DEWS2003 6-C-01

半構造データ系列のオンライン予測とXMLデータ圧縮への応用

河野正太郎† 有村 博紀† 有川 節夫†

† 九州大学大学院システム情報科学府・研究院
〒 812–8581 福岡市東区箱崎 6–10–1
E-mail: †{s-kawano,arim,arikawa}@i.kyushu-u.ac.jp

あらまし 離散データ系列からのオンライン予測は，パターン発見や，例外検出，遺伝子配列解析，データ圧縮などの広い応用をもつ．本稿では，半構造データ系列からのオンライン予測問題を考察し，半構造データ系列に対する予測モデル XPST（XML Prediction Suffix Tree）を提案する．XPST モデルは，1 次元の離散データ系列に対する予測モデル PST（Prediction Suffix Tree）を 2 次元の半構造データに自然に拡張したものになっている．さらに，XPST のオンライン予測アルゴリズムを与え，このアルゴリズムが限定されない XML データストリームにおいて効率よく働くことを示す.

キーワード XML，半構造データ，テキスト DB，情報検索，データ圧縮，PPM

Online Prediction of Semi-structured Data Streams and Its Application to XML Data Compression

Shotaro KAWANO†, Hiroki ARIMURA†, and Setsuo ARIKAWA†

† Department of Informatics, Kyushu University
Hakozaki 6–10–1, Higashi-ku, Fukuoka, 812–8581, Japan
E-mail: †{s-kawano,arim,arikawa}@i.kyushu-u.ac.jp

Abstract Discrete sequence prediction has a wide range of applications, e.g., sequence pattern discovery, anomaly detection, biological sequence analysis, data compression. This paper studies online prediction of XML data streams. We present a probabilistic prediction model for XML data streams, called XPST (XML Prediction Suffix Tree), which is a natural generalization of variable-length context-based prediction model PST (Probabilistic suffix tree) for semi-structured data streams. Then, we develop an efficient online prediction algorithm for XPST based on the online suffix tree/trie construction technique. Finally, we show that the prediction and update time of this algorithm is proportinal to the depth of XPST.

Key words XML, Semi-structured data, Text database, Information retrieval, Data compression, PPM

1. はじめに

近年，膨大な量の電子データがネットワーク上で交換・蓄積されるようになってきており，XML [7] に代表される半構造データが，その基盤技術として注目されている．現在，半構造データの蓄積や，交換，圧縮，構造変換，検索のための様々な技術が盛んに研究されている [1].

本稿では，半構造データ系列に対するオンライン予測アルゴリズムを提案する．一般に，離散データ系列からのオンライン予測アルゴリズムは，ストリームから文字を読み込み，現在までに読み込んだ文字列から出現確率を予測し，モデルの更新を行うというプロセスを繰り返して動き続ける．離散データ系列のオンライン予測と，系列学習，データ圧縮は密接な関係があり，効率と精度の高いアルゴリズムを得ることができれば，次のような問題に応用可能である．

- 系列予測．現在の予測モデルを用いて次に出現する文字を予測する．
- データ圧縮．予測を用いて受け取った文字を符号化する [6], [10].
- 系列の自動分類．訓練例により予測モデルを訓練しておき，その予測モデルを用いてクラスが未知の系列を分類する．
- 例外検出．オンラインで離散データ系列を学習しながら，予測損失の変化を監視し，急激な増大によってトレンド変化や異常データの到着を検出する [18].

これらの離散データ系列予測問題に対して，高い予測精度を達成可能であり，モデルの構造が単純で，計算効率も良い，新し

いタイプのオンライン系列予測アルゴリズムが提案されている [4]〜[6], [8], [10], [11]．これらのアルゴリズムは，ストリームから現在までに受け取った文字列である文脈 σ を用いて，次の文字 a の出現確率 $P(a\,|\,\sigma)$ を予測するという方式を採用している．さらに，系列アルゴリズムの技術を取り込むことで，従来の確率モデルや圧縮法と比較して長い文脈を扱うことができ，これにより，広いクラスの離散データ系列に対して，高い予測精度の達成に成功している．

1.1 本稿の結果

本稿では，文脈を用いた離散データ系列予測を，XML データに代表される半構造データに拡張することを目標とする．そのために，離散データ系列の予測モデルの 1 つである PST（Prediction Suffix Tree）モデル [11] を，XML データと DOM 木データモデルに拡張して，XML データに対する確率的予測モデル XPST（XML Prediction Suffix Tree）を提案する．

XPST モデルは，1 次元の文字列に対する PST モデルを，2 次元の木構造をもつ XML データへ自然に拡張したものである．XPST モデルでは，オンライン予測アルゴリズムは，XML データストリームから文字を読み込みながら，新しい文字に対応する DOM 木中の節点のパス文脈 π とテキストの文脈 σ を同時に管理し，2 種類の文脈に基づいて，新しい文字 a の出現確率 $P(a\,|\,\pi,\sigma)$ を予測する．

まず，PST モデルに基づいて XPST モデルを定義する．次に，PST のオンライン版と考えられる離散データ系列予測モデル TDAG（Transition Directed Acuclic Graph）[8]，および，独立に提案されたテキスト圧縮法 PPM（Prediction with Partial Match）[10] の機構を半構造データに一般化して，XPST のオンライン構築アルゴリズム XPST_ONLINE を開発する．解析では，XPST_ONLINE が，各時点においてモデルの高さ H の線形時間で予測と更新を行なうことを示す．さらに，予測のみをおこなう場合には，モデルのサイズと無関係に，$O(1)$ ならし時間で予測可能であることを示す．

また，XPST モデルを用いた圧縮プログラムを実装し，実際の XML 文書に適用した実験について示す．この実験の結果，PST モデルを用いた圧縮プログラムと比較して，同程度の計算時間で，より高い圧縮率を得ることができた．

