知 能 と 複 雑 系 128−26
(2 0 0 2 . 5 . 2 4)

テキストデータベースからの構文構造のマイニング

工藤 拓† 山本 薫‡ 坪井 祐太†† 松本 裕治†

† 奈良先端科学技術大学院大学 情報科学研究科
‡ 理化学研究所ゲノム科学総合研究センター
†† 日本 IBM 東京基礎研究所

†{taku-ku,matsu}@is.aist-nara.ac.jp, ‡kaorux@gsc.riken.go.jp, ††yuutat@jp.ibm.com

テキストマイニングの分野は決して新しいものではなく, 単語の共起やクラスタリングといった多くの手法が現在まで提案されている. しかし, それらの多くは, テキストを単語の集合や 1 次元的な列という簡単な構造で表現しており, 単語間の関係や構文的な係り受け関係を無視してしまっている. テキストに対して, 平坦な構造しか仮定していない場合には, 文がもつ本当に意味のある構造をうまくとらえらない. 本稿では, チャンキングや係り受け解析といった自然言語処理ツールを使用し, テキストデータを具体的な内容をよりよく反映した半構造化データに変換し, そこから有益なパターンを抽出する手法を提案する. 具体的には, テキストマイニングを半構造化されたデータから頻出する部分構造の抽出ととらえ, シーケンシャルパターンのマイニング手法である PrefixSpan アルゴリズムを拡張し, この問題を効率よく解決するアルゴリズムを提案する.

キーワード: テキストマイニング, シーケンシャルパターンマイニング, 半構造化データ, PrefixSpan

Mining Syntactic Structures from Text Database

Taku Kudo† Kaoru Yamamoto‡ Yuta Tsuboi†† Yuji Matsumoto†

†Graduate School of Information Science, Nara Institute Science and Technology
‡RIKEN Genomic Sciences Center
††IBM Research, Tokyo Research Laboratory, IBM Japan, Ltd.

†{taku-ku,matsu}@is.aist-nara.ac.jp, ‡kaorux@gsc.riken.go.jp, ††yuutat@jp.ibm.com

Text mining has gained the focus of attention recently, in particular, the success in word clustering have been reported. However, many of these bag-of-word or sequence-of-word approaches ignore the implicit dependency relations between words which are critical to understanding of the original text. In this paper, we apply syntactic parser to convert a raw text into a semi-structured text from which useful patterns are extracted. We extend the PrefixSpan algorithm, one of an efficient algorithms for sequential pattern mining, to efficiently extract sub-structures from a text data annotated by syntactic parser.

Keywords: Text Mining, Sequential Pattern Mining, Semi-structured Data, PrefixSpan

1 はじめに

データマイニングとは, 膨大なデータの中から有用, あるいは興味のあるパターンを明示的な知識として発掘する方法についての科学的研究である.

その対象をテキストデータとしたテキストマイニングの分野は, 決して目新しいものではない. 例えば, 語と語の共起関係や相関関係を調べたり, 語のクラスタリングといった処理は, 古くから研究されてきている [2, 3]. これらの研究の多くは, 形態素解析といった比較的「浅い」言語処理技術のみを利用し, テキスト全体を単語の集合や一次元的な列とみなして処理を行っている. しかし, このようなデータ構造は, 機械処理に向いているといはいえ, 各単語のテキスト中の役割, 単語間の係り受け関係などを考慮に入れないため, 文がもつ本当に意味のある構造をうまくとらえらないという問題がある.

その一方で, Text Chunking, 係り受け解析 (構文解析), 固有名詞抽出といった高度な自然言語処理ツールが近年利用できるようになってきた. これらのツールにより, テキスト中の単語の役割がより明確になり, テキストの具体的な内容をよりよく反映したデータ構造に変換することができる. 自然言語処理ツールを利用することで, 生のテキストは, ある構造を持ったデータ (半構造テキストデータ) に変換される. しかし, 複雑なデータ構造になればなるほど, 単純に単語の集合を前提としていたようなアプローチは有効に機能しなくなってくる. 今後, このような半構造化されたテキストデータからいかに有用なパターンを発見するかが大きな課題となってくる.

