大規模ドキュメント集合に適用可能な木構造マイニング技術

Subtree Mining Technology for Extracting Useful Information in Huge Document Sets

要　旨

近年、報告書や帳票などのオフィス文書に加えて、ウェブページや XML（eXtended Markup Language）ドキュメントが企業内に爆発的に増加している。文書管理システムにドキュメントが基本的に木構造で格納されること、XML のタグセットや報告書の構成などドキュメント内容も木構造で表現されることから、ドキュメント集合中に埋もれている共通の木構造パターンをマイニングする技術の研究開発を進めている。本稿ではわれわれが開発した 2 種類の embedded subtree mining（先祖子孫の木構造抽出）技術についてアルゴリズムの概略と実験評価結果を示し、各技術の適用例を紹介する。評価実験の結果、われわれが開発した技術は、速度およびメモリ使用量において従来技術に対して飛躍的な性能を示すことができた．また、ドキュメントヒストリーなど実ドキュメントデータを用いた実験でも有効な抽出パターンが得ることができた。今後、応用に向けた技術開発を更に進め、企業などに価値の高いドキュメントサービスを提供していく。

Abstract

Recently, the digital documents used by companies have dramatically increased, including not only office documents such as reports and business forms, but also web and xml documents. We are developing embedded subtree mining technologies to extract useful patterns in document sets, because documents are generally stored in tree structures in a management system of documents, and document contents such as xml tag sets or logical structures in a report are frequently represented in a tree structure. This paper introduces two types of embedded subtree mining technology that can extract ancestor-descendant relationships. In an experiment using a test dataset, our technologies outperformed conventional ones in terms of both processing speed and memory consumption. In an experiment using real document data such as usage logs, we could extract useful patterns. We now intend to develop useful mining applications to deliver valuable document services.

執筆者

林　千登（Kazuto Hayashi）[*1]
吉岡　健（Takeshi Yoshioka）[*2]

[*1] 研究技術開発本部 システム要素技術研究所
（System Technology Laboratory, Research & Technology Group）
[*2] 研究技術開発本部 基盤技術研究所
（Key Technology Laboratory, Research & Technology Group）

1. 緒言

近年、報告書や帳票などのオフィス文書に加えて、ウェブページや XML（eXtended Markup Language）ドキュメントが企業内に爆発的に増加している。また、これらのドキュメントに紐付けされて、大量の購買データやアクセス履歴がメタデータとして管理されている。このため、1990 年代から大量データを解析してデータ中に潜む項目間の共起や順序などのパターンを探し出すデータマイニング技術の研究技術開発が盛んに行なわれ、マーケティングやバイオメディカルなど様々な分野で適用され成果を上げている[4)、9)]。

報告書やウェブページなどのドキュメントは、タイトルや内容などの論理的な意味構造と、テキストや図表、写真などの配置を表すレイアウト構造を持っており、一般的に非定型データとして計算機上に表現されている。さらに、ドキュメント中の引用やハイパーリンクなど他のドキュメントとの関係を示す情報や、編集、閲覧、印刷などユーザのドキュメントに対する操作履歴情報まで含め計算機で処理させようとすると、従来のデータマイング技術が主に扱ってきた、関係データベース中の｛属性-属性値｝ペアでのデータ表現では不十分である。このため、木構造やネットワーク構造で大規模データを表現し、その中から意味のある情報を探索する要求が近年高まり、様々な研究が進められている[8)]。われわれは、DocuShare®などの文書管理システムにドキュメントが基本的に木構造で格納されること、XML のタグセットや報告書の構成などドキュメント内容も木構造で表現されることから、木構造パターンをマイニングする技術[1)、2)、5)、6)、10)]（以下サブツリーマイニング技術：木構造で表現されたデータ群の中から共通パターンである部分木を抽出するため、サブツリーマイニング技術と呼ばれている）の研究開発を進めている。

本稿では、ドキュメントサービスの要素技術としてわれわれが開発しているサブツリーマイニング技術を簡単な事例と共に紹介する。本節に続く第 2 節ではサブツリーマイニング技術の概要を数学的に定式化しながら説明する。第 3 節では、我々が研究開発した、木構造データをルートノードから深さ優先で探索しパターンを抽出する「Root Search Tree Mining（以下 RSTM）」について、アルゴリズム、実験結果、適用例を紹介する。第 4 節では、大規模データを効率的に扱うために、RSTM を改良した、下位ノード等からパターンを柔軟に探索する「Flexible Search Tree Mining（以下 FSTM）」について実験結果と適用例を説明し、第 5 節で、今後の展望を述べ、まとめとする。

2. サブツリーマイニング

サブツリーマイニングは木構造のデータが複数ある場合に、共通に内在する部分木構造パターンを抽出する技術である。親子関係を保持してパターンを抽出する induced subtree mining と先祖子孫関係を抽出する embedded subtree mining があり、embedded subtree mining の方がパターン抽出能力が高い。例えば図 1 の(a)、(b)、(c)のツリーデータに共通の構造として embedded subtree mining では(d)の木構造パターンを抽出できるが、induced subtree mining では D、E、F のノードの存在により(d)の木構造パターンが抽出できない。ドキュメントの操作や配布を例にとっても処理の流れの途中に差異が生じることは少なくないことから、embedded subtree mining に注目して、処理アルゴリズムや応用技術等の研究を行なっている。