1.2 関 連 研 究

半構造データ圧縮は，本研究の可能な応用の一つである．XML 等の半構造データは，人間にとって内容を理解し易く，ネットワーク上の通信の共通フォーマットとして有用であるが，専用フォーマットと比較してかさばり易い．そのため，XML 技術を半構造データのネットワーク上での交換・蓄積のための基盤技術として用いるには，高速かつ高圧縮率の半構造データ圧縮アルゴリズムが必要である．さらに，アルゴリズムは，広いクラスのデータに対して，パラメータ調整が不要で，容易に安定して使えるものである必要がある．

最近，Liefke と Suciu [9] は，XML データに対する圧縮システム Xmill を提案した．この Xmill は，XML データのパス情報を利用することで，様々な従来型の圧縮アルゴリズムを使い分け，圧縮率を飛躍的に高めることができる．文献 [9] では，XML データに適用した場合に，Xmill が代表的な辞書式圧縮手法である gzip と比較して，同程度の計算時間で，2 倍程度に及ぶ圧縮率を達成したと報告されている．一方，Xmill は固定したパス情報を利用しており，高い圧縮率を得るには，利用者がパス情報をうまく指定する必要がある．また，Xmill では，タグの圧縮に XML がもつ構造を利用していない．これと対照的に，本稿で提案する XPST モデルとオンライン予測アルゴリズムは，構造と内容に応じて XML データストリームの適応的圧縮を行なう技術であり，Xmill を補完する技術であると考えられる．

1.3 本稿の構成

第 2 章では，XML 文書と DOM 木についての準備を行なう．第 3 章では，PST モデルとそのオンライン構築アルゴリズムについて述べる．第 4 章では，XML 文書に対する系列予測モデル XPST を導入する．第 5 章では，XPST モデルのオンライン構築アルゴリズムを与え，その計算量の解析を与える．第 6 章では，XPST モデルを用いた適応型圧縮について述べる．第 7 章では，XPST モデルを用いた圧縮プログラムの実装と，圧縮・展開プログラムを実際の XML 文書に適用した実験について述べる．最後に，8 章で本稿の結論と今後の課題について述べる．

2. 準　　備

2.1 アルファベットと文字

アルファベット Σ に対して，Σ の要素数を $|\Sigma|$ で表す．Σ 上の文字列全体の集合を Σ^* で表し，空文字列を ε で表す．文字列を $s \in \Sigma^*$ に対して，$s = xyz$ となる文字列 x を s の**接頭辞**（prefix），文字列 y を s の**部分文字列**（substring），文字列 z を s の**接尾辞**（suffix）とよぶ．文字列 $s \in \Sigma^*$ の接頭辞全体の集合を $Prefix(s)$，接尾辞体の集合を $Suffix(s)$ で表す．また，文字列の集合 $S \subseteqq \Sigma^*$ に対して，$Prefix(S) = \bigcup_{s\in S} Prefix(s)$，$Suffix(S) = \bigcup_{s\in S} Suffix(s)$ と定義する．文字列集合 S に対して，$Prefix(S) \subseteqq S$（$Suffix(S) \subseteqq S$）であるとき，S は接頭辞（接尾辞）演算で閉じているという．

2.2 文字列の頻度

アルファベット Σ 上のストリーム $\mathcal{I} \in \Sigma^*$ に対して，文字列 $s \in \Sigma^*$ がストリーム $\mathcal{I}$ の部分文字列であるとき，文字列 s はストリーム $\mathcal{I}$ に出現するという．ストリーム $\mathcal{I}$ に文字列 s が出現する回数を**頻度**（count）と呼び，$count(s)$ で表す．また，ストリーム $\mathcal{I}$ に対して，文字列 $s \in \Sigma^*$ を読み込んだ直後の文字 $a \in \Sigma$ の出現確率を

$$P(a|s) = \frac{count(sa)}{\sum_{x\in\Sigma} count(sx)}$$

と定義する．

2.3 XML 文書と DOM 木

名前集合を L とする．名前 $n \in \Gamma$ をもつ**開始タグ**（start tag）を <n> で表す．任意の開始タグに対して，対応する唯一の**終了タグ**（end tag）を </ETG> で表す．ETG は L に含まれない終了タグのラベルである．タグ集合を $\Gamma = \{\text{<}n\text{>}\}_{n\in L} \cup \{\text{</ETG>}\}$ で表す．また，Σ をテキスト文字集合とする．

文字集合を $\Delta = \Sigma \cup \{\texttt{<}n\texttt{>}\}_{n \in L} \cup \{\texttt{</ETG>}\}$ とおく．このとき，Δ 上の **XML 文書**（XML document）は，*Document* を開始記号とする次の文脈自由文法を用いて生成される文字列 $X \in \Delta^*$ である．

$$
\begin{array}{lcl}
\mathit{Document} & ::= & \mathit{Element} \\
\mathit{Element} & ::= & \texttt{<}n\texttt{>}\ \mathit{Content}\ \texttt{</ETG>}\ \ (n \in L) \\
\mathit{Content} & ::= & \mathit{Text}\ (\mathit{Element}\ \mathit{Text}\ )^* \\
\mathit{Text} & ::= & s \cdot \mathrm{ETX}\ \ (s \in \Sigma^*)
\end{array}
$$

非終端記号 *Element* は**要素**（element）を表す．XML 文書は 1 つの要素からなり，要素はテキストと要素の列からなる．この文脈自由文法によって生成される XML 文書全体の集合を $\mathcal{L}_\Delta$ で表す．この定義では，終了タグは </ETG> の 1 種類のみであり，実際の XML 文書とは異なるが，XML 文書の構文解析時に動的に変換することができる．また，簡単のため，タグの属性やコメントは含まないものとする．