実際に, 半構造データベースに関して, 現在多くの研究が行われている. 安部ら [5] は, 半構造データとして, 木構造のデータをとりあげ, その集積から頻出する部分構造を効率良く発見する手法を提案している. 松澤 [6] は, 構文解析済みのテキストデータから頻出する部分構造を効率良く発見する手法を提案している.

本稿では, このような流れを受け, 自然言語処理ツールの結果から得られた半構造化されたテキストデータに対するマイニング手法提案する. 具体的には, まず, 半構造データマイニングを与えられた半構造データから頻出する部分構造を発見する問題と定式化する. さらに, シーケンシャルパターンのマイニング手法である PrefixSpan アルゴリズム [4] を拡張し, 構文解析済みの大量のテキストデータから, 頻出する部分構文木 (部分構文構造) を高速に発見するアルゴリズムを提案する. さらに, 実際のテキストデータを用いた実験結果を示し, 提案手法の有効性について議論する.

2 シーケンシャルパターンマイニングと PrefixSpan

Agrawal[1] らは, シーケンシャルパターンマイニングを以下のように定式化している [1]

$I = \{i_1, i_2, \ldots i_n\}$ を, **アイテム** 集合とする. **系列** とは, アイテムの列であり, $\langle u_1, u_2, \ldots, u_n \rangle$, と表記する. u_k は任意のアイテムである. ある系列 α 中のすべての アイテムが, 別の系列 β 中に存在し, その順番が保持されている場合, 系列 β は系列 α を「含む」と言い, $\alpha \sqsubseteq \beta$ と表記する. **系列データベース** S とは, 系列 id sid と系列 s のタプル $\langle sid, s \rangle$ の集合である. $S = \{\langle sid_1, s_1 \rangle, \langle sid_2, s_2 \rangle, \ldots, \langle sid_2, s_n \rangle\}$. さらに, 系列 α の系列データベース S における**サポート** $support_S(\alpha)$ とは, S 中のすべての系列のうち, 系列 α を含むタプルの数と定義される.

シーケンシャルパターンマイニングとは, 系列データベース S と 任意の整数 (**最小サポート値 (minimum support)** ξ) に対し, $support_S(\alpha) \geq \xi$ となるような系列 α を全て列挙するタスクの事を指す.

この問題に対し, Pei ら [4] は, 深さ優先探索で多頻度パターンを抽出する手法 PrefixSpan を提案している. PrefixSpan を説明する前に, そのアルゴリズムの核となる**射影 (project)** という操作を説明する. ある系列 $s = \langle a_1, a_2, \ldots a_m \rangle$, アイテム a に対し, $a_1 \neq a,\ a_2 \neq a, \ldots, a_{j-1} \neq a,\ a_j = a$ となるような整数 $j (1 \leq j \leq m)$ が存在する場合, 系列 $\langle a_1, a_2, \ldots a_j \rangle$ を s の a に対する **prefix** ($prefix(s, a)$) と定義する. また, 系列 $\langle a_{j+1}, \ldots, a_m \rangle$ を s の aに対する **postfix** ($postfix(s, a)$) と定義する. もし j が存在しない場合は, prefix,postfix は未定義 (ε) となる. ある系列データベース S に対し, アイテム a によって 射影し, **射影データベース** $S|a$ を作成するとは, S 中のそれぞれの系列 s に対し, $postfix(s, a)$ を作成し, それらを改めて系列データベースとする操作と定義される.

$$S|a = \{\langle sid, s' \rangle | (\langle sid, s \rangle \in S) \wedge (s' = postfix(s, a)) \wedge (s' \neq \varepsilon)\}$$

図 1 に, $\xi = 2$ とした場合の PrefixSpan の動作過程を示す. まず, アイテム数 1 の多頻度系列 a, b, c を抽出する. d は, サポートが 1 であるため, 多頻度系列ではない. 多頻度系列は, これら 3 種類あるが, そのうち a について考え, S の a による 射影データベース $S|a$ を作成する (図 1 中央上). この データベースから, 多頻度系列 b, c を生成する. 以下再帰的にこれらの作業を繰り返すことで, すべての多頻度系列を抽出することができる. PrefixSpan の擬似コードを 図 2 に示す.

