頻出順序木発見のための効率的な列挙手法の提案

Proposal of Efficient Enumeration Method for Frequent Ordered Subtree Discovery

比戸 将平♠ 河野 浩之♦
Shohei HIDO Hiroyuki KAWANO

近年，半構造データにおけるデータマイニングが盛んに研究されている．本稿では，左右木結合を用いて頻出候補木を効率良く列挙する方法を提案し，木構造データにおける頻出順序木発見アルゴリズムに適用する．XML を用いた実験によって比較したところ，提案手法は代表的なアルゴリズムよりも数倍高速という結果を得た．

Recently many studies have been conducted for data mining on semi-structured data. In this paper we propose right-and-left tree join as a enumeration method of candidate subtrees. We apply the method to discover all frequent ordered subtrees in tree-structured data. The experiment with XML shows that our method is several times faster than a typical algorithm.

1. はじめに

近年，HTMLやXML，RNAやタンパク質の2次構造，化学物質の分子構造など，木やグラフで表現可能な半構造データが普及しつつある．それらのデータから有用な知識を発見するデータマイニング技術に対する需要は非常に大きい．しかしながら，半構造データは平坦なリレーショナルデータに比べて柔軟な記述が可能である反面，極めて不均質かつ複雑であるため，従来のデータマイニング手法をそのまま適用することは難しい．そこで，半構造データ向けの新たなデータマイニング技術が必要とされている．

データマイニングにおける代表的な処理は，データベース中に多く出現する頻出パターンを発見することである．リレーショナルデータベースから頻出アイテム集合を発見する問題は以前から研究されており，Apriori[1]アルゴリズムが広く知られている．一方，半構造データ中に頻出する部分構造を探し出す頻出部分構造マイニングはここ数年盛んに研究されており，様々なアルゴリズムが提案されてきた[1], [2].

本稿では半構造データの中でも木構造データを対象とし，データ木に頻出する順序木パターンを発見する問題を扱う．代表的なアルゴリズムとしては，FREQT[4]やTreeMiner[5]があり，最右拡張による候補木列挙が用いられている．我々は，左右木結合を用いて冗長な候補木の生成を抑制する新たな頻出候補木の列挙手法を提案し，それによって頻出順序木パターンを高速に求めるアルゴリズムを構築する．

♠ 学生会員 京都大学大学院情報学研究科システム科学専攻修士課程 hido@sys.i.kyoto-u.ac.jp
♦ 正会員 南山大学数理情報学部情報通信学科 kawano@it.nanzan-u.ac.jp

2. 頻出順序木パターン発見問題

2.1 順序木のデータモデル

$L = \{l_0, l_1, \cdots\}$をラベルの有限集合とする．L上のラベル付き順序木は$T = \{V, E, B, label, root\}$として表現される．$V$は節点の集合であり，$E \subseteq V^2$は親節点と子節点の接続を表す二項関係である．また$B \subseteq V^2$は隣接兄弟節点間の順序を保持している．$label : V \to L$は節点$v \in V$に対応するラベル$l \in L$を求めるラベル関数である．$root$は木$T$で唯一の親を持たない根節点である．子を持たない節点は葉節点と呼ぶ．また，節点列$P_v = (v_0, v_1, \cdots v_n) \subseteq V$において，任意の$0 \leqq i < n$に対して$(v_i, v_{i+1}) \in E$が成立する場合，$P_v$を$v_0$から$v_n$への経路と呼ぶ．節点$v$の深さ$depth(v)$を根節点から$v$までの経路の節点数とする．

以後，ラベル付き順序木を順序木または単に木と呼ぶ．

2.2 順序木の出現

データ木D中の順序木Tの出現を次のように定義する．

定義 2.1. ラベルと親子関係と兄弟関係を保存するような節点集合間の単射$\Phi : V_T \to V_D$が存在する時，TはDに出現するという．

本稿では順序木の出現数の計算に根出現数を用いる．根出現数は，データ木D中の順序木Tの出現の中で，重複を除いた根節点$root$の出現数$|Occ(root)|$である．また，D中のTの出現頻度$F_D(T)$を以下のように定義する．

