# 大規模半構造データストリームからの知識発見

安部　賢治† 　川副　真治† 　浅井　達哉† 　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: †{k-abe,s-kawa,t-asai,arim,arikawa}@i.kyushu-u.ac.jp

**あらまし**　本稿では，半構造データストリームからのデータマイニング問題を考察する．我々は，半構造データとパターンのモデルとしてラベルつき順序木を採用し，与えられた半構造データストリームから，任意の時点で現在の頻出パターンを出力するオンライン型の半構造データマイニングアルゴリズム StreamT を開発する．このアルゴリズムは，無限長の半構造データストリームに対して，少ない計算領域しか用いずに高速に動作する．さらに，本稿では，候補集合の管理手法について詳細に議論する．我々は複数の候補集合管理戦略を提案し，それぞれの戦略におけるアルゴリズムの振る舞いを実験にて検証した．また，アルゴリズムに忘却の概念を導入した結果，過去のデータに影響されず，データの変化に柔軟に追従することを実証した．

**キーワード**　XML, データストリーム, 半構造データマイニング, 頻出パターン発見, 忘却型オンラインアルゴリズム

# Knowledge Discovery from Large Semi-structured Data Streams

Kenji ABE†, Shinji KAWASOE†, Tatsuya ASAI†, Hiroki ARIMURA†, and Setsuo ARIKAWA†

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

**Abstract**　In this paper, we study an online data mining problem from a stream of semi-structured data such as XML data. Modeling semi-structured data and patterns as labeled ordered trees, we present an online algorithm StreamT that receives fragments of an unseen possibly infinite semi-structured data in the document order through a data stream, and can return the current set of frequent patterns immediately on request at any time. Moreover, we discuss on a candidate management policy of StreamT. We present some candidate management policies and empirically study behavior of algorithms with each policy. Experiments show that the algorithm of forgetting model computes really current frequent patterns from the data stream, without influenced by past events.

**Key words**　XML, data stream, semi-structured data mining, frequent pattern discovery, online algorithm with forgetting factor

## 1. はじめに

データマイニングとは，データベースに蓄積された大量のデータから，自明でない規則性やパターンを半自動的にとりだす方法についての科学的研究であり，ビジネス分野や科学技術分野などのさまざまな対象分野で，その適用が盛んに行なわれている．また，高速なネットワークと安価な大容量記憶装置の発達によって，ウェブページや XML 文書に代表される半構造データ [3], [21] がネットワーク上に蓄積されており，大規模半構造データを対象としたデータマイニング（半構造データマイニング）に関する研究が盛んに行なわれつつある [1], [2], [6], [10], [16], [17], [22], [24].

一方，高速ネットワークとウェブ技術の発達によって，近年，電子商取引やウェブサイト管理など，新しい形態のネットワーク上の応用プログラムが普及し始めている．これらの応用プログラムでは，データは静的なレコードの集合ではなく，膨大な半構造データがネットワーク上を流れ続けるデータストリームの形態をしていることが特徴である．今後，このような**大規模半構造データストリーム**を対象とした，効率のよいデータマイニング手法の重要性が増すものと予想される．

本稿で考察する半構造データストリームは，

（1） **膨大な量**のデータが，
（2） **時間的に変化**しながら，
（3） **連続して**流れ続ける

（4） **高速な**データストリーム

であることが特徴である．よって，このような大規模半構造データストリームを対象とするアルゴリズムは，制限された計算資源しか使わずに長時間働き続け，任意の時点で有益な近似解を提供できることが望ましい．このアルゴリズムが実現すれば，ネットワークログからの異常検出や XML データストリームからのトレンド検出などへの応用が期待できる．

我々は，半構造データとパターンのモデルとしてラベルつき順序木を採用し，先に開発した順序木の効率よい枚挙法 [6] を用いて，半構造データストリームをラベルつき順序木の節点列として定式化する．これは，データストリーム上を，巨大な XML データが連続的に配送される場合をモデル化している．そして，半構造データストリームに対する頻出パターン発見アルゴリズム StreamT を開発する [7]．このアルゴリズムでは，掃木枝上のボトム出現を逐次的に更新することにより，パターンの出現位置を漸増的に計算している．この手法と Hidber [12] による候補集合の管理戦略を組み合わせることにより，アルゴリズムは無限のデータストリームに対して有限の資源しか用いずに効率的に働く．

本稿では，候補集合の管理手法について，さらに詳しく議論する．我々は複数の管理戦略を提案し，それぞれに理論的な特徴づけを与え，実データ上での実験により各々の管理戦略を用いたのアルゴリズムの振る舞いを検証する．また，忘却の概念を用いて過去の影響力が指数的に逓減するような仕組みを構築し，忘却型のアルゴリズムが過去のデータに影響されず，データの変化に柔軟に追従することを実証する．

本稿の構成は次のとおりである．2. 節で，半構造データストリームからのオンラインマイニング問題の定式化を与え，3. 節では，それを解くオンラインアルゴリズム StreamT [7] について述べる．4. 節では，様々な候補集合管理戦略について，詳細に議論する．5. 節では実データを用いた実験を行い，忘却型アルゴリズムと新たに提案する候補集合管理戦略の有効性を評価する．最後に 6. 節で本稿をまとめる．

### 1.1 関 連 研 究

半構造データベース [3], [21] の普及にともない，半構造データマイニングの研究が盛んに行なわれるようになってきた [6], [10], [16], [17], [22], [24]．これらの多くは，Apriori [4] 風にパターンの探索を行なっている．それに対して，Zaki [24] と Asai ら [6] は，それぞれ独立に，半構造データからの効率的な順序木パターン探索法（**最右拡張**）を開発した．これは，Bayardo [8] による集合枚挙木を用いたアイテムセット探索法の自然な拡張である．本稿で提案するアルゴリズムは，部分的に最右拡張の概念を用いるものの，上述 [6], [24] の半構造データマイニング手法とはまったく異なることに注意されたい．

最近，オンライン型データマイニングの研究が盛んになってきている [11], [15], [19]．Hidber [12] は，トランザクションデータのストリームに対するオンライン型の相関規則マイニングアルゴリズムを開発した．Parthasarathy ら [18] は，時系列パターンを発見するオンラインマイニングアルゴリズムを与えている．また，Mannila ら [15] は，エピソードと呼ばれる特殊な時系列パターンの発見問題を考察している．Yamanishi ら [23] は，効率的なオンライン異常検出システム SmartSifter を開発した．このシステムはネットワークへの侵入検出などに応用されている．

