other hand, PSTMM handle both the parent-child and sibling dependencies, and so the algorithm must deal with both upward and downward probabilities as well as forward and backward probabilities. In PSTMM, every child depends on its parent, and the upward (downward) probability of a node can be directly updated from its child (parent) easily. In contrast, the parent-child dependencies in OTMM are limited to the dependencies of the eldest siblings on their parents. Thus, for OTMM, we needed to develop an algorithm by which we could compute the four (upward, downward, forward and backward) probabilities under the constraint of using the limited parent-child dependencies.

We empirically evaluated the effectiveness of our proposed scheme using synthetically generated datasets as well as real datasets in bioinformatics. From the results we obtained, we found that for any test setting, our approach significantly reduced the computation time for learning PSTMM and avoided overfitting to the training data, while either maintaining or even improving the predictive power of PSTMM. In particular, we emphasize that a typical increase in computation time ranged from four- to seven-fold in our experimental setting, and that this factor increases as the number of states increase. In addition to these comparison results, we analyzed biological data by our method using data from glycobiology, which is the study of carbohydrate sugar chains, or glycans, that can be modeled as labeled ordered trees. Glycans are considered the third major class of biomolecules next to DNA and proteins [20], and it is known that the ordering of their leaves is used in recognition and signaling events in various biological processes. Thus we studied the patterns learned from this data and verified known facts related to sibling-dependencies in glycans that were captured by our model.

2. NOTATIONS

We describe the notations that will be used throughout this paper. A *tree* is an acyclic connected graph. In this paper, we refer to a vertex of a tree as a *node* of the tree. A *rooted tree* is a tree that has a special node called the root. Any node x on a unique path from the root to a node y is called an *ancestor* of node y, in which case y is called a *descendant* of x. Each of the closest descendants of x (that is, a node that is only one edge away from node x) is called a *child* of x, in which case x is called the parent of the child. We call nodes x and y siblings if x and y have the same parent. We call a node having no children a *leaf*. A *subtree* of tree T is a tree consisting of all descendants of a node. An *ordered tree* is a rooted tree in which the children of each node are ordered. A *labeled tree* is a tree in which a label is attached to each node. We will often simply use the term *tree* in place of an ordered, labeled and rooted tree.

Let $\mathbf{T} = \{T_1, \ldots, T_{|\mathbf{T}|}\}$ be a set of labeled ordered trees, where $T_u = (V_u, E_u)$ and $V_u (= \{x_1^u, \ldots, x_{|V_u|}^u\})$ and $E_u \subseteq V_u \times V_u$ are a set of nodes and a set of edges, respectively. Let x_1^u be the root of tree T_u, and $|V| = \max_u |V_u|$. We assume that nodes are indexed by level order, which can be done by traversing the tree in breadth-first order. An example of trees with their indices is shown in Figure 1. From this indexing of nodes, for a node j, we can refer to the immediately elder and younger siblings of j as $j-1$ and $j+1$, respectively. Let $t_u(i)$ be a subtree of T_u, having x_i^u as the root of $t_u(i)$. Let $x_{\leftarrow}^u(p)$ and $x_{\rightarrow}^u(p)$ be the eldest and youngest children of node p, respectively. Let $C_u(p) \subseteq \{1, \ldots, |V_u|\}$ be a set of indices of children of x_p^u in T_u and $|C| = \max_{u,p} |C_u(p)|$. Let $X_u(j)$ be a set of indices of all the younger siblings of x_j^u in T_u. Each node x_j^u has label $o_j^u \in \Sigma$, where $\Sigma = \{\sigma_1, \ldots, \sigma_{|\Sigma|}\}$ is a set of labels. For simplicity, we will often use node j instead of x_j^u and p as a parent node, if understood from the context.

Let θ denote a set of parameters of a probabilistic model. For simplicity, we may use $\theta = \{\theta_1, \ldots, \theta_n\}$ as a set of parameters, such that $\theta_i = \{\theta_{i,1}, \ldots, \theta_{i,|\theta_i|}\}$ and $\sum_{j=1}^{|\theta_i|} \theta_{i,j} = 1$ for $i = 1, \ldots, n$. A probabilistic model has 'states,' each of which is a so-called *latent* (or *hidden*) variable that cannot be seen directly, and each state has a probabilistic parameter which probabilistically generates a label at a node. Let $S = \{s_1, \ldots, s_{|S|}\}$ be a set of states, and $z_j^u \in S$ be the state of node j in a tree. For simplicity, we may also use j instead of z_j^u and q as the state of a parent node, if understood from the context.

3. PROPOSED MODEL: ORDERED TREE MARKOV MODEL (OTMM)

We propose a new efficient probabilistic model for mining labeled ordered trees which we hereafter call OTMM, for Ordered Tree Markov model. Here we briefly describe the differences between OTMM and two other similar tree Markov models, called the Hidden Tree Markov Model (HTMM) [6] and the Probabilistic Sibling-dependent Tree Markov Model (PSTMM) [18, 19].

OTMM is a first-order Markov chain model, meaning that a state depends on only one state. This is also true of HTMM, which is a probabilistic model for labeled trees. Figure 2 illustrates the dependencies in HTMM for the tree T_u in Figure 1 (a), where the state of a node depends on the state of its parent node. In OTMM, the state of a node depends on either the state of its parent or its immediately elder sibling. Figure 3 shows the dependencies in OTMM for the tree T_u in Figure 1 (a). As shown in the figure, if the node of a state is the eldest sibling, this state depends on its parent; otherwise, this state depends on its immediately elder sibling. Thus an important difference between OTMM and HTMM is that sibling dependencies are considered in

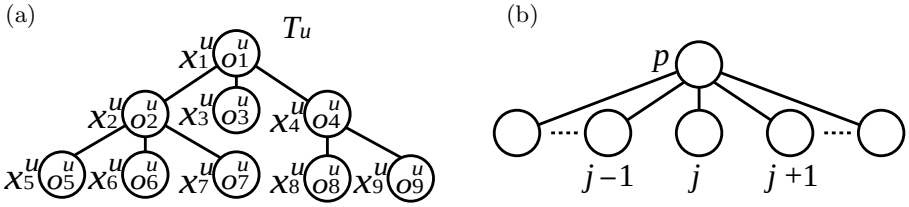

Figure 1: Notations for labeled ordered trees. (a) Labeled ordered tree T_u. (b) Indices for a parent and its children.