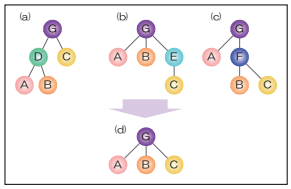

図 1. 部分木パターンの抽出 Embedded subtree mining

以下、サブツリーマイニングに関する用語を順に定義する。

図 2 の左側のように閉路を持たない連結グラフは木(tree、ツリー)と呼ばれる。ここでは図 2 の右側のようにノードの一つがルート(root)として指定されており、各ノードにラベルが付与されている、ラベル付き根付き木(labeled rooted tree)を対象とする。以降では単にツリーと呼ぶときにラベル付き根付き木を指すものとする。

図 2 に示したツリーはノードの集合 $V=\{v_0, v_1,$

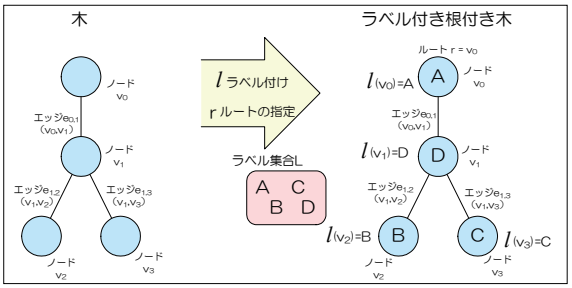

図 2. ラベル付き根付き木
A labeled rooted tree

v_2, v_3}、エッジの集合 E={(v_0,v_1), (v_1,v_2), (v_1,v_3)}、ラベル集合 L={A, B, C, D}、ルート r=v_0、ラベル付け関数 l (l (v_0)=A, l (v_1)=D, l (v_2)=B, l (v_3)=C)の組（V, E, r, L, l）で表される。

次に親子関係と先祖子孫関係を定義する。ルートとの位置関係により各エッジ(v_i,v_j)は方向付けられているものとし、(v_i,v_j)が存在するときには v_iを v_jの親(parent)、v_jを v_iの子(child)と呼ぶ。図３ではエッジ(v_0,v_1)により v_0は v_1の親、v_1は v_0の子であることを例示している。図３の v_2と v_3のように、親が同じであるノード同士は兄弟(siblings)関係となる。

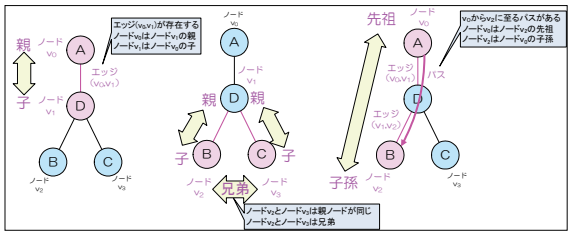

図 3. 木構造におけるノード間の関係
Relationships among nodes in a tree structure

パスを用いて先祖子孫関係を定義する。ノード v_iからノード v_jへのパスがあるとき、v_iを v_jの先祖(ancestor)と呼び、v_jを v_iの子孫(descendant)と呼ぶ。図３では v_0から v_2へはエッジ(v_0,v_1)とエッジ(v_1,v_2)を通るパスが存在するため、v_0が v_2の先祖であり、v_2が v_0の子孫であることを例示している。子ノードを持たない図３の v_2と v_3のようなノードをリーフ(leaf、葉)と呼び、リーフ以外のノード（図３では v_0と v_1）を内部ノードと呼ぶ。

２つのツリーS と T があるとき、S が T のサブツリーとなる条件を説明する。

ツリーS= (V_S, E_S, r_S, L, l_S) とツリーT= (V_T, E_T, r_T, L, l_T) について、下の条件を満たす V_Sから V_Tへの１対１写像φがあるとき、S を T のサブツリー(subtree、部分木)と呼ぶ。

１）エッジ(u_i, u_j) $\in E_S$(u_i, $u_j \in V_S$)があるとき、
$\varphi(u_i)$は$\varphi(u_j)$の親/先祖である。
($\varphi(u_i)$, $\varphi(u_j)$ $\in V_T$)

２）$l_S(u_i) = l_T(\varphi(u_i)) \in L$

条件１）で$\varphi(u_i)$ ($\in V_T$)を$\varphi(u_j)$($\in V_T$)の親とした場合に induced subtree、先祖とした場合に embedded subtree を表現することとなる。図４に embedded subtree の場合の条件１)、２）の各式の関係を図示する。条件１）は S のエッジと T の先祖子孫関係が対応することを規定したものであり、条件２は写像φではラベルの値が保存されることを規定したものである。

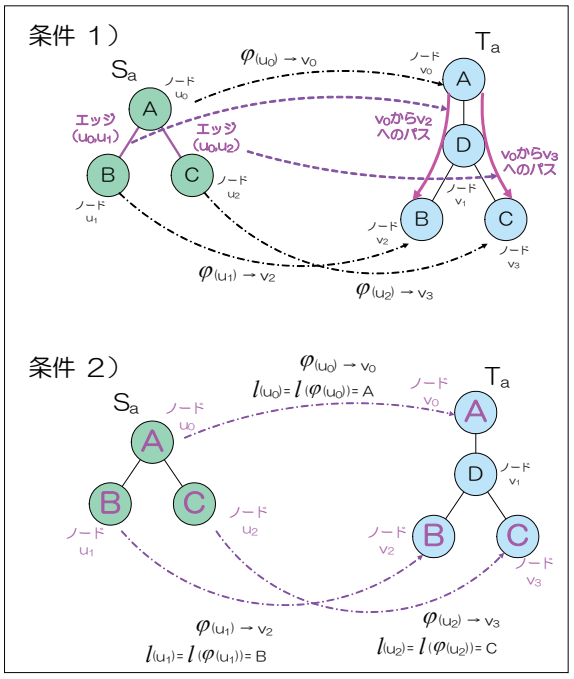

図 4. φの2つの条件
Two conditions for φ

embedded subtree の場合にはこれら２つの条件だけでは図５に示すように兄弟関係と先祖子孫関係を対応させる不適切な写像φ'を許容してしまうため、φに次の条件３）を加える。