XML 文書構造のモデルとして，**DOM 木**（Dom tree）と呼ぶラベルつき順序木を用いる．順序木とは子供の集合に順序が付けられた木である [3]．ラベルはそれぞれ要素とテキストに対応する．形式的には，Δ 上の DOM 木を, 6 項組 $\mathcal{D}_\Delta = (V, E, <, v_0, \Gamma, \ell)$ で表す．ここに，V は節点の集合を表し，$E \subseteqq V^2$ は節点の親子関係を，$E \subseteqq V^2$ は節点の兄弟関係を表し，(V, E, v_0) は v_0 を根とする木を形成する．$\ell : V \to \Gamma$ はラベル関数であり，節点 $v \in V$ のラベルを $\ell(v)$ で表す．

E で連結された節点列に対応するラベル列のことを**パス**（path）と呼ぶ．パスのうち，E によって連結された根 v_0 から $v \in V$ の親までの節点列に対応するラベル列のことを，ラベル $ell(v)$ の**パス文脈**（path context）と呼ぶ．同じパス文脈をもつテキストについて，現在までに読み込んだ文字列を**テキスト文脈**（text context）と呼ぶ．

XML 文書 $X \in \mathcal{L}_\Delta$ と DOM 木 $\tau(X) \in \mathcal{D}_\Delta$ は，次のように自然な対応をもつ．要素 $E = \texttt{<}n\texttt{>}\ E_1 \cdots E_n\ \texttt{</ETG>}$ に対して，根のラベルが $n \in \Gamma$ で，左から右に子供 $\tau(E_1), \ldots, \tau(E_n)$ をもつような DOM 木 $\tau(E)$ が対応する．

3. テキスト予測モデル

本章では，テキスト文字列に対するテキスト予測モデル PST (Prediction Suffix Tree) [11] について述べる．まず，PST モデルを定義し，次に，PST モデルのオンライン構築アルゴリズムと予測アルゴリズムを与える．そして，次章で，この PST モデルを部品として用いて，XML 文書に対する予測モデルである XPST を導入する．

3.1 テキスト予測モデル PST

［定義 1］ PST モデルは，5 項組

$$\mathcal{S} = (S \cup \{\bot\}, \varepsilon, g, f, \{count(x)\}_{x \in S})$$

である．ここに，$\mathcal{S}$ は次の要素からなる．

テキスト文脈集合 $S \subseteqq \Sigma^*$．S は接頭辞演算と接尾辞演算で閉じているものとする．各 $x \in S$ を**節点**（node）と呼び，節点 $\varepsilon \in S$ を**根**（root）と呼ぶ．また，$\bot$ を根の親となるような仮想的な節点とする．

遷移関数 $g : S \times \Sigma \to S$．ここに，任意の $x, y \in S, a \in \Sigma$ に対して，$g(x, a) = y$ iff $xa = y$ が成立する．また，すべての $a \in \Sigma$ に対して $g(\bot, a) = \varepsilon$ とする．S は接頭辞演算について閉じているので，g は $\varepsilon \in S$ を根とするトライ（trie）(注1) を形成する．

接尾辞リンク（suffix link）$f : S \to S \cup \{\bot\}$．任意の節点 $x, y \in S$ に対して，$f(x) = y$ iff ある $a \in \Sigma$ に対して $x = ay$ と定義する．ただし，$x = \varepsilon$ ときは $f(\varepsilon) = \bot$ とする．

各節点 $x \in S$ は，テキスト x の頻度 $count(x) \geqq 0$ を属性としてもつ．

［定義 2］ PST モデルの各節点 $x \in S \cup \{\bot\}$ に対して，Σ 上の離散確率分布 $p_x : \Sigma \to [0, 1]$ を次のように定義する．$p_\bot$ は，Σ 上の一様分布である．すなわち，任意の $a \in \Sigma$ に対して，$p_\bot(a) = 1/|\Sigma|$ である．節点 $x \in S$ に対して，x が $a \in \Sigma$ によって遷移可能なとき，

$$p_x(a) = \frac{count(x, a)}{\sum_{y \in child(x)} count(y)}$$

と定義する．ここに，$child(x)$ は節点 x から遷移可能な子節点の集合であり，$count(x, a)$ は節点 x から文字 a による遷移先の節点の頻度である．x が文字 $a \in \Sigma$ によって遷移できないとき，$p_x(a) = 0$ と定義する．明らかに $\sum_{a \in \Sigma} p_x(a) = 1$ が成立するので，p_x は正しく定義されている。

［定義 3］ S を PST モデルとする．Σ 上のストリーム $\mathcal{I}$ からテキスト文字列 $\sigma \in \Sigma^*$ を読み込んだ直後に, テキスト文字 $a \in \Sigma$ が出現する確率 $P(a|\sigma)$ を

$$P_S(a|\sigma) = p_x(a)$$

と定義する．ここに，テキスト文字列 $x \in \Sigma^*$ は, $x \in S$ を満たすテキスト文脈 σ の最長の接尾辞である．

テキスト文字集合 Σ 上のストリーム $\mathcal{I} \in \Sigma^*$ が与えられたとき，PST モデルは $\mathcal{I}$ から文字 $a \in \Sigma$ を読み込みながらオンライン構築することができる．

3.2 PST モデルのオンライン構築アルゴリズム

図 1 に PST モデルのオンライン構築アルゴリズム PST_ONLINE を示す．PST_ONLINE ではまず，図 2 の PST の生成手続き $Create()$ によって，根と補助節点からなる PST モデルを生成する．$\mathcal{I}$ からテキスト文字 $a \in \Sigma$ を読み込むと，図 3 の予測手続き $Predict(S, a)$ によって，テキスト文字 a の出現確率 $P(a|\sigma)$ を予測し，図 4 の更新手続きに $Update(S, a)$ によって，PST モデルを更新する．各手続きにおいて，$root(S)$ は PST S の根を指すポインタであり，$base(S)$ は現在のテキスト文脈に対する最長の接尾辞を指すポインタである．

3.3 拡張可能性述語

ポインタ $base(S)$ の更新において，節点 $x \in S$ を新しいテキスト文字 $a \in \Sigma$ で伸長するかどうかを，**拡張可能性述語**（extendible predicate）$Extensible(x)$ で決める．

(注1)：デジタル探索木（digital search tree）または接尾辞木（suffix tree）ともいう [3].