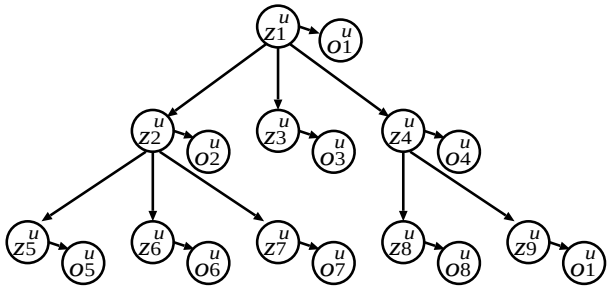

Figure 2: Graphical representation of HTMM for tree T_u in Figure 1.

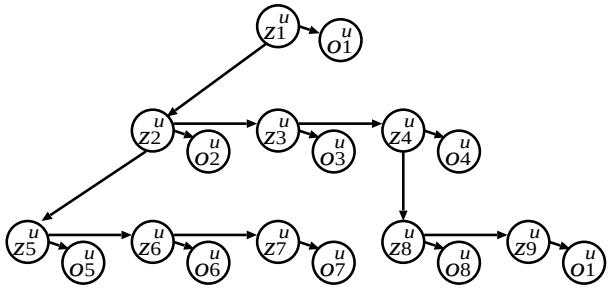

Figure 3: Graphical representation of OTMM for tree T_u in Figure 1.

OTMM but not in HTMM. Obviously, this difference makes the expressive power of OTMM for labeled ordered trees greater than that of HTMM.

On the other hand, PSTMM is a probabilistic model for labeled ordered trees, and the state of a node always depends on two different states, except for the eldest siblings. Figure 4 illustrates the dependencies in PSTMM for the tree in Figure 1 (a). As shown in the figure, the dependencies on the immediately elder sibling and on the parent are both considered in PSTMM. This feature gives PSTMM rich expressive power, but it is computationally expensive and requires more memory space in estimating the probabilistic parameters of PSTMM. In addition, this high complexity of PSTMM has the additional risk of overfitting to data. In fact, these problems have appeared when applying PSTMM to real data. OTMM avoids these problems and achieves better performance in practical situations where PSTMM suffers from the above problems.

In addition, recall that hidden Markov models (HMMs) enable the indirect capture of distant (long-range) dependencies in a sequence. We emphasize that OTMM can also capture such indirect dependencies in a similar manner. For example, a dependency between a node and its distant sibling, which cannot be captured by HTMM, may be detected by OTMM (and PSTMM). A dependency between a node and its distant sibling's descendant may also be captured

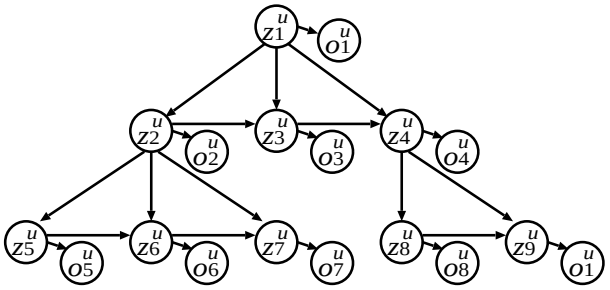

Figure 4: Graphical representation of PSTMM for tree T_u in Figure 1.

by OTMM (and PSTMM) but clearly not by HTMM. From the viewpoint of capturing such indirect dependencies, we can say that the expressive power of OTMM is at the same level as that of PSTMM and that it is greater than that of HTMM.

The algorithms for OTMM are derived by making some significant modifications to the algorithms of PSTMM, although OTMM is a simplification of PSTMM. These modifications are necessary to account for the reduced dependencies between parent and child. Thus in OTMM, the ancestor information cannot be transfered through parent-child dependencies directly except to the eldest siblings, although before, this information was easily (and directly) sent from a parent to all its children in PSTMM. As a result, it was necessary to reconstruct both the definitions of some probabilities and the equations using these probabilities in the dynamic programming procedure for learning PSTMM.

OTMM has three types of probability parameters, π, a, and b. The initial state probability $\pi[l]$ $(= P(z_1^u = s_l; \theta))$ is the probability that the state z_1^u of the root node x_1^u is s_l. The state transition probability a further takes two types: $\alpha[q,m](= P(z_j^u = s_m | z_p^u = s_q; \theta))$ and $\beta[l,m](= P(z_j^u = s_m | z_{j-1}^u = s_l; \theta))$. $\alpha[q,m]$ is the conditional probability that the state of a node is s_m given that the state of its parent is s_q. $\beta[l,m]$ is the conditional probability that the state of a node is s_m given that the state of the immediately elder sibling is s_l. The label output probability $b[l,\sigma_h](= P(o_j^u = \sigma_h | z_j^u = s_l; \theta))$ is the conditional probability that the output label of a node is σ_h given that the state of this node is s_l. Note that $\sum_l \pi[l] = 1$, $\sum_m \alpha[q,m] = 1$, $\sum_m \beta[l,m] = 1$, and $\sum_h b[l,\sigma_h] = 1$.

Given the probabilistic model, there are three key problems of interest that must be solved for the model to be used in real world applications [17]: 1) Likelihood computation: computing the likelihood of a given tree, 2) Learning: estimating the probability parameters from a set of given trees, and 3) Parsing (Prediction): finding the most likely state transition for a given tree using the estimated probability parameters. In the following three subsections, we

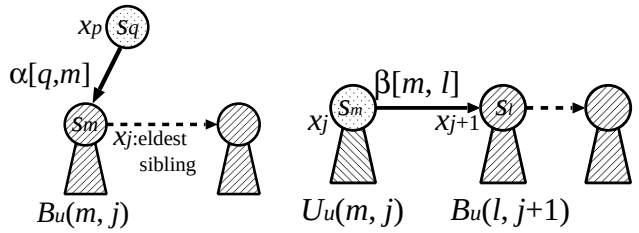

Figure 5: Updating (left) $U_u(q,p)$ and (right) $B_u(m,j)$. The sparse shaded node is p for $U_u(q,p)$ and j for $B_u(m,j)$. Dense shaded areas are used for updating.

will explain our efficient algorithms for OTMM for each of the above three problems.

3.1 Likelihood Computation

3.1.1 Upward and Backward Probabilities

We define an upward probability and a backward probability. The upward probability $U_u(q,p)$ is the probability that all labels of subtree $t_u(p)$ are generated and the state of node p is s_q. The backward probability $B_u(l,j)$ is the probability that for node j, all labels of a subtree for each of the younger siblings and node j are generated, and the state of j is s_l.

