半構造データからの頻出パターン発見アルゴリズム

Finding Frequent Patterns from Semi-structured Data

（論文番号: C5–1）

浅井達哉 † Tatsuya Asai
安部賢治 † Kenji Abe
川副真治 † Shinji Kawasoe
坂本比呂志 † Hiroshi Sakamoto
有村博紀 †,‡ Hiroki Arimura
有川節夫 † Setsuo Arikawa

† 九州大学大学院システム情報科学府・研究院
Department of Informatics, Kyushu University
‡ さきがけ研究 21
PRESTO, JST
E-mail: †{t-asai,k-abe,s-kawa,hiroshi,arim,arikawa}@i.kyushu-u.ac.jp

概 要

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，近年，ウェブページや XML 文書に代表される半構造データの利用が急速に進んでいる．そのため，これらのデータに対する効率よいアクセス手法の開発が急務となっている．本論文では，半構造データマイニング問題を，与えられた半構造データの集積から出現頻度の高い部分構造を発見する問題として定式化し，効率のよいマイニングアルゴリズムを与える．さらに，提案アルゴリズムの正当性を証明する．我々は，半構造データのモデルとしてラベル付き順序木を採用し，Bayardo（SIGMOD'98）の手法を順序木に拡張することにより，効率のよい半構造データマイニングアルゴリズムを実現している．実際のウェブデータを用いた実験では，このアルゴリズムの有効性を確認した．

キーワード: データマイニングアルゴリズム，ウェブマイニング，半構造データ，XML，パターンマッチング，木構造パターン

1 はじめに

データマイニング（Data mining）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る．データマイニングは，現在，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われている [2].

一方，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 [1, 17] 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）に対する要求が高まっている [4, 5, 9, 10, 12, 13, 14, 15, 18, 19]．しかし，半構造データは関係データベースとは違い，明示的な構造をもたないので，関係データベースを対象とした従来のデータマイニング手法を半構造データに直接適用することは困難である．したがって，木構造やグラフ構造に対する高速マイニング手法の開発が不可欠である．

我々は，半構造データマイニング問題を，与えられた半構造データの集積から出現頻度の高い部分構造を発見する問題と定式化し，Bayardo

[8] の手法に基づき，この問題を効率よく解決するアルゴリズム FREQT を開発した [6, 7].

1.1 関連研究

半構造データマイニングに関する研究はそれほど多くない [14, 19]．Wang と Liu [19] は，半構造データベースにおけるスキーマ発見の問題を考察し，節点への経路をアイテムとする結合規則を発見するアルゴリズムを与えている．宮原ら [14] は，半構造データベースから，タグ木パターンを発見するアルゴリズムを与えている．

半構造データマイニングではないが，木構造やグラフ構造を対象としたマイニング手法が研究されている．Wang，Shapiro，Shasha ら [18] は，ゲノムデータにおける RNA 二次構造に対するマイニング手法を与え，Dehaspe ら [9] は有機化合物に対するマイニング手法を与えている．また，松田と元田ら [13] は，逐次ペア拡張と呼ばれる手法を用いて，ラベル付きグラフから統計的指標を最適化する経験的アルゴリズムを与えている．

代表的なデータマイニング問題の一つである結合規則発見問題に関して，Agrawal ら [2] は，アプリオリと呼ばれるアルゴリズムを開発した．このアルゴリズムは，アイテム集合の部分集合束を利用して，頻出アイテム集合を効率よく発見する．上記の [9, 19] は，アプリオリに基づいている．しかし，アプリオリアルゴリズムは少数の非常に大きな頻出アイテム集合が存在するときには，極めて効率が悪いことが指摘されている．

この問題に対して，Bayardo [8] は大きな頻出アイテム集合を効率よく発見するアルゴリズムを提案した．このアルゴリズムは，すべてのアイテム集合を，集合枚挙木を用いて重複なく枚挙する方法に基づいている．また，瀬々ら [16] は，各アイテム集合の出現リストの併合に基づく手法を集合枚挙木と組み合わせて，最適アイテム集合を発見するアルゴリズムを与えている．

我々の提案するアルゴリズムは，集合枚挙木と出現リストのアイディアを，順序木へ拡張したものである．

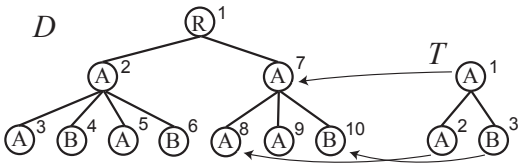

図 1: データ木 D とパターン木 T

Zaki [20] は，我々とは独立に，順序木の集積から頻出部分構造パターンを発見するアルゴリズムを提案している．このアルゴリズムは，我々の提案手法と本質的に同じである．