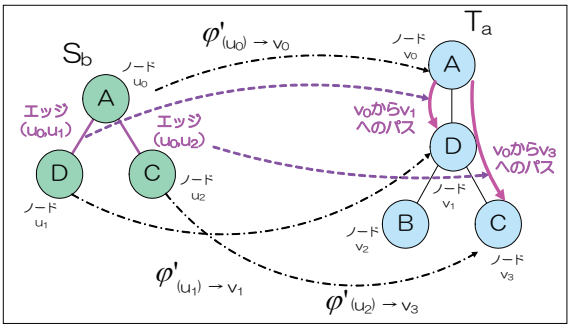

図 5. 条件 1、2 を満たす不適切な写像
Invalid mapping satisfying condition 1) and 2)

3）u_i ($\in V_S$)と $u_j \in (V_S)$が兄弟ノードであれば、$\varphi(u_i)$ は $\varphi(u_j)$の先祖でも子孫でもない。($\varphi(u_i)$, $\varphi(u_j) \in V_T$)

条件3）はTで先祖子孫関係にある2つのノードをSの兄弟関係にあるノードに対応付けることを防ぐ。

ツリーSがツリーTに含まれる（SがTのサブツリーである）とき、$d_T(S)=1$、含まれないときに$d_T(S)=0$となる関数d_Tを考える。あるツリーデータベース（以下TDB）に含まれるツリーについてSの d_T の値の集計値をサポート(support)と呼ぶ。直感的にはSが出現するツリーTの数を表現している。

$$\text{support}_{TDB}(S) = \sum_{T \in TDB} d_T(S)$$

以降、特に誤解のない場合には$\text{support}_{TDB}(S)$を単にsupport(S)またはsupportと書く。Sのサポートがある閾値minsup (minimum support)以上である場合にSは頻出(frequent)であるといい、SをTDBにおけるパターンと呼ぶ。

$$\text{support}_{TDB}(S) \geq \text{minsup}$$

以上より、embedded subtree miningは、あるツリーデータベースTDBと閾値が与えられたとき、指定された閾値以上となるサポートを持つ全てのサブツリーのパターンを列挙する処理と定義できる。

以下、われわれが研究開発したサブツリーマイニングとして、ルートから探索しパターンを抽出するRSTMと、大規模データを効率的に扱うために下位ノード等からフレキシブルに探索しパターンを抽出するFSTMを紹介する。

3. RSTM（Root Search Tree Mining ）

3.1 パスを用いたパターンの伸張

パターンの列挙は、見つけた頻出パターンP_nに一つノードを追加したパターンP_{n+1}を探索することを繰り返していくことで実現できる。木構造パターンのため、ノードの追加が可能な箇所は通常複数存在する。重複や抜け洩れなく効率的にパターンの列挙を実現するために、文献1)、文献5)で提案されているパターンの右端のパス(Right Most Path、以下RMP)[1), 5)]上のノードを新たなノードの追加箇所とする方法を採用する。RMP上に追加するノードの候補を図6に示す。パターンScの伸張はRMP上のノードV_{c0}、V_{c3}、V_{c4}のいずれかについて新たな子供ノードを追加して実現できる。V_{c4}に対応するデータツリー上のノードの子孫ノード全てが、新たなノード候補 C_{c1} のデータツリー上での出現箇所になりえる。

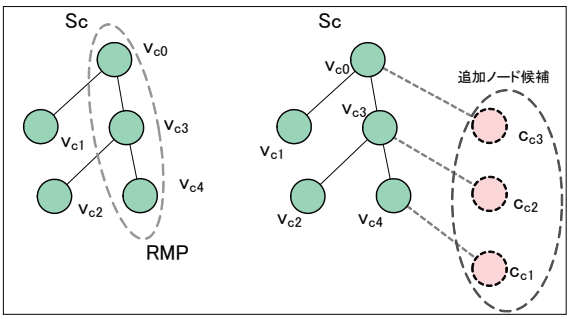

図6. RMPから探すノード候補
New node candidates from RMP

新たなノード候補C_{c2}とC_{c3}の探索は他のノードの兄弟ノードを見つける処理となる。例えばC_{c2}ではV_{c3}の子供でV_{c4}の新しい兄弟ノードを探すことになる。

C_{c2}やC_{c3}のようにRMPから分岐する場合に、C_{c2}やC_{c3}の出現可能範囲を算出する方法について説明する。パターンをS_p、出現するツリーデータをT_dとする。embedded subtree miningではS_p上の親子関係がT_d上の先祖子孫関係に対応するため、S_p上のパスはT_d上に対応するパスを持つ。ここで、ツリーのルートからリーフに至るパスをRoot-Leaf Pathと名づけ（以下RLP)、RLPを用いて新規ノードの探索範囲を規定する。