定義 2.2. 順序木Tのデータ木Dにおける出現頻度$F_D(T)$を，$F_D(T) = |Occ(root)|/|D|$とする．

以後，節点数kである頻出順序木パターンを単にk-パターンと表す．また，k-パターン全体の集合をF_k，節点数kの頻出候補木の集合をC_kとする．

2.3 頻出順序木パターン発見問題

ある木が頻出か否かを決める出現頻度の閾値として，最小サポートσ($0 \leqq \sigma \leqq 1$)を与える．これを用いて，データ木Dにおける頻出順序木を以下のように定義する．

定義 2.3. D中の順序木Tの出現頻度$F_D(T)$が最小サポートσ以上である時，順序木Tは頻出とする．

次に，頻出順序木パターン発見問題を次のように定式化する．

問題 2.4. データ木$D = \{V, E, B, label, root\}$，最小サポート$\sigma$が与えられた時，$D$中の出現頻度$F_D(T)$が$F_D(T) \geqq \sigma$を満たすような頻出順序木$T$を全て求めよ．

例として図1のデータ木Dにおいて最小サポートσ = 20%とした頻出順序木パターン発見問題を考える．頻出候補木T_1は，根出現数が3であり，出現頻度$F_D(T_1)$ = 3/15 = 20%より頻出である．一方，頻出候補木T_2は3回出現しているように見えるが根出現数としては2であるため，$F_D(T_2)$ = 2/15 < 20%より非頻出である．

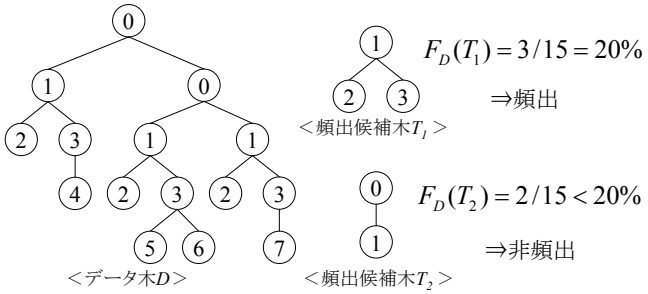

図1 データ木中の頻出木

Fig.1 Frequent Tree in Tree Data

3.頻出順序木発見における関連研究

3.1 問題の性質

まず頻出順序木パターンに関わる公理を示す.

公理 3.1. k-パターンから，いずれかの葉節点を除いた節点数k-1の部分木は全て(k-1)-パターンである.

この公理は頻出アイテム集合発見問題においてAprioriアルゴリズムで用いられている部分アイテム集合の逆単調性に類似している．公理3.1に基づき，多くの頻出木パターン発見アルゴリズムは以下のように構成されている.

STEP 1. データ木D を走査しF_1 とF_2 を求める．$k=2$とする.
STEP 2. F_k から節点数k+1の候補木C を列挙しC_{k+1} に加える.
STEP 3. 各候補木$C \in C_k$ のデータ木D における出現頻度を求め，$F_D(C) \geqq \sigma$ であればC をF_{k+1} に加える.
STEP 4. $|F_{k+1}| = 0$ならば終了．そうでなければ$k = k + 1$としてSTEP 2に戻る. □

3.2 頻出順序木発見における既存アルゴリズム

前節のアルゴリズムで重要となるのは，STEP 2においてどのように候補木を列挙し，C_{k+1} を生成するかということである．単純な列挙手法では，明らかに頻出ではない冗長な候補木の列挙や，同じ候補木の複数回列挙が発生する場合が多く，STEP 3においてデータ木を走査に必要するためのコストが非常に大きくなってしまう．また，k が大きくなるにつれて候補木集合の大きさ$|C_{k+1}|$が増大し，現実的な時間では動作しなくなる．そこで，頻出候補木の効率的な列挙に最右拡張を用いるアルゴリズムとして，浅井らによってFREQT[4]が，ZakiによってTreeMiner[5]がそれぞれ独立に提案された．本節では最右拡張とFREQTの概要を説明する.

