数理解析研究所講究録 1375 巻 2004 年 113-119

大規模木構造データからの頻出無順序木パターン発見アルゴリズム
An Efficient Algorithm for Discovering Frequent Patterns from Large Unordered Trees

浅井達哉† Tatsuya Asai　房延慎二† Shinji Fusanobu　有村博紀† Hiroki Arimura　宇野毅明‡ Takeaki Uno　中野眞一* Shin-ichi Nakano

† 九州大学大学院システム情報科学府・研究院
Department of Informatics, Kyushu University
‡ 国立情報学研究所
National Institute of Informatics
* 群馬大学工学部
Department of Computer Science, Gunma University

1 はじめに

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文書に代表される構造化テキストデータがネットワーク上に大量に蓄積されている．これらは，一般的に半構造データ [2] と呼ばれ，新しい形態の大規模データとして急速に利用が進んでいる．さらに，ウェブサイトのリンク構造や化合物データベースなど，従来の関係データとして扱えない非定型・非正規データも多い．そのため，大規模半構造データを対象としたデータマイニング，すなわち，半構造データマイニング [1, 5, 6, 11, 17, 19] に対する要求が高まっている．

しかし，半構造データは，木やグラフ構造で表わされる複雑な非正規データである．そのため，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 [3] を，直接に半構造データに適用することは難しく，効率よい半構造データマイニング手法の開発が緊急の課題となっている．

本稿では，与えられたラベルつき無順序木の集合から，多頻度で出現するすべての部分構造パターンを発見する問題を考える．**ラベルつき無順序木**とは，有向根付き木の各節点にラベルをつけたものである．

グラフ構造のデータマイニングは，パターンの同型性判定や出現の計算など，計算困難な問題を含む [9, 10, 16, 18]．我々は，無順序木の正規形表現を導入し，すべての無順序木の正規形表現を重複せずに列挙することにより，無順序木パターンに対する同型性判定問題を回避する．また，パターンの出現として埋め込み出現を定義し，これを漸増的に計算することにより，無順序木パターンの出現位置を効率よく計算する．

我々は，以上の手法を組み合わせて，与えられた無順序木の集積に頻出するすべての無順序木パターンを効率よく計算するアルゴリズムUNOTを開発した [7]．このアルゴリズムは，パターン1つあたり $O(kb^2m)$ 時間ですべての頻出パターン T を計算する．ここに，k は T の大きさであり，b はデータ木の最大枝分かれ数，m は T のデータ木への総出現数である．また，UNOTの入力を無順序木から一般のグラフに拡張することは容易である．応用として化合物データベースからの木パターン発見などがある（図1）．

無順序木パターンを扱うデータマイニングの研究は，あまり行われていない．Termierら [15] は，半構造データからの頻出無順序木パターン発見問題を考察し，これを解くアルゴリズムTreeFinderを開発した．しかし，この手法の完全性は保障されていない．最近，Nijssenら [14] が同じ問題を考察し，我々とは独立に，UNOTに似た効率よいアルゴリズムを与えている．

本稿の構成は以下のとおりである．2節では，定義と記法を導入し，本稿で考察する問題を定式化する．3節で無順序木の正規形を導入する．4節で頻出無順序木パターン発見アルゴリズムUNOTを説明する．5節で実データを用いた実験について述べる．最後に，6節で本稿をまとめる．

2 準備

2.1 半構造データのモデル

集合 A に対して，A の大きさを $|A|$ であらわす．本稿では，半構造データとパターンの形式的なモデルとして，ラベルつき無順序木を採用する．ここで，ラベル付き無順序木とは，閉路をもたない根付き有向グラフ (DAG) であり，根以外の節点がちょうど一つの親をもつようなものである [4]．形式的には，次のように定義する．

$\mathcal{L} = \{\ell, \ell_1, \ell_2, \ldots\}$ をラベルの可算集合とする．$\mathcal{L}$ 上には全順序 $\leq_{\mathcal{L}}$ が与えられていると仮定する．$\mathcal{L}$ 上の**ラベルつき無順序木** (labeled unordered tree) とは，次の条件を満たす四項組 $U = (V, E, r, label)$ である．

[0]連絡先：浅井達哉，〒812-8581 福岡市東区箱崎 6-10-1，九州大学大学院システム情報科学府情報理学専攻，Phone: 092-642-2697, FAX: 092-642-2698, Mail: t-asai@i.kyushu-u.ac.jp

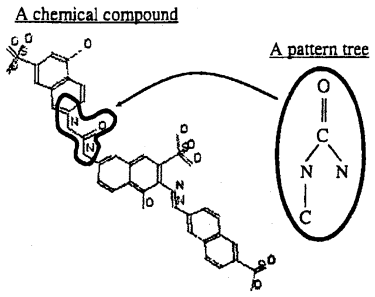

図 1: 化合物データからの木パターン発見

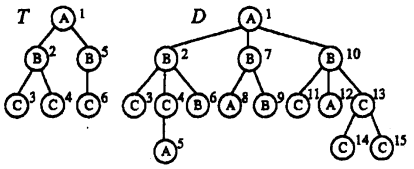

図 2: ラベルつき無順序木：T をパターン，D をデータ木と思うと，T は D に出現している．

$G = (V, E, r)$ は $r \in V$ を根とする木である．V は節点の集合をあらわす有限集合であり，辺集合 $E \subseteq V^2$ は G の親子関係を示す．$label : V \to \mathcal{L}$ はラベル関数をあらわす．$(u, v) \in E$ ならば，u は v の**親** (parent) である，および v は u の**子** (child) であるという．節点 $v \in V$ の**深さ** $dep(v)$ を，U の根 r から v への経路の長さ，すなわち経路上の辺の数と定義する．