RMPにノードを追加する場合のRLPの変化を図7に示す。RMPに分岐する形でノードが追加される場合には、RLP_{Sp4a}、RLP_{Sp4b}のように追加するノードが含まれた新たなRLPができる。RLP_{Sp4a}、RLP_{Sp4b}がパターン候補の一部となるのは、RLP_{Sp4a}、RLP_{Sp4b}、それぞれに対応するパスがデータツリー側に存在する場合である。上記パスが見つかった場合のみ、そのパス上にノードの候補を探索する。

以下、図7の中央下に示したRLP_{Sp4a}を例にし

て、図 8 を用いて、このノードの探索を図解する。

最初に、探索範囲となる RLP の抜け落ちを防ぐために、パターンの RMP 上の分岐ノード C と対応可能である、より上位のノードをデータツリー上で選ぶ（図 8(a)）。

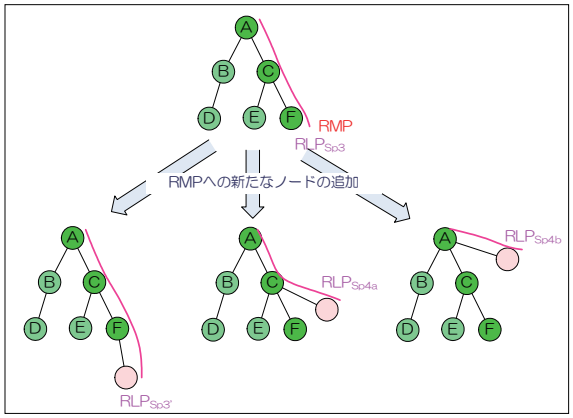

図 7. RMP へのノードの追加と新たな RLP
New RLPs with node addition

次に、兄弟ノードであるラベル E、F ノードに対してより下位の出現箇所を選ぶ。これにより、対応する RLP を RLP$_{Td3}$、RLP$_{Td4}$、RLP$_{Td5}$ に絞り込む。

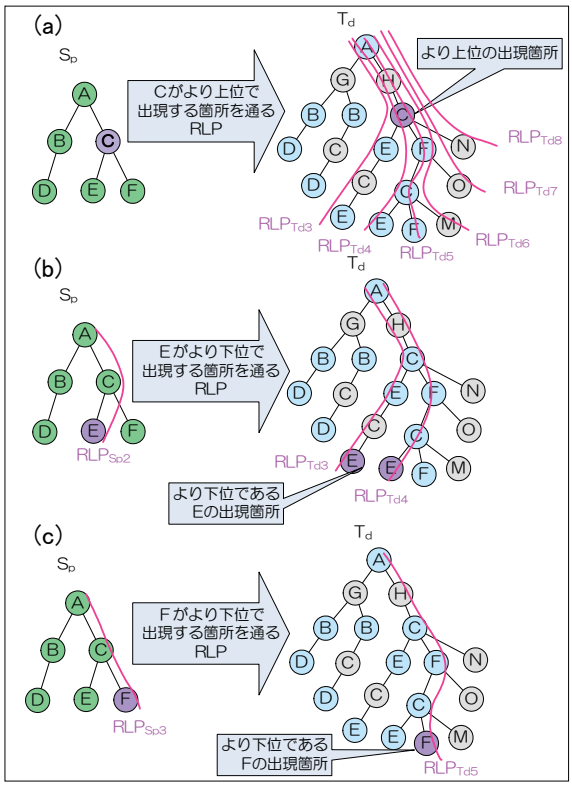

図 8. 探索範囲計算に用いる RLP
RLPs used to calculate node search space

そして、分岐箇所である親側の C に対応する RLP 集合から兄弟ノード E、F に対応する RLP の集合の差分を計算し、新しいノードが T_d 上で出現可能な範囲を求める。より詳しくは、差分計算で得られた RLP について、兄弟ノードに対応して選んだ RLP との共通部分を除いた範囲となる。なお、図 9 に示すように、兄弟関係の順序の有無で探索範囲が異なる。

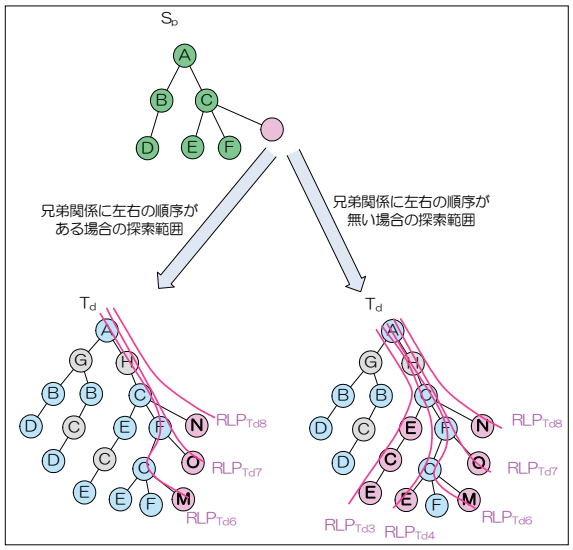

図 9.ノードの探索範囲
Node search space

RSTM は上記のように、追加するノードの候補を探す前にデータツリー上でのノードの探索範囲を算出するところに特徴をもっている。パターン上の一つのノードが出現する箇所は複数箇所あり、ノード間の関係を正しく保ってパターンを漸進的に探索するためにはノードの出現箇所の組み合わせを管理する必要が生じる。RSTM では探索範囲計算において後の処理に影響を及ぼさない出現箇所の処理を省き、更に、出現箇所情報をパスとして効率的に扱うことにより、従来技術に対して計算量の削減を実現している。