[1] この定式化は, エレメントが無いため厳密ではない. 簡略化のためにエレメントは省いている.

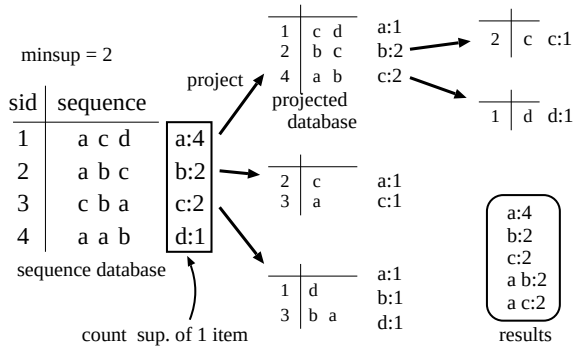

図 1: PrefixSpan の動作過程

```
call PrefixSpan(ε, S)
procedure PrefixSpan (α, S|α)
begin
  B ← {b|(s ∈ S|α, b ∈ s) ∧ (support_S|α(⟨b⟩) ≥ ξ)}
  foreach b ∈ B
  begin
    (S|α)|b ← {⟨sid, s'⟩|(⟨sid, s⟩ ∈ S|α)
                ∧ (s' = postfix(s, b)) ∧ (s' ≠ ε)}
    call PrefixSpan (αb, (S|α)|b)
  end
end
```

図 2: PrefixSpan の擬似コード

3 PrefixSpan の木構造への拡張

3.1 定式化

シーケンシャルパターンマイニングと対比させながら, 順序木のマイニング問題を定式化する. $I = \{i_1, i_2, \ldots i_n\}$ を, **アイテム** 集合とする. **順序木** とは, アイテムを節点とし, 兄弟間には全順序が定義される木構造の事を言う.

ある順序木 α が, 別の順序木 β の部分木である時, β は α を「含む」と定義し, $\alpha \sqsubseteq \beta$ と表記する. 具体的には以下の条件を満たすような射影 $\phi : \alpha \to \beta$ が存在する必要がある.

(1) ϕ は単射である.

(2) ϕ は親子関係, 兄弟関係を保持する. ただし α 中で隣接する兄弟は, β 中で隣接する必要はない.

順序木データベース T とは, 順序木 id tid とそれに対応する順序木 t のタプル $\langle tid, t\rangle$ の集合である. $T = \{\langle tid_1, t_1\rangle, \langle tid_2, t_2\rangle, \ldots, \langle tid_2, t_n\rangle\}$. さらに, 部分順序木 α の順序木データベース T におけるサポートとは, T 中のすべての順序木のうち, 部分順序木 α を含むタプルの数と定義される.

$$support_T(\alpha) = \left|\{\langle tid, t\rangle | (\langle tid, t\rangle \in T) \wedge (\alpha \sqsubseteq t)\}\right|$$

順序木のマイニングとは, 順序木データベース T と任意の整数 (最小サポート値 (minimum support) ξ) に対し, $support_T(\alpha) \geq \xi$ となるような部分順序木 α を全て列挙するタスクの事を指す.

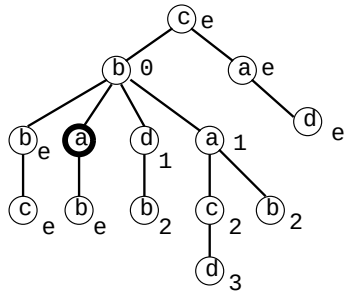

図 3: 節点 a から見た他の節点との関係

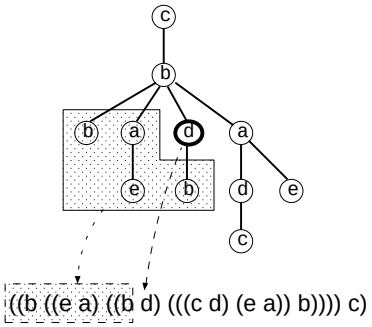

図 4: 系列表現への変換

3.2 節点間の関係

順序木 t 中の任意の節点 i と i とは異なる別の節点 j の 2 つの引数を取る関数 $\psi : i \to j$ を以下のように定義する. ψ は, 一般に非対称の関数となる.