次に，$\mathcal{L}$ 上の**ラベルつき順序木** (labeled ordered tree) とは，次の条件を満たす五項組 $T = (V, E, B, r, label)$ である．$V, E, r, label$ は，無順序木の場合と同様に定義される．$B \subseteq V^2$ は，T の兄弟関係をあらわす半順序関係である [5]．

$\mathcal{U}$ と $\mathcal{T}$ を，それぞれ $\mathcal{L}$ 上の無順序木と順序木のクラスと定義する．ラベルつき木 $T = (V, E, r, label)$ に対して，文脈から明らかな場合は，V および $E, r, label$ をそれぞれ $V_T, V_E, r_T, label_T$ と表記する．

T を任意の (順序または無順序) 木とする．T の任意の節点 u, v に対して，$(u, v) \in E^*$ ならば，u は v の**先祖** (ancestor) である，または u は u の**子孫** (decendant) であるという．節点 v に対して，v を根とし，T における v のすべての子孫によって構成される無順序木を，U の v に関する**部分木** (subtree) と呼び，$T(v)$ と表記する．T の**大きさ** $|T|$ を，$|T| = |V|$ と定義する．空木 $\perp$ を，大きさが 0 の木と定義する．

以降では，大きさ $k \geq 1$ のラベルつき順序木 $T = (V, E, B, r, label)$ に対して，その節点集合は $V = \{1, \ldots, k\}$ であり，それらはプリオーダー順で番号付けられていると仮定する．プリオーダーとは，順序木を反時計回りに深さ優先探索して，各節点を，はじめて通ったときに出力して得られる順序である [4]．

このとき，T の根は $root(T) = 1$ であり，最右葉は $rml(T) = k$ であることに注意されたい．また，T の**最右枝** (rightmost branch) を，T の根から最右葉への経路 $RMB(T) = (r_0, \ldots, r_c)$ $(c \geq 0)$ と定義する．

2.2 パターンと照合写像と出現

無順序木パターン (または，パターン) とは，任意のラベルつき無順序木である．自然数 $k \geq 0$ に対して，大きさ k のパターンを **k-パターン**と呼ぶ．無順序木の有限集合 $\mathcal{D} = \{D_1, \ldots, D_n\} \subseteq \mathcal{U}$ を**データベース**と呼び，各 $D_i (i = 1, \ldots, n)$ を**データ木** (または**文書**) と呼ぶ．$\mathcal{D}$ 中の全節点の集合を $V_{\mathcal{D}}$ と表記する．また，$\|\mathcal{D}\| = |V_{\mathcal{D}}| = \sum_{D \in \mathcal{D}} |V_D|$ と定義する．

無順序木パターンの意味は，以下で定義される照合写像で与えられる．T と D を，それぞれ $\mathcal{L}$ 上のパターンとデータ木とする．このとき，任意の $x, y \in V_T$ に対して，次の (1)–(3) を満たす写像 $\varphi : V_1 \to V_2$ が存在するならば，T は D に**出現する**という：

(1) φ は単射である．すなわち，任意の $x, y \in V_1$ に対して，$x \neq y$ ならば $\varphi(x) \neq \varphi(y)$ が成り立つ．

(2) φ は親子関係を保存する．すなわち，任意の $x, y \in V_1$ に対して，$(x, y) \in E_1 \iff (\varphi(x), \varphi(y)) \in E_2$ が成り立つ．

(3) φ はラベル値を保存する．すなわち，任意の $x \in V_1$ に対して，$L_1(x) = L_2(\varphi(x))$ が成り立つ．

φ を T から D への**照合写像** (matching) と呼ぶ．データ木 D への照合写像 $\varphi : V_T \to V_D$ を，データベース $\mathcal{D}$ への照合写像 $\varphi : V_T \to 2^{V_{\mathcal{D}}}$ に自然に拡張する．また，パターン T からデータベース $\mathcal{D}$ へのすべての照合写像の集合を，$\mathcal{M}^{\mathcal{D}}(T)$ であらわす．

定義 1 自然数 $k \geq 1$ に対して，$T \in \mathcal{U}$ を任意の k パターンとし，$\mathcal{D}$ をデータベースとする．また，$\varphi : V_T \to V_{\mathcal{D}} \in \mathcal{M}^{\mathcal{D}}(T)$ を T から $\mathcal{D}$ への任意の照合写像とする．このとき，T の $\mathcal{D}$ への出現として，次の 4 つを定義する：