アルゴリズム PST_ONLINE:
入力：Σ 上のストリーム $\mathcal{I}$
出力：予測のストリーム $\mathcal{O}$
手法：

```
S := Create();
base(S) := root(S);
repeat
    I から次のテキスト文字 a ∈ Σ を読み込む;
    Predict(S, a) を O に出力;
    base(S) := Update(S, a);
```

図 1　PST のオンライン構築アルゴリズム
Fig. 1　An online algorithm for constructing PST

```
Create(): /* PST S = ({ε, ⊥}, ε, g, f, {count(ε)}) の生成 */
    節点 ε, ⊥ を生成する; count(ε) := 0;
    f(ε) := ⊥; 任意のテキスト文字 a ∈ Σ に対して, g(⊥, a) := ε;
    return S;
```

図 2　PST の生成手続き
Fig. 2　A subprocedure for creating PST

```
Predict(S, a):
    x := base(S);
    while x の子節点が存在しない do
        x := f(x);
    return p_x(a);
```

図 3　PST の予測手続き
Fig. 3　A subprocedure for prediction with PST

例として，図 5 に，PPM 圧縮 [6], [10] で用いられている，最大深さ H を用いた拡張可能性述語を示す．この拡張可能性述語は，深さ $depth(x)$ が H 以下の節点 x のみ，拡張可能であるとするものである．

この他に，TDAG [8] のように，最大深さ H に加えて，節点を訪問する頻度や，伸長による条件付確率の変化度を用いることも有用と思われる．

4. XML 文書に対する予測モデル

本章では，前章で定義したテキストに対する予測モデル PST を拡張して，XML 文書に対する予測モデル XPST（XML Prediction Suffix Tree）を導入する．ここでは，XPST を，DOM 木の節点ラベルをその文脈から予測するモデルとして定義する．そして，次章で，定義した XPST モデルのオンライン構築アルゴリズムと予測アルゴリズムを与える．

4.1 XPST の定義

XPST モデルは，XML 文書から得られるテキスト文字とタグの系列において，現在のパス文脈 π とテキスト文脈 σ に基づいて，次に出現する文字 $a \in \Delta$ の出現確率 $P(a \mid \pi, \sigma)$ を予測する確率モデルである．テキスト文字列の予測モデル PST をラベル付き順序木上のデータに拡張して，XML 文書の系列予測モデル XPST を与える．

```
Update(S, a):
    x := base(S);
    do
        if x の子節点 y = g(x, a) が存在する then
            coutn(y) := count(y) + 1;
        else if Extensible(x) then
            x の子節点 y = g(x, a) を新たに生成する;
            count(y) := 1;
            if x ≠ base(S) then f(last_y) := y;
        last_y := y;
        x := f(x);
    while x ≠⊥
    return g(base(S), a);
```

図 4　PST の更新手続き
Fig. 4　A subprocedure for updating PST

```
Extensible(x):
    if depth(x) ≦ H then return True;
    else return False;
```

図 5　拡張可能性述語の例
Fig. 5　An example of extensible predicate

［定義 4］　XPST モデルは，組

$$\mathcal{T} = (T \cup \{\perp\}, \varepsilon, g, f, \{count(x)\}_{x \in T}, \{model(x)\}_{x \in T})$$

である．ここに，$\mathcal{T}$ は次の要素からなる．

パス文脈集合 $T \subseteqq \Gamma^*$．T は接頭辞演算と接尾辞演算で閉じているものとする．各 $x \in T$ を節点と呼び，節点 $\varepsilon \in T$ を根と呼ぶ．また，$\perp$ を根の親となるような仮想的な節点とする．

遷移関数 $g : T \times \Gamma \to T$．ここに，任意の $x, y \in T, a \in \Gamma$ に対して，$g(x, a) = y$ iff $xa = y$ が成立する．また，すべての $a \in \Gamma$ に対して $g(\perp, a) = \varepsilon$ とする．T は接尾辞演算について閉じているので，g は $\varepsilon \in T$ を根とするトライを形成する．

接尾辞リンク $f : T \to T \cup \{\perp\}$．任意の節点 $x, y \in T$ に対して，$f(x) = y$ iff ある $a \in \Sigma$ に対して $x = ay$ と定義する．ただし，$x = \varepsilon$ ときは $f(\varepsilon) = \perp$ とする．

各節点 $x \in T$ は，Σ 上のテキスト予測モデル $model(x)$ を属性としてもつ．このモデル $model(x)$ はテキスト文脈に基づいて文字を予測するのに用いる．

各節点 $x \in T$ は，パス x の頻度 $count(x) \geqq 0$ を属性としてもつ．

［定義 5］　XPST の各節点 $x \in T \cup \{\perp\}$ に対して，Γ 上の離散確率分布 $p_x : \Gamma \to [0, 1]$ を次のように定義する．

$p_\perp$ は，Γ 上の一様分布である．すなわち，任意の $a \in \Gamma$ に

対して，$p_\perp(a) = 1/|\Gamma|$.