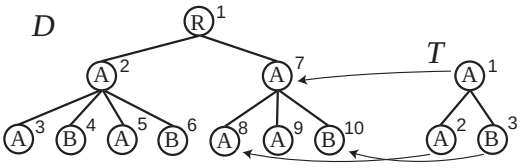


図 1　データ木 $D$ とパターン木 $T$

## 2. 定　式　化

### 2.1 半構造データとラベルつき順序木

本稿では，XML 文書などの半構造データ [3] をラベルつき順序木 [5] でモデル化する．

**ラベルつき順序木**　$\mathcal{L} = \{\ell, \ell_0, \ell_1, \ldots\}$ をラベルの有限集合とする．半構造データベースとパターンの形式的なモデルとして，ラベルつき順序木を用いる．ラベルは半構造データの属性名，タグつきテキストのタグに対応する．順序木とは子供の集合に順序が付けられた木である．形式的には，$\mathcal{L}$ 上のラベルつき順序木を次の五項組 $T = (V, E, B, L, v_1)$ で定義する．$G = (V, E, v_1)$ は $v_1$ を根とする木である．$V$ は節点の集合を，$E \subseteqq V^2$ は親子関係を表す．$L : V \to \mathcal{L}$ はラベル関数を，二項関係 $B \subseteqq V^2$ は $G$ における兄弟関係を表す．

ラベルつき順序木 $T$ の大きさを，節点数 $|T| = |V|$ と定義する．節点 $v \in V$ の深さを，根節点 $v_1$ から $v$ への経路に含まれる辺の数と定義する．$T$ の節点集合は $V_T = \{1, \ldots, k\}$ であり（$k = |T|$），これらは $T$ の前置順巡回（preorder traversal）で番号付けられていると仮定する．このとき，$T$ の根は $v_1 = 1$ であり，一番右の葉（最右葉）$rml(T)$ は $rml(T) = k$ となることに注意されたい．$T$ の最右枝を $RMB(T)$ であらわす．

**パターンの出現**　$T$ と $D$ を $\mathcal{L}$ 上の順序木とする．ここで，$T$ を**パターン木**，$D$ を**データ木**と呼ぶことにする．以下の条件を満たす関数 $\varphi : V_T \to V_D$ が存在するとき，$T$ が $D$ に出現すると定義する：(i) $\varphi$ は単射であり，(ii) $\varphi$ は親子関係，兄弟関係とラベルの値を保存する．ただし，パターン木 $T$ 中で隣接する兄弟が，それらの出現先で隣接する必要はない．この $\varphi$ を，$T$ から $D$ への**マッチング関数**と呼ぶ．

以後，パターン木 $T$ を単にパターンと呼ぶ．

$D$ をデータ木，$T$ を大きさ $k$ のパターン，$\varphi$ を $T$ から $D$ へのマッチング関数とする．$\varphi$ に関する $T$ の $D$ への**埋め込み**（embedding）を，$\varphi$ による $T$ の像 $\varphi(T) \subseteqq V_D$ と定義する．また，$\varphi$ に関する**ルート出現**（root occurrence）を $roc(\varphi) = \varphi(1)$ と，**最右葉出現**（rightmost-leaf occurrence）を $rmo(\varphi) = \varphi(k)$ と定義する．

例として，図 1 のデータ木 $D$ とパターン $T$ を考える．図中の矢印は，$D$ への埋め込みが $E_T = \{7, 8, 10\}$ となるようなマッチング関数 $\varphi$ を表している．$T$ の $D$ への全ルート出現は節点 2 と節点 7 の 2 種類であり，最右葉出現は節点 $4, 6, 10$ の 3 種

類である．

### 2.2 ラベルつき順序木のストリーム表現

本節では，ラベルつき順序木の記号列による表現を導入する．この表現は，データストリームから XML 文書が，テキスト順に順次読み出される場合に対応する．

［定義 1］ $\mathcal{L}$ 上のラベルつき順序木 $T$ の**深さ・ラベル対表現**とは，記号列 $s(T) = ((d_1, \ell_1), \ldots, (d_k, \ell_k))$ のことである．ここに，$(d_i, \ell_i) \in \mathbf{N} \times \mathcal{L}$ は，$T$ のプリオーダー順で $i$ 番目の節点 $v_i$ の深さとラベルの組であり，$k = |T|$ である．

例えば，図 1 のデータ木 $D$ に対する深さ・ラベル対表現は $s(D) = ((0,R),(1,A),(2,A),(2,B),(2,A),(2,B),(1,A),(2,A),(2,A),(2,B))$ となる．次で定義する**最右拡張** [6] を用いることにより，任意のラベルつき順序木 $T$ に対して，その深さ・ラベル対表現 $s(T)$ が一意に定まることが分かる．

［定義 2］ $S$ を大きさ $k-1$ のラベルつき順序木，$d_{max} \geqq 0$ を $S$ の最右葉 $v_{k-1} = k-1$ の深さとする．任意の自然数 $0 \leqq d \leqq d_{max}+1$ と任意のラベル $\ell \in \mathcal{L}$ に対して，$S$ の $(d,\ell)$ 拡張とは，$S$ にラベル $\ell$ をもつ新たな節点 $v_k = k$ を追加して得られるラベルつき順序木 $T$ である．ただし，$S$ の最右葉 $v_{k-1}$ の先祖で深さが $d-1$ であるような節点を $z$ とするとき，$v_k$ は $z$ の一番右の子として追加する．空木 $\perp$ の $(d,\ell)$ 拡張を，ラベル $\ell$ をもつ大きさ 1 の順序木と定義する．

$T$ が $S$ の $(d,\ell)$ 拡張であるとする．このとき $T$ は $S$ の**前者**，$S$ は $T$ の**後者**という．また，上の定義より，明らかに $v_k$ は $T$ の最右葉となる．したがって，$s$ には次の逆変換 $s^{-1}$ が存在する：(i) 空列 $\varepsilon$ に対して $s^{-1}(\varepsilon) = \perp$ であり，(ii) 正の長さをもつ深さ・ラベル対の列 $(\mathcal{S}, (d,\ell))$ に対して，$s^{-1}(\mathcal{S}, (d,\ell))$ は $s^{-1}(\mathcal{S})$ の $(d,\ell)$ 拡張である．

### 2.3 データマイニング問題

集合 $A$ に対して，$A^\infty$ は $A$ 上のすべての無限列をあらわす．

$D$ をデータ木とする．$D$ の深さは有限であると仮定する（幅は無限であってもよい）．$D$ に対する**半構造データストリーム**（semi-structured data stream）とは，無限列 $\mathcal{S} = s(D) = (\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_i, \ldots) \in (\mathbf{N} \times \mathcal{L})^\infty$ のことである．ここに，$\boldsymbol{v}_i = (d_i, \ell_i)$ は $D$ の $i$ 番目の節点 $v_i = i$ の深さ・ラベル対表現である．このとき，$\boldsymbol{v}_i$ を $\mathcal{S}$ の $i$ 番目の節点と呼ぶことにする．我々は節点 $v_i = i$ と時刻 $i$ を同一視する．