1. T の**全出現** (total occurrence) とは，k 項組 $TO(\varphi) = \langle \varphi(1), \ldots, \varphi(k) \rangle \in (V_{\mathcal{D}})^k$ である．
2. T の**埋め込み出現** (embedding occurrence) とは，節点集合 $EO(\varphi) = \{\varphi(1), \ldots, \varphi(k)\} \subseteq V_{\mathcal{D}}$ である．
3. T の**根出現** (root occurrence) とは，$\mathcal{D}$ の節点 $RO(\varphi) = \varphi(1) \in V_{\mathcal{D}}$ である．
4. T の**文書出現** (document occurrence) とは，$EO(\varphi) \subseteq V_{D_i}$ が成り立つような文書番号 $DO(\varphi) = i$ $(1 \leq i \leq |\mathcal{D}|)$ である．

任意の出現の種類 $\tau \in \{TO, EO, RO, DO\}$ に対して，パターン T のデータベース $\mathcal{D}$ への **τ-出現数**を

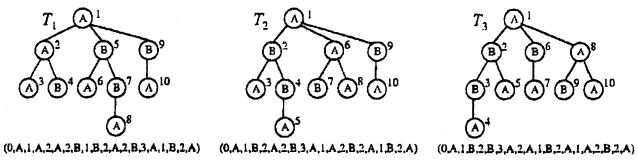

図 3: ラベルつき順序木の深さラベル列

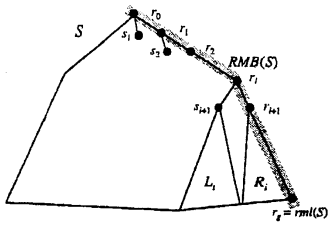

図 4: 正規形表現に関する記法

$|\tau^{\mathcal{D}}(U)|$ で定義する．また，T の $\mathcal{D}$ への τ-**出現頻度**（または，頻度）を，$\tau \in \{TO, EO, RO\}$ のときは $freq_{\mathcal{D}}(T) = |\tau^{\mathcal{D}}(T)|/||\mathcal{D}||$ と定義し，$\tau = DO$ のときは $freq_{\mathcal{D}}(T) = |DO^{\mathcal{D}}(T)|/|\mathcal{D}|$ と定義する．パターンの出現頻度の閾値として，ユーザは**最小支持度**（minimun support）と呼ばれる任意の実数 $0 \leq \sigma \leq 1$ を与える．以上にもとづき，本稿で考察するデータマイニング問題を導入する．

出現種類 τ に関する頻出無順序木パターン発見問題
与えられたデータベース $\mathcal{D} \subseteq \mathcal{U}$ と最小支持度 $0 \leq \sigma \leq 1$ に対して，$\mathcal{D}$ への τ-出現頻度が σ 以上となるような無順序木パターン $T \in \mathcal{U}$ を，すべて見つけよ．

本稿では，EO に関する頻出無順序木パターン発見問題について議論する．この問題の RO や DO への拡張は容易である．

3 無順序木の正規形表現

本節では，[13] にしたがい，無順序木の正規形表現を導入する．

3.1 ラベルつき順序木の深さラベル列

本稿では，任意のラベルつき無順序木をラベルつき順序木で表現する．すなわち，任意の無順序木 $U \in \mathcal{U}$ に対して，T から兄弟関係 B_T を無視して得られる木が U と一致するような順序木 $T \in \mathcal{T}$ が存在するので，これをこのような T を U の表現として用いる．2つの順序木 $T_1, T_2 \in \mathcal{T}$ が同一の無順序木 $U \in \mathcal{U}$ を表現するとき，T_1 と T_2 は互いに**同値である**といい，$T_1 \equiv T_2$ と表記する．

大きさ k のラベルつき順序木を，以下のように符号化する [6, 12, 19]．T を大きさ k のラベルつき順序木とする．このとき，T の**深さラベル列**（depth-label sequence）とは，系列

$$C(T) = ((dep(v_1), label(v_1)), \ldots, (dep(v_k), label(v_k))$$

である．ただし，$(v_1, \ldots, v_k)$ は，T の節点をプリオーダー順に並べた系列であり，$(dep(v_i), label(v_i)) \in \mathbf{N}\times\mathcal{L}$ は，節点 $v_i \in V_T$ の**深さラベル対**（depth-label pair）である．$\mathcal{T}$ と $\mathbf{N}\times\mathcal{L}$ は一対一に対応するので，以降は順序木 T とその深さラベル列 $C(T)$ を同一視する．図 3 に深さラベル列の例を示す．

次に，深さラベル対上の全順序 $\geq \subseteq (\mathbf{N}\times\mathcal{L})^2$ を導入する．任意の深さラベル対 $(d_i, \ell_i) \in \mathbf{N}\times\mathcal{L}$ $(i = 1, 2)$ に対して，$(d_1, \ell_1) > (d_2, \ell_2)$ であるのは，(i) $d_1 > d_2$ が成り立つとき，または (ii) $d_1 = d_2$ かつ $\ell_1 > \ell_2$ が成り立つときである．全順序 $\geq$ から生成される辞書式順序 $\geq_{\mathrm{lex}}$ は，深さラベル列上の全順序を与える．すなわち，任意の順序木 T_1, T_2 に対して，$C(T_1) \geq_{\mathrm{lex}} C(T_2)$ と $C(T_2) \geq_{\mathrm{lex}} C(T_1)$ のいずれかが成り立つ．