We can compute these two probabilities using a bottom-up (B-up) dynamic programming (DP) procedure. This computation is formulated as follows:

$$U_u(q,p) = \begin{cases} \text{If } C_u(p) = \emptyset \text{ then } b[q, o_p^u] \\ \text{otherwise} \\ b[q, o_p^u] \sum_{m=1}^{|S|} \alpha[q,m] B_u(m,j) \\ (\text{s.t. } x_j^u = x_{\leftarrow}^u(p)) \end{cases} \quad (1)$$

$$B_u(m,j) = \begin{cases} \text{If } x_j^u = x_{\rightarrow}^u(p) \text{ then } U_u(m,j), \\ \text{otherwise} \\ U_u(m,j) \sum_{l=1}^{|S|} \beta[m,l] B_u(l,j+1) \end{cases} \quad (2)$$

Figure 5 depicts the above calculation of the upward and backward probabilities. The upward probability at a node is computed using the backward probability of its eldest child, so this computation is repeated from the (eldest) child to its parent. The backward probability at a node is computed using the backward probability of its immediately younger sibling, meaning that this computation is successively repeated from a node to its immediately elder sibling. So the whole computation proceeds from the leaves to the root, in reverse breadth-first order, going bottom-up and right-to-left (R-to-L) using dynamic programming.

We can compute the likelihood for a given tree T_u by using the upward probability at the root of the tree, as follows:

$$L(T_u) = \sum_{l=1}^{|S|} \pi[l] U_u(l,1).$$

The likelihood for a given set of trees is defined as the product of the likelihood for each tree in the set.

$$L(\mathbf{T}) = \prod_{u=1}^{|\mathbf{T}|} L(T_u) = \prod_{u=1}^{|\mathbf{T}|} \sum_{l=1}^{|S|} \pi[l] U_u(l,1).$$

The above computation is relatively similar to that of PSTMM. A significant difference is that the backward probability of OTMM is a tri-tuple, while that of PSTMM is a quatro-tuple. This feature of OTMM reduces both the time and space complexity of PSTMM.

3.2 Learning: Maximum Likelihood Estimation

Maximum likelihood is a general criterion used to estimate the probability parameters of a probabilistic model from the given training examples. We employ an EM algorithm [5, 15], a general and popular scheme to maximize the likelihood for a given set of examples.

3.2.1 Forward and Downward Probabilities

We define forward and downward probabilities and use them with the backward and upward probabilities, both of which were defined in the previous section. The forward probability $F_u(l,j)$ is the probability that all labels of tree T_u except for those of subtree $t_u(j)$ and of all subtrees $t_u(k)$ ($k \in X_u(j)$) are generated and that the state of node x_j^u is s_l. The downward probability $D_u(l,j)$ is the probability that all labels of tree T_u except for those of subtree $t_u(j)$ are generated and that the state of x_j^u is s_l.

We can compute these two probabilities using a top-down (T-down) dynamic programming procedure. This computation is formulated as follows:

$$F_u(l,j) = \begin{cases} \text{If } x_j^u = x_{\leftarrow}^u(p) \text{ then } \sum_{q}^{|S|} \alpha[q,l] D_u(q,p) b[q, o_p^u], \\ \text{otherwise} \\ \sum_{m=1}^{|S|} \beta[m,l] F_u(m,j-1) U_u(m,j-1) \end{cases}$$

$$D_u(l,j) = \begin{cases} \text{If } j = 1 \text{ then } \pi[l], \\ \text{else if } x_j^u = x_{\rightarrow}^u(p) \text{ then } F_u(l,j), \\ \text{otherwise} \\ F_u(l,j) \sum_{m=1}^{|S|} \beta[l,m] B_u(m,j+1) \end{cases}$$

Figure 6 depicts these calculations for the forward probability at the eldest sibling and at another node. The forward probability at the eldest sibling is computed using the downward probability of its parent, meaning that this computation is repeated from a parent to its eldest child. The forward probability at another node is computed using the forward and upward probabilities of its immediately elder sibling, meaning that this computation is repeated from a node to its immediately younger sibling. Figure 7 depicts the calculation of the downward probability. The downward probability at a node is computed using its forward probability (and the backward probability of its younger sibling), and so at each node the downward probability must be computed after the forward probability is computed. Overall, the entire computation proceeds from the root to the leaves, in breadth-first order, in a top-down and left-to-right (L-to-R) dynamic programming procedure.

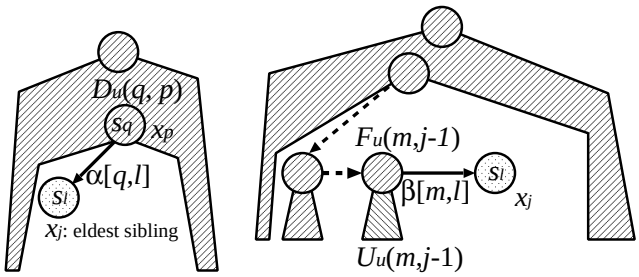

Figure 6: Updating $F_u(l, j)$ (left) at the eldest sibling and (right) at another node. The sparse shaded node is j, and dense shaded areas are used for updating.

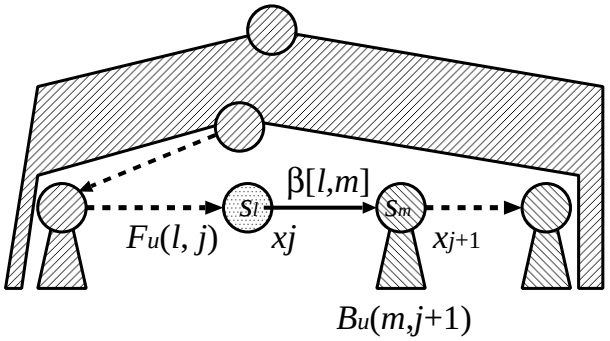

Figure 7: Updating $D_u(l, j)$. The sparse shaded node is j, and dense shaded areas are used for updating.

The updating rules of the forward and downward probabilities are significantly different from those of PSTMM. In PSTMM, the downward probability of a node was easily computed by using the downward probability of its parent. However, in OTMM, the state of a node (except the eldest siblings) does not depend on the state of its parent. Therefore, the downward probability needed to be computed from somewhere else, either the forward and/or backward probabilities. That is, we needed to incorporate the downward (i.e. *parent to child*) dependencies into either the forward or the backward probabilities. This was not possible for the backward probabilities, because the parent-child dependencies are limited to the eldest siblings only. Thus, at the eldest sibling, we compute the forward probability using the downward probability of its parent, and the forward probability carries the downward dependencies from parent to child. As a result, the forward probability of OTMM contains richer dependency information than that of PSTMM, making it completely different from that of PSTMM.

We note that the forward probability of OTMM is a tri-tuple, while it is a quatro-tuple in PSTMM. Thus we can expect that the algorithms for OTMM will run faster than those for PSTMM. We also note that the likelihood of a tree can be computed by OTMM using the upward and downward probabilities at node i as follows:

$$L(T_u) = \sum_l U_u(l, i)D_u(l, i).$$

3.2.2 EM (Expectation-Maximization) Algorithm

The EM algorithm for OTMM iterates the following E- and M-steps alternately, using the above four probabilities.