1.2 本稿の構成

本稿の構成は次のとおりである．2 節では半構造データマイニング問題を定式化する．3 節では，頻出パターン発見アルゴリズムについて述べる [6, 7]．4 節では，実際のウェブデータを用いた実験について述べる．最後に，5 節で本稿をまとめる．

2 定式化

集合 A に対し，A の要素数を $\#A$ で表す．非負整数 n と関数 $f : A \to A$ に対し，f の n 回の合成を f^n と表す．つまり，$f^0(x) = x$ であり，$f^n(x) = f(f^{n-1}(x))(n \geq 1)$ である．

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六項組 $T = (V, E, \mathcal{L}, L, v_0, \preceq)$ で定義する．$G = (V, E, v_0)$ は v_0 を根とする木である．V は節点の集合を，$E \subseteq V^2$ は親子関係を表す．$L : V \to \mathcal{L}$ はラベル関数を，二項関係 $\preceq \subseteq V^2$ 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．任意の節点 $v \in V$ に対し，v の親節点を $\pi_T(v)$ と書く．

順序木 T が**標準形**であるとは，T における節点集合が $V_T = \{1, \ldots, k\}$ であり，V_T の要素が T の前置順巡回（preorder traversal）[3] で番号付けられていることを言う． このとき，T の根は $v_0 = 1$ 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．

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**と呼ぶことにし，さらに T は標準形順序木であることを仮定する．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) φ は単射であり，(ii) φ 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 φ 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．また，大きさ $|T| = k$ のパターンを k-**パターン**と呼ぶ．

出現頻度　k-パターン T とデータ木 D，T から D へのマッチング関数 φ に対して，φ に関する**全出現**を $Total(\varphi) = (\varphi(1), \ldots, \varphi(k)) \in (V_D)^k$ と定義する．$Total(\varphi)$ は，T を標準形順序木と仮定すると，マッチング関数 $\varphi : V_T \to V_D$ の表現そのものとなることに注意せよ．次に，φ に関する**ルート出現**を $Root(\varphi) = \varphi(1)$ と定義し，$Occ(T) = \{Root(\varphi) \mid \varphi$ は T から D へのマッチング関数$\}$ と定義する．すなわち，$Occ(T)$ は T の根の出現先の集合である．我々は，パターン T のデータ木 D への**出現頻度**（または**頻度**）を $freq_D(T) = \#Occ(T)/|D|$ と定義する．$0 < \sigma \leq 1$ なる正数 σ に対して，パターン T が $freq_D(T) \geq \sigma$ を満たすとき，T を σ 頻出と呼ぶ．

例として，図 1 のデータ木 D とパターン木 T を考える．図中の矢印は，$Total(\varphi_1) = (7, 8, 10)$ なるマッチング関数 φ_1 を表しており，φ_1 に対するルート出現 $Root(\varphi_1)$ は，節点 7 である．T から D への全出現 $Total(\varphi)$ は，全部で $(2, 3, 4), (2, 3, 6), (2, 5, 6), (7, 8, 10), (7, 9, 10)$ の 5 種類が存在する．したがって，ルート出現 $Root(\varphi)$ は節点 2 と節点 7 の 2 種類であり，以上より，パターン T のデータ木 D への出現頻度は $freq_D(T) = \#Occ(T)/|D| = 2/10$ となる．

Algorithm FREQT
入力: データ木 D, ラベル集合 $\mathcal{L}$, 最小サポート σ.
出力: D における σ 頻出パターンの集合 $\mathcal{F}$.
手法:
```
1  σ 頻出 1-パターン集合 F_1 と
2  最右葉出現リスト RMO_1 を計算する;
3  k := 2;
4  while F_{k-1} ≠ ∅ do begin
5    ⟨C_k, Roc_k⟩ := Expand-Trees(F_{k-1}, RMO_{k-1});
6    F_k := ∅;
7    foreach pattern T ∈ C_k do
8      RMO_k(T) から freq_D(T) を計算する;
9      if freq_D(T) ≥ σ then
10       F_k = F_k ∪ {T};
11   k := k + 1;
12 end /* while-loop */
13 return F = F_1 ∪ ··· ∪ F_{k-1};
```