ラベルつき無順序木の正規形表現を，以下で定義する．

定義 2 ([13]) T を任意のラベルつき順序木とする．T と同値な任意のラベルつき順序木 S に対して，$C(T) \geq_{\mathrm{lex}} C(S)$ が成り立つとき，T を**正規形**と定義する．

ラベルつき無順序木 $U \in \mathcal{U}$ を表現する正規形順序木 $T \in \mathcal{T}$ を，U の**正規形表現**と呼ぶ．また，$\mathcal{L}$ 上のラベル付き無順序木の正規形表現全体を $\mathcal{C} \subseteq \mathcal{T}$ と定義する．

次の補題は，無順序木の正規形表現に関する重要な特徴づけを与える．

補題 1 (左荷重条件 ; Left-heavy condition [13]) ラベルつき順序木 T がある無順序木の正規形表現となることの必要十分条件は，T が**左荷重**（*left-heavy*）となること，すなわち，T の任意の節点 $v_1, v_2 \in V$ に対して，$(v_1, v_2) \in B$ ならば $C(T(v_1)) \geq_{\mathrm{lex}} C(T(v_2))$ が成り立つことである．

図 3 に示される 3 つの相異なる順序木 T_1, T_2, T_3 は，同一の無順序木を表現する．ラベル間の順序として $A > B > C$ を仮定すると，T_1 は左荷重条件を満たすことから正規形であることが分かる．一方，T_2 と T_3 は左荷重条件を満たさないので，正規形ではない．

3.2 逆探索と最右拡張

逆探索（reverse search）とは，列挙問題を高速に解くための効率よいアルゴリズム構成法の 1 つである [8]．逆探索では，列挙問題の解空間 $\mathcal{S}$ に対して，任意の解 $X \in \mathcal{S}$ が唯一の親 $P(X)$ をもつように，親子関係 $P \subseteq \mathcal{S}\times\mathcal{S}$ を導入する．このとき，親子関係 P は解

Algorithm UNOT($\mathcal{D}, \mathcal{L}, \sigma$)
入力：データベース $\mathcal{D} = \{D_1, \ldots, D_m\}$ $(m \geq 0)$，ラベル集合 $\mathcal{L}$，最小支持度 $0 \leq \sigma \leq 1$.
出力：頻出無順序木パターンの集合 $\mathcal{F} \subseteq \mathcal{C}$.
手法：
1. $\mathcal{F} := \emptyset$; $\alpha := \lceil |\mathcal{D}|\sigma \rceil$; /* 初期化 */
2. 任意のラベル $\ell \in \mathcal{L}$ について，以下をおこなう：
   $T_\ell := (0, \ell)$;
   Expand($T_\ell, \mathcal{O}, 0, \alpha, \mathcal{F}$);
3. $\mathcal{F}$ を出力する; /* 頻出パターン集合 */

図 5: 頻出無順序木パターン発見アルゴリズム UNOT

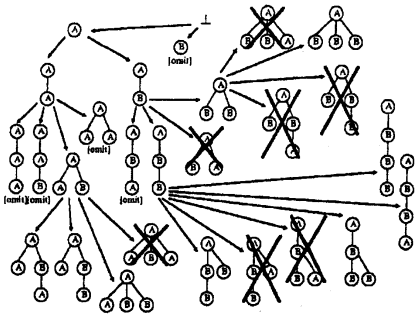

図 6: ラベルつき無順序木の探索木

空間 $\mathcal{S}$ 上の探索木を構成するので，根からはじめて，順に子供を計算することにより，すべての解を重複せずに列挙することができる．

T を大きさ 2 以上のラベルつき順序木とする．T から最右葉 $rml(T)$ を除去して得られる順序木を $P(T)$ と定義する．$P(T)$ を T の親と呼び，T を $P(T)$ の子と呼ぶ．補題 1 より，次の補題が成り立つ．

補題 2 ([13]) 任意のラベルつき順序木 $T \in \mathcal{T}$ に対して，T が正規形ならば，その親 $P(T)$ も正規形である．すなわち，$T \in \mathcal{C}$ ならば $P(T) \in \mathcal{C}$ が成り立つ．

この補題は，無順序木の正規形表現も逆探索手法で列挙可能なことを示している．

定義 3 ([5, 12, 19]) $S \in \mathcal{T}$ を $\mathcal{L}$ 上のラベルつき順序木とする．S の最右枝 $RMB(S)$ 上の節点に，新たな節点 v を一番右の子となるように付け加えて得られるラベルつき順序木 $T \in \mathcal{T}$ を，S の**最右拡張** (rightmost expansion) という．特に，$(dep(v), label(v)) = (d, \ell)$ のとき，T を S の (d, ℓ) **拡張**と呼ぶ．$\perp$ の $(0, \ell)$ 拡張を，ラベル ℓ をもつ大きさ 1 の木と定義する．