我々のデータマイニング問題を以下に記す．

**半構造データストリームからのオンライン型頻出パターン発見問題**

**入力：** 半構造データストリーム $\mathcal{S} = (\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_i, \ldots)$，実数 $0 < \sigma \leqq 1$（**最小サポート**）．

**問題：** 時刻 $i$ で $\boldsymbol{v}_i$ が入力されると仮定する（$i \geqq 1$）．このとき，出現頻度が $\sigma$ 以上となるような頻出パターン $T$ を，各時刻 $i$ において出力せよ．

オンライン型マイニングアルゴリズムの目標は，無限のデータストリームに対して有限の資源しか用いずに半永久的に働き，任意の時点で利用者の質問に迅速に応えることである．

パターンの出現頻度として，我々は以下の 3 つのモデルを定義する．ただし，$1 \leqq j \leqq i$ に対して，$hit_j^{(i)}$ は次のように定義される関数である．もし $v_j \in V_D$ が $T$ の $D_i$ へのルート出現ならば $hit_j^{(i)}(T) = 1$ であり，それ以外のときは $hit_j^{(i)}(T) = 0$.

**Algorithm** StreamT$(\mathcal{L}, \mathcal{S}, \sigma)$

**入力**: ラベル集合 $\mathcal{L}$，データ木 $D$ に対応する半構造データストリーム $\mathcal{S} = s(D) = (\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_i, \ldots)$，最小サポート $0 < \sigma \leqq 1$.

**出力**: パターン集合の列 $(\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_i, \ldots)$（$i \geqq 1$）．ただし，任意の $j \geqq 1$ に対して $\mathcal{F}_j \subseteqq \mathcal{T}$.

**変数**: 候補集合 $\mathcal{C} \subseteqq \mathcal{T}$，掃木枝スタック $B = (B[0], \ldots, B[Top])$.

**手法**:

1. $\mathcal{C}$ を大きさ 1 の全パターンの集合とする;
   $B := \emptyset$, $Top := -1$, $i := 1$;
2. 次の節点 $\boldsymbol{v}_i = (d, \ell)$ が存在するかぎり，以下を行なう：

2–(a) 掃木枝スタックのバケット $B[0], \ldots, B[d-1]$ を更新する：
   $(B, EXP) :=$ UpdateB$(B, \mathcal{C}, (d,\ell), i)$;

2–(b) 候補集合 $\mathcal{C}$ とバケット $B[d]$ を更新する：
   $(B, \mathcal{C}) :=$ UpdateC$(EXP, B, \mathcal{C}, (d,\ell), i)$;

2–(c) $\mathcal{F}_i = \{\, T \in \mathcal{C} \mid freq_i(T) \geqq \sigma \,\}$ を出力する; $i = i+1$;

図 2　オンライン型半構造データマイニングアルゴリズム

［定義 3］ $T$ を任意のパターン，$\mathcal{S}$ を半構造データストリームとする．このとき，時刻 $i$ における $T$ の**頻度** $freq_i(T)$ を，以下のように定義する．

**基本型**　このモデルでは，時刻 $i$ における $T$ の $D_i$ への異なるルート出現数 $count_i(T)$ を数える．

$$freq_i(T) = \tfrac{1}{i} count_i(T) = \tfrac{1}{i} \textstyle\sum_{j=1}^{i} hit_j^{(i)}(T). \qquad (1)$$

**忘却型**　これは，過去のデータの影響力を，過ぎ去った時間に対して指数的に逓減させるモデルである [13], [23]．$0 < \gamma < 1$ を実数（**忘却定数**）とし，$Z_i = \sum_{j=1}^{i} \gamma^{i-j}$ とおく．

$$freq_{\gamma,i}^{\mathrm{fg}}(T) = \tfrac{1}{Z_i} \textstyle\sum_{j=1}^{i} \gamma^{i-j}\, hit_j^{(i)}(T). \qquad (2)$$

## 3. オンライン型マイニングアルゴリズム

本節では，基本型モデルに対するオンラインマイニングアルゴリズム StreamT を与える．

### 3.1 アルゴリズムの概要

我々の提案するアルゴリズム StreamT を，図 2 に示す．このアルゴリズムは，各ステージ $i = 1, 2, \ldots$ において，半構造データストリーム $\mathcal{S}$ から節点 $\boldsymbol{v}_i = (d_i, \ell_i)$ を受け取り，候補パターンの頻度を計算する．アルゴリズムは，利用者からの要請に応じて，ステージ $i$ で $\sigma$ 以上の頻度をもつ頻出パターンの集合 $\mathcal{F}_i \subseteqq \mathcal{T}$ を出力する．

データストリームから継続的に頻出パターンを計算するために，アルゴリズムはデータ木 $D$ の上で**掃木枝**（sweep branch）と呼ばれる $D$ の根 $v_1$ から現在の節点 $v_i$ への経路を右へずらしていき，掃木枝と交わるすべてのパターンを探索する．これを実現するために，アルゴリズムはステージごとに以下の情報を更新する：

- **候補集合** $\mathcal{C} \subseteqq \mathcal{T}$.
- **掃木枝スタック** $B = (B[0], B[1], \ldots, B[Top])$.

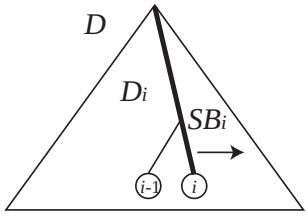


図 3　時刻 $i$ における掃木枝 $SB_i$

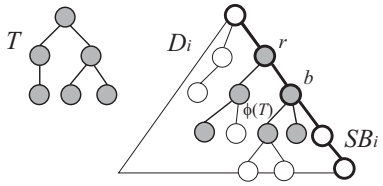


図 4　掃木枝 $SB_i$ に関する $T$ のルート出現 $r$ とボトム出現 $b$

任意の候補パターン $T \in \mathcal{C}$ は，$T$ の $D_i$ 中のルート出現数 $count(T)$ をもつ．また，深さ $r$ の最新のルート出現 $Rto_T[r]$ を，深さごとに保持する．

### 3.2　掃木枝を用いた漸増的なパターン発見

候補パターンの出現位置を見つけるために，パターンの埋め込みを完全に保持する必要はない．アルゴリズムは，掃木枝を $D$ 上で右へ走らせて，パターン $T$ の埋め込み $\varphi(T)$ と掃木枝の交差をステージごとに更新しながら保持し，漸増的に候補パターンの出現を計算する．パターン $T$ の埋め込みと掃木枝の交差に関して，次の性質が成り立つ．

［補題 1］　$\varphi(T)$ をパターン $T$ の任意の埋め込み，$SB_i$ を時刻 $i$ の掃木枝とする．このとき，$\varphi(T)$ の最右枝と $SB_i$ との交差 $I_T = RMB(\varphi(T)) \cap SB_i$ が空集合でなければ，$I_T$ は $D$ の経路となり，$\varphi(T)$ の根を含む．　□