(1) j は i の親: $\psi = 0$

(2) j の $k-1(k \geq 1)$ 代目の祖先は, i の右側の兄弟: $\psi = k$

(3) (1),(2) 以外: $\psi = \varepsilon$

図 3 に, 関係関数 $\psi : i \to j$ の例を示す.

3.3 木構造の系列表現

順序木 t をそれと等価な**系列表現**に変換することを考える. 系列として表現することで, PrefixSpan が適用可能となる. ただし, 系列に変換しても, 前節で定義した節点間の関係はそのまま保持されていることとする. つまり, 順序木としての性質はこの変換で失われることはない. 具体的には, 木の右から左へ, 前順操作 (pre-order traverse) で節点を列挙した結果を反転する. この変換により, 順序木中の任意の節点の左隅の部分構造が, 系列中では, 節点に対応するアイテムの左側にある系列として表現される.

図 4 に, 順序木を系列表現に変換した例を示す. ただし, 木構造の関係を明確にするため, 系列は S 式として表現している. 順序木中の太丸で囲んだ節点 d の左隅の構造が, 系列中のアイテム d の左側の系列に対応する.

3.4 アルゴリズム

順序木データベース T は, 順序木 id tid と順序木 t のタプル $\langle tid, t\rangle$ の集合であった. ここでは, PrefixSpan を適用するために, 順序木データベースを tid と, 系列表現 t' とのタプルの集合 $\langle tid, t'\rangle$ として取り扱う. この操作は, 問題の定式化自身を変えない.

順序木マイニングアルゴリズムの基本的なアイデアは, 系列表現を, 単純な系列データとみなして PrefixSpan を動作させることにある. しかし, このままでは, 木構造の情報が欠落してしまい, 本来の目的が達成されない. そこで, 前々節で説明した関係関数 ψ を使用する. 具体的には, PrefixSpan に対し次の変更を加える.

(1) 通常のアイテムを, 節点名と関係のタプル $\langle i, r\rangle$ という形で表現する.

(2) アルゴリズムを動作させる前に, 系列データベースの全アイテムを $\langle i, 0\rangle$ (i は各節点名) に初期化しておく.

(3) アイテム $\langle i, r_1\rangle$ で射影をする場合, 各系列中のアイテム $\langle i, r_1\rangle$ の右側にある 全アイテム $\langle j, r_2\rangle$ に対し, $r_3 = \psi(i, j)$ を求める. 射影データベース中のアイテム $\langle j, r_2\rangle$ を, $\langle j, r_3\rangle$ に置きかえる.

(4) 1 つの系列に, 同一アイテムが複数存在する場合は, 最左のアイテムだけでなく, すべてのアイテムに対し射影を行う[2].

(5) (4) より, 同一の系列から, 複数の系列が生成されるが, 各アイテムの support は, 系列単位で算出する.

(6) $\langle i, \varepsilon\rangle$ (i は任意の節点) のように, 関係が ε となる場合は, 数え上げの対象に含めない.

系列表現での prefix は, 順序木では, 左隅部分構造に対応する. つまり, このアリゴリズムは, 木の左隅から部分木パターンを深さ優先で広げていき, 頻出パターンを抽出していく. 結果として得られるパターンは, 節点名と関係のタプルの系列である. この系列を, 順序木に戻せば, 最終的な結果を得ることができる.

図 6 に, 実際の動作例を示す. ただし図 6 中では, タプル $\langle a, b\rangle$ を 便宜的に a-b と表記している.

実際にアルゴリズムを動作させるにあたり, 射影時に毎回タプルを作成していたのでは効率が悪い. そのため, 実際には, 系列データの id と prefix の開始位置を記憶しておき, それらの情報を用いて射影を行う. 図 5 に, 実際のアルゴリズムの擬似コードを示す.