E-step:

We compute the expectation values $\mu_u(\alpha[q, l])$, $\mu_u(\beta[q, l])$, $\mu_u(b[m, \sigma_h])$ and $\mu_u(\pi[m])$ from the current probability parameters and from the auxiliary probabilities F_u, B_u, U_u, D_u.

$$\mu_u(\alpha[q, l]) = \frac{1}{L(T_u)} \sum_{p:C_u(p)\neq\emptyset} D_u(q, p)b[q, o_p^u]\alpha[q, l]B_u(l, j)$$
$$(\text{s.t. } x_j^u = x_{\leftarrow}^u(p))$$

$$\mu_u(\beta[q, l]) = \frac{1}{L(T_u)} \sum_{j:X_u(j)\neq\emptyset} F_u(q, j)\beta[q, l]B_u(l, j+1)U_u(q, j)$$

$$\mu_u(b[m, \sigma_h]) = \frac{1}{L(T_u)} \sum_{i:o_i^u=\sigma_h} D_u(m, i)U_u(m, i),$$

$$\mu_u(\pi[m]) = \frac{1}{L(T_u)}\pi[m]U_u(m, 1),$$

M-step:

We update the probability parameters using these expectation values computed in the E-step:

$$\hat{\alpha}[q, l] = \frac{\sum_u \mu_u(\alpha[q, l])}{\sum_u \sum_{l'} \mu_u(\alpha[q, l'])},$$

$$\hat{\beta}[q, l] = \frac{\sum_u \mu_u(\beta[q, l])}{\sum_u \sum_{l'} \mu_u(\beta[q, l'])},$$

$$\hat{b}[m, \sigma_h] = \frac{\sum_u \mu_u(b[m, \sigma_h])}{\sum_u \sum_i \mu_u(b[m, \sigma_i])},$$

$$\hat{\pi}[m] = \frac{\sum_u \mu_u(\pi[m])}{\sum_u \sum_k \mu_u(\pi[k])}.$$

We repeat these E- and M-steps alternately until a certain convergence criterion is satisfied[1]. A possible criterion is that the increase of the likelihood at an iteration is less than a minimal threshold value.

Figure 8 is sample pseudocode for this EM algorithm for OTMM. This algorithm starts with parameter initialization and likelihood $L(\mathbf{T})$ computation using the initial parameter values (lines 1-2). The parameters can be randomly

[1]Note that it has been proven that the EM algorithm theoretically converges to a local maximum solution [15].

1: **for each** $\theta_{i,j}$ **do** Initialize θ;
2: Calculate $L^{(0)}(\mathbf{T})$ using the initial θ;
3: $t := 0$;
4: **repeat**
5: **for each** $\theta_{i,j}$ **do** $\mu_{\mathbf{T}}(\theta_{i,j}) := 0$;
6: **for** $u := 1$ **to** $|\mathbf{T}|$ **do**
7: **for** $k := |V_u|$ **downto** 1 **do** /* B-up & R-to-L DP */
8: **for each** $q \in S$ **do** Calculate $U_u(q, k)$;
9: **for each** $m \in S$ **do** Calculate $B_u(m, k)$;
10: **for** $k := 1$ **to** $|V_u|$ **do** /* T-down & L-to-R DP */
11: **for each** $q \in S$ **do** Calculate $D_u(q, k)$;
12: **for each** $m \in S$ **do** Calculate $F_u(m, k)$;
13: **for each** $\theta_{i,j}$ **do** Calculate $\mu_u(\theta_{i,j})$;
14: **for each** $\theta_{i,j}$ **do** $\mu_{\mathbf{T}}(\theta_{i,j}) := \mu_{\mathbf{T}}(\theta_{i,j}) + \mu_u(\theta_{i,j})$;
15: **for each** $\theta_{i,j}$ **do** Update $\theta_{i,j}$ using $\mu_{\mathbf{T}}(\theta_{i,j})$;
16: $t := t + 1$;
17: Calculate $L^{(t)}(\mathbf{T})$ using the current θ;
18: **until** $|L^{(t)}(\mathbf{T}) - L^{(t-1)}(\mathbf{T})| < \epsilon$
19: **output** θ;

Figure 8: Pseudocode of the EM algorithm for OTMM.

initialized, satisfying $\sum_j \theta_{i,j} = 1$ for all i. We then start repeating the E- and M-steps (lines 3-4). In each iteration, we first initialize the expectation value to $\mu_{\mathbf{T}}(\theta_{i,j}) := 0$ (line 5). In the E-step, we compute the upward and backward probabilities by a bottom-up and right-to-left dynamic programming procedure (lines 7-9) and the downward and forward probabilities by a top-down and left-to-right dynamic programming procedure (lines 10-12). We then update the expectation values $\mu_{\mathbf{T}}(\theta_{i,j})$ (lines 13-14). In the M-step, we update the probability parameters using the expectation values (line 15). At the final part of the iteration, we compute the likelihood $L(\mathbf{T})$ (line 17) and confirm whether some fixed convergence criterion is satisfied (line 18). After the iterations, we output the probability parameters learned from the given trees (line 19).

3.3 Parsing (Finding the Most Likely State Transition)

Once the model is trained, we need to retrieve what was learned by retrieving the most likely state paths. Let Z_u^* ($= \{q_1^*, \cdots, q_{|V_u|}^*\}$) be the most likely state transition of a given tree T_u. We first reformulate the upward and backward probabilities to find the maximum probabilities, $\phi_u^U(q, p)$ and $\phi_u^B(m, j)$, respectively. That is, $\phi_u^U(q, p)$ is the maximum probability that all labels of subtree $t_u(p)$ are generated and the state of node p is s_q (maximum probability of $U_u(q, p)$), and ϕ_u^B is the corresponding maximum probability for $B_u(m, j)$. In order to obtain these two "maximum probability" values, we first replace the $\sum$ in Equations (1) and (2) with max and then iteratively compute these probabilities using the bottom-up and right-to-left dynamic programming procedure. This procedure is generally called the *Viterbi* algorithm, and the probability that all labels are outputted along the most likely state transition is given as $P^* = \max_l \pi[l]\phi_u^U(l, 1)$.

We further define two functions $\psi_u^U(q, p)$ and $\psi_u^B(m, j)$, each of which returns the most likely state for the given node, and it can be computed by replacing $\sum$ in (1) and (2) with $arg\max$. The most likely state transition starts with the most likely state for the root, which can be obtained by $q_1^* = arg\max_l \pi[l]\phi_u^U(l, 1)$. We can then retrieve the most likely state transition by computing the following equations in the top-down and left-to-right manner as done for the forward and downward probabilities: $q_j^* = \psi_u^U(q_p^*, p)$ if $x_j = x_{\leftarrow}^u(p)$; otherwise $q_j^* = \psi_u^B(q_{j-1}^*, j-1)$.