順序木T において，最も右に位置する葉節点，つまり根節点から深さ優先で木を走査した時に最後に到達する葉節点を最右葉($rml(T)$)と呼ぶ．また，根節点から最右葉に至るまでの経路を最右枝とする.

定義 3.2. 任意の順序木S において，最右枝上の節点に対し新たな節点v を最右葉として追加して得られる順序木T をSの最右拡張という.

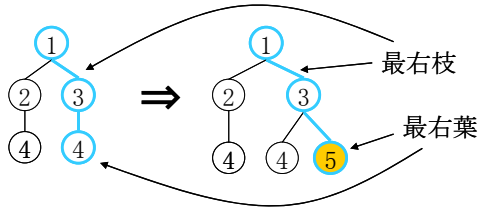

図 2 最右拡張の例

Fig.2 Example of Right Most Expansion

任意の木T から最右葉を除いた親木P (T)は一意に定まる．また公理3.1より節点数k+1の頻出木の親木は必ずF_k に含まれている．よって，F_kの各節点に対する最右拡張により，全ての節点数k+1の頻出木は頻出候補木集合C_{k+1} の中に重複無く列挙される.

FREQTでは，順序木は深さラベル列を用いて表現されている．深さラベル列は，根節点から深さ優先で走査して現れた順に節点の深さとラベルの組を並べることで，順序木に一意的な表現を与える．また，最右拡張は元の木の深さラベル列の末尾に新たな節点の深さとラベルを加えることにより，定数時間で行える.

T =(($depth(root)$, $label(root)$), (($depth(v_1)$, $label(v_1)$)…

FREQTにおいては，各k-パターンの$T \in F_k$ の最右葉が出現する位置$RMO_k(T)$が保存されている．T の最右拡張を行って得られた$C \in C_{k+1}$ の出現数を調べる際には，$RMO_k(T)$を利用し，$RMO_{k+1}(C)$を求める．ただしその際，同じ節点が複数回出現する可能性がある．そこでFREQTでは，重複検出と呼ばれる方法によって重複を完全に回避している.

FREQTにより，全ての頻出順序木パターンとその最右葉出現を重複無しに求められることが保証されている.

4. 左右木結合を用いた頻出順序木発見

4.1 左右木結合

順序木T において，3.2節で与えた最右葉と同様に，根節点から深さ優先で辿った時に最初に現れる葉節点を最左葉($lml(T)$)と呼ぶ．ある木T から最右葉を除いた部分木を左木$Left(T)$，最左葉を除いた部分木を右木$Right(T)$と表す．最左葉と最右葉を両方除いた部分木を中央木$Center(T)$とする.

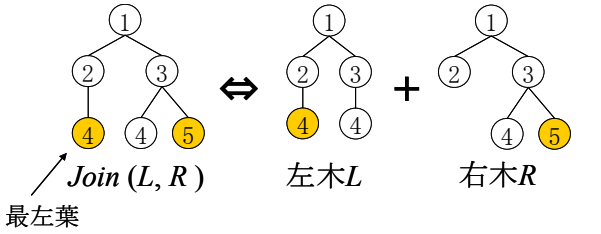

図 3 左右木結合

Fig.3 Right-and-Left Tree Join

定理 4.1. 任意の順序木T に対し，

$$Center(T) = Right(Left(T)) = Left(Right(T))$$

が成立する.

証明. 最左葉を取り除く操作は最右葉の表現に影響を与えず，最右葉を除く操作も最左葉に影響を及ぼさない．よって，

$$Right(Left(T)) = Left(Right(T))$$

が成立する. □

$Right(T_1) = Left(T_2)$である2つの順序木の左右木結合(right-and-left tree join)を以下のように定義する.