この補題にしたがい，次の定義を与える．パターン $T$ の埋め込み $\varphi(T)$ と掃木枝 $SB_i$ に対して，$\varphi(T)$ と $SB_i$ の交差を $I_T = RMB(\varphi(T)) \cap SB_i$ とする（図 4）．このとき，$I_T$ 中でもっとも浅い節点 $v_r$ と深い節点 $v_b$ を，それぞれ $SB_i$ に関する $I_T$ の**ルート出現**（root occurrence），**ボトム出現**（bottom occurrence）と呼ぶ．$I_T$ のルート出現は $\varphi$ に関する $T$ のルート出現と一致することに注意せよ．また，$RMB(\varphi(T)) \subseteqq SB_i$ が成り立つときは，$I_T$ のボトム出現と $\varphi$ に関する $T$ の最右葉出現は一致する．

次の補題は，最右葉出現の漸増的な特徴づけを与える．

［補題 2］　$T$ を大きさ $k > 1$ のパターンとする．時刻 $i$ において，$T$ が現在の節点 $v_i$ を最右葉出現にもつことの必要十分条件は，次の (i),(ii) を満たす大きさ $k-1$ のパターン $S$ が存在することである．(i) $S$ はボトム出現 $v_b$ として $v_i$ の親節点をもち，(ii) $T$ は $S$ の $(d-r, \ell)$ 拡張である．ここに，$d$ は $v_i$ の深さ，$r$ は $S$ のルート出現 $v_r$ の深さ，$\ell$ は $v_i$ のラベルである．　□

この補題により，候補パターン $T \in \mathcal{C}$ のすべての最右葉出現を，$T$ の後者 $S$ のボトム出現を用いて計算できることが分かる．我々のアルゴリズム StreamT は，掃木枝スタック $B = (B[0], B[1], \ldots, B[Top])$ を管理することにより，パターンの埋め込みと掃木枝の交差を記録する．ここで，掃木枝スタックの任意の要素（**バケット**と呼ぶ）$B[b]$ は，$(T, r, b)$ の形の**三項組**を保持する．$T \in \mathcal{C}$ は候補パターン，$r$ と $b$ は，$T$ の $SB_i$ に関するルート出現 $v_r$ とボトム出現 $v_b$ の深さである．

掃木枝スタック $B = (B[0], B[1], \ldots, B[Top])$ が以下の条件 (i),(ii) を満足するとき，$B$ は時刻 $i$ で up-to-date であるという．(i) スタック $B$ の長さが掃木枝 $SB_i$ の長さと一致，すなわち $Top = |SB_i|$ が成り立ち，(ii) 任意の $0 \leqq h \leqq Top$ に対して，$B$ の $h$ 番目のバケット $B[h]$ は，三項組 $\tau = (T, r, b) \in \mathcal{T} \times \mathbf{N} \times \mathbf{N}$ のうちでボトム出現が $b = h$ となるものをすべて含む．

次の補題は，時刻 $i$ における掃木枝スタック $B_i$ の各バケットに関する再帰的な特徴づけを与える．

［補題 3］　$\boldsymbol{v}_i = (d, \ell)$ をデータストリーム $\mathcal{S}$ における現在の節点，$B_k = (B_k[0], B_k[1], \ldots, B_k[Top_k])$ を時刻 $k \in \{i-1, i\}$ における掃木枝スタックとする．このとき，掃木枝スタック $B_{i-1}$ と $B_i$ が共に up-to-date であると仮定すると，次の 1–4 が成り立つ．

1.　任意の $0 \leqq b < d-1$ に対して，$\tau \in B_i[b]$ の必要十分条件は $\tau \in B_{i-1}[b]$ である．

2.　$b = d-1$ のとき，$\tau = (T, r, d-1) \in B_i[d-1]$ の必要十分条件は，以下の (i),(ii) のどちらかが成り立つことである．

2–(i)　$\tau \in B_{i-1}[d-1]$.

2–(ii)　$r \leqq d \leqq h$ なる整数 $h$ が存在し，$(T, r, h) \in B_{i-1}[d] \cup \cdots \cup B_{i-1}[Top_{i-1}]$ を満たす．

3.　$b = d$ のとき，$\tau = (T, r, d) \in B_i[d]$ の必要十分条件は，以下の条件 (i),(ii) のどちらかが成り立つことである．

3–(i)　$T$ はラベル $\ell$ をもつ大きさ 1 の順序木である．

3–(ii)　三項組 $(S, r, d-1) \in B_i[d-1]$ が存在し，$T$ は $S$ の $(d-r, \ell)$ 拡張である．

4.　任意の $b > d$ に対して，$B_i[b]$ は未定義である．

(証明)　条件 1, 2, 3–(i), 4 は，いずれも自明である．したがって，3–(ii) のみを証明する．

$T$ を大きさ $k$ のパターン，$\varphi(T)$ を $T$ の $D$ へのある埋め込みとする．$\varphi(T)$ の最右枝 $RMB(\varphi(T))$ と現在の掃木枝 $SB_i$ が交差し，そのボトム出現の深さを $b = d$ であるとする．仮定より $v_i = (d, \ell)$ なので，$v_i$ は $T$ の最右葉出現となっている．また，明らかに $\varphi(T)$ は大きさ $k$ の順序木を形成し，その最右葉は $v_i$ である．ここで，$\varphi(T)$ から節点 $v_i$ を取り除いて得られる集合を $C_T$ とおく．$C_T$ は $T$ の後者 $S$ の埋め込みであり，その最右葉出現は $v_i$ の親節点 $w$ となる．$S$ は $T$ の後者なので $depth(w) = d-1$ が成り立ち，したがって，$C_T$ に対応する三項組 $(S, r, d-1)$ は $B_i[d-1]$ に属する．このとき，$T$ が $S$ の $(d-r, \ell)$ 拡張であることは明らか．

最後に，逆の命題が成り立つことは自明である．　□

図 5 に，補題 3 に基づいて掃木枝スタック $B_{i-1}$ を更新する様子を示す．今，データストリーム $\mathcal{S}$ から $i$ 番目の節点 $\boldsymbol{v}_i = (d, \ell)$ を受け取ったと仮定する．このとき，UNCHANGE バケット $(B_{i-1}[0], \ldots, B_{i-1}[d-1])$ に属する三項組は，時刻 $i$ になっても同じ深さのバケットに留まる．$B_{i-1}[d], \ldots, B_{i-1}[Top]$ に