新しく付加される節点 v は，T のプリオーダーにおける最後の節点となることから，S の (d, ℓ) 拡張を $S \cdot (d, \ell)$ とも書く．

4 頻出無順序木パターン発見

本節では，埋め込み出現に関する頻出無順序木パターン発見問題を効率よく解くアルゴリズム UNOT について述べる．

4.1 アルゴリズムの概要

図 5 にアルゴリズム UNOT を示す．このアルゴリズムは，与えられたデータベース $\mathcal{D}$ に頻出するすべての無順序木の正規形表現を効率よく発見する．

アルゴリズムの基本アイディアは，無順序木の正規形表現の高速な列挙と，パターンの埋め込み出現の漸増的な計算である．UNOT は，まず，図 5 の部分手続き FindAllChildren を用いて，パターン 1 つ当たり定数時間ですべての正規形表現を重複せずに列挙する．次に，図 10 の部分手続き UpdateOcc を用いて，パターン 1 つあたり $O(kb^2m)$ 時間ですべての埋め込み出現を計算する．ここに，k はパターン T の大きさであり，b はデータ木の最大枝分かれ数，m は T のデータ木への総出現数である．

図 6 は，アルゴリズム UNOT が $\mathcal{L} = \{A, B\}$ 上のサイズ 4 以下のラベルつき無順序木を計算する様子を図示している．図中の矢印はラベルつき順序木の親子関係をあらわし，×印は正規形でない順序木をあらわす．[omit] 印のところでは図を省略しているが，他と同様に列挙を行うものとする．

4.2 無順序木の列挙

はじめに，いくつかの概念と記法を準備する (図 4)．T をラベルつき順序木とし，その最右枝を $RMB(T) = (r_0, r_1, \ldots, r_g)$ とおく．任意の自然数 $i = 0, 1, \ldots, g$ に対して，r_i が 2 つ以上の子供をもつとき，r_i の子供かつ r_{i+1} の直前の兄である節点を s_{i+1} と書く．すなわち，s_{i+1} は r_i の最後から 2 番目の子供である．また，$L_i = T(s_{i+1})$ を r_i の**左木**と，$R_i = T(r_{i+1})$ を r_i の**右木**と呼ぶ．r_i がちょうど 1 つの子供 r_{i+1} をもつとき，$L_i = \mathsf{T}_\infty$ と定義する．ここで，無限木 T_∞ は，任意の $S \in \mathcal{T}$ に対して $\mathsf{T}_\infty >_{\mathrm{lex}} S$ が成り立つ特別な木である．

与えられた順序木 T が左荷重かどうか調べるために，アルゴリズムは，T の各深さにおける左木と右木の大小関係のみをチェックすれば十分である [13]．

$RMB(T) = (r_0, r_1, \ldots, r_g)$ を T の最右枝とする．任意の $i = 0, 1, \ldots, g-1$ に対して，$C(R_i)$ が $C(L_i)$ の接頭辞のとき，節点 r_i を**アクティブ**と呼ぶ．T の**コピー深さ** (copy depth) とは，T のアクティブ節点の中でもっとも浅い節点の深さである．また，最右葉 r_g は常にアクティブであるとする．このとき，T が r_g 以外にアクティブ節点をもたないならば，コピー深さが g となることに注意されたい．

Procedure Expand($S, \mathcal{O}, c, \alpha, \mathcal{F}$)
入力：正規形表現 $S \in \mathcal{U}$，埋め込み出現 $\mathcal{O} = EO^{\mathcal{D}}(S)$，コピー深さ c，非負整数 α，頻出パターン集合 $\mathcal{F}$.
手法：
- $|\mathcal{O}| < \alpha$ ならば手続きを終了する；それ以外のとき，$\mathcal{F} := \mathcal{F} \cup \{S\}$ とする；
- 任意の $\langle S{\cdot}(i,\ell), c_{\mathrm{new}} \rangle \in$ FindAllChildren(S, c) について，以下をおこなう：
  - $T := S{\cdot}(i,\ell)$;
  - $\mathcal{P} :=$ UpdateOcc$(T, \mathcal{O}, (i,\ell))$;
  - Expand$(T, \mathcal{P}, c_{\mathrm{new}}, \alpha, \mathcal{F})$;

図 7: 深さ優先手続き Expand

以降では，与えられた正規形表現 $T \in \mathcal{C}$ の子供（**正規最右拡張**と呼ぶ）をすべて生成するアルゴリズム FindAllChildren（図 8）について述べる．アルゴリズム FindAllChildren が，解 1 つあたり定数時間ですべての正規最右拡張を列挙できるよう，実装上の工夫としてパターンに図 9 のデータ構造を導入する．

- 深さラベル対の配列 $code : [1..size] \to (\mathbf{N} \times \mathcal{L})$. ここにはパターン T（大きさ $size \geq 0$）の深さラベル列を格納する．
- 三項組 $(left, right, cmp)$ のスタック $RMB : [0..top] \to (\mathbf{N}\times\mathbf{N}\times\{=,\neq\})$. $(left, right, cmp) = RMB[i]$ に対して，$left$ と $right$ は，それぞれ $code$ における左木 L_i と右木 R_i の開始位置を指すポインタである．フラグ $cmp \in \{=,\neq\}$ は，$L_i = R_i$ が成り立つか否かを記録する．パターンの最右枝の長さを，$top \geq 0$ であらわす．

