部分文字列増幅法による共通パターン発見アルゴリズム

IKEDA DAISUKE
(Present address)Kyushu University Library

山田 泰寛
九州大学大学院システム情報科学府

廣川 佐千男
九州大学情報基盤センター

出版情報：情報処理学会論文誌. 数理モデル化と応用. 46 (2), pp.56-66, 2005-01-15. 一般社団法人情報処理学会
バージョン：

部分文字列増幅法による共通パターン発見アルゴリズム

池田大輔[†,☆] 山田泰寛[††] 廣川佐千男[†]

本論文では，複数の文字列に共通な部分を見つける問題を考察する．まず，この問題をパターンから生成された文字列の集合が与えられたときに，そのパターンの定数部分を見つける問題（テンプレート発見問題）として定式化する．パターンとは定数と変数からなる文字列で，パターンが生成する語は変数を定数文字列で置きかえて得られる．置きかえに用いられる文字列中の部分文字列の頻度分布はベキ分布に従うことを仮定し，高確率でテンプレート発見を解くアルゴリズムを構築する．共通部分の発見問題の 1 つである最長の共通部分列を探す問題は NP 完全であることが知られているが，問題の再定式化，部分文字列の集合による定数部分の表現方法，部分文字列の頻度と総出現数から共通部分を発見する手法により，テンプレート発見問題は高確率で $O(n)$ 時間で解けることを示す．ここで，n は入力文字列の長さの和である．さらに，このアルゴリズムがノイズに対し頑健であることと，複数のテンプレートが混在する場合でも有効であることを，Web 上の実データに適用することで実証する．

A Pattern Discovery Algorithm by Substring Amplification

Daisuke Ikeda,[†,☆] Yasuhiro Yamada[††] and Sachio Hirokawa[†]

In this paper, we consider to find common parts among given strings. We define this problem as the *template discovery problem* which is, given a set of strings generated by some pattern, to find constant parts of the pattern. A pattern is a string over constant and variable symbols, and generates strings by replacing variables into constant strings. We assume that the frequency of replaced constant strings follows a power-law distribution, and construct an algorithm which solves the problem with high probability. Although the longest common subsequence problem, which is one of the famous common part discovery problems, is well-known to be NP-complete, we show that the template discovery problem can be solved in $O(n)$ with high probability, where n is the total length of input strings. This complexity is achieved due to the following our contributions: reformulation of the problem, using a set of substrings to express a template, and using string frequency and all occurrences to find substrings common to input strings. Moreover, using data on the Web, we show noise robustness and effectiveness for the case that input strings are generated by different patterns.

1. はじめに

ゲノム配列や HTML/XML などの半構造化データ，メールやニュースのアーカイブなど大量のテキストデータが容易に入手可能となってきた．複数のテキストデータが与えられたときに，これらの多くに共通する部分を見つけることは一般的な問題であり，様々な分野に応用可能である．

たとえば，共通する部分として部分列がよく用いられる．ある文字列 w の**部分列**とは w から 0 個以上の文字を削除して得られる文字列である．たとえば *cbbbab* は *ccbbcbabbabca* の部分列である．共通する部分列の中で最も長いものを探す問題は最長共通部分列問題（longest common subsequence problem）と呼ばれ，NP 完全であることが知られている[11]．この問題の入力は任意の（複数の）文字列である．しかし，一般に共通な部分を見つけたい場合は，ある程度共通なものを含んでいる入力を仮定している場合が多い．たとえば，複数の DNA 配列に共通なパターンを見つけるときは多くの配列が共通なパターンを持つと考えてよい．Web マイニングでは情報抽出の研究がさかんに行われている[3),9),10),17)] が，情報抽出も共通のテンプレートを発見する問題と考えられる．情報抽出の入力はテンプレートを持つファイルが与えられる．

次に，共通な部分の表現方法としての部分列について考えてみる．上述の例では *cbbbab* は *ccbbcbabbabca*

† 九州大学情報基盤センター
Computing and Communications Center, Kyushu University
†† 九州大学大学院システム情報科学府
Department of Informatics, Kyushu University
☆ 現在，九州大学附属図書館
Presently with Kyushu University Library

の部分列であると述べた．しかし，下線部の取り方として *cc*bb*c*ba*bbab*c*a* としても，部分列 *cbbbab* が得られる．部分列は元の文字列の部分文字列を連接して得られるため，1 つの部分列を構成する部分文字列の列の取り方は複数存在する．このことは，生物学的に意味のあるまとまりを共通部分として取り出したい場合には不適当である．また，情報抽出の点から見ても，テンプレートを構成する部分文字列が変化するということは不自然である．