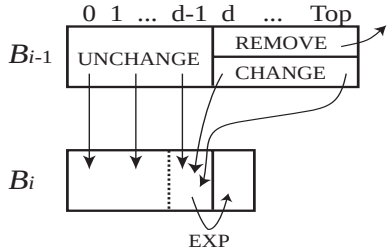


図 5　時刻 $i-1$ と時刻 $i$ における掃木枝スタック

**Algorithm** UpdateB$(B, \mathcal{C}, (d,\ell), i)$

**入力：** 掃木枝スタック $B = (B[0], B[1], \ldots, B[Top])$，候補集合 $\mathcal{C}$，節点 $v_i = (d,\ell)$，現在の時刻 $i$.

**出力：** 三項組の集合 $EXP \subseteqq \mathcal{T} \times \mathbf{N} \times \mathbf{N}$.

**手法：**

1. もし $d \leqq Top$ ならば，以下を行なう：
   - $BELOW := B[d] \cup \ldots \cup B[Top]$; $Top = d-1$;
   - $REMOVE := \{\, (T,r,b) \in BELOW \mid r \geqq d \,\}$;
   - $CHANGE := \{\, (T,r,b) \in BELOW \mid r \leqq d-1 \,\}$;
   - $B[d-1] := B[d-1] \cup \{(T,r,d-1) \mid (T,r,b) \in CHANGE\}$;
2. ラベル $\ell$ の 1-パターン $T_\ell$ に対して，$EXP := \{(T_\ell, d, d)\}$;
3. 任意の $(S, r, d-1) \in B[d-1]$ に対して以下を行なう：
   - $S$ の $(d-r, \ell)$ 拡張 $T$ に対して，$EXP := EXP \cup \{(T,r,d)\}$;
4. $(B, EXP)$ を出力する;

図 6　掃木枝スタック更新アルゴリズム

属する三項組は，REMOVE と CHANGE に分けられる．REMOVE バケットに属する三項組は時刻 $i$ では捨てられるが，CHANGE バケットに属する三項組は $B_i[d-1]$ に移される．$B_i[d-1]$ に属するすべての三項組については，それらに最右拡張を適用し，得られた拡張に関する三項組を EXP に加える．

図 6 に，掃木枝スタックを再帰的に更新するアルゴリズム UpdateB を示す．このアルゴリズムは，任意の時刻 $i$ において，古い掃木枝スタック $B_{i-1}$ と現在の節点 $v_i = (d,\ell)$ を受け取り，新しい掃木枝スタック $B_i$ の中で最初の $d$ 個のバケット $B_i[0], \ldots, B_i[d-1]$ を出力する．最後のバケット $B_i[d]$ は，次の手続きで必要となるため，他のバケットとは別に $EXP = B_i[d]$ として出力される．

次の系は，補題 3 よりただちに示される．

［系 4］　図 2 のアルゴリズム StreamT における while ループが時刻 $i$ にて発動したと仮定する．このとき，アルゴリズム UpdateB$(B, \mathcal{C}, (d,\ell), i)$ は，以下の (i),(ii) を出力する．(i) $d$ 番目のバケット $B[d]$ を除くすべてのバケット $B[0], \ldots, B[d-1]$ が時刻 $i$ で up-to-date であるような掃木枝スタック $B$，(ii) 時刻 $i$ で up-to-date であるような $d$ 番目のバケット $B[d]$．　□

パターンの頻度を計算するにあたって，そのルート出現を重複せずに探索することは非常に重要である．我々は，図 6 のアルゴリズム UpdateB が，各時刻において任意のパターン $T \in \mathcal{C}$ のルート出現を重複せずに枚挙することを確認している [7].

### 3.3　候補集合の管理

本節では，候補集合 $\mathcal{C}$ の管理法について，その概要を述べる．

**Algorithm** UpdateC$(EXP, B, \mathcal{C}, (d,\ell), i)$

**入力:** 三項組の集合 $EXP$, 掃木枝スタック $B = (B[0], B[1], \ldots, B[Top])$, 候補集合 $\mathcal{C}$，$i$ 番目の節点 $v_i = (d,\ell)$，現在の時刻 $i$.

**出力：** 更新後の掃木枝スタックと候補集合 $(B, \mathcal{C})$.

**手法：**

1. 任意の三項組 $(T,r,b) \in EXP$ に対して，そのルート出現 $\rho$ が $\rho \neq Rto_T[r]$ を満たすならば以下を行なう：
   - $Rto_T[r] := \rho$;
   - $T \in \mathcal{C}$ ならば，$count(T) := count(T) + 1$;
   - $T \notin \mathcal{C}$ かつ $T$ の後者が頻出ならば，
     $count(T) := c_I$ かつ $\mathcal{C} := \mathcal{C} \cup \{T\}$;
2. $B[d] := \emptyset$; $Top := d$; $freq_i(T) := count(T)/i$;
3. 非頻出な後者をもつパターン $T \in \mathcal{C}$ に対して，以下を行なう：
   - $\mathcal{C} = \mathcal{C} \backslash \{T\}$;
4. 任意の三項組 $(T,r,b) \in EXP$ に対して，以下を行なう：
   - $T \in \mathcal{C}$ ならば $B[d] := B[d] \cup \{(T,r,b)\}$;
5. $(B, \mathcal{C})$ を出力する;

図 7　候補集合の更新アルゴリズム

4. 節では，候補集合の様々な管理戦略について，さらに詳しく議論する．

図 7 に，候補集合 $\mathcal{C}$ を管理するアルゴリズム UpdateC を示す．我々のアルゴリズムは，Hidber [12] と同様に，候補パターンとして頻出パターン集合の *negative border* [18]（後者パターンが頻出となるような非頻出パターン）までを保持する．候補集合の管理は，以下の 4 つの手順で行われる．

**初期化**　大きさ 1 の全パターンを候補集合 $\mathcal{C}$ に入れる．これはメインアルゴリズム（図 2）の最初に行なわれる．

**インクリメント**　現在の節点 $v_i$ が最右葉出現となるような任意の候補パターン $T \in \mathcal{C}$ に対して，$count(T)$ を 1 増やす．

**挿入**　パターン $T \notin \mathcal{C}$ がストリームに出現したとき，その後者パターン $P \in \mathcal{C}$ が頻出ならば，$\mathcal{C}$ に $T$ を加える．

**削除**　頻出な候補パターン $T \in \mathcal{C}$ が非頻出になったら，$T$ のすべての前者パターン（最右拡張）を $\mathcal{C}$ から削除する．ただし，大きさ 1 のパターンは削除しないものとする．

### 3.4　忘却モデルへの拡張

本節では，アルゴリズム StreamT を，定義 3 の忘却モデルに適用する方法を示す．

忘却モデルにおけるパターン $T$ の頻度は，定義 3 の (2) 式で定義される．すなわち，忘却モデルでは，時刻 $i$ における頻度は各々のルート出現が発見された時刻 $j$ に関するすべての重み $\gamma^{i-j}$ に依存する．