このデータ構造を用いることにより，図 8 の FindAllChildren において，すべての演算が定数時間で動くように実装できる．ただし，解である正規最右拡張は，親の木との差分のみを出力すると仮定する．

次の補題は，ラベルの扱いを除けば，[13] と同様に証明できる．

補題 3 ([13]) 任意の正規形表現 S と，そのコピー深さ $k \geq 0$ に対して，図 8 のアルゴリズム FindAllChildren は，S のすべての正規最右拡張 T を，解 1 つあたり $O(1)$ 時間で計算する．ただし，T と S との差分のみを出力すると仮定する．

補題 1 に基づいて，アルゴリズム FindAllChildren を素直に構築すると，パターンの大きさ k に対して，解である正規最右拡張 1 つあたり $O(k^2)$ 時間の計算時間を要する．したがって，このアルゴリズムは，素朴なアルゴリズムに比べて非常に効率がよいといえる．

Procedure FindAllChildren(S, k)
入力：無順序木の正規形表現 S と，S のコピー深さ k.
手法：S の正規最右拡張 T と，T のコピー深さ c の組 $\langle T, c \rangle$ をすべて出力する．T と c は，以下の場合分けにしたがって計算される：

Case I $C(L_k) = C(R_k)$ のとき：
- S の正規最右拡張は，$S{\cdot}(1,\ell_1), \ldots, S{\cdot}(k+1,\ell_{k+1})$ である．ただし，任意の $i = 1, \ldots, k+1$ について，$label(r_i) \geq \ell_i$ を満たすとする．一方，$S{\cdot}(k+2,\ell_{k+2}), \ldots, S{\cdot}(g+1,\ell_{g+1})$ は正規形でない．
- 任意の $i = 1, \ldots, k+1$ に対して，$S{\cdot}(i,\ell_i)$ のコピー深さは，$label(r_i) = \ell_i$ ならば $i-1$ であり，それ以外のときは i である．

Case II $C(L_k) \neq C(R_k)$ のとき：
- $m = |C(R_k)| + 1$ とし，$w = (d,\ell)$ を $C(L_k)$ の m 番目の要素とする．このとき，S の正規最右拡張は $S{\cdot}(1,\ell_1), \ldots, S{\cdot}(d,\ell_d)$ である．ただし，任意の $i = 1, \ldots, d-1$ について $label(r_i) \geq \ell_i$ を満たし，$i = d$ のときは $\ell \geq \ell_d$ を満たすとする．
- 任意の $i = 1, \ldots, d-1$ に対して，$S{\cdot}(i,\ell_i)$ のコピー深さは，$label(r_i) = \ell_i$ ならば $i-1$ であり，それ以外のときは i である．$S{\cdot}(d,\ell_d)$ のコピー深さは，$w = v$ ならば k であり，それ以外のときは d である．

図 8: すべての正規最右拡張を計算する手続き

4.3 出現リストの更新

本節では，正規形表現 S の埋め込み出現 $EO^{\mathcal{D}}(S)$ から，その正規最右拡張 T の埋め込み出現 $EO^{\mathcal{D}}(T)$ を漸増的に計算する方法を与える．図 10 に，これを実現する部分手続き UpdateOcc を示す．

T を，大きさ k のラベルつき無順序木の正規形表現とする．また，T から $\mathcal{D}$ への照合写像を $\varphi \in \mathcal{M}^{\mathcal{D}}(T)$ とおく．T の φ に関する全出現と埋め込み出現は，それぞれ $TO(\varphi) = \langle \varphi(1), \ldots, \varphi(k) \rangle$ と $EO(\varphi) = \{\varphi(1), \ldots, \varphi(k)\}$ で与えられる．文脈から明らかな場合は φ と $TO(\varphi)$ を同一視する．

我々は，埋め込み出現 EO を，$EO = EO(\varphi)$ であるような全出現 φ の 1 つを用いて符号化する．しかし，

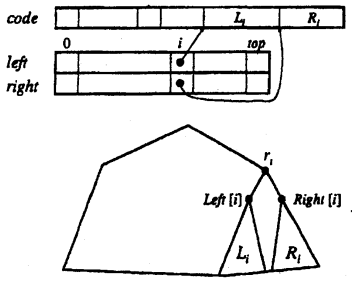

図 9: パターンのデータ構造

Algorithm UpdateOcc($T, \mathcal{O}, d, \ell$)
入力： S の正規最右拡張 T，全出現リスト $\mathcal{O} = TO^{\mathcal{D}}(S)$，$T$ の最右葉の深さラベル対 (d, ℓ).
出力： 更新された全出現リスト $\mathcal{P} = TO^{\mathcal{D}}(T)$.
手法：
- $\mathcal{P} := \emptyset$;
- 任意の $\varphi \in \mathcal{O}$ に対して，以下をおこなう：
  + $x := \varphi(r_{d-1})$;
  + x の任意の子節点 y に対して，以下をおこなう：
    - $label_D(y) = \ell$ かつ $y \notin E(\varphi)$ ならば，$\xi := \varphi \cdot y$ とする;
    - ξ が正規全出現ならば，$\mathcal{P} = \mathcal{P} \cup \{\xi\}$ とする;
- $\mathcal{P}$ を出力する;