節点 $x \in T$ に対して，x が $a \in \Gamma$ によって遷移可能なとき，

$$p_x(a) = \frac{count(x,a)}{\sum_{y \in child(x)} count(y)}$$

と定義する．x が $a \in \Gamma$ によって遷移できないとき，$p_x(a) = 0$ と定義する．明らかに $\sum_{a\in\Gamma} p_x(a) = 1$ が成立するので，p_x は正しく定義されている．

与えられたパス文脈 $\pi \in \Gamma^*$ の XPST モデル T における**最長パス文脈**（longest path context）とは，節点 $x \in T$ において，(i) x が π の接尾辞であり，(ii) 長さ $|x|$ が最大となる節点 $lpc(\pi) = x \in T$ である．$lpc(\pi)$ は，常に，そして一意に存在する．

4.2 XML 文書の生成確率

［定義 6］ $X \in \mathcal{L}_\Delta$ を XML 文書とし，T を XPST モデルとする．現在のパス文脈，テキスト文脈をそれぞれ π，σ とする．このとき，XPST モデルによるテキスト文字 $a \in \Sigma$ の生成確率を，

$$P(a \mid \pi, \sigma) = P_{model(lpc(\pi))}(a \mid \sigma)$$

と定義する．これは，T の最長一致パス文脈 $lpc(\pi)$ で定まるテキスト予測モデルが返す予測値である．また，XPST モデルによるタグ a の条件付き生成確率を，

$$P(name(a) \mid \pi, \sigma) = p_{lpc(\pi)}(name(a))$$

と定義する．ここに，$name(a)$ はタグ a に対する名前を表す．

［定義 7］ XPST T による長さ $N \geqq 1$ の XML 文書データ系列 $X = a_1 a_2 \cdots a_N \in \Delta^N$ の生成確率を

$$P_T^N(X) = \prod_{i=1}^{N} p(a_i \mid \pi_i, \sigma_i)$$

と定義する．ここに，π_i と σ_i は, それぞれ文字 a_i のパス文脈とテキスト文脈である．

5. XPST のオンライン構築アルゴリズム

本章では，XPST モデルのオンライン予測アルゴリズムについて述べる．XPST モデルのオンライン予測は，PST モデルと同様に，文字を読み込み，その文字の出現確率を予測し，XPST モデルを更新するというプロセスを繰り返しながら，動きつづける．アルゴリズムの詳細についての説明の後に，計算時間の解析を与える．

5.1 アルゴリズムの詳細

図 6 に，XPST モデルのオンライン構築アルゴリズム XPST_ONLINE を示す．オンライン構築アルゴリズムは，XML 文書のストリーム $\mathcal{S} \in \Delta^*$ を受け取り，文字 $a \in \Delta$ の読み込み，出現確率 $P(a \mid \pi, \sigma)$ の予測，XPST モデル $\mathcal{T}$ の更新のプロセスを繰り返しながら動き続ける．

図 7 に XPST モデルによる予測手続き $Predict(base, a)$ を示す．文字 $a \in \Delta$ の出現確率の予測は，基本的に $base$ ポインタの指す節点がもつテキスト予測モデルによって行なう．

図 8 に XPST モデルの更新手続き $Update(Stack, base, a)$ を

アルゴリズム XPST_ONLINE
入力： XML データのストリーム $\mathcal{I}$
出力： 予測のストリーム $\mathcal{O}$
手法：

XPST T の節点 $\varepsilon, \perp$ を生成する;
$count(\varepsilon) = 0$; $model(\varepsilon) := Create()$;
$f(\varepsilon) := \perp$; 任意の名前 $a \in \Gamma$ に対して，$g(\perp, a) := \varepsilon$;
$base := \varepsilon$; $Stack := \emptyset$;
repeat
  $\mathcal{I}$ から次の文字 $a \in \Delta$ を受け取る;
  $Predict(Stack, base, a)$ を $\mathcal{O}$ に出力;
  $\langle Stack, base \rangle := Update(Stack, base, a)$;

図 6 XPST のオンライン構築アルゴリズム
Fig. 6 An online algorithm for constructing XPST

$Predict(base, a)$:
  $x := base$;
  if a がタグ **then**
    while x に子節点が存在しない **do**
      $x := f(x)$;
    return $q_x(name(a))$;
  else if a がテキスト文字 **then**
    return $Predict(model(x), a)$;

図 7 XPST の予測手続き
Fig. 7 A subprocedure for prediction with XPST

示す．予測アルゴリズムは，現在の最長パス文脈を変数 $base$ に保持しながら動作する．初期状態では，$base$ ポインタは XPST の根 ε を指しており，ストリーム $\mathcal{I}$ から文字 $a \in \Delta$ を読み込むたびに，$base$ ポインタを次のように更新する．

タグ a に対しては，パス文脈が伸長するので，XPST モデルが Aho-Corasick パターン照合機械 [2] と同形であることを利用して，常に最長一致パス文脈を指すように $base$ ポインタを更新する．つまり，先の $base$ から $name(a) \in \Gamma$ で遷移可能ならば遷移先を $base$ とし，遷移不可能なら $name(a)$ で遷移可能になるまで, 接尾辞辺で上に戻る．a が開始タグのときは，古い $base$ ポインタは，対応する終了タグが来たときのためにスタックに積む．a が終了タグのときは，パス文脈が縮むので，スタックをポップして，先頭の値を $base$ ポインタに代入する．テキスト文字 $a \in \Sigma$ に対しては，パス文脈は変化しないので，$base$ ポインタは変えない．以上の木構造の更新は，トライのオンライン構築アルゴリズム [15] をパス文字列集合に拡張したものとみることができる．

テキスト予測モデルに関しては，$base$ ポインタから始めて，接尾辞辺を根までさかのぼり，各節点 $x \in T$ がもつテキスト予測モデル $model(x)$ を更新していく．