定義 4.2. 2つの順序木T_1とT_2が$Right(T_1) = Left(T_2)$ を満たす時，その左右木結合$join(T_1, T_2)$を以下のように表す.

$$join(T_1, T_2) = T_1 \cup rml(T_2) = lml(T_1) \cup T_2$$

任意の順序木において，左右木結合の完全性と一意性を表す次の2つの定理が成り立つ.

定理 4.3. 任意の順序木T はその左木と右木の左右木結合として表現される.

証明. 定理4.1 より，任意の順序木T について

$$Right(Left(T)) = Left(Right(T))$$

が成立する．よって，定義4.2 に従いT は

$$T = join(Left(T), Right(T))$$

と表現することができる. □

定理 4.4. 左木と右木の組と，その左右木結合によって得られる順序木は一対一に対応する．つまり，

$$join(T_1, T_2) = join(S_1, S_2) \Leftrightarrow T_1 = S_1 \text{かつ} T_2 = S_2$$

が成り立つ.

証明. (十分性) 右木と左木の表現は一意に定まるので，

$$Left(join(L, R)) = L$$
$$Right(join(L, R)) = R$$

である．よって$join(T_1, T_2) = join(S_1, S_2)$の両辺の左木及び右木をとれば$T_1 = S_1$かつ$T = S_2$となる.

(必要性) 左右木結合の定義より明らか. □

4.2 頻出候補木への適用

左右木結合を頻出候補木列挙に適用する方法について説明する．ここで(k+1)-パターン$T \in F_{k+1}$ について考える．Tの左木T_1=*Left*(T)と右木T_2=*Right*(T)は，公理3.1よりk-パターンである．よって，全ての(k+1)-パターンT は，T_1, $T_2 \in F_k$ の左右木結合*join*(T_1, T_2)により得られる．

左右木結合によって頻出候補木を列挙するために，結合候補クラスを作成する．結合候補クラスは節点数k-1の中央木C_T と左木候補リスト$J_L(C_T)$，右木候補リスト$J_R(C_T)$から成る構造$J_C(C_T) = \{C_T, J_L(C_T), J_R(C_T)\}$である．ある結合候補クラス$J_C(C_T)$内の左右木候補リスト$J_L(C_T)$，$J_R(C_T) \subseteq F_k$ において，$(L, R) \in J_L(C_T) \times J_R(C_T)$を用いた左右結合により頻出候補木$C = join(L, R)$を生成する．結合候補クラスを用いた左右木結合による頻出候補木列挙の例を図4に示した．節点数4の中央木C_Tを右木に持つ左木候補L_1, L_2及びC_Tを左木として持つ右木候補R_1, R_2 を用いて左右木結合による頻出候補木が4つ生成されている．