図 10: パターンの埋め込み出現更新アルゴリズム

ある埋め込み出現 EO に対応する全出現は，最悪で指数個存在する．そこで，3 節で順序木の正規形を導入したように，全出現にも正規形を導入し，これを埋め込み出現をあらわす表現として用いる．

$EO(\varphi_1) = EO(\varphi_2)$ が成り立つとき，2 つの全出現 φ_1 と φ_2 は**同値である**という．φ_1 が，自然数列 $\mathbf{N}^*$ として辞書式順序で φ_2 より大きいことを，$\varphi_1 \geq_{\mathrm{lex}} \varphi_2$ と表記する．埋め込み出現の正規形表現を以下で定義する．

定義 4 T をラベルつき無順序木の正規形表現とし，$EO \subseteq V_{\mathcal{D}}$ を T の $\mathcal{D}$ への埋め込み出現とする．このとき，EO の正規形表現 $CR(EO)$ を，同値類 $\{\varphi' \in \mathcal{M}^{\mathcal{D}}(T) \mid \varphi' \equiv \varphi\}$ の中で辞書式順序が最大となる全出現と定義する．

$\varphi = \langle \varphi(1), \ldots, \varphi(k) \rangle$ を T の全出現とする．φ から最後の要素 $\varphi(k)$ を除去して得られる全出現を，φ の**親出現** (parent occurrence) と呼び，$P(\varphi)$ と書く．T の節点 $v \in V_T$ に対して，部分木 $T(v)$ への φ の制限を $\varphi(T(v)) = \langle \varphi(i), \varphi(i+1), \ldots, \varphi(i+|T(v)|-1) \rangle$ とする．ここで，$\langle i, i+1, \ldots, i+|T(v)|-1 \rangle$ は，プリオーダー順に並べた $T(v)$ の節点列である．

以上の準備に基づき，埋め込み出現の漸増的な計算法を与えよう．S を正規形順序木とし，φ を S の $\mathcal{D}$ への正規全出現とする．$T = S \cdot v$ を S の正規最右拡張とし，その最右枝を $(r_0, \ldots, r_g)$ とおく．このとき，データ木の節点 $w \in V_{\mathcal{D}}$ に対して，写像 $\xi = \varphi \cdot w$ が T の正規全出現になるための必要十分条件は，以下の条件 (1)–(4) が成り立つことである [7]：

(1) $label_D(w) = label_T(v)$.
(2) 任意の $i = 1, \ldots, k-1$ に対して，$w \neq \varphi(i)$ である．
(3) w は $\varphi(r_{d-1})$ の子節点である．ただし，$d = dep(v)$.
(4) 任意の $i = 0, \ldots, g-1$ に対して，$C(L_i) = C(R_i)$ ならば $\xi(root(L_i)) < \xi(root(R_i))$ が成立する．

図 9 のデータ構造に加えて，最右枝の各深さにおいて右木と左木の最長共通接頭辞長 $LCP \in \mathbf{N}$ を管理することにより，このアルゴリズムにおける $C(L_i) = C(R_i)$ の判定を定数時間で行うことができる．また，すべての正規形順序木は，少なくとも 1 つの正規最右拡張をもつことに注意せよ．

以上より，本稿の主結果が導出される．

定理 4 $\mathcal{D}$ をデータベースとし，$0 \leq \sigma \leq 1$ を最小支持度とする．このとき，図 5 のアルゴリズム UNOT は，埋め込み出現に関するすべての頻出無順序木の正規形表現 T を，パターン 1 つあたり $O(kb^2m)$ 時間で計算する．ここで，k はパターンの最大サイズであり，b は入力データ木の最大枝分かれ数，m はパターン T の埋め込み出現数である．

5 実験

本節では，実際の XML データを用いて実験を行い，アルゴリズムの性能を評価する．

5.1 方法

我々は，アルゴリズム UNOT を Java で実装し，PC (Pentium4 2.5GHz, 1GB RAM, Windows XP) 上で実験を行った．実装においては，大きさ 1 の非頻出パターンを用いた枝刈り [5, 18] を導入している．

実験に用いるデータは，オンライン論文目録データベース DBLP で公開されている XML データ[1]である．このデータは，対応するデータ木の兄弟におけるタグの出現順序がほぼ固定されているので，簡単なスクリプトを用いて，データ木の兄弟節点をランダムに並べ替えた．さらに，上の XML データから属性とテキスト値を除去した．

以上の加工を施して得られた 11MB の XML ファイルを実験データとして用いる．対応するデータ木の大きさは 1,056,223(nodes) であり，異なるラベル数は 31 個である．

5.2 規模耐性

はじめに，アルゴリズム UNOT の規模耐性を検証する．実験では，最小支持度を $\sigma = 2(\%)$ に固定して，データサイズを 10K(nodes) から 1000K(nodes) まで増やしながら，アルゴリズムの計算時間を計測した．その結果を図 11 に示す．