[2]通常の PrefixSpan は最左のアイテムのみ考慮する.

func $seq(T, tid)$:= T 中の 系列 id tid に対応する系列表現
func $node(T, tid, p)$:= $seq(T, tid)$ の p 番目の節点
func $\psi(T, tid, p, q)$:= $node(T, tid, p)$,$node(T, tid, q)$ の関係
ただし, $\psi(T, tid, 0, q) := 0$

```
# 順序木の系列表現データ
```
$T = \{\langle tid_1, t'_1\rangle, \langle tid_2, t'_2\rangle, \ldots, \langle tid_n, t'_n\rangle\}$
```
# tid と 位置情報のポインタ
```
$P = \{\langle tid_1, 0\rangle, \langle tid_2, 0\rangle, \ldots, \langle tid_n, 0\rangle\}$

call $PrefixSpan(\varepsilon, P)$

proc $PrefixSpan\ (\alpha, P|\alpha)$
begin
```
  # B は，タプルをキー，
  # タプルの集合を値とするマップ
```
  $B \leftarrow \{\}$
  foreach $\langle tid, l\rangle \in P|\alpha$
  begin
    foreach $k \leftarrow l + 1$ **to** $|seq(T, tid)|$
    begin
      $b \leftarrow node(T, tid, k)$
      $r \leftarrow \psi(T, tid, l, k)$
      $B[\langle b, r\rangle] \leftarrow B[\langle b, r\rangle] \cup \langle tid, k\rangle$
    end
  end
  foreach $\langle b, r\rangle \in$ keys of B
  begin
    if $(support|_{P_\alpha}(\langle b, r\rangle) < \xi)$ **continue**
    call $PrefixSpan\ (\alpha\langle b, r\rangle, B[\langle b, r\rangle])$
  end
end

図 5: 順序木に対する PrefixSpan の擬似コード

4 実験

アルゴリズムの性能を評価するために, 以下の 2 つの実テキストデータを用いて実験を行った.

- 新聞記事
  京都大学コーパス 3.0 全文, 38,383 文. このコーパスは, 95 年 1 月の毎日新聞記事に対し, 形態素解析, 構文解析 (文節係り受け解析) が施されたものである.
- 小説
  夏目漱石の「我輩は猫である」全文 9,298 文[3]. 形態素解析 (文/単語認定), 構文解析 (文節係り受け解析) には, ChaSen[4] 及び CaboCha[5] を使用した.

構文木としては, 文節の表層を単位とする依存構造木を考える. また, 単純な順序木の場合, 「(本を (太郎に あげる))」と「((太郎に (本をあげる))」の 2 つの構文木は, 別の木となってしまうため, 兄弟関係にある節点をソートし非順序木に変換した. 実験は, PC (XEON 2.2 GHz, RAM 3.5GB, Linux) 上で行った. 実装には Perl を使用した.

4.1 実験結果

図 7 に 新聞記事を用い, 最小サポート値を 3 とした場合のデータサイズの増加に対するアルゴリズム

[3]http://www.aozora.gr.jp/
[4]http://chasen.aist-nara.ac.jp/
[5]http://cl.aist-nara.ac.jp/~ taku-ku/software/cabocha/