図 2: 頻出パターン発見アルゴリズム

半構造データマイニング問題　我々は，半構造データからのデータマイニング問題を，データ木 D に頻出するパターン T を発見する問題として，次のように定式化する：

頻出パターン発見問題
入力：ラベル集合 $\mathcal{L}$，$\mathcal{L}$ 上のデータ木 D，実数 $0 < \sigma \leq 1$（**最小サポート**）．
問題：D に対して σ 頻出なパターン T をすべて見つけよ．

この問題は，相関規則マイニング [2] における頻出アイテム集合発見問題の一般化である．

3　アルゴリズム FREQT

3.1　FREQT の概要

本節では，頻出パターン発見問題を効率よく解決するアルゴリズム FREQT を与える．このアルゴリズムは，**最右拡張**と呼ばれる効率のよい順序木枚挙法に基づいている．ここで，$(k-1)$-パターン S の最右拡張とは，S の最右枝に節点 k を追加して得られる k-パターン T である．ただし，k は T の兄弟関係において末弟であるように追加する．特に，S の最右葉 $rml(S) = k - 1$

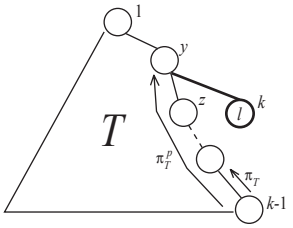

図 3: T の (p, ℓ) 拡張

の p 代前の親 $\pi_T^p(k-1)$ に，ラベル ℓ をもつ節点 k を追加して得られた S の最右拡張を，(p, ℓ) 拡張と呼ぶ（図 3）．また，サイズ 0 のパターン $\perp$（空木）を仮定し，任意の 1-パターンは $\perp$ の最右拡張であるとする．

空木 $\perp$ からはじめて，最右拡張を繰り返すことにより，すべてのパターンを重複なく生成することができる [6, 7]．この順序木枚挙手法は，Bayardo [8] による集合枚挙木に基づくアイテム集合枚挙法を，順序木に拡張したものである．

図 2 にアルゴリズム FREQT を示す．正整数 $k \geq 1$ に対して，σ 頻出 k-パターンの集合を $\mathcal{F}_k$ とする．$\mathcal{F}_k$ の**候補集合** $\mathcal{C}_k$ とは $\mathcal{F}_{k-1}$ の各要素の最右拡張すべてからなる集合である．

最初に，このアルゴリズムはデータ木 D を巡回することにより頻出 1-パターンの集合 $\mathcal{F}_1$ を計算し，同時に，その出現位置を RMO_1 に格納する．次に $k \geq 2$ において，$\mathcal{F}_{k-1}$ と RMO_{k-1} より最右拡張手法をもちいて，$\mathcal{C}_k$ と RMO_k を同時に計算する．さらに，各 $T \in \mathcal{C}_k$ に対して $RMO_k(T)$ から $freq_D(T)$ を計算し，σ 頻出であれば T を $\mathcal{F}_k$ に加える．この操作を停止するまで繰り返すことにより，アルゴリズムは全ての σ 頻出パターンを発見する．

3.2 出現リストの更新

頻出パターン発見問題を高速に解くためには，パターン T からデータ木 D へのマッチング関数 φ を効率よく保持する必要がある．我々のアルゴリズムでは，T から D への全出現 $Total(\varphi) = (\varphi(1), \ldots, \varphi(k))$ を保持せずに，その部分的な情報を保持する戦略を採用する．

任意の k-パターン T に対して，φ に関する**最右出現**を $Rmo(\varphi) = \varphi(k)$ と定義し，$RMO(T) = \{Rmo(\varphi) \mid \varphi$ は T から D へのマッチング関数 $\}$ と定義する．すなわち，$RMO(T)$ は T の最右葉の出現先すべてからなる集合である．この集合を**最右葉出現リスト**と呼ぶ．例として，図 1 のデータ木 D とパターン木 T を考えよう．この例において，T は 3 つの最右出現（節点 4，節点 6，節点 10）をもつ．また，T の 2 つのルート出現（節点 2 と節点 7）は，それぞれの最右出現の親節点として簡単に計算できることに注意されたい．

S をパターン，T を S の最右拡張とする．このとき，次の補題 1 は $RMO(S)$ と $RMO(T)$ の関係を必要十分に特徴付ける．

補題 1 $(k-1)$-パターン S がデータ木 D に出現するとし，$\varphi : V_S \to V_D$ を S から D へのマッチング関数とする．また，S の (p, ℓ) 拡張を T とし，$\psi : V_T \to V_D$ を φ の関数としての拡張とする．すなわち，任意の $i = 1, \ldots, k-1$ に対して $\psi(i) = \varphi(i)$ が成り立つとする．このとき，以下の 2 条件 (1), (2) が成り立つことと，ψ が T から D へのマッチング関数であることは必要十分である．

(1) $\psi(k)$ は ${\pi_D}^p(\varphi(k-1))$ の子であり，$p \geq 1$ ならば ${\pi_D}^{p-1}(\varphi(k-1))$ より右の兄弟である．
(2) $L_D(\psi(k)) = \ell$.