3.2 RSTM のアルゴリズム

RSTM は、頻出ラベルの探索範囲演算と頻出ラベルの調査、ノードの出現箇所情報のメンテナンス機構で構成される。図 10 と図 11 に概略のアルゴリズムを示す。図 10 では一つの頻出ノードだけからなるパターンを順に作成して、パターンを再帰的に探索する処理(図 11)を呼び出している。

```
MINE(minsup,TDB)
1    freqLs ←FINDFQLB(ALL,minsup,TDB)
2    foreach(lab, occls) in freqLs
3    do
4       pat ←mkpat(NULL,lab)
5       stat ← makestat(occls, NULL)
6       RSTM( pat, stat, minsup, TDB)
```

図 10. メインプログラム
Main program

```
RSTM(pat, stat, minsup,TDB)
1    Output(pat)
2    foreach Node v on RMP
3    do
4      foreach tr(v,TDB)
5      do
6        sRegionLs ← calcNodeSearchRegion(v)
7        foreach sRegion in sRegionLs
8        do
9          labelOccurrenceCheck
10     freqLs ← FINDFQLB(minsup)
11     foreach (lab, occls) in freqLs
12     do
13       newPat←mkPat(oat, lab, v)
14       newoccls ← makestat(occls, stat)
15       RSTM ( newpat, newoccls
                    , minsup, TDB)
```

図 11. RSTM 再起処理部
RSTM recursive procedure

図 11 ではパターン(pat)の RMP 上のノード毎に、探索領域を算出し(ライン 6)、データツリー中の対応するノードで出現するラベルを調べ（ライン 9)、RMP 上に追加するノードの候補を選択する(ライン 10、11)。新たなパターンを作成(ライン 13)後、出現箇所情報を含めて状態変数を作成(ライン 14)する。以上の処理を再帰的に繰り返す(ライン 15)ことで、ツリーデータベース TDB 内に閾値 minsup 個以上のツリーデータで出現するパターンを全て列挙する。

3.3 評価実験

RSTM を評価するために我々は人工的に作成したツリーデータと現実世界のデータから作成されたツリーデータを複数用いて実験を行なった。ここでは、これらの実験のうち、subtree mining の研究分野で評価実験によく用いられているデータセット CSLOGS[1]を用いた実験結果を紹介する。

CSLOGS はある Web サイトに訪れたユーザのナビジェーション履歴のツリー構造から作成された 59691 個のツリーデータから構成される、ノードのラベルの異なり数が 13661 のデータセットである。

実験環境として 4 Gbyte のメインメモリ、core2duo® 3GHz の CPU を搭載した PC を用意し、OS は Fedora8 の 64bit 版を用いた。比較対象を他の文献でもよく利用されている TreeMiner[1] とし、TreeMiner も RSTM も GNU g++の-O3 オプションを用いてコンパイルして処理時間とメモリの使用量を測定した。

なお、比較実験の前に、それぞれのプログラムが同じ入出力を行なっていることを確認した。抽出パターンのリストを比較した結果、TreeMiner が正常終了した場合については CSLOGS だけでなく、他の実験データに関しても両プログラムの抽出結果は完全に一致した。

図 12 に処理時間の測定結果を示す。横軸はパターン抽出の閾値である minimum support の設定値、縦軸は処理時間（秒）を対数で示している。

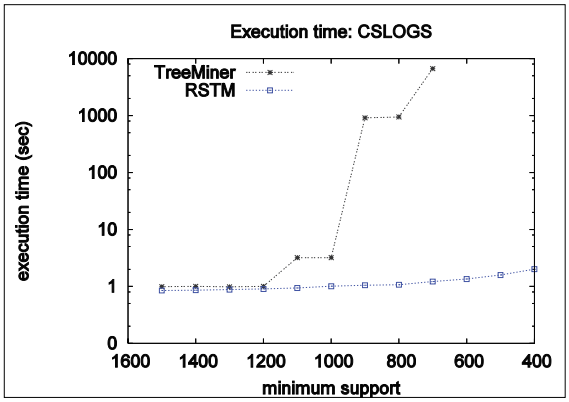

図 12. RSTM の処理時間(CSLOGS)
Execution Time of RSTM (CSLOGS)

TreeMiner は閾値 (minimum support) を 1100 以下、さらには 900 以下に設定した場合に処理時間が急峻に増加し、700 以下では正常終了しなかった。これに対して RSTM では実験した全ての閾値において、処理時間に特段の変化は観測されなかった。更に、TreeMiner が数時間処理を実行して異常終了する、700 以下に閾値(minimum support)を設定しても RSTM は数秒で処理を正常に完了した。

図 13 に使用メモリ量の測定結果を示す。TreeMiner では閾値 1100 以下で使用量が増加をはじめ、閾値 700 ではメインメモリ 4Gbyte の実験環境では必要なメモリを獲得できず処理

が異常終了した。これに対して RSTM が使用したメモリ量は、実験した全ての閾値において約 100MByte で一定していた。

以上の実験結果から、RSTM はより小さな抽出閾値を設定しても TreeMiner より高速に動作する性能を持つといえる。小さな抽出閾値を設定すれば、情報の多いより大きなパターンを抽出可能になるため、RSTM は実用面で TreeMiner に対し優位性を持っていることになる。