拡張可能性述語 $Extensible(x)$ については，PST モデルと同様のものである．

```
Update(Stack, base, a):
    x := base;
    if a がタグ then
        while x ≠⊥ do
            if x の子節点 y = g(x, name(a)) が存在する then
                count(y) := count(y) + 1;
            else if Extensible(x) then
                x の子節点 y = g(x, name(a)) を新たに生成する;
                count(y) := 1;  model(y) := Create();
                if x ≠ base then f(last_y) := y;
            last_y := y;  x := f(x);
        if a が開始タグ then
            Push(base, Stack);
        else if a が終了タグ then
            Pop(Stack);  base := Top(Stack);
    else if a がテキスト文字 then
        while x ≠ ⊥ do
            Update(model(x), a);
            x := f(x);
        return ⟨Stack, base⟩;
```

図 8　XPST の更新手続き
Fig. 8　A subprocedure for updating XPST

5.2　計算時間の解析

テキスト予測モデルとして，PST モデル（TDAG [8], PPM [10]）を仮定する．このとき，アルゴリズムの構成から，次の結果が容易に示される．

［定理 1］ 任意の入力 XML 文書ストリーム $\mathcal{I}$ と，正整数 $N \geqq 0$ に対して，図 6 のアルゴリズム XPST_ONLINE の時刻 N における計算時間は $O(H)$ である．ここに，$H > 0$ は XPST モデルの節点の最大深さである．

モデルの更新を行なわず，予測のみを行なう場合には，文献 [2] と同様の議論から次のことが成立する．

［定理 2］ 予測のみを行なう場合，任意の時刻 $N \geqq 0$ までの，図 6 のアルゴリズム XPST_ONLINE の総計算時間は $O(N)$ であり，予測 1 回あたりの計算時間は $O(1)$ である．

6.　XPST を用いた XML 文書の適応型圧縮

適応型圧縮法（adaptive compression method）は．次に出現する文字を予測する確率モデル $\mathcal{M}$ と，予測された確率を符号化・復号化するエンコーダ $\mathcal{E}$ を用いて圧縮を行う手法である．本章では，予測モデルとして XPST，符号化に算術符号（arithmetic coding）[16] を用いた適応型圧縮アルゴリズムについて説明する．以下では，テキスト文字集合 Σ 上と名前集合上 Γ に全順序 $<$ を仮定する．

6.1　XPST による確率区間の計算

データ圧縮では，ゼロ確率問題と呼ばれる問題が存在する．予測モデルの節点 x で文字 a を予測する場合，節点 x から文字 a による遷移がないとき，予測モデルによって求められる a の出現確率が 0 になるという問題である．このような場合，文字 a を算術符号化することができない．この問題を解決するために，PPM 圧縮 [6], [10] と同様に，**エスケープ文字**（escape symbol）ESC を導入する．エスケープ文字とは，任意の文字 $a \in \Sigma \cup \Gamma$ に対して $a <$ ESC を満たす文字 ESC $\notin \Sigma \cup \Gamma$ である．エスケープ文字はデータ中に出現しないので，その出現確率を明示的に与える必要がある．以下では，XPST モデルにおけるエスケープ文字を含む任意の文字の確率区間を定義する．

予測モデル $\mathcal{M}$ の各節点 x について，節点 x から文字 ESC によって遷移可能であり，遷移先の子節点の頻度は $count(x, \mathrm{ESC}) = |child(x)|$ [(注2)] であると定める．$\mathcal{M}$ の節点 x から遷移可能な文字の集合 $\{a_1, \ldots, a_N, a_{N+1} = \mathrm{ESC}\}$ 中の文字 $a_k (k = 1, \ldots, N+1)$ に対する確率は，次の 3 つの数で表すことができる．

$$low(x, a_k) = \sum_{i=1}^{k-1} count(x, a_i)$$

$$high(x, a_k) = \sum_{i=1}^{k} count(x, a_i)$$

$$total(x) = \sum_{i=1}^{N+1} count(x, a_i)$$

つまり，$\mathcal{M}$ の節点 x における文字 a_k の確率区間は次のようになる．

$$\left[\frac{low(x, a_k)}{total(x)}, \frac{high(x, a_k)}{total(x)} \right)$$

6.2　算 術 符 号

エンコーダ $\mathcal{E}$ として，算術符号を用いる．ここでは，算術符号は副手続きとして用いるので，手続き呼びだしのインタフェースだけを与える．手続きの実装については，教科書 [12], [17] を参考されたい．

算術符号のエンコーダは以下の手続きを与える．

- $Encode(low, high, total)$：low，$high$，$total$ の 3 つの数で表される確率区間を符号化し，出力ストリームに書き込む．

算術符号のデコーダは次の 2 つの手続きを与える．

- $Index(total)$: $total$ の値から，次に復号化される文字に対応する区間 $[0, total)$ の間の値を返す．
- $Decode(low, high, total)$: 符号化された low, $high$, $total$ の 3 つの数で表される確率区間をストリームから削除する．

6.3　XPST による文字の符号化

以上の確率モデルとエンコーダを組み合わせて，適応型算術符号化と復号化のアルゴリズムを与える．確率モデル $\mathcal{M}$ として，XPST モデルとその各節店がもつテキスト予測モデルを用いる．エンコーダ $\mathcal{E}$ として，算術符号を用いる．

図 9 に文字の符号化手続き $ArithmeticEncode(x, a)$ を，図 10 に文字の復号化手続き $ArithmeticDecode(x)$ を示す．符号化アルゴリズム $ArithmeticEncode(x, a)$ では，$\mathcal{M}$ の節点 x から文字 a によって遷移できない間，文字 ESC の確率区間を