Table 1: Time and space complexity

	Time	Space
U	$O(\|\mathbf{T}\| \cdot \|S\|^2 \cdot \|V\|)$	$O(\|S\| \cdot \|V\|)$
B	$O(\|\mathbf{T}\| \cdot \|S\|^2 \cdot \|V\|)$	$O(\|S\| \cdot \|V\|)$
F	$O(\|\mathbf{T}\| \cdot \|S\|^2 \cdot \|V\|)$	$O(\|S\| \cdot \|V\|)$
D	$O(\|\mathbf{T}\| \cdot \|S\|^2 \cdot \|V\|)$	$O(\|S\| \cdot \|V\|)$
$\mu(a)$	$O(\|\mathbf{T}\| \cdot \|S\|^2 \cdot \|V\|)$	$O(\|S\|^2)$
$\mu(b)$	$O(\|\mathbf{T}\| \cdot \|S\| \cdot \|V\|)$	$O(\|S\| \cdot \|\Sigma\|)$
$\mu(\pi)$	$O(\|\mathbf{T}\| \cdot \|S\| \cdot \|V\|)$	$O(\|S\|)$
$\hat{a}$	$O(\|\mathbf{T}\| \cdot \|S\|^2)$	$O(\|S\|^2)$
$\hat{b}$	$O(\|\mathbf{T}\| \cdot \|S\| \cdot \|V\|)$	$O(\|S\| \cdot \|\Sigma\|)$
$\hat{\pi}$	$O(\|\mathbf{T}\| \cdot \|S\|)$	$O(\|S\|)$

Table 2: Time and Space Complexity Comparison of HTMM and PSTMM

	Time
OTMM	$O(\|\mathbf{T}\| \cdot \|S\|^2 \cdot \|V\|)$
HTMM	$O(\|\mathbf{T}\| \cdot \|S\|^2 \cdot \|V\|)$
PSTMM	$O(\|\mathbf{T}\| \cdot \|S\|^3 \cdot \|V\| \cdot \|C\|)$

	Space
OTMM	$\max\{O(\|S\| \cdot \|V\|), O(\|S\|^2), O(\|S\| \cdot \|\sum\|)\}$
HTMM	$\max\{O(\|S\| \cdot \|V\|), O(\|S\|^2), O(\|S\| \cdot \|\sum\|)\}$
PSTMM	$\max\{O(\|S\|^2 \cdot \|V\|), O(\|S\|^3), O(\|S\|^2 \cdot \|\sum\|)\}$

3.4 Time and Space Complexity

Table 1 summarizes the time and space complexity of the above algorithms for OTMM, including the likelihood computation and parameter estimation. We note that the complexity of the parsing is the same as that of the likelihood computation. As clearly shown in the table, the time complexity of the most time consuming parts is $O(|\mathbf{T}| \cdot |S|^2 \cdot |V|)$, and the space complexity is upper-bounded by $\max\{O(|S| \cdot |V|), O(|S|^2), O(|S| \cdot |\sum|)\}$.

Table 2 shows the comparison of the time and space complexities between OTMM and the other two tree Markov models, HTMM and PSTMM. This table illustrates that the space and time complexity of OTMM are kept at the same as those of HTMM and are significantly lower than those of PSTMM.

Furthermore, we note that algorithms used for estimating probability parameters of probabilistic models for more general (complex) objects like graphs have higher computational complexity. For example, both the time and space complexity of the junction tree algorithm [13] for probabilistic inference in a Bayesian network reaches $O(|\mathbf{T}| \cdot |S|^3 \cdot |V|)$ for a directed acyclic graph with $|V|$ nodes. Thus we can say that our model and its learning algorithm can provide a relatively low complexity by confining them in labeled ordered trees.

4. EXPERIMENTAL RESULTS

We examined the performance of our proposed model on both synthetic and real datasets, comparing it with that of PSTMM in terms of predictive accuracy and computation time. The predictive accuracy was computed in a supervised learning manner. That is, we trained the model using a set of positive examples and examined the ability of the trained model to discriminate the positive examples from the negatives. Using the real dataset, we confirmed the validity of our method by examining the results from a variety of biological viewpoints.

4.1 Synthetic Data

4.1.1 Data Generation Procedure

The models were trained using positive examples only, and so we synthetically generated three datasets, i.e. positive training, positive test and negative test datasets. We kept the size of each of these three datasets the same, which is denoted by $|\mathbf{T}|$, and in our experiments, we tested various values of $|\mathbf{T}|$ and numbers of states $|S|$. We set $|\sum| = 10$ and $|V_u| = 20$ in all of our experiments.

Each positive example contains a tree fragment as a pattern. Figure 9 shows the six tree fragments Q1 to Q6 that we used in our experiments. In each tree fragment in Figure 9, the solid circle indicates a fixed label, and the dashed circle indicates that the label is randomly selected. So, for example, in Q5, the labels of the eldest and third siblings are fixed, whereas the labels of the parent and the second sibling are randomly applied. We generated K different label patterns for each tree fragment. In positive example generation for a particular tree fragment pattern, a positive example was generated by the following two steps. First, we generated a random tree by iteratively: randomly generating zero to five children and randomly assigning a label to each of the children, until the number of generated nodes reached twenty. Second, we randomly embedded one of the K label patterns of the tree fragment into the tree. A negative example was generated in the same manner as the first step above, except that the random generation of labels follows the distribution of parent-child labels in the positive examples.

4.1.2 Performance Comparison with PSTMM

We evaluated the discriminative performance of the models by AUC, the area under the ROC (Receiver Operator Characteristic) curve [7, 8]. We can compute the AUC by first sorting examples by their computed likelihoods and then by using Equation (3).

$$\text{AUC} = \frac{R_n - \frac{n_n \cdot (n_n+1)}{2}}{n_n \cdot n_p}, \tag{3}$$

where n_n (n_p) is the number of negative (positive) examples and R_n is the sum of the ranks of the negative examples. We note that $n_n = n_p$ in our experiments.

We first evaluated the amount of overfitting to the training data that occurred, using Q1 as the tree fragment. In this experiment, $|\mathbf{T}| = 100$ and $K = 1$, such that the complexity of the data was relatively low and the tendency to overfit would be higher. Figure 10 shows the AUC for the training[2] and test datasets, setting the range of $|S|$ between two and twelve. The AUCs of the two models for the training examples increased with $|S|$ and reached around 100% when $|S|$ was eight or more. On the other hand, the AUCs for the test examples decreased with $|S|$. In particular, the AUC of PSTMM for the test data went down to 70% from around 95%, which was the highest obtained when $|S|$ was four. This phenomenon clearly illustrates overfitting to the training data. A similar tendency was found with the AUC of OTMM but it was always more than 85%, which was around 15% better than the worst AUC of PSTMM. Thus, can say that OTMM reduced the overfitting problems of PSTMM. We can infer from these results that PSTMM with more than four states is too complex to be trained from the dataset we used in this experiment and that OTMM is more appropriate for this dataset.