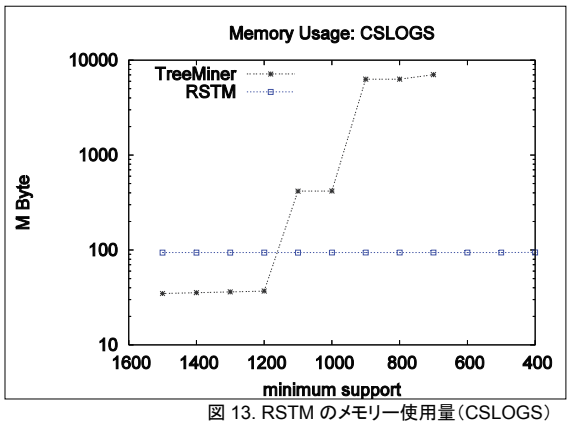

図 13. RSTM のメモリー使用量（CSLOGS）
Memory Usage of RSTM（CSLOGS）

3.4　ドキュメントの操作ログへの適用例

本小節では、RSTM によるパターン抽出事例として、DocumentHistory[3]（ドキュメントの操作履歴を統合的に管理するシステム）のログデータに適用した例を紹介する。

DocumentHistory は各ドキュメントについて誰が作成し、誰が閲覧し、誰が編集したか、誰が複写したかなどの一連のドキュメント操作を記録するシステムである。ドキュメントが複製や配布されることで操作履歴が分岐するため、ログデータはツリー構造として記録される（図 14）。

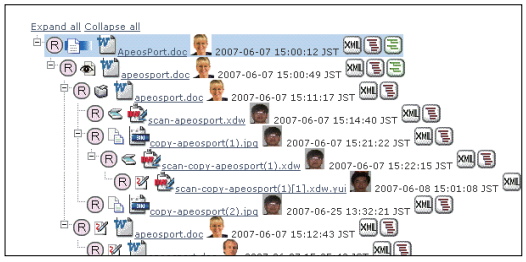

図 14. Document History での記録構造の例
A sample of tracable record of Document History

この DocumentHistory のプロトタイプシステムに記録された約 1 年分のログデータから RSTM を用いてパターンの抽出を行なったところ、幾つかの典型的なパターンが抽出された。以下ではそのうちの一事例として、DocumentHistory のデモが行なわれていたプロセスの抽出に関して紹介する。

図 15 はログの中の比較的初期の時期から抽出されたパターン（a）とパターンが出現したデータの一例（b）を示したものである。図 15(a) はＤさんがドキュメントを作成(53)、編集(67)、印刷(54)した左側の紙のフローと、Ｄさんが再度閲覧(57)した後でＦｂさんが編集(84)してＤさんが再度閲覧(57)している右側の電子ドキュメントのフローを持つパターンを示している。

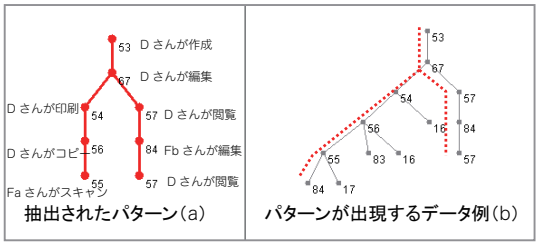

図 15. 第 1 期に抽出されたパターン
A pattern from first phase

図 15 のパターン抽出された期間とは異なる期間に、図 16（a）に示すパターンが抽出された。図 16（a）のパターンでは分岐の右下が図 15（a）の場合と異なっており、Fb さんの編集が Fa さんの編集(69)に変更され、ドキュメントを編集した後に元の文書を閲覧するパスが現れている（図 16（a）の一番右の 57)。

当事者は当初同じシナリオを用いてデモを行なっていたと記憶していた。しかしながら、図 15、図 16 に示すパターン変化を提示すると、当事者は途中でシナリオを変えたことを思い出した。以上より、プロセスの変化が正しく抽出できたことを確認できたと考えている。

RSTM は本事例のように活動記録などから典型的なパターンに加えて、変化の兆候を抽出することができ、トラブルなどの検知にも貢献できる可能性がある。

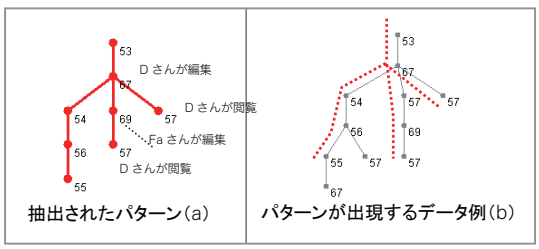

図 16. 第 2 期で抽出されたパターン
A pattern from second phase

4. FSTM(Flexible Search Tree Mining)

RSTM のパターン探索処理では、最初に一個または二個のノードからなるパターンを発見した後に、そのパターンにノードを追加したパターンを見つけることを再帰的に処理を繰り返す。この処理手順においては、RMP を利用してパターンをルートノードから右下に向かって伸張する。このため、図 17 に例示するように上位の構造に共通性が高いツリーデータを処理する場合には、上位の共通部分だけからなる実用上無駄な探索を大量に繰り返すこととなり、処理効率が悪くなる。