補題 1 より，データ木 D の任意の節点 u に対して，$u \in RMO_k(T)$ が成立するための必要十分条件は，ある節点 $v \in RMO_{k-1}(S)$ が存在して，その $(p-1)$ 代前の親節点（更新原点と呼ぶ）$W = \pi_D^{p-1}(v)$ に対して，u が W の弟であり，ラベル ℓ を持つことである．この性質を利用して，入力 $\langle RMO(S), p, \ell \rangle$ から $RMO(T)$ を計算するアルゴリズム Update-RMO を，図 4 に示す．ここに，S は任意のパターン，T は S の (p, ℓ) 拡張である．

さて，補題 1 の性質を単純に利用して最右葉出現リスト更新アルゴリズムを実装すると，計算されるリストの中に同一節点が重複して現れる可能性がある [6]．そこで，図 4 の Update-RMO では，**重複検出** (Duplicate-Detection) と呼ばれる手法を用いて，最右葉出現リストから重複を除去して

Algorithm Update-RMO(RMO, p, ℓ)

1. $RMO_{\rm new}$ を空リスト ε で初期化する．$check := null$ とする．
2. 任意の $x \in RMO$ に対して，以下を繰り返す．
   (a) $p = 0$ ならば，y を x の長男とする．
   (b) それ以外のとき（$p \geq 1$），
   - $check = \pi_D^p(x)$ ならば，x への操作を終了してステップ 2 の最初に戻る．
   - それ以外のとき，$y := next(\pi_D^{p-1}(x))$, $check := \pi_D^p(x)$ とする．
     /*$next(v)$ で v の次の弟を表す*/
   (c) $y \neq null$ である限り，以下を繰り返す．
   - $L_D(y) = \ell$ ならば，リスト $RMO_{\rm new}$ に y を加える．
   - $y := next(y)$ を y の右隣の弟とする．
3. $RMO_{\rm new}$ を出力する．

図 4: 最右葉出現リスト更新アルゴリズム

いる．ここで，重複検出手法とは，アルゴリズム Update-RMO における以下の一連の操作のことである：$p \geq 1$ のとき，任意の $x \in RMO(T)$ に対して，アルゴリズムは変数 $check$ の値をチェックして $check = \pi_D^p(x)$ が成り立つかどうかを調べる．もし成り立つならば，アルゴリズムは更新原点 $W = \pi_D^{p-1}(x)$ の弟の走査をスキップして，$RMO(T)$ の次の要素の計算に移る．成り立たない場合は，$check$ の値を $\pi_D^p(x)$ に更新する．4 節の実験結果によると，重複検出手法は FREQT の計算効率の向上に著しく貢献する．

これから，アルゴリズム Update-RMO の正当性を証明する．集合 A に対して，$s = a_1 \cdots a_n \in A^*$ を A 上の文字列，$\leq_A$ を A 上の二項関係とする．$1 \leq i < j \leq n$ を満たす任意の i と j に対して $a_i <_A a_j$（$resp.$ $a_i \leq_A a_j$）が成り立つとき，s は $\leq_A$ に関して**単調**（$resp.$ **非交差**）であるという．文字列 $s = a_1 \cdots a_n \in A^*$ と，関数 $\xi : A \to A^*$ に対して，$\xi(s) = \xi(a_1) \cdots \xi(a_n)$ を仮定する．以下の議論では，$RMO(T)$ を集合ではなく節点列として扱う．

補題 2 T を任意のパターンとする．このとき，以下の 2 条件 (i),(ii) を仮定すれば，アルゴリズム Update-RMO が計算する最右葉出現リスト $RMO(T)$ には，T のすべての最右葉出現が重複せずに枚挙される．(i) 任意の 1-パターン T に対して，$RMO(T)$ のすべての要素が D のプリオーダー順で並んでおり，(ii) アルゴリズムは $RMO(T)$ のすべての要素を $RMO(T)$ の順番で走査する．