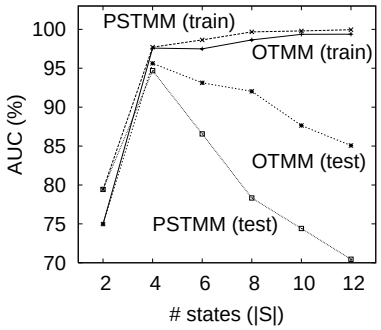

Figure 10: Performance comparison of OTMM with PSTMM when $|\mathbf{T}| = 100$.

Figure 11 shows how the AUC and computation time for the test examples change with different values of $|\mathbf{T}|$ and $|S|$, using $K = 3$ and Q1 as the tree fragment. We note that the complexity of the dataset increases with $|\mathbf{T}|$, and the complexity of the model increases with $|S|$. So when $|\mathbf{T}|$ was relatively small, say 100, and $|S|$ was large, say ten, overfitting occurred. However, when $|\mathbf{T}|$ was large, say 400 and 600, and $|S|$ was small or a moderate size, say 2 to 6, the AUCs remained at the maximum, meaning that overfitting was avoided. Under such conditions, e.g. $|\mathbf{T}| = 400$ and $|S| = 6$, we can see that the two models achieved almost the same predictive performance, indicating that OTMM kept approximately the same predictive power as that of PSTMM. On the other hand, regarding the computation time, the two models were clearly different. For example, for $|\mathbf{T}| = 400$ and $|S| = 10$, the computation time of PSTMM reached 4,000 seconds while that of OTMM was just under 1,000 seconds. OTMM clearly reduces the computational cost of PSTMM greatly, keeping its predictive power and avoiding overfitting. This result indicates that OTMM is more practical for mining from large datasets of labeled ordered trees compared to PSTMM.

We next fixed $|\mathbf{T}|$ at 200 and changed K, still using $Q1$ as the tree fragment. As $|\mathbf{T}|$ increases with K, so does the complexity of the data, and so the results in Figure 12 are similar to those in Figure 11. However, we note that in this case, under the conditions when overfitting did not occur, OTMM achieved better performance than PSTMM. That is, when K was set between two and four, we found that PSTMM achieved the highest AUC at $|S| = 6$, where overfitting did not occur, and that the AUC of OTMM was clearly better than that of PSTMM. This indicates that in this experi-

[2]We used the negative examples in the test set to compute the AUC of the training data.

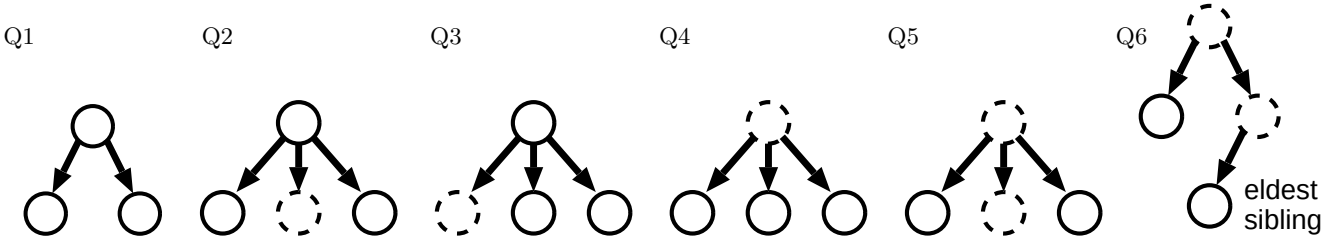

Figure 9: Six patterns of tree fragments used in our experiments.

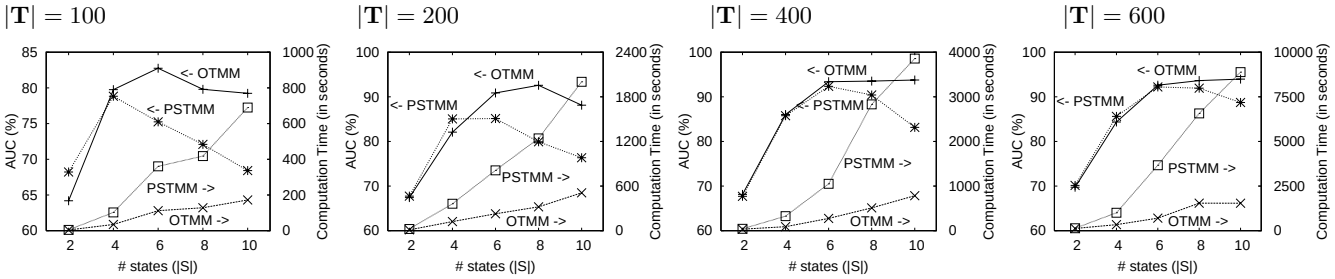

Figure 11: AUC and computation time for $K = 3$.

mental setting, OTMM had better predictive power than PSTMM.

Finally, we checked the predictive performance and computation time of the two models for all the six tree fragments, Q1 to Q6. The parameter settings we used in this experiment were $K = 3$, $|\mathbf{T}| = 400$ and $|S| = 6$, the same conditions as when overfitting did not occur for Q1 using both OTMM and PSTMM. Table 3 shows the AUC and computation time. The predictive performances of the two models were almost equivalent except for Q6, where the AUC of OTMM was roughly seven percent better than that of PSTMM. Thus we can say that OTMM has almost the same or better predictive power compared to PSTMM. On the other hand, the difference in computation time between the two models was significant. In Table 3, we indicate the ratio of the computation time of OTMM to that of PSTMM in parentheses, which shows that the typical computation time improved by a factor of at least three to at most six, which increases as $|S|$ increases. Consequently, we have clearly demonstrated the advantage of OTMM in computation time and that it is more practical than PSTMM for mining complex labeled ordered trees.

4.2 Real Data: Carbohydrate Sugar Chains

4.2.1 About the Data

We used real data derived from glycobiology (an overview of this field is provided in a book by Varki *et al.* [20]), which is the study of carbohydrate sugar chains, or glycans. Glycans can be modeled as branched and directed tree structures. The basic component of glycans is the monosaccharide unit (or sugar), which corresponds to a label, and each sugar may have one or more child sugars bound to it, such that they are ordered. Thus glycans can be considered labeled ordered trees. The glycans we used in our experiments are all derived from the KEGG GLYCAN database [9, 10].