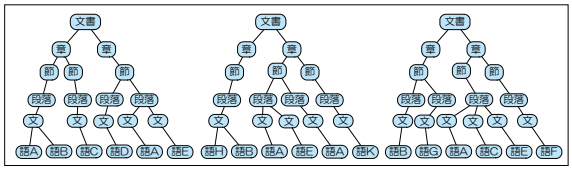

図 17. 上位構造の共通性が高いツリーの例
Examples of trees sharing main upper nodes

このため、下位のノードを含むパターンを効率的に探索するアルゴリズム FSTM（Flexible Search Tree Mining）を研究開発した。以下、FSTM について簡単に説明する。

4.1 リーフノードからのパターン探索

FSTM は RMP へのノードの追加を再帰的に行なってパターンの伸張を行なう点等は RSTM と共通である。RSTM がパターンをルートから右下にパターンを伸張するのに対して、FSTM では RMP のリーフノードを先に求めてからルートノードとの間を埋めていく点が異なる。FSTM でパターンを伸張していく流れを図 18 に簡単に示す。

FSTM は最初に下位のノードをパターンに追加するため、データツリーの上位構造に共通性が高いデータ群を処理対象とした場合も、上位構造の探索を後に行なうことが可能になるため、処理時間の面でも使用メモリ量の面でも効率的に処理が実行できる。

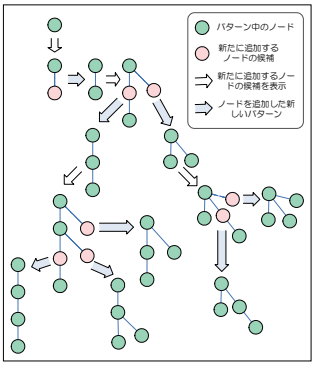

図 18. FSTM のパターン伸張の流れ
Pattern expansion of FSTM

FSTM では、先祖子孫関係にある 2 つのノードの間に存在するノードをデータツリーから探索する必要がある。このため、FSTM では、データツリーごとに各 RLP 毎に探索可能な深さをリーフノードの位置から算出して格納したテーブル情報を用意して処理を効率化している。パターン抽出の再帰処理が進行しても、パターンに RLP が追加されない場合は同じテーブル情報を使用することで、テーブル情報作成の計算量を抑えている。

4.2 評価実験

FSTM も、RSTM と同様に、同じ実験環境で TreeMiner と比較実験を行なった。実験では、ツリーの上位構造の共通性が高い例、文献 2)でも実験に用いている英文構文解析結果の XML から変換した 52681 ツリーからなるデータを用いた（http://www.cs.washington.edu/research/xmldatasets/）。図 19 に処理時間の測定結果を示す。

パターン抽出の閾値を下げていくと TreeMiner は閾値 35000 程度で処理時間が増大し異常終了した。また、RSTM は TreeMiner より効率が優れているが、閾値を小さくするにつれ処理時間が増え、閾値が 100 以下の場合を計算することは不可能であった。しかしながら、FSTM は抽出閾値を 10 以下にしても数秒内に処理が終了した。

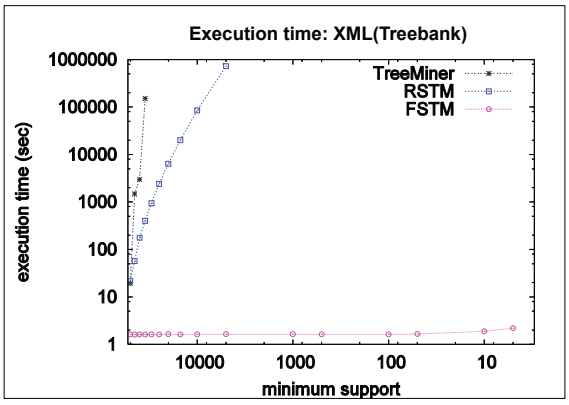

図 19. FSTM の処理時間(Treebank)
Execution Time of FSTM(Treebank)

図 20 はメモリ使用量を示したグラフである。

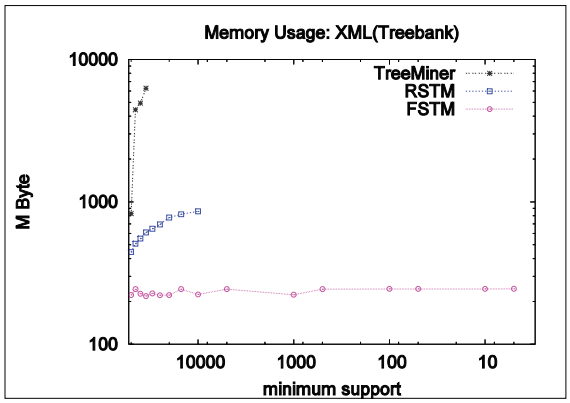

図 20. FSTM のメモリ使用量（Treebank）
Memory Usage of FSTM（Treebank）

TreeMiner は抽出閾値 35000 程度で処理に必要なメモリが確保できず処理が終了している。FSTM では各データツリーに RLP 毎に探索可能な深さを記録したテーブル情報を追加して、そのためのメモリも余分に使用しているにもかかわらず RSTM に対して使用メモリ量は数分の一であった。FSTM では、リーフノードから探索するために抽出ツリーパターン候補を保持する数が減少し、また、処理対象のデータツリーの数を抑えることができる。これによる使用メモリ量の削減効果の方が、FSTM でテーブル情報を追加した影響よりも大きいことが分かる。