(注2)：このようなエスケープ文字の頻度の定義に基づく PPM を PPMC [10] という．

ArithmeticEncode(x, a)
while x の子節点 xa が存在しない **do**
$Encode(high(x, \mathrm{ESC}),\ low(x, \mathrm{ESC}),\ total(x))$
$x = f(x)$;
$Encode(high(x, a),\ low(x, a),\ total(x))$;

図 9　文字の符号化手続き
Fig. 9　A procedure of encoding symbol

ArithmeticDecode(x):
do
$Index(total(x)) \in [low(x, a), high(x, a))$ となる文字 a を見つける;
$Decode(high(x, a),\ low(x, a),\ total(x))$;
$x = f(x)$;
while a = ESC

図 10　文字の復号化手続き
Fig. 10　A procedure of decoding symbol

符号化し，接尾辞リンクをたどるという動作を繰り返す．

これとは逆に，復号化アルゴリズム *ArithmeticDecode*(x) では，$\mathcal{M}$ の節点 x において，$Index(total(x))$ によって見つけた文字が ESC である間，文字 ESC の確率区間を符号化したものをストリームから削除し，接尾辞リンクをたどるという動作を繰り返す．

7. 実　　験

本章では，XPST のオンライン構築アルゴリズムを用いた圧縮・伸張プログラム xpst の実装について述べ，実装した圧縮・伸張プログラムを実際に XML 文書に適用した実験について報告する．実験は，OS Windows2000，プロセッサ PentiumIII 650MHz，メインメモリ 256MB という環境で行った．

7.1　圧縮プログラムの実装

圧縮プログラム xpst は，すべて Java（JDK1.4.0）によって実装した．算術符号化の部分については，パッケージ com.colloquial.arithcode (注3) を用いた．このパッケージの中には，算術符号のエンコーダであるクラス ArithEncoder と，デコーダであるクラス ArithDecoder が用意されている．また，適応型圧縮に用いるモデルのインタフェースとして *ArithCodeModel* が用意されている．XPST モデルとその各節点がもつテキスト予測モデルをこのインタフェースで実装した．拡張可能性述語には図 5 と同じものを採用し，XPST の節点の拡張は高さ 2 で制限し，テキスト予測モデルはすべて高さ 4 で制限するようにした．

圧縮は，XML 文書を SAX [13] パーサによって解析し，XPST モデルを構築しながら，エンコーダによって符号化を行う．展開は，圧縮のときと同じ XPST モデルを再構築しながら，デコーダによって復号化を行う（図 11）．

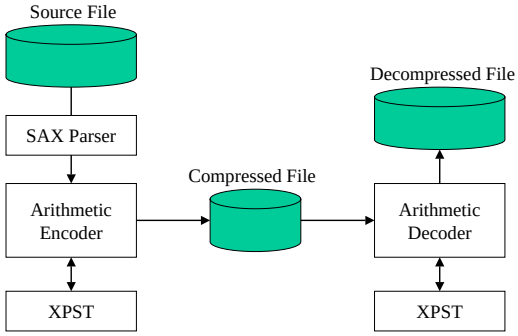

図 11　XPST を用いた圧縮・展開
Fig. 11　Compression/decompression using XPST

表 1　実験データ
Table 1　Data Sources

Data Source	File Size	Tag	Text	Depth	Tags
soap_list.xml	93.4KB	85.1%	14.9%	3	11
much_ado.xml	195.1KB	37.1%	62.9%	4	17
periodic.xml	87.1KB	70.7%	29.3%	2	20
weblog.xml	2919.9KB	62.5%	37.5%	2	12

7.2　実験データ

実験データには，表 1 に示す，性質の異なる 4 種類のデータを用いた(注4)．表における Depth は XML 文書に対応する DOM 木の深さであり，Tags は XML 文書中のタグの種類の数である．Tag と Text は XML 文書全体に対するタグとテキストの割合である．データの説明を以下に示す．

（1）　soap_list.xml．人工的に生成された SOAP [14] 形式の XML 文書．短いテキストを内容とする要素からなる平坦な構造をもつ．

（2）　much_ado.xml．シェークスピアの劇「から騒ぎ（Much Ado about Nothing）」の脚本を XML 文書化したもの．比較的長いテキストを内容とする要素からなり，複雑な構造をもつ．

（3）　periodic.xml．元素の周期表を XML 文書化したもの．主に短いテキストを内容とする要素からなり，複雑な構造をもつ．

（4）　weblog.xml．XML 形式のウェブサーバのアクセスログファイル．ページヒットを表す約 10000 の要素からなり，それぞれの子要素はテキストを内容とする複数の要素からなる．

7.3　pst と xpst の比較実験

PST を予測モデルとする適応型圧縮プログラム pst を実装し，圧縮率と圧縮・展開時間について，xpst との比較を行った．

7.3.1　圧　縮　率

実験データを pst と xpst によって圧縮した結果を図 12 に示す．図における圧縮率（compression ratio）は，次の式によって求められる値である．

(注3)：Compression via Arithmetic Coding in Java 1.0. http://www.colloquial.com/ArithmeticCoding/.

(注4)：XML Benchmark Documents. http://www.sosnoski.com/opensrc/xmlbench/documents.html.