証明： 補題を証明するために，D の節点列 $RMO(T)$ が，k **次プリオーダー関係**と呼ばれる V_D 上の二項関係に関して，単調性を保存することを示す．まず，D の k **次プリオーダー列** Π_k とは，以下の (1),(2) で再帰的に定義される D の節点列である：(1) Π_1 は，D のプリオーダー順に並べられた，D のすべての節点からなる列であり，(2) $k \geq 2$ に対しては，$\Pi_k = C_D(\Pi_{k-1})$ である．ただし，$C_D : V_D \to (V_D)^*$ は，入力 $v \in V_D$ に対して，v のすべての子供が兄弟関係 $\preceq_D$ の順番でちょうど 1 回ずつ現れるような節点列 $C_D(v)$ を返す関数である．このとき，k 次プリオーダー関係 $\leq_k$ は，以下で定義される：任意の $v_1, v_2 \in V_D$ に対して，$v_1 \leq_k v_2$ iff k 次プリオーダー列 Π_k において v_1 は v_2 以前に出現する．この定義により，D の節点列 s が $\leq_k$ に関して単調であることと，s が Π_k の部分列であることは同値である．また，二項関係 $\leq_k$ に対して，

$$v_1 \leq_k v_2 \text{ ならば } {\pi_D}^p(v_1) \leq_{k-p} {\pi_D}^p(v_2) \quad (1)$$

が成り立つことが容易に証明できる．S を任意のパターンとし，S の最右葉の深さを $d = depth(rml(S))$ とおく．このとき，$RMO(S)$ を d 次プリオーダー関係 $\leq_d$ に関して単調な節点列だと仮定すると，(1) 式より節点列 ${\pi_D}^p(RMO(S))$ は $\leq_{d-p}$ に関する非交差列となる．ここで，アルゴリズム Update-RMO は，重複検出手法を用いて ${\pi_D}^p(RMO(S))$ 中の節点の重複を回避している．したがって，入力 $\langle RMO(S), p, \ell \rangle$ に対してアルゴリズムが出力する節点列 $RMO(T)$ は，$\leq_{d-p+1}$ に関して単調である．ここで，T は S の (p, ℓ) 拡張であり，このとき $d - p + 1 = depth(rml(T))$ が成り立つ．以上により，任意のパターン T に対して，アルゴリズムが計算する節点列 $RMO(T)$ は $\leq_d$ に関して単調であることが証明された．ただし $d = depth(rml(T))$ である．このことは，$RMO(T)$ が重複する節点を

Algorithm Expand-Trees($\mathcal{F}, RMO$)

1. $\mathcal{C} := \emptyset$; $RMO_{\rm new} := \emptyset$;
2. 任意のパターン $S \in \mathcal{F}$ と任意の $(p,\ell) \in \{0,\ldots,d-1\} \times \mathcal{L}$ に対して以下を行う．ただし，$d = depth(rml(S))$ である．
   - S の (p,ℓ) 拡張 T を計算する;
   - $RMO_{\rm new}(T)$:= Update-RMO($RMO(S), p, \ell$);
   - $\mathcal{C} := \mathcal{C} \cup \{T\}$;
3. $\langle \mathcal{C}, RMO_{\rm new} \rangle$ を出力する;

図 5: 候補パターン生成アルゴリズム

含まないことを意味する．したがって補題が示された． □

3.3 アルゴリズムの解析

図 5 に，$\mathcal{F}_k$ の候補集合 $\mathcal{C}_k$ と，対応する最右葉出現リストの集合 RMO_k を計算するアルゴリズム Expand-Trees を示す．ここで，RMO_k は，任意のパターン $T \in \mathcal{C}_k$ に対して T の D への最右葉出現リスト $RMO_k(T)$ を返す関数である．以上により，図 2 のアルゴリズム FREQT の正当性が証明される．

定理 3 ラベル集合 $\mathcal{L}$，$\mathcal{L}$ 上のデータ木 D，最小サポート $0 < \sigma \leq 1$ に対して，アルゴリズム FREQT はすべての σ 頻出パターンを重複せずに計算する．

アルゴリズム FREQT の計算時間は $O(k^2 bLN)$ である．ここに，k は極大頻出パターンのサイズ，b はデータ木 D の最大分岐数，$L = \#\mathcal{L}$，N は頻出パターンの最右葉出現リストの長さの総和である．また，M を極大頻出パターンの大きさの総和とすると，FREQT はたかだか $O(kLM)$ 個のパターンしか生成しない．