$freq^{\mathrm{fg}}_{\gamma,i}(T)$ と $hit^{(i)}_j(T)$ を，それぞれ $fr_i$，$hit_j$ と略記する．また，時刻 $i$ におけるパターン $T$ が最後に出現した時間を $lt(i)$ とする．このとき，次の補題が成り立つ．

［補題 5］　任意のパターン $T$ と時刻 $i$ に対して，以下の漸化式が成り立つ．

$$\begin{aligned} fr_0 &= 0, \\ fr_i &= \tfrac{lt(i)}{i}\gamma^{i-lt(i)}\, fr_{lt(i)} + \tfrac{1}{i}\, hit_i \quad (i > 0) \end{aligned}$$

□

この補題から，$lt(i)$ を保持することにより，忘却モデルにお

ける $T$ の頻度 $freq^{\mathrm{fg}}_{\gamma,i}(T)$ を効率よく計算できることが分かる．

### 3.5 アルゴリズムの解析

ステージ $i$ におけるアルゴリズム StreamT の計算量は以下のとおりである．

時刻 $i$ における掃木枝スタックを $B_i$ とする．このとき，ステージ $i$ におけるアルゴリズム UpdateB と UpdateC の時間計算量は，それぞれ $O(\sum_{j=d-1}^{Top}|B_{i-1}[j]|)$ 時間，$O(kC+D)$ 時間となる．ここに，$d$ は時刻 $i$ の入力節点 $\boldsymbol{v}_i$ の深さ，$k$ はパターンの最大サイズ，$C$ は $v_i$ に最右葉出現をもつ候補パターン数，$D$ はステージ $i$ で削除された候補パターンの数である．また，ステージ $i$ における領域計算量は，$O(\sum_{j=1}^{Top}|B_i[j]|+k|\mathcal{C}|)$ 領域である．

したがって，我々のアルゴリズムは，掃木枝スタックと候補集合の大きさ程度の時間と領域しか使用しない．

## 4. 候補集合の管理戦略

本節では，3.3 節で触れた候補集合の管理戦略について，さらに詳しく議論する．ここでは基本型のアルゴリズムについてのみ述べるが，本節の結果は忘却型のアルゴリズムにも適用可能である．

### 4.1 挿入されるパターンの初期頻度

まず，候補集合 $\mathcal{C}$ に新たに挿入されるパターンの，初期頻度の与え方について考察する．今，時刻 $i$ でパターン $T$ が候補集合 $\mathcal{C}$ に挿入されると仮定する．このとき，時刻 $i$ における $T$ の正確な出現回数 $c_R$ を知ることは困難なので，カウンタ $count(T)$ の初期値として，ある自然数 $c_I \in \mathbf{N}$ を与える．$T$ の後者パターンを $P$ とすると，常に $1 \leqq c_R \leqq count(P)$ が成り立つことから，$c_I$ として 1 以上 $count(P)$ 以下の自然数を選べばよい．

以下では，$c_I = 1$ とする候補集合の管理戦略を**節制型**（lazy）と呼び，逆に，$c_I = count(P)$ とする管理戦略を**贅沢型**（eager）と呼ぶ．また，図 2 のアルゴリズム StreamT が出力するパターン集合列を $(\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_i, \ldots)$，実際の頻出パターン集合の列を $(\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_i, \ldots)$ とする．

#### 4.1.1 節制型の管理戦略

節制型の管理戦略は，挿入されたパターンの初期頻度として挿入時刻 $i$ で検出された 1 回のみを与えるという，非常に慎重な候補集合管理戦略である．したがって，この戦略を用いれば，パターンの頻度を実際より多く計算することはありえない．

［定理 6］ アルゴリズム StreamT が節制型の候補集合管理戦略を用いると仮定する．このとき，任意のステージ $i$ において $\mathcal{A}_i \subseteqq \mathcal{F}_i$ が成立する．すなわち，節制型を用いたアルゴリズムは，非頻出なパターンを解として出力することがない**健全**（sound）なアルゴリズムである． □

一方，節制型以外のアルゴリズム（$C_I > 1$）は，パターンの出現頻度を実際より大きく計算し得るので，非頻出なパターンを頻出パターンとして出力する可能性があり，健全性は保障されない．

#### 4.1.2 贅沢型の管理戦略

贅沢型の管理戦略では，候補集合 $\mathcal{C}$ に挿入されたパターン $T$ の初期頻度として，$T$ の後者 $P$ の頻度をそのまま譲り渡す．また，任意のパターン $T$ とその後者 $P$ の頻度は，任意の時刻 $i$ において，$freq_i(P) \geqq freq_i(T)$ の単調性を満たす．したがって，贅沢型の戦略は，候補パターンの頻度を実際の頻度より常に大きく計算する特徴がある．

贅沢型を採用することにより，節制型を用いるよりも多くの意味のあるパターンを発見できることが期待される．

表 1 4 種類のアルゴリズム

| | オンライン戦略 | 候補集合管理戦略 |
|---|---|---|
| *lazy-basic* | 基本型 | 節約型 |
| *eager-basic* | 基本型 | 贅沢型 |
| *lazy-forget* | 忘却型 | 節約型 |
| *eager-forget* | 忘却型 | 贅沢型 |

### 4.2 パターン挿入の遅延

次に，候補集合への挿入に関する**遅延法**について述べる．基本型のオンラインアルゴリズムに対して，3.3 節の管理手法のままパターンを候補集合に挿入すると，アルゴリズムの開始直後に膨大な候補パターンを生成してしまい，計算が困難になる場合がある．そこで，我々は，TDAG [14] におけるヒューリスティックを用いて，候補パターンの挿入を遅延させる．

[14] に基づく遅延法は，以下のように行う．$T \notin \mathcal{C}$ を大きさ $k$ のパターンとする．時刻 $i$ において $T$ がデータストリーム中に出現すると仮定する．このとき，$i \leqq Lk^2$ が成り立つならば，$T$ の後者パターンが頻出であっても，$T$ を候補集合 $\mathcal{C}$ に挿入しない．ここに，$L(\geqq 1)$ は，遅延定数と呼ばれる自然数である．

$L$ として充分に小さい自然数をとれば，遅延法を用いて，計算の序盤における候補パターンの爆発を防ぐことができる．実際，遅延法を実装すると，基本型アルゴリズムの性能が著しく向上する．

節制型の管理手法と遅延法を組み合わせても，アルゴリズム StreamT の健全性は保存される．

## 5. 実　　験

本節では，実データを用いた実験について述べる．

### 5.1 方　　法

我々は，基本型と忘却型のアルゴリズムを SAX と Java で実装し，PC 上で実験を行った．実験には，以下のスペックのマシンを用いた．

- PentiumIII 1GHz, 1500 megabytes RAM, Windows2000（5.3 節 規模耐性実験）
- PentiumIII 1.2GHz, 1000 megabytes RAM, Windows2000（5.4 節 アルゴリズムの比較実験）