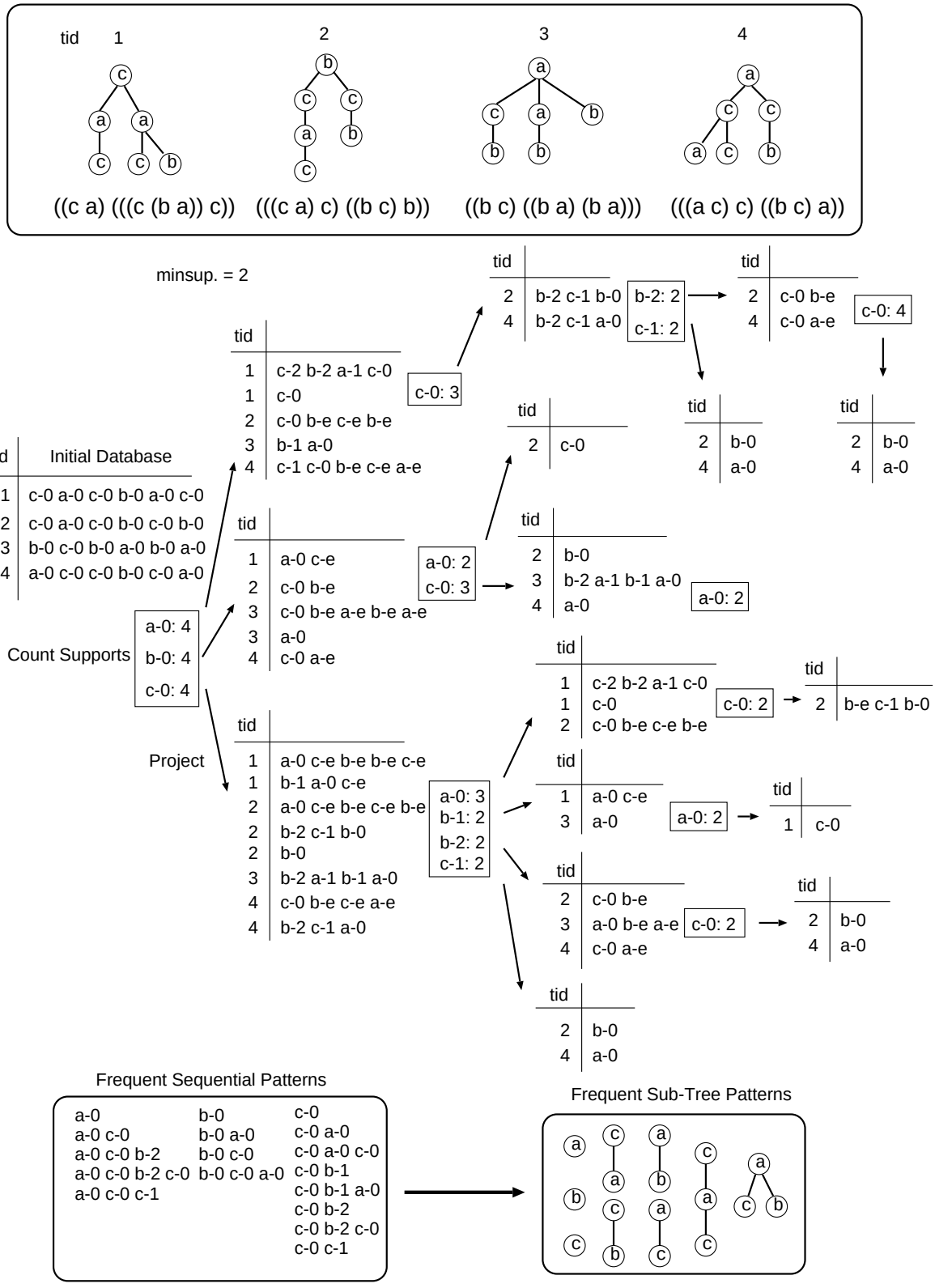

図 6: マイニングの動作例

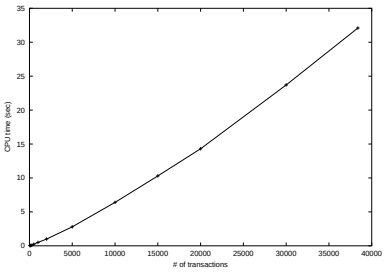

図 7: データサイズと計算時間

表 1: 最小サポート値 vs. 実行時間

minsup	2	3	5	10	20
新聞記事 (秒)	434.0	31.3	26.7	23.7	21.4
小説 (秒)	7.4	6.3	5.5	4.8	4.3

の計算時間の増加を示す. この図から, 提案アルゴリズムは充分な規模耐性を持つことが分かる.

表 1 に, 最小サポート値を変化させた時の, アルゴリズムの計算時間の増減を示す. 最小サポートを 2 (0.01%) にした時でさえ, 本手法が, 全パターンを発見するのに要した時間は 7 分程度であった.