3.4 枝刈り

本節では，FREQT に対する枝刈り手法について述べる．

Node-Skip この枝刈りは，頻出 1-パターンの集合 $\mathcal{F}_1$ を用いた手法である．すなわち，ラベル $\ell \in \mathcal{L}$ をもつ 1-パターンが D において σ 頻出ではないと仮定すると，いかなる σ 頻出パターンの中にもラベル ℓ は出現しない．したがって，この枝刈り手法では，ℓ が非頻出ラベルならば，任意のパターン S に対する (p,ℓ) 拡張の作成と Update-RMO($RMO(S), p, \ell$) の呼び出しをスキップする。

Edge-Skip この枝刈りは，頻出 2-パターンの集合 $\mathcal{F}_2$ を用いた手法である．上の Node-Skip 手法と同様に，ラベル対 (ℓ_1, ℓ_2) をもつ 2-パターン E が非頻出パターンであると仮定すると，いかなる σ 頻出パターンの中にもラベル対 (ℓ_1, ℓ_2) は出現しない．したがって，この枝刈り手法では，(ℓ_1, ℓ_2) が非頻出ラベル対であるならば，$\pi^p_D(rml(S))$ のラベルが ℓ_1 となるような任意のパターン S に対して，その (p, ℓ_2) 拡張の作成と Update-RMO($RMO(S), p, \ell_2$) の呼び出しをスキップする。

3.5 動作例

入力を図 1 のデータ木 D，$\sigma = 0.2$，$\mathcal{L} = \{A, B\}$ としたときに，FREQT がすべての σ 頻出パターン求める様子を図 6 に示す．この図から，最右拡張によるパターンの生成経路は木を成しており，各パターンは一意な経路をとることが分かる．

4 実験

アルゴリズムの性能を評価するために，実際のウェブデータ上で実験を行った．実験では，以下に示される 3 種類のアルゴリズムを Java（SUN JDK1.3.1）と DOM（OpenXML）を用いて実装した．

- *FREQT without Duplicate-Detection*: 図 2 のアルゴリズム FREQT における重複検出

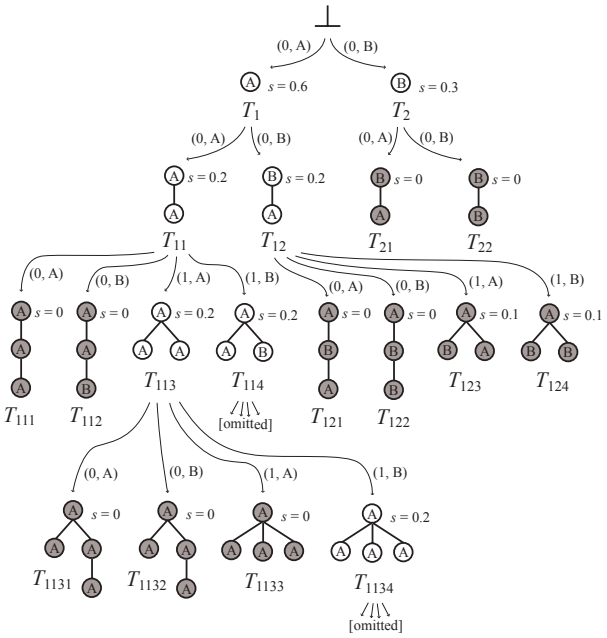

図 6: FREQT の動作例：パターン S からパターン T への矢印は，T が S の (p, ℓ) 拡張であることを表す．各パターンに付けられた値 s は出現頻度であり，白いパターンは頻出パターン，灰色のパターンは非頻出パターンを表す．

手法を用いず，かわりに出現リスト生成後に明示的に重複除去を行うアルゴリズム．

- *FREQT*: FREQT の素直な実装．
- *FREQT with Node-Edge-Skip*: FREQT に，3.4 節で述べた枝刈り手法を適用したアルゴリズム．

実験は，PC（Pentium III 600MHz, RAM 512MB, Linux 2.2.14）上で行った．

4.1 データ

実験のテストデータとして，*Citeseers*（5.6MB）と *Allsite*（3.6MB）を用いる．*Citeseers* は，オンライン論文目録サイト ReseachIndex[1]にて収集したウェブページ 189 文書の集積であり，*Allsite* は [11] に引用されている 30 のウェブサーチエンジンで収集したウェブページ 288 文書からなる混合データである．このようにして得られたテストデータは，DOM ツリーに変換される．さらに，DOM ツリー内の任意の属性-値対を，以下のようにして節点の集合に変換する．節点 v が属性-値対 $(Attr, Val)$ を持つとする．このとき，根ノードと葉ノードのラベルがそれぞれ $Attr$ と Val であるような，大きさ 2 の順序木 T を作成する．そして，T を $(Attr, Val)$ の辞書順で節点 v の下にぶら下げる．