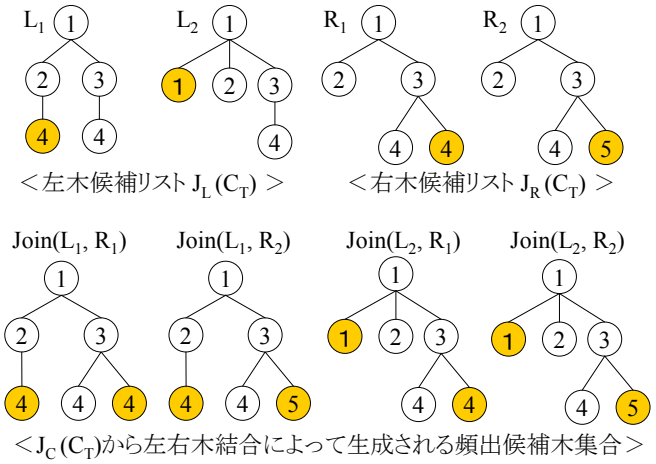

図 4 左右木結合による頻出候補木の列挙

Fig. 4 Enumerate Candidates by Right-and-Left Tree Join

4.3 アルゴリズム

左右木結合を用いて頻出候補木を列挙し，頻出順序木パターンを発見するアルゴリズムの流れを示す．

STEP 1. データ木D を走査し，F_1, F_2を求める．k = 2とする．

STEP 2. 各$T \in F_k$ に対し(a)-(d)を行う．

(a) 中央木に*Left*(T)を持つ結合候補クラスJ_C(*Left*(T))を選ぶ．

(b) 各左木候補$T_L \in J_L$(*Left*(T))について，左右木結合*join*(T_L, T)を頻出候補木としてC_{k+1} に加える．

(c) 右木候補リストJ_R(*Left*(T))にT を加える．

(d) 同様に，結合候補クラスJ_C(*Right*(T))を選び，各右木候補$T_R \in J_R$(*Right*(T))とT の左右木結合*join*(T, T_R)を頻出候補木としてC_{k+1}に加え，T を最後にJ_L(*Right*(T))に加える．

STEP 3. 各候補木$C \in C_{k+1}$ のデータ木D における出現頻度$F_D(C)$を求め，$F_D(C) \geqq \sigma$であればC をF_{k+1} に加える．

STEP 4. $|F_{k+1}| = 0$であるならば終了．そうでなければ$k = k + 1$としてSTEP 2に戻る． □

ここで葉節点数が1の木，つまり根節点から唯一つの葉節点までの経路として表せる木を直列木と呼ぶ．直列木は葉節点数が2以上である木と異なり，上記の左右木結合による方法では列挙を行うことができない．そこで節点数kの頻出木パターン集合に属する直列木において，葉節点の子として1つ節点を加えることで節点数k+1の頻出候補直列木への拡張を行う．以上の手法により、全ての頻出候補木を重複無く列挙することができる．

最後に出現数の計算方法と木の表現方法を考える．定義4.2より，左右木結合によって得られた木T は，左木*Left*(T)におけるラベル*label*(*rml*(*Right*(T)))を用いた最右拡張に等しい．つまり出現数や木の表現はFREQTと同様の方法で扱うことができる．よって，頻出木の最右葉がデータ木に出現する位置を保持することにより，重複検出を行いながら出現数を計算する．また深さラベル列を木のデータ表現に用いる．

4.4 アルゴリズムの考察と最右拡張との比較

左右木結合による頻出候補木の列挙が線形時間で行えることを示す．前節で述べたように左右木結合の操作は左木に対する最右拡張と等価である．つまり左右木結合による木の生成は，最右拡張と同じく定数時間で実行可能である．また定理4.4より，全ての頻出候補木は重複無しに1回ずつ左右木結合によって生成される．よって，左右木結合による頻出候補木の列挙は線形時間で可能である．さらに頻出直列木の拡張による頻出候補直列木の列挙も同様に線形時間で行える．以上より，提案アルゴリズムにおける頻出候補木の列挙は，その個数に対して線形時間で実行可能である．

次に生成される候補木列挙の効率を比較する．上述したように左右木結合で生成される頻出候補木は必ず最右拡張によっても生成される．一方，最右拡張によって生成される頻出候補木は左右木結合によって生成されないことがある．例えば，図4の左木候補L_1において根節点にラベル1を持つ節点を加える最右拡張S を考える．右木候補リスト$J_R(C_T)$を見れば分かるようにS の右木*Right*(S)は頻出ではないため，左右木結合によっては列挙されない．このように，最右拡張による節点数kの頻出候補木集合C_k^Fと提案アルゴリズムによる節点数kの頻出候補木集合C_k^{LR} の間には$C_k^{LR} \subseteq C_k^F$，つまり要素数としては$|C_k^{LR}| \leqq |C_k^F|$という関係が常に成り立つ．冗長な候補木の個数が少なければ，データ木の走査に必要な時間が短縮できる．以上より，左右木結合を用いた列挙は最右拡張よりも効率の良い列挙を言える．