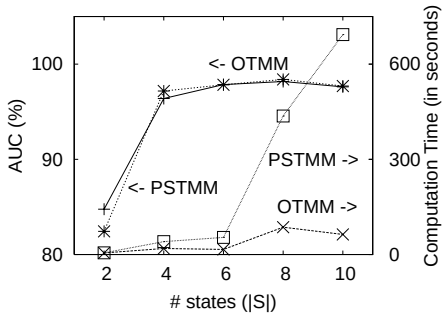

Figure 13: Performance comparison of OTMM with PSTMM using real datasets.

4.2.2 Performance Comparison with PSTMM

Glycans are classified based on their structural properties, and we selected two of the major classes called "N-Glycans" and "O-glycans," which were used as the positive and negative datasets, respectively. We examined whether N-Glycans could be discriminated from O-Glycans by the model trained using N-Glycans. We performed a five-fold cross-validation to evaluate the performance of OTMM and PSTMM. That is, we randomly divided a given dataset into five blocks of roughly equal size. Then the first block would be reserved as test data while the remaining four were used as training data. This was repeated for each of the five blocks. We repeated this random division five times, and the results were averaged over the total 25 ($= 5 \times 5$) runs. We used those glycans containing five to eighteen sugars only, because the dataset of the other glycans was very small. In total, we used 1,826 N-Glycans and 606 O-Glycans. In this experiment, $|S| = 6$ since the predictive performance of OTMM and PSTMM was always the best under this setting in the synthetic data experiments.

Figure 13 shows the AUC and computation time of the two models for the real datasets. The two models achieved approximately the same AUC values for all $|S|$ we used. However, the amount of computation time of OTMM was much smaller than that of PSTMM. That is, at $|S| = 8$, the computation time improvement reached a factor of approximately six to seven. Thus, from the real datasets, we

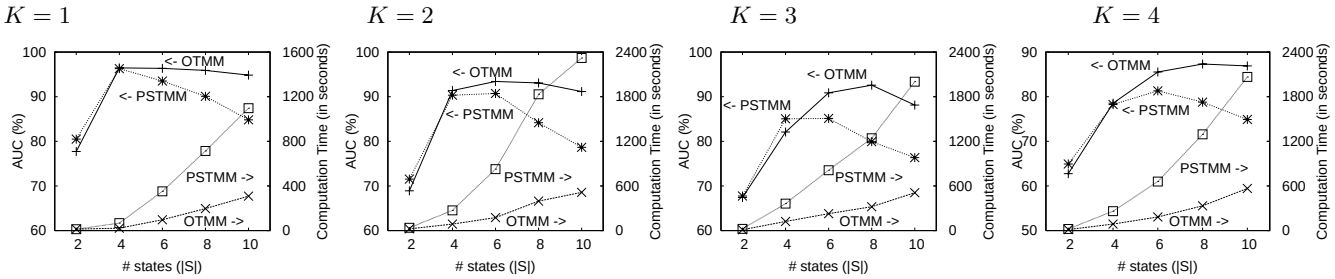

Figure 12: AUC and computation time for $|\mathbf{T}| = 200$.

Table 3: AUC and computation time for the six tree fragments in Figure 9.

	Model	Q1	Q2	Q3	Q4	Q5	Q6
AUC	OTMM	**93.4**	87.2	88.6	**96.6**	**81.8**	**82.0**
(%)	PSTMM	92.3	**89.9**	**91.8**	95.0	79.9	75.2
Comp. Time	OTMM	**438.7** (0.204)	**583.8** (0.269)	**608.8** (0.309)	**379.5** (0.193)	**829.8** (0.239)	**581.4** (0.179)
(in seconds)	PSTMM	2145.6 (1.0)	2173.8 (1.0)	1970.8 (1.0)	1961.9 (1.0)	3475.4 (1.0)	3257.1 (1.0)

confirmed that the time efficiency of OTMM was consistent with the results using the synthetic datasets.

4.2.3 *Results with Biological Significance*

Next, we verified the actual patterns learned in the data to verify the biological characteristics retrieved. We focus on the set of N-Glycan structures because it is the most well-studied among all the glycan classes. Using the N-Glycan data set from the previous experiment, we extracted the most likely state paths and found some interesting patterns reflecting the biological properties of sugars. Figure 14 is an example a glycan tested and the most likely state transitions for it. Here we see reflected the fact that the Gal (galactose) residues and GalNAc (N-acetylgalactosamine) residues end up with the same state. This corresponds with the fact that GalNAc is a modification of Gal, such that they may be considered very similar and sometimes appear in place of each other. Thus they may be aligned together as well. A rather large number of glycans have this pattern of sugars at the leaves, where Gals may be GalNAcs and vice versa.

In another trial, we assessed the most likely state transitions for glycans representing the three sub-classes of N-Glycans: High-mannose, Complex, and Hybrid, given in Figure 15. Here we see these three classes being distinguished based on the states learned. In particular, the states at the tri-mannose core distinguishes these three sub-classes. The High-mannose type is characterized by state 2 appearing at both child mannoses. For the Complex type, these mannoses are both state 1. Consequently, the Hybrid type, which is a hybrid of these two types having one each of a High-mannose type subtree and a Complex-type subtree, receives state 1 on one mannose, and state 2 on the other. Thus, the states learned at just the core N-Glycan structure can determine the sub-class without needing to actually traverse the rest of the structure.

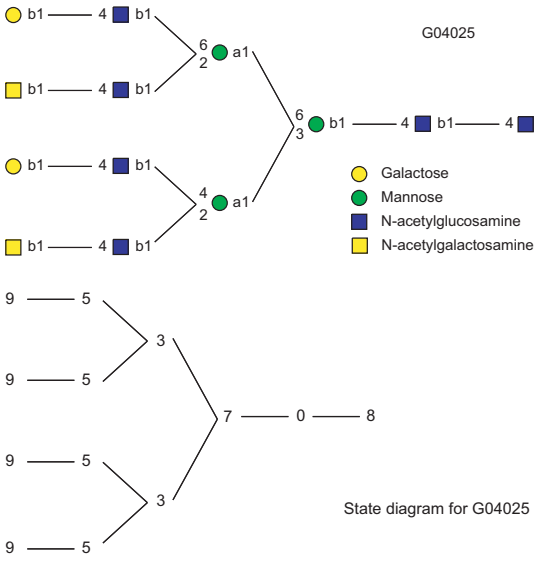

Figure 14: Example of the states learned using OTMM for a specific glycan structure.