[1] http://citeseer.nj.nec.com/

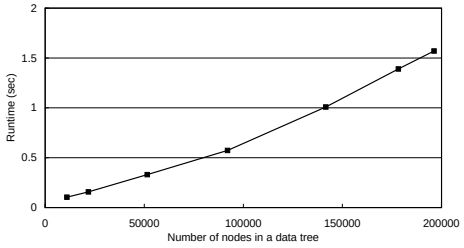

図 7: データサイズと計算時間

以上の前処理を *Citeseers* と *Allsites* に施して得られたデータ木のサイズは，節点数がそれぞれ 196,247，192,468 であり，異なるラベル数はそれぞれ 7,125，9,696 であった．

4.2 規模耐性

はじめに，*Citeseers* のデータサイズを 316KB から 5.62MB まで変化させて，それぞれの計算時間を測定した．最小サポート $\sigma = 3(\%)$ に対する実験結果を図 7 に示す．この実験において発見された極大頻出パターンの総数は，どのデータサイズについてもほぼ同数であった．一方，アルゴリズムが走査した最右出現の総数は，12,844 から 223,003 に線形的に増加した．

実験によると，アルゴリズムはデータ木のサイズ n に関して線形を少々上回る時間で動作する．この非線形性は，n が増加するにつれて，データ木の平均枝分かれ数が微妙に増加することが原因であると考えられる．

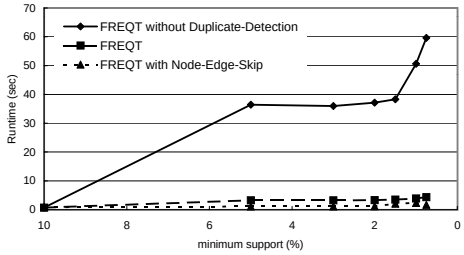

図 8: アルゴリズムの比較

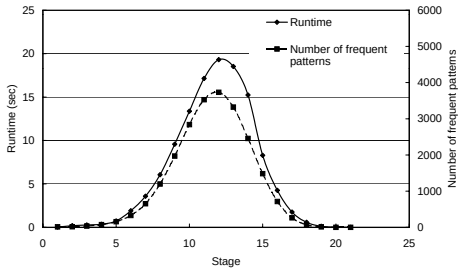

図 9: 計算時間と頻出パターン数

4.3 アルゴリズムの比較

次に，実装した 3 つのアルゴリズムに入力データ *Allsite* を与え，最小サポート σ を 10(%) から 1.75(%) まで変化させて計算時間を測定した．結果を図 8 に示す．$\sigma = 10(\%)$ のときは，頻出パターンとして 1-パターンしか発見されなかったので，アルゴリズムの性能に差は見られない．一方，$\sigma < 10(\%)$ のときは，*FREQT* は *FREQT without Duplicate-Detection* に比べて 10 倍以上高速であり，さらに，*FREQT with Node-Edge-Skip* は *FREQT* よりも 3 倍程度高速であることが観測された．例えば，$\sigma = 2(\%)$ における各アルゴリズムの計算時間は，それぞれ 37.15 (sec), 3.285 (sec), 1.151 (sec) である．

4.4 ステージごとの動作比較

我々は，アルゴリズム *FREQT* の動作を，より詳細に分析した．図 9 に，最小サポート $\sigma = 2(\%)$ に対する *FREQT* の計算時間と発見された頻出パターン数を，ステージごとに測定した結果を示す．この実験に用いた入力データは，*Citeseers* の一部（1.49MB）である．相関規則マイニングでは，ステージ 2 において頻出パターン数が最大になることが報告されており，図 9 に示される結果とは大きく異なる．このことから，木構造のマイニングには，相関規則マイニングとは異なる最適化が必要である．

No. 68, Size 6, Hit 1162, Freq 1.30%

```
<a href="_">
   <font color="#6F6F6F"> #text_1 </font>
</a>
```