次に，共通でない部分について考えてみる．ゲノム配列の場合，異なる部分は進化の過程でランダムに変化してきたものと考えられている．情報抽出では，共通でない部分にはコンテンツとして自然言語の単語や文章が入る．このような単語や文章中の文字列の頻度と総出現数は線形の関係があり，べキ分布に従うことが経験的に知られている．よって，情報抽出のモデルとして共通でない部分の文字列の頻度はベキ分布に従うと仮定する．

以上の考察から，共通な部分を探す問題として**テンプレート発見問題**が自然に定式化できる．これは，未知のパターンから生成された複数の文字列が与えられたときに，パターンの定数部を探す問題である．ここで，パターンとは定数と変数からなる文字列で，パターンから生成される文字列とは，パターン中の変数を定数文字列で置きかえたものである．置きかえた部分の部分文字列の出現頻度と総出現数はベキ分布に従うものと仮定する．

パターン言語は機械学習の分野でよく研究されていて，変数の出現回数や種類を制限しないと，効率の良い学習はできないことが知られている[2),8),15)]．しかし，情報抽出の観点でいえば，これらの制限は実用的でない．変数の種類の固定は代入できるコンテンツの種類が固定されていることに対応し，変数の出現回数の固定は同じ種類のコンテンツの出現回数の固定に対応する．内部に木構造を持つ半構造化データを扱う場合は，同じ種類のデータを複数回出現することは普通であり，これらの制限は致命的である．一方，テンプレート発見問題では，発見するのはパターンの定数部分だけだが，変数の個数や種類には依存せず，情報抽出の定式化として妥当である．

問題の定式化から，テンプレート部分とそうでない部分の部分文字列の頻度の差を利用してテンプレート発見が可能であると予想できる．しかし，単純に頻度を数えるだけでは，最も短い 1 文字からなる部分文字列の頻度が高くなる．このような場合，自然言語の N グラム統計では，扱うデータの種類や使える計算機資源を考えて，あらかじめ頻度を数える文字列の長さを固定する[14),19)]．データマイニングの大きな枠組みの 1 つである頻出パターンマイニングでもサポートの最小値を与え，最小値より多く出現したものだけを有用とする[1),4),5)]．

本論文では，このような前提知識や負例・背景集合を要求せずに，正例のみからテンプレート発見問題を高速に解くアルゴリズムを提案する．ただし，テンプレート部分とそうでない部分の頻度の差が明確になるように，入力を生成するパターンの定数文字列はある程度の長さを持っていなければならない．このアルゴリズムは，代入される文字列の出現頻度と総出現数がベキ分布に従うことを利用しており，高確率で正しくテンプレートを発見できる．

アルゴリズムの時間計算量は $O(n)$ である．ここで，n は入力文字列の長さの和である．この計算量を実現するために，パターンの定数文字列をさらに短い文字列の集合で表す手法☆と，頻度 f ではなく f 回出現する文字列の総出現数 $F(f)$ を数えあげる**部分文字列増幅法**という手法を提案する．

提案アルゴリズムを実装し，Web 上の同一テンプレートを持つ複数のデータセットを用いて，実際にテンプレート発見に有効であることを示した．さらに，ノイズが多く含まれる場合や，1 ファイル中に繰り返し同じ定数文字列が含まれる場合，複数の異なるパターンから生成された文字列の和集合の場合についても，提案アルゴリズムは高速にテンプレートを発見できることを示した．このことは，提案アルゴリズムがパターン言語だけでなく，繰返しや和集合といった正規表現の一部が用いられている場合にもうまく動くことを示している．

本論文の構成は以下のとおりである．まず，2 章でパターン言語と接尾辞木を中心に用語の定義を与える．3 章では，共通部分を発見するということを情報抽出の観点からとらえ直し，テンプレート発見問題として定式化する．4 章で，部分文字列増幅法を用いたアルゴリズムを示した後，正当性の証明と計算量を与える．現実のデータによる実験結果を 5 章で与え，ノイズが含まれる場合や，繰返しを含むパターンやパターンの和から生成された入力であっても部分文字列増幅法により共通な定数文字列が発見できることを示す．