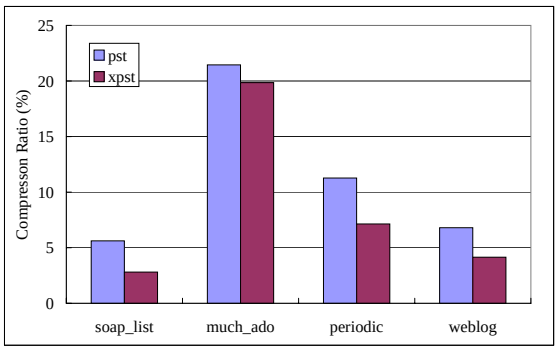

図 12　pst との圧縮率の比較

Fig. 12　Compression Ratio

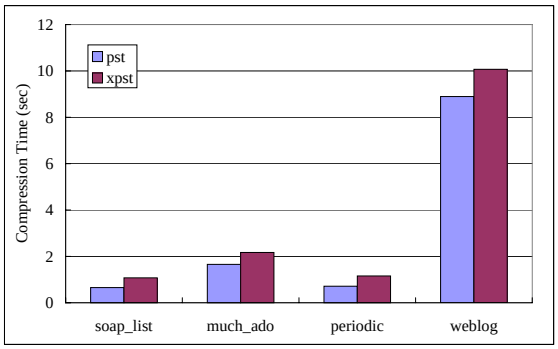

図 13　pst との圧縮時間の比較

Fig. 13　Compression Time

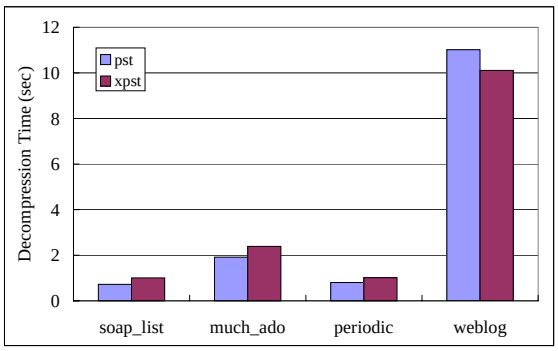

図 14　pst との展開時間の比較

Fig. 14　Deompression Time

$$\text{圧縮率 (\%)} = \frac{\text{圧縮されたファイルのサイズ}}{\text{元のファイルサイズ}} \times 100$$

結果より，すべての実験データについて，pst に比べて xpst の方が高い圧縮率を得られることが分かる．よって，DOM 木のパス文脈を用いた予測精度が高いと考えられる．

7.3.2　圧 縮 時 間

pst と xpst による圧縮時間を図 13 に，展開時間を図 14 に示す．図より，すべての実験データについて，pst とほぼ同程度の時間で圧縮・展開できることが分かる．

8. おわりに

本稿では，半構造データストリームからのオンライン系列予測問題を考察し，XML データストリームのための予測モデル XPST と，そのオンライン構築アルゴリズムを提案した．また，XPST モデルを用いた圧縮プログラムを実装し，複数の XML 文書に対する実験を行った．

その結果，テキストの予測モデルである PST を用いた圧縮プログラムと比較して，同等の計算時間で，高い圧縮率を得ることがきた．したがって，本稿で提案した XPST モデルは，PST モデルと比べて，予測精度の高いモデルであるということが分かった．

8.1　今後の課題

本稿では，XPST モデルが対象とする XML 文書は，属性やコメントを含まないものに限定していた．よって，今後の課題は，属性やコメントを含む，より一般的な XML 文書に対する予測モデルへと XPST を拡張することである．

また，XPST モデルでは，次の文字の予測に，DOM 木における節点の親子関係であるパスを文脈として用いていたが，これに加えて節点の兄弟関係も文脈として考えることにより，予測の精度をさらに上げることができると予測される．

さらに，XPST モデルを用いた XML 文書の自動分類や，トレンド解析，情報抽出等への応用も今後の課題である．

文　　献

[1] S. Abiteboul, P. Buneman, and D. Suciu. Data on the Web. Morgan Kaufmann, 2000.

[2] A. V. Aho, M. J. Corasick. Efficient string matching: an aid to bibliographic search. *CACM*, 18(6):333–340, 1975.

[3] A. V. Aho, J. E. Hopcroft, and J. Ullman. The Design and Analysis of Computer Algorithms. Addison-Wesley, 1974.

[4] A. Apostolico and G. Bejerano. Optimal amnesic probabilistic automata or how to learn and classify proteins in linear time and space. In *Proc. RECOMB2000*, 25–32, ACM Press, 2000.

[5] G. Bejerano and G. Yona. Modeling protein families using probabilistic suffix trees. In *Proc. RECOMB'99*, 15–24, ACM Press, 1999.

[6] J. G. Cleary and I. H. Witten. Data compression using adaptive coding and partial string matching. *IEEE Trans. Commun.* COM-32(4): 396–402, 1984.

[7] Extensible Markup Language (XML) 1.0, Second Edition, `http://www.w3.org/TR/REC-xml`.

[8] P. Laird and R. Saul. Discrete sequence prediction and its applications. *Machine Learning*, 15(1): 43–68, 1994.

[9] H. Liefke and D. Suciu. XMill: an ecient compressor for XML data. In *Proc. SIGMOD2000*, 153–164, 2000.

[10] A. Moffat. Implementing the PPM data compression scheme. *IEEE Trans. Commun.* COM-38(11): 1917–1921, 1990.

[11] D. Ron, Y. Singer, and N. Tishby. The power of amnesia: learning probabilistic automata with variable memory length. *Machine Learning*, 25(2-3): 117– 149, 1996.

[12] D. Salomon. Data Compression: The Complete Reference, Secondd Edition. Springer-Verlag, 1998.

[13] Simple API for XML (SAX) 2.0. `http://www.saxproject.org/`.

[14] Simple Object Access Protocol (SOAP) 1.1. `http://www.w3.org/TR/SOAP/`.

[15] E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3), 249–260, 1995.

[16] I. H. Witten, R. Neal, and J. G. Cleary. Arithmetic coding for data compression. *CACM*, 30(6): 520-540, 1987.

[17] I. H. Witten, A. Moffat, and T. C. Bell. Managing Gigabytes, Second Edition. Morgan Kaufmann, 1999.

[18] K. Yamanishi and J. Takeuchi. A unifying framework for detecting outliers and change points from non-stationary time series data. In *Proc. SIGKDD2002*, 2002.