次に，メモリ使用量を比較する．最右拡張を用いたFREQTやTreeMinerは深さ優先方式で実行でき，少ないメモリ使用量で動作することが可能である．一方，提案アルゴリズムは結合候補クラスなどを用いているため原理的には幅優先方式である．よって，同時にメモリ上に保持するべき木の数が大きく，メモリ使用量は必然的に大きくなる．しかし，木の性質と左右木結合の特徴を利用することで探索空間を分割し，メモリ使用量をある程度削減することも可能である．例えば2つの木において最大深さや葉節点数が2以上異なる場合，それらの木同士に左右木結合が適用されることはない．そのため，例えば葉節点数1の木の集合から1ずつ節点数を増やしながら左右木結合を行った場合，葉節点数3の木集合に至る前に葉節点数1の木は全て不要となるため，メモリ上から削除することが可能となり，同時に使用するメモリ量を低減できる．

5. 性能評価実験

5.1 実験環境

本稿で提案したアルゴリズム及びFREQTをJavaで実装し，性能評価実験を行った．実験に使用したPCはCPUがPentium4 2.5GHz，メインメモリが512メガバイト，OSはWindows XP Professionalである．

5.2 XMLを用いた実験

実際の木構造データの例としてXMLを用いた実験を行った．XMLファイルは容量約500キロバイトから最大約10メガバイトのGoogleの検索結果を格納したものを用いた．そのラベル集合の要素数は408であり，節点数はそれぞれ8677から172932の間である．各XMLデータを読み込んでデータ木Dに変換した後，最小サポートσ = 0.05%として我々の頻出順序木パターン発見アルゴリズムとFREQTを順に適用した．その結果，提案アルゴリズムはFREQTと同様，全ての頻出順序木パターンを重複無く発見していることが確認された．

全ての頻出順序木パターンを発見し終わるまでにかかる時間を計測したところ，図5のようなグラフを得た．

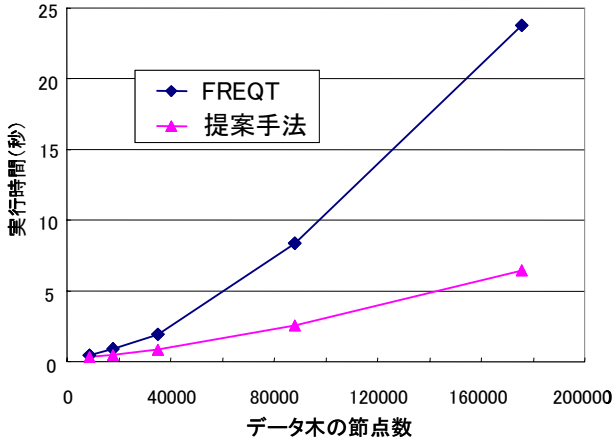

図 5 実行時間

Fig.5 Running Time

図5によって，提案アルゴリズムはデータ木の節点数に対してほぼ線形時間で頻出順序木パターンを求めることが可能であることがわかる．さらに，FREQTと比べるとデータ木の節点数が増えるにつれて約2倍から4倍高速に動作している．これは，左右木結合による頻出候補木の列挙において冗長な候補木の列挙が抑制され，データ木の走査にかかる時間が少ないことによる効果であると思われる．

そこで実際に我々のアルゴリズムが頻出候補木を効率的に列挙していることを確かめるために，上記の$|V|$ = 87853における実験中に，各サイズkにおいて生成される頻出候補木の個数$|C_k|$と，k-パターン集合の要素数$|F_k|$を求め，図6に示した．