我々は，オンライン戦略および候補集合管理集合の違いにより，表 1 に示す 4 種類のアルゴリズム *lazy-basic*, *eager-basic*, *lazy-forget*, *eager-forget* を実装した．

3.3 節で述べた候補集合の管理手法のうち，候補パターンの削除は計算負荷が大きい．そこで，パターンの削除は，5.3 節の規模耐性実験では節点を 500 個処理するごとに，5.4 節の比較実験では節点を 100 個処理するごとに，それぞれ 1 回ずつ

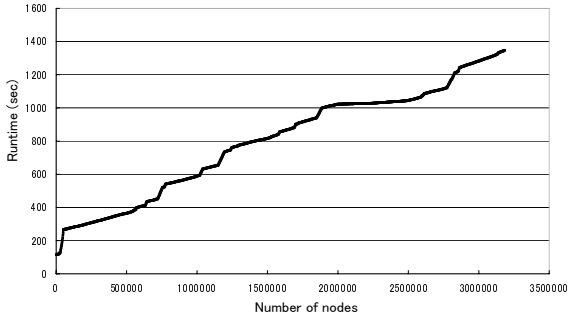


図 8　規模耐性実験

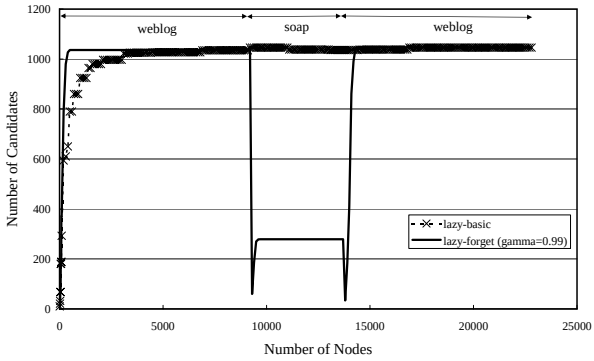


図 9　基本型 vs 忘却型

行う.

予備実験において，アルゴリズムは計算開始直後に候補パターンが爆発的に増大し，計算が困難になることが分かった．この問題は，4.2 節で述べた TDAG [14] における経験的手法を用いることにより，大幅に改善された．

### 5.2 デ ー タ

我々は，実験データとして，2 種類のデータセット *dblp* と *wsw* を用意した．

*dblp* は計算機科学分野の論文目録サイト DBLP [9] で公開されている XML データ `dblp.xml`（130MB）であり，対応する半構造データストリームの節点数は 3,185,138，異なるタグ数は 22 である．このデータは，5.3 節の規模耐性実験に用いる．

もう 1 つのデータセット *wsw* は, [20] で提供されている XML データ `weblog.xml` と `soap.xml` から作成されるデータである．今，`weblog.xml` の後半を切り落として得られた 9164 (nodes) のデータを *weblog* と呼び，`soap.xml` 全体からなる 4502 (nodes) のデータを *soap* と呼ぶことにする．このとき，データセット *wsw* は，2 つのデータ *weblog* と *soap* を，*weblog*, *soap*, *weblog* の順に結合して得られたデータである．また，*weblog* と *soap* に含まれるタグの種類は，それぞれ 13 種類と 11 種類であり，双方に共通するタグは存在しない．このように，*wsw* データセットは，時間変化する半構造データストリームを意図して人工的に作成されたデータである．このデータは，5.4 節のアルゴリズムの比較実験に使用する．

### 5.3 規 模 耐 性

最初に，大規模な XML データを用いた規模耐性実験を行い，アルゴリズム StreamT の性能を評価する．

我々は，アルゴリズム *eager-basic* を用いて StreamT の規模耐性実験を行った．この実験では，*dblp* データに対して最小サポートを $\sigma = 1(\%)$ に固定し，データストリームの節点が 2000 個処理されるごとにアルゴリズムの計算時間を測定した．実験結果を図 8 に示す.

実験より，アルゴリズム StreamT は，入力サイズに対してほぼ線形時間で動作する．この結果は，提案するオンラインアルゴリズムが，高速に流れ続ける大規模な半構造データストリームに対して，効率よく働き続けることを示している．

### 5.4 アルゴリズムの比較

本節では，表 1 の 4 種類のアルゴリズムを実験により比較し，忘却の有無や候補集合管理戦略の違いが，アルゴリズムの実際の振る舞いに与える影響を検証する．本節で述べるすべての実験において，データセットは *wsw* を用いる．

#### 5.4.1 オンライン戦略の比較

本節では，2. 節で述べた基本型と忘却型のアルゴリズムを，実験により比較する．

図 9 に，最小サポートを $\sigma = 5(\%)$ に固定して，基本型アルゴリズム *lazy-basic* と，忘却定数 $\gamma = 0.99$ に対する忘却型アルゴリズム *lazy-forget* が，各時刻にそれぞれ計算した候補パターン数を示す.

実験の結果，基本型アルゴリズムが計算した候補パターンは，常に *weblog* データ中に出現するパターンのみであり，*soap* データ中のパターンを見つけることはなかった．一方，忘却型のアルゴリズムは，*weblog* と *soap* のデータが入れ替わるたびに，候補パターンとしてそれぞれのデータに出現するパターンを計算した．この結果から，忘却型のアルゴリズムは，データストリームの傾向が変わると，速やかに過去のパターンを削除し，代わりに現在のパターンを候補集合に加えることが分かる．これは，忘却型のアルゴリズムが，過去に出現したパターンの出現頻度を指数的に逓減させることからである．

このように，忘却型アルゴリズムは，基本型アルゴリズムよりストリームデータの時間変化に柔軟に対応する．したがって，忘却の手法は，異常検出やトレンド変化検出などの応用に有効である．

#### 5.4.2 忘却定数の値による比較

次に，上で述べたオンライン戦略の比較実験と同じ環境で，忘却定数を $\gamma = 0.99, 0.999, 0.9999, 0.99999$ と変化させたときの忘却型アルゴリズム *lazy-forget* の動作を調べた．図 10 に，最小サポート値 $\sigma = 5(\%)$ に対してアルゴリズムが各時刻に計算した候補パターン数を，$\gamma$ の値ごとに示す．なお，忘却定数 $\gamma = 0.99, 0.999, 0.9999, 0.99999$ の半減期は，それぞれ，約 $70, 700, 7000, 70000$ である．

結果より，$\gamma$ の値が大きくなるほど，過去に出現したパターンの出現頻度の逓減がなかなか進まず，データストリームの時間変化への追従が遅れることが分かる．また，$\gamma = 0.99999$ の