この図より，アルゴリズムはデータサイズに関して線形時間で動作することが分かる．よって，アルゴリズムは充分な規模耐性を持つといえる．

[1] http://dblp.uni-trier.de/xml/dblp.xml

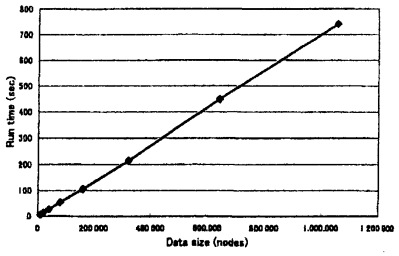

図 11: 規模耐性実験

表 1: UNOT と FREQT の比較：( ) 内は FREQT が計算した順序木パターンのうち，無順序木として相異なるものの個数をあらわす

最小支持度 (%)	頻出パターン数		最大サイズ	
	UNOT	FREQT	UNOT	FREQT
1	521	284 (90)	10	5
3	297	67 (41)	9	4
5	136	43 (32)	8	3
10	37	11 (11)	6	2

5.3 順序木マイニングとの比較

次に，今回提案する頻出無順序木発見アルゴリズム UNOT と，先に開発した頻出順序木発見アルゴリズム FREQT[5] を比較する．表 1 に，データサイズを 1000K(nodes) に固定して，最小支持度を $\sigma = 1(\%)$ から $\sigma = 10(\%)$ まで変化させた時に，UNOT と FREQT がそれぞれ計算した頻出パターンの数と，頻出パターンの最大サイズを記す．この表より，すべての最小支持度について，UNOT は FREQT が見つけるより多い頻出パターンを発見し，発見したパターンの最大サイズは UNOT の方が FREQT よりも 2 倍以上大きいことが分かる．また，FREQT が計算する頻出順序木パターンには，無順序木として同型なパターンが数多く含まれている．以上から，無順序木データからの知識発見という観点では，UNOT は FREQT より有益である．

最小支持度 $\sigma = 5(\%)$ において，UNOT が見つけた頻出無順序木パターンの例を図 12 に示す．このパターンに対応する頻出順序木パターンを，FREQT は 1 つも計算しなかった．

```
<inproceedings>
  <author> </author>
  <author> </author>
  <booktitle> </booktitle>
  <crossref> </crossref>
  <title> </title>
  <url> </url>
  <year> </year>
</inproceedings>
```

図 12: 見つかった頻出無順序木パターンの例

6 おわりに

本稿では，与えられた半構造データの集合から頻出無順序木パターンを効率よく計算するアルゴリズム UNOT を提案した．今後の課題として，テキストマイニングとの融合やパターンの自由木（根なし木）への拡張などが挙げられる．

謝辞 本研究の一部は，国立情報学研究所の共同研究費による．

参考文献

[1] 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, 2002.

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

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

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

[5] T. Asai, K. Abe, S. Kawasoe, H. Arimura, H. Sakamoto, S. Arikawa, Efficient Substructure Discovery from Large Semi-structured Data, In *Proc. SIAM SDM'02*, 158–174, 2002.

[6] T. Asai, H. Arimura, K. Abe, S. Kawasoe, S. Arikawa, Online Algorithms for Mining Semi-structured Data Stream, In *Proc. IEEE ICDM'02*, 27–34, 2002.

[7] T. Asai, H. Arimura, T. Uno, S. Nakano, Discovering Frequent Substructures in Large Unordered Trees, In *Proc. DS'03*, LNAI 2843, 47-61, 2003.

[8] D. Avis, K. Fukuda, Reverse Search for Enumeration, *Disc. Appl. Math.*, 65(1–3), 21–46, 1996.

[9] A. Inokuchi, T. Washio, H. Motoda, An Apriori-Based Algorithm for Mining Frequent Substructures from Graph Data, In *Proc. PKDD'00*, 13–23, 2000.

[10] M. Kuramochi, G. Karypis, Frequent Subgraph Discovery, In *Proc. IEEE ICDM'01*, 2001.

[11] 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.

[12] S. Nakano, Efficient Generation of Plane Trees, Information Processing Letters, 84, 167–172, 2002.

[13] S. Nakano, T. Uno, Efficient Generation of Rooted Trees, *NII Technical Report NII-2003-005E*, ISSN 1346-5597, Natinal Institute of Informatics, 2003.

[14] S. Nijssen, J. N. Kok, Effcient Discovery of Frequent Unordered Trees, In *Proc. MGTS'03*, 2003.

[15] A. Termier, M. Rousset, M. Sebug, TreeFinder: a First Step towards XML Data Mining, In *Proc. IEEE ICDM'02*, 450–457, 2002.

[16] N. Vanetik, E. Gudes, E. Shimony, Computing Frequent Graph Patterns from Semistructured Data, In *Proc. IEEE ICDM'02*, 458–465, 2002.

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

[18] X. Yan, J. Han, gSpan: Graph-Based Substructure Pattern Mining, In *Proc. IEEE ICDM'02*, 721–724.

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