No. 10104, Size 20, Hit 1039, Freq 1.17%

```
<p> #text_2
    <b> #text_3 <!-- CITE -->
        <font color="green"> #text_4 </font>
        #text_5
    </b>
    #text_6 <br /> <br />
    <font color="#999999">
        #text_7
        <i> #text_8 </i>
        #text_9
    </font>
</p>
```

図 10: 発見された頻出パターンの例

4.5 発見された頻出パターンの例

図 10 に，実験で見つかった頻出パターンのうちの 2 つを HTML 形式で表示した．これらは，最小サポート $\sigma = 1.17(\%)$ に対して，アルゴリズム *FREQT* がテストデータ *Citeseers* から発見したパターンである．これらの頻出パターンとテストデータを見比べると，論文目録データベースにおける書誌情報の反復を，アルゴリズムが正確に抜き出していることが観察される．

5 まとめ

本稿では，半構造データマイニング問題を効率よく解決するアルゴリズム FREQT を示し，その正当性を証明した．さらに，様々なウェブデータをもちいた実験を行い，アルゴリズムが充分な規模耐性を持つこと，および，重複検出手法と簡単な枝刈り手法が有効であることを示した．

実験により，大規模ウェブデータからの正規構造抽出に対する我々のアルゴリズムの有効性が明らかになった．したがって，本手法の応用とし

て，ウェブデータからの情報抽出（Information Extraction）[11, 15] などが考えられる．

このような，より現実的な応用を行うためには，半構造データの属性やテキストを扱えるよう提案手法を拡張する必要がある．また，グラフ構造 [10, 12, 13] や一階述語モデル [9] への拡張も今後の課題である．

参考文献

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

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

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

[4] H. Arimura, Efficient Learning of Semi-structured Data from Queries, In *Proc. the 12th International Conference on Algorithmic Learning Theory (ALT'01)*, LNAI, Washington, D.C., Nov. 2001. (To appear)

[5] H. Arimura, S. Arikawa, S. Shimozono, Efficient discovery of optimal word-association patterns in large text databases, *New Generation Computing*, 18, pp. 49–60, 2000.

[6] 浅井達哉，安部賢治，川副真治，有村博紀，有川節夫，“ 半構造データマイニングのための部分構造パターンの効率的探索 ”，信学技報，Vol.101, No.342, pp. 1–8, 2001.

[7] T. Asai, K. Abe, S. Kawasoe, H. Arimura, H. Sakamoto, S. Arikawa, Efficient Substructure Discovery from Large Semi-structured Data, In. *Proc. the Second SIAM Int'l Conf. on Data Mining (SDM2002)*, 2002. (To appear)

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

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

[10] A. Inokuchi, T. Washio, H. Motoda, An Apriori-Based Algorithm for Mining Frequent Substructures from Graph Data, In *Proc. PKDD 2000*, LNAI 1910, 2000, pp. 13-23.

[11] N. Kushmerick, Wrapper induction: efficiency and expressiveness, *Artificial Intelligence*, Vol.118, 2000, pp. 15–68.

[12] H. Mannila, C. Meek, Global Partial Orders from Sequential Data, In *Proc. KDD2000*, 2000, pp. 161–168.

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

[14] T. Miyahara, T. Shoudai, T. Uchida, K. Takahashi, H. Ueda, Discovery of Frequent Tree Structured Patterns in Semistructured Web Documents, In *Proc. PAKDD-2001*, pp. 47–52, 2001.

[15] K. Taniguchi, H. Sakamoto, H. Arimura, S. Shimozono and S. Arikawa, Mining Semi-Structured Data by Path Expressions, In *Proc. the 4th Int'l Conf. on Discovery Science*, LNAI, 2001. (To appear)

[16] 瀬々潤，森下真一，“ 最適なアソシエーションルールの効率的探索 ”，第 2 回データマイニングワークショップ資料集，日本ソフトウェア科学会，pp. 38–47, 2001.

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

[18] J. T. L. Wang, B. A. Shapiro, D. Shasha, K. Zhang, C.-Y. Chang, Automated Discovery of Active Motifs in Multiple RNA Seconary Structures, In *Proc. KDD-96*, pp. 70–75, 1996.

[19] K. Wang, H. Liu, Schema Discovery for Semistructured Data, In *Proc. KDD'97*, pp. 271–274, 1997.

[20] M. J. Zaki, Efficiently Mining Frequent Trees in a Forest, Computer Science Department, Rensselaer Ploytechnic Institute, PRI-TR01-7-2001, 2001.
`http://www.cs.rpi.edu/~zaki/PS/TR01-7.ps.gz`