比較として, 安部ら [5] の順序木のマイニング手法を, 同一の新聞データで実験したところ, 最小サポート値が 2 の場合は, 半日たっても終了しなかった. 実験環境や実装に用いた言語が異なるため, 厳密な比較はできないが, 提案手法は最小サポート値が小さい時に有効な手法と言える.

以下に, 抽出された頻出パターンのいくつかを紹介する. これらがどれだけ有用で興味深いかの客観的な評価は行えないが, 文としての構造を崩していないため, それ自身で何かしらの意味を持つようなパターンが抽出されている.

・新聞
((ついて 述べ、)(記者会見で、 明らかにした。)) 2 回
((機能を 果たしておらず、)(存在感すら 薄れている)) 3 回
・小説
(休養を (また (吾輩は 要する。))) 2 回
((云う 声が)(また する。)) 2 回

5 アルゴリズムの改良の可能性

実験により, 本アルゴリズムは, 充分な規模耐性を持つことが分かったが, まだ改善の余地がある. ここでは, それらのいくつかを挙げる.

- 非連結部分木の枝刈り
  図 6 の例には, c-0 b-2 c-0 のように, 連結していない部分木パターンが抽出されている[6]. 非連結パターンのうち, これ以上サイズが大きくならないものを何らかの方法で検出し, それらを早期に枝刈りすることで, 高速化が見込める.

[6]逆に言えば, 提案手法は, 兄弟関係のみのパターンを抽出できるアルゴリズムと言え, マイニングの対象によっては利点となる.

- 関係の算出/格納方法
  実験では, すべての関係をデータの読み込み時に計算し, メモリに格納した状態でアルゴリズムを動作させた. しかし, 関係値は, 各系列の節点の数の二乗に比例した記憶容量が必要なため, 大規模なデータでは, メモリに納まらない可能性がある. 速度を落とすことなく, 規模耐性を上げるためには, 効率の良い関係算出/格納アルゴリズムを採用する必要がある.

6 まとめと今後の課題

本稿では, 構文解析済みのテキストに対するマイニング手法提案した. 具体的には, シーケンシャルパターンのマイニング手法である PrefixSpan アルゴリズムを拡張し, 順序木の集合から, 頻出する部分木をマイニングできるよう拡張した. 実際のテキストデータを用いた実験を行い, 提案アルゴリズムが十分な規模耐性を持ち, 最小サポートが小さい時でも現実的な時間で終了することが分かった.

本稿では, 実際のテキストデータから頻出するパターンを抽出する実験のみに留まり, 抽出されたパターンの客観的な有効性に関する評価は行っていない. 今後は, 構文木 (係り受け構造) に対する本マイニング手法と従来の単語の集合による手法を比較し, 抽出されるパターンにどのような違いが現われ, それらが具体的な応用においてどれほどの有意差を生じるか評価を行いたいと考える. また, グラフ構造といった一般的な構造に対する拡張や, それらの完全性や健全性についても議論していきたいと考える.

参考文献

[1] Rakesh Agrawal and Ramakrishnan Srikant. Mining sequential patterns. In Philip S. Yu and Arbee L. P. Chen, editors, *Proc. 11th Int. Conf. Data Engineering, ICDE*, pp. 3–14. IEEE Press, 6–10 1995.

[2] Robert Dale, Hermann Moisl, and Harold Somers. *Handbook of Natural Language Processing*. Marcel Dekker, 2000.

[3] Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. The MIT Press, 1999.

[4] Jian Pei, Jiawei Han, and et al. Prefixspan: Mining sequential patterns by prefix-projected growth. In *Proc. of International Conference of Data Engineering*, pp. 215–224, 2001.

[5] 安部賢治, 川副真治, 浅井達哉, 有村博紀, 坂本比呂志, 有川節夫. 頻出順序木パターン発見に基づくウェブマイニング. 人工知能学会研究会資料 SIG-FA/KBS-J22, pp. 133–139, 2001.

[6] 松澤裕史. 大規模データベースからの頻出構造化パターンの抽出. 情報処理学会論文誌, Vol. 42, No. 8, 2001.