5. CONCLUDING REMARKS

In summary, we have developed a new probabilistic model, OTMM, and an efficient learning scheme for mining labeled ordered trees. We empirically evaluated the effectiveness of OTMM using both synthetic and real datasets and found that in all of the experimental settings we conducted, OTMM achieved equivalent or better performance compared to the more complex probabilistic model, PSTMM, for mining labeled ordered trees. In particular, the amount of computation time of OTMM is significantly less than that of PSTMM in all experimental settings.

The key property of our method that contributes to these improvements is its concise model structure, in which the state of a node depends on only one state, and the efficient learning algorithm by which the model captures both the sibling and parent-child dependencies in labeled ordered trees. Thus, we have developed the ultimate model in terms of efficiency and accuracy for mining labeled ordered trees.

6. REFERENCES

[1] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web: from relations to semistructured data and XML.* Morgan Kaufmann, 2000.

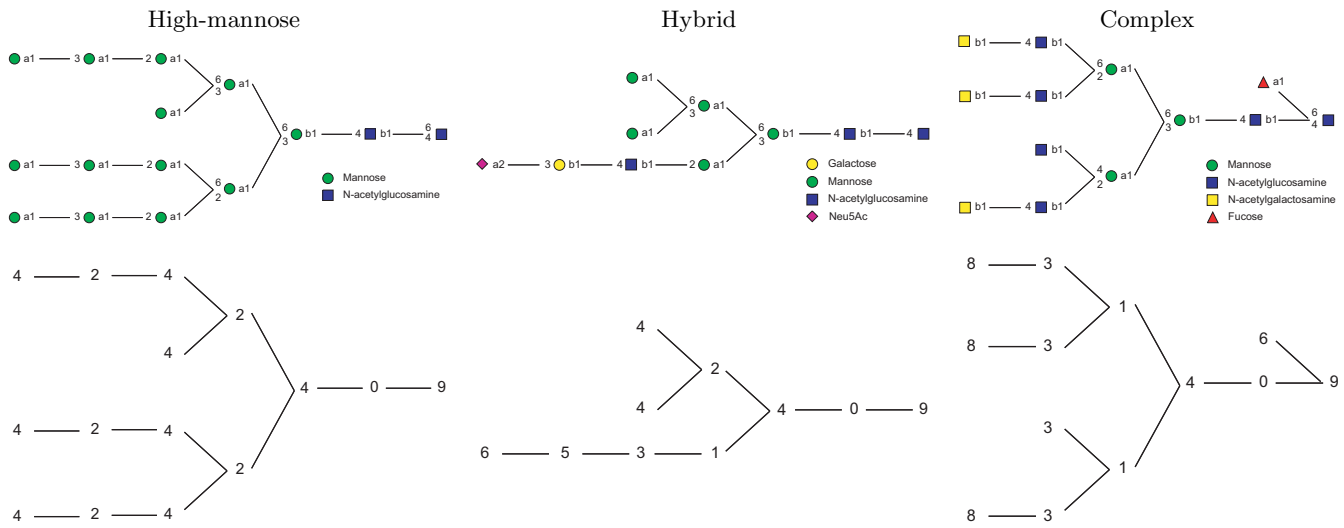

Figure 15: (top) The actual glycans, and (bottom) the most likely state paths.

[2] K. F. Aoki, N. Ueda, A. Yamaguchi, T. Akutsu, M. Kanehisa, and H. Mamitsuka. Managing and analyzing carbohydrate data. *ACM SIGMOD Record*, 33(2), 2004.

[3] E. Baum and T. Petrie. Statistical inference for probabilistic functions of infinite state Markov chains. *Ann. Math. Stat.*, 37:1554–1563, 1966.

[4] S. Chakrabarti. *Mining the Web: Analysis of Hypertext and Semi Structured Data*. Morgan Kaufmann, 2002.

[5] A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *J. Royal Stat. Soc., B*, 39:1–38, 1977.

[6] M. Diligenti, P. Frasconi, and M. Gori. Hidden tree Markov models for document image classification. *IEEE Trans. PAMI*, 25(4):519–523, 2003.

[7] D. J. Hand and R. J. Till. A simple generalisation of the area under the ROC curve for multiple class classification problems. *Mach. Learn.*, 45:171–186, 2001.

[8] J. A. Hanley and B. J. McNeil. The meaning and use of the area under a receiver operating characteristic (ROC) curve. *Radiology*, 143:29–36, 1982.

[9] K. Hashimoto, S. Goto, S. Kawano, K. Aoki-Kinoshita, N. Ueda, M. Hamajima, T. Kawasaki, and M. Kanehisa. KEGG as a glycome informatics resource. *Glycobiology*, 16(5):63R–70R, 2005.

[10] M. Kanehisa, S. Goto, M. Hattori, K. F. Aoki-Kinoshita, M. Itoh, S. Kawashima, T. Katayama, M. Araki, and M. Hirakawa. From genomics to chemical genomics: New developments in KEGG. *Nucl. Acids Res.*, 34:D354–D357, 2006.

[11] H. Kashima and T. Koyanagi. Kernels for semi-structured data. In *Proc. 19th ICML*, pages 291–298, 2002.

[12] K. Lari and S. J. Young. The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language*, 4:35–56, 1990.

[13] S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems (with discussion). *J. Royal Stat. Soc. B*, 50(2):157–224, 1988.

[14] C. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.

[15] G. J. McLachlan and T. Krishnan. *The EM Algorithm and Extensions*. Wiley-Interscience, 1996.

[16] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

[17] L. R. Rabiner and B. H. Juang. An introduction to hidden Markov models. *IEEE ASSP Magazine*, 3(1):4–16, 1986.

[18] N. Ueda, K. F. Aoki, and H. Mamitsuka. A general probabilistic framework for mining labeled ordered trees. In *Proc. Fourth SDM*, pages 357–368, 2004.

[19] N. Ueda, K. F. Aoki-Kinoshita, A. Yamaguchi, T. Akutsu, and H. Mamitsuka. A probabilistic model for mining labeled ordered trees: Capturing patterns in carbohydrate sugar chains. *IEEE Trans. Know. Data Eng.*, 17(8):1051–1064, 2005.

[20] A. Varki, R. Cummings, J. Esko, H. Freeze, G. Hart, and J. Marth, editors. *Essentials of Glycobiology*. Cold Spring Harbor Laboratory Press, New York, 1999.

[21] S. M. Weiss, N. Indurkhya, T. Zhang, and F. J. Damerau. *Text Mining: Predictive Methods for Analyzing Unstructured Information*. Springer, 2005.

[22] M. Zaki and C. Aggarwal. Xrules: An effective structural classifier for XML data. In *Proc. Ninth ACM KDD*, pages 316–325, 2003.

[23] M. J. Zaki. Efficiently mining frequent trees in a forest: Algorithms and applications. *IEEE Trans. Know. Data Eng.*, 17(8):1021–1035, 2005.