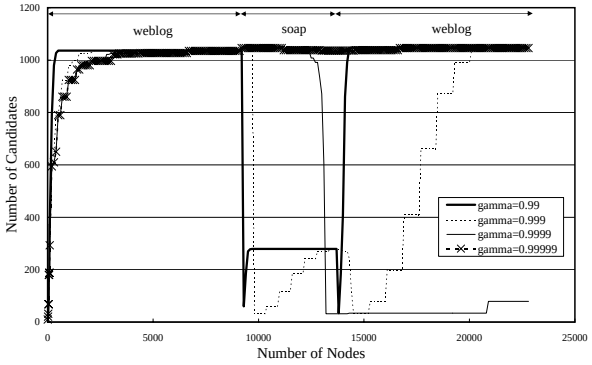


図 10　忘却定数の値による比較

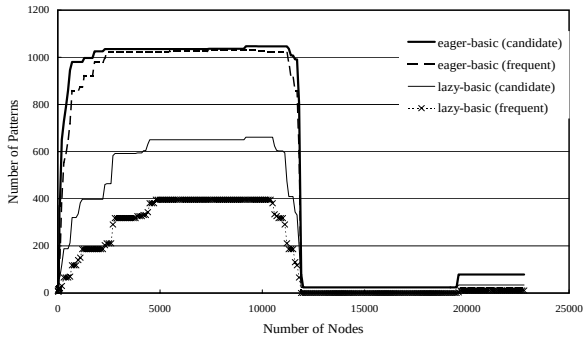


図 11　節制型 vs 贅沢型

場合，半減期がデータサイズより遥かに大きいため，図 9 における基本型アルゴリズムの挙動とほとんど変わらなかった．

**5.4.3**　候補集合管理戦略の比較

最後に，3. 節で述べた節制型と贅沢型の候補集合管理戦略を比較する．

図 9 に，最小サポートを $\sigma = 7(\%)$ に固定して，アルゴリズム *lazy-basic* と *eager-basic* が，各時刻にそれぞれ計算した候補パターン数および頻出パターン数を示す．実験により，贅沢型の候補戦略の方が，節制型よりも多くの候補パターンと頻出パターンを計算することが分かる．

## 6. 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データマイニングを考察し，効率よいオンライン半構造データマイニングアルゴリズム StreamT を与えた．このオンラインアルゴリズムについて，いくつかの候補集合管理戦略を考察し，それぞれの戦略を用いたアルゴリズムの動作を実験で検証した．また，アルゴリズムに忘却の概念を導入した結果，過去の影響を受けずにデータの時間変化に柔軟に追従することを実証した．

半構造データストリームからの異常検出やトレンド変化検出などに，提案手法を応用することが今後の課題である．

## 文　献


[1] K. Abe, S. Kawasoe, T. Asai, H. Arimura, H. Sakamoto, and S. Arikawa. Mining Frequent Substructures from Web, *Active Mining*, H.Motoda(Eds.), 83–94, IOS Press, 2002.

[2] K. Abe, S. Kawasoe, T. Asai, H. Arimura, and S. Arikawa. Optimized Substructure Discovery for Semi-structured Data, In *Proc. PKDD'02*, 1–14, LNAI 2431, Springer, 2002.

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

[4] R. Agrawal, R. Srikant, Fast Algorithms for Mining Assoiciation Rules, In *Proc. the 20th VLDB*, 487–499, 1994.

[5] Aho, A. V., Hopcroft, J. E., Ullman, J. D., *Data Structures and Algorithms*, Addison-Wesley, 1983.

[6] T. Asai, K. Abe, S. Kawasoe, H. Arimura, H. Sakamoto, S. Arikawa, Efficient Substructure Discovery from Large Semi-structured Data, In *Proc. the 2nd SIAM Int'l Conf. on Data Mining (SDM2002)*, 158–174, 2002.

[7] T. Asai, H. Arimura, K. Abe, S. Kawasoe, and S. Arikawa. Online Algorithms for Mining Semi-structured Data Stream, In *Proc. the 2002 IEEE Int'l Conf. on Data Mining (ICDM'02)*, 27–34, 2002.

[8] R. J. Bayardo Jr., Efficiently Mining Long Patterns from Databases, In *Proc. SIGMOD98*, 85–93, 1998.

[9] DBLP, `http://dblp.uni-trier.de/`

[10] L. Dehaspe, H. Toivonen, R. D. King, Finding Frequent Substructures in Chemical Compounds, In *Proc. KDD-98*, 30–36, 1998.

[11] P. B. Gibbons, Y. Matias, Synopsis Data Structures for Massive Data Sets, In *External Memory Algorithms*, DIMACS Series in Discr. Math. and Theor. Compt. Sci., Vol. 50, AMS, 39–70, 2000.

[12] C. Hidber, Online Association Rule Mining, In *Proc. SIGMOD'99*, 145–156, 1999.

[13] 石川佳治，北川博之，“ 忘却の概念に基づく文書クラスタリング手法の一般化について ”，信学技法，DE2002-3，13–18，2002.

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

[15] H. Mannila, H. Toivonen, A. I. Verkamo, Discovering Frequent Episode in Sequences, In *Proc. KDD-95*, 210–215, 1995.

[16] T. Matsuda, T. Horiuchi, H. Motoda, T. Washio, K. Kumazawa, N. Arai, Graph-Based Induction for General Graph Structured Data, In *Proc. DS'99*, 340–342, 1999.

[17] T. Miyahara, Y. Suzuki, T. Shoudai, T. Uchida, K. Takahashi, H. Ueda, Discovery of Frequent Tag Tree Patterns in Semistructured Web Documents, In *Proc. PAKDD-2002*, 341–355, 2002.

[18] S. Parthasarathy, M. J. Zaki, M. Ogihara, S. Dwarkadas, Incremental and Interactive Sequence Mining, In *Proc. CIKM'99*, 251–258, 1999.

[19] R. Rastogi, Single-Path Algorithms for Querying and Mining Data Streams, In *Proc. SDM2002 Workshop HDM'02*, 43–48, 2002.

[20] Sosnoski Software Solution, Inc., XML benchmark, `http://www.sosnoski.com/opensrc/xmlbench/`

[21] W3C, Extensive Markup Language (XML) 1.0 (Second Edition), *W3C Recommendation*, 06 October 2000. `http://www.w3.org/TR/REC-xml`

[22] K. Wang, H. Liu, Discovering Structural Association of Semistructured Data, TKDE, 12(2), 353–371, 2000.

[23] K. Yamanishi, J. Takeuchi, A Unifying Framework for Detecting Outliers and Change Points from Non-Stationary Time Series Data, In *Proc. SIGKDD 2002*, ACM, 2002.

[24] M. J. Zaki. Efficiently Mining Ffrequent Trees in a Forest, In *Proc. SIGKDD 2002*, ACM, 2002.