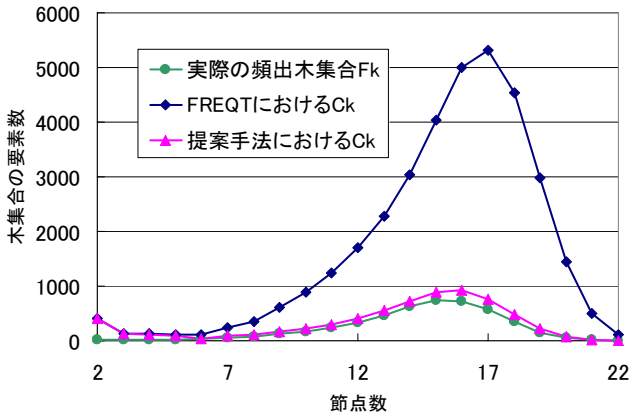

図6 候補木列挙の効率

Fig.6 Efficiency of Candidate Enumeration

木のサイズ別に見ると，FREQT では頻出木が多くなるほど頻出候補木の個数が著しく増加している．それに比べ提案アルゴリズムでは頻出木と頻出候補木の個数がほぼ一致しており，非常に効率の良い列挙が行われていることがわかる．

6. まとめ

本稿では頻出順序木パターン発見問題に対し，左右木結合による頻出候補木列挙を提案し，アルゴリズムを構築した．XMLデータを用いた実験によって，提案アルゴリズムは全ての頻出順序木パターンをデータサイズに対して概ね線形時間で重複無く発見できることを示した．同じ頻出順序木パターン発見アルゴリズムであるFREQTと比較すると，提案手法は数倍高速であるという結果を得た．

今後の課題を2つ挙げる．まず，アルゴリズムの性能がデータの構造的な性質に依存することがわかっているため，今回実験で用いたデータ以外の様々なスキーマを持つXMLデータや，RNAの2次構造データなどを用いた性能評価実験を行い，どのようなデータに対しても高い性能を示すことを確かめる必要がある．また4.4節で述べたとおり，提案手法は幅優先方式であるために実験においてもFREQTよりも多くのメモリを使用していた．これに関しては左右木結合における頻出候補木の生成についてより詳細な解析を行い，アルゴリズムの完備性と高速性を失うことなく同時に保持する木の個数を削減する方法を考案し，改善を進めていきたい．

[謝辞]

本研究の一部は，文部科学省の科学研究費補助金(課題番号: 13680482, 16016248)の支援を受けた．

[文献]

[1] 浅井達哉，有村博紀，“半構造データマイニングにおけるパターン発見技法”，電子情報通信学会論文誌，vol.J87-D1，no.2，pp.79-96，2004.

[2] S. Nijssen, “Frequent structure mining: Efficiency issues”, Proceedings of the 2nd International Workshop on Mining Graphs, Trees and Sequences (MGTS'04), 2004.

[3] T. Asai, K. Abe, S. Kawasoe, H. Arimura, H. Sakamoto, and S. Arikawa, “Efficient substructure discovery from large semistructured data”, Proceedings of the 2nd SIAM International Conference on Data Mining (SDM'02), pp.158-174, 2002.

[4] M.J. Zaki, “Efficiently mining frequent trees in a forest”, Proceedings of the 8th International Conference on Knowledge Discovery and Data Mining (ACM SIGKDD'02), pp.71-80, 2002.

[5] 比戸将平，河野浩之，“左右木結合を用いた頻出順序木パターン発見アルゴリズム”，電子情報通信学会第 16 回データ工学ワークショップ(DEWS'05)，6C-i9，2005.

比戸 将平 Shohei HIDO

京都大学大学院情報学研究科システム科学専攻修士課程在学中．情報処理学会，日本データベース学会各学生会員．

河野 浩之 Hiroyuki KAWANO

南山大学数理情報学部情報通信学科教授．京都大学工学部数理工学教室助手，同大学大学院情報学研究科システム科学専攻助教授を経て現職．工学博士．AAAI，ACM，IEEE，電子情報通信学会，情報処理学会，人工知能学会，日本データベース学会各会員．