4.3　FSTM の適用例

FSTM の適用例として、日本語の構文解析結果から抽出したパターンの例を紹介する。図 21 はある病院の読影レポートを Cabocha[7]を用いて構文解析したものに、さらに単語に分類属性を付与して構造化したデータからパターン抽出を行なった結果の一例である。

この抽出実験では単語と、単語の分類情報を、パターンのリーフ候補ラベルとして分類して処理を行なっている。このため、単語まで一致する箇所については具体的な単語が、分類情報までしか一致しない箇所については分類情報までの情報が、文節情報などの上位の構造と共にパターンとして抽出されている。図 21 の例では、「((医療-部位名)（医療-領域）内)（(明らか)(医療-病状)）認める」という形で、繰り返し使用された部分表現が構文構造の情報を含めて抽出できている。上記パターンを抽出することで、

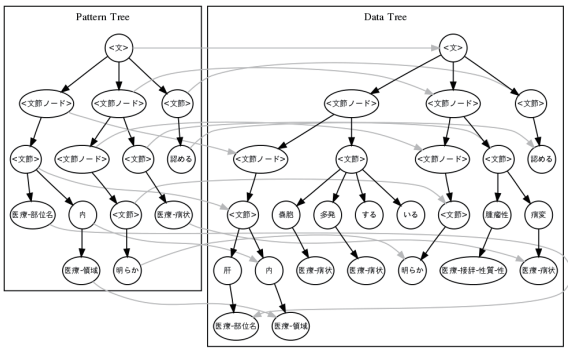

図 21. 汎化情報を加えた構文解析結果からのパターン抽出
An extracted pattern from analyzed text collection

電子カルテなどに入力するメニュー作成の支援が可能である。

FSTM ではこの例のようにツリー構造の上部に共通性が高い構造があるデータ群に対してしても、注目したい情報を含めたパターンを効率的に抽出することができる。本節では説明を簡単にするために FSTM をパターンのリーフノードを先行して見つける処理として紹介したが、内部ノードから探索を開始しパターンを抽出する処理にも適用できる。すなわち、FSTM では、優先的に抽出したい情報に注目したうえで、その周辺の構造情報を Flexible に探索してパターン抽出を行なうことが可能である。

5.　結言

木構造データ群からパターン情報を抽出する 2 種類の基本技術の開発・評価を行ない、適用例を紹介した。ルートからリーフ方向にパターンを伸張していく RSTM はドキュメントの操作履歴など情報の接続関係が木構造となる場合に効率的にパターンを抽出できる。一方、下位ノードや中間ノードからパターンを抽出できる FSTM は、XML など部分情報をまとめ上げてできる木構造から効率的にパターンを抽出することができる。

これらの技術には従来技術に対して性能面で大きな優位性を示す実験結果を得ており、更なる高性能化と高機能化に向けて研究開発を行なっている。また同時に、ドキュメントサービス＆コミュニケーションのサービスの差別化に貢献するマイニング・アプリケーションの研究開発を進めている。今後、クラウド技術の普及な

により大規模なドキュメントのリポジトリが実現されてくることが予想される。このため、サブツリーマイニング技術を活用し、ドキュメントのリポジトリ中の有用なパターン情報を抽出可能にし、企業などに価値の高いソリューションを提供していく。

6. 商標について

7. 参考文献

1) M. Zaki. Efficiently mining frequent trees in a forest. 8th ACM SIGKDD Int'l Conf Knowledge Discovery and Data Mining, July, (2002).
2) S. Tatikonda, S. Parthasarathy, and T. Kurc. Trips and tides: New algorithms for tree mining. 15th ACM international conference on Information and knowledge management, pages 455–464, (2006).
3) W. Peng, T. Terao, Y. Masuda, N. Yoshida, and J. Miyazaki. Document history system and its application to abnormal document access pattern detection with a probabilistic model. 22nd International Conference on Advanced Information. Networking and Applications(AINA2008). (2008).
4) R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo. Fast discovery of association rules. In U. F. et al., editor, Advances in Knowledge Discovery and Data Mining, pages 307–328. Menlo Park, Calif.: AAAI Press, (1996).
5) 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).
6) H. Tan, T. Dillon, F. Hadzic, E. Chang, and L. Feng. MB3-miner: mining eMBedded subTREEs using tree model guided candidate generation. Proc of the 1st International Workshop on Mining Complex Data, held in conjunction with ICDM'05, (2005).
7) 工藤 拓、松本 裕治. チャンキングの段階適用による日本語係り受け解析. 情報処理学会論文誌 Vol.43, No.6. pp.1834-1842. (2002).
8) Han, J., Kamber, M. Data mining: Concepts and Techniques, 2nd Edition. Morgan. Kaufmann Publishers Inc., San Francisco. (2006).
9) Mohammed J. Zaki. Efficiently Mining Frequent Trees in a Forest: Algorithms and Applications. IEEE Transactions on Knowledge and Data Engineering, 17(8), 1021–1035. August (2005).
10) 浅井達哉、安部賢治、川副信治、坂本比呂志、有村博紀、有川節夫. 半構造データからの頻出パターン発見アルゴリズム. 第 13 回データ工学ワークショップ（DEWS-2002） (2002).

筆者紹介

林　千登
研究技術開発本部 システム要素技術研究所に所属
専門分野：知識処理

吉岡　健
研究技術開発本部 基盤技術研究所に所属
専門分野：知識処理