2. 準　　備

Σ を有限アルファベットとする．Σ の要素の列 $w =$

☆ このアイデアは文献 7) において導入された．

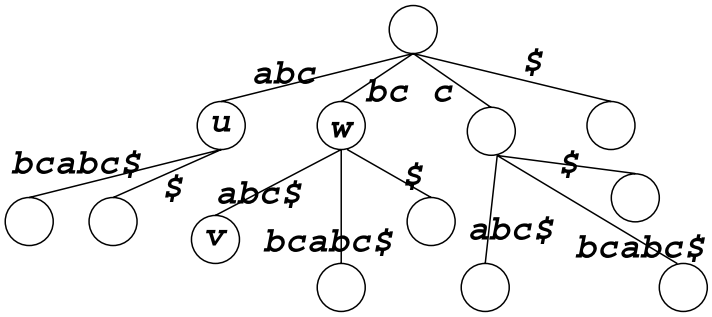


図 1 *abcbcabc* に対する接尾辞木
Fig. 1 The suffix tree for *abcbcabc*.

$a_1a_2\cdots a_n$ $(a_i \in \Sigma)$ を Σ 上の**文字列**と呼ぶ．Σ 上の文字列の集合を Σ^* で表し，Σ^* から空文字列を除いた集合を Σ^+ で表す．文字列 $w \in \Sigma^*$ に対し，$tuv = w$ なる文字列 t，u，v が存在するとき，u を w の**部分文字列**，t を w の**接頭辞**，v を w の**接尾辞**と呼ぶ．

2.1 パターン言語

$V = \{x_1, x_2, \ldots\}$ を Σ と交わらない集合とする．V の要素を**変数**と呼ぶ．$\Sigma \cup V$ 上の文字列を**パターン**と呼ぶ．ただし，パターンは少なくとも Σ の要素を 1 文字含んでいるものとする．V との対比から，パターンに現れる Σ 上の文字列を**定数**文字列と呼ぶ．各変数がたかだか 1 回しか出現しないパターンを**正則**パターンという．

V から Σ^* への写像 θ を**代入**と呼ぶ．代入 θ が変数 x へ文字列 u を代入することを $[x := u]$ と書き，複数の同時代入を $[x_1 := u_1, \ldots, x_m := u_m]$ と表す．

パターン p と代入 $\theta = [x_1 := u_1, \ldots, x_m := u_m]$ に対し，$p\theta$ により，パターン中の変数 x_i を u_i で置きかえて得られる文字列を表す．パターン p の**言語**とは，文字列の集合 $\{w \in \Sigma^+ \mid \exists\theta; p\theta = w\}$ のことであり，$L(p)$ で表す．

文字列の有限集合 $S \subset \Sigma^*$ を**サンプル**と呼ぶ．

2.2 接 尾 辞 木

\$ を，任意の文字 $a \in \Sigma$ に対し $a \neq$ \$ となる特別な文字とする．文字列 w に対し $A = w$\$ とおき，A_p $(1 \leq p \leq |A|)$ で A の p 文字目から始まる接尾辞を表す．$A_{p_1}, A_{p_2}, \ldots, A_{p_n}$ を，辞書式順序☆に並べた A のすべての接尾辞とする．w に対する**接尾辞木**とは $A_{p_1}, A_{p_2}, \ldots, A_{p_n}$ に対するコンパクトなトライのことである．接尾辞木のノード v に対し，根から v に至るパス上のラベルの文字列を連接してできた文字列を $string(v)$ で表し，**分岐語**と呼ぶ．

例 1 *abcbcabc* に対する接尾辞木を**図 1** に示す．ノード u と v に対し，それぞれ $string(u) = abc$ と $string(v) = bcabc\$$ である．

u を任意の接尾辞木のノードとする．$string(u)$ の出現回数は u の子孫となる葉の数である．実際，例 1 において，$string(w) = bc$ の子孫となる葉は 3 つであり，*abcbcabc* 中におけるその出現回数も 3 回である．

v を u の親ノードとすると，$string(v)$ は $string(u)$ の（真に短い）接頭辞である．さらに，$string(u)$ の接頭辞で，$string(v)$ を真に含むものの出現回数は $string(u)$ の出現回数と等しい．このとき，$string(u)$ と出現回数の等しい接頭辞は**等価**である，という．たとえば，上の例では $string(u) = abc$ の出現回数は，その接頭辞である a や ab の出現回数と等しい．つまり，部分文字列の頻度を数える場合，すべての部分文字列を数える必要はなく，分岐語のみを数えれば十分である．

3. テンプレート発見問題

本章では，情報抽出の観点から共通部分発見について考察を加え，テンプレート発見問題として定式化する．

情報抽出では，同じ形式のファイルが入力として与えられ，ファイル間に共通でない部分が情報として抽出される．このとき，まず共通部分を見つける必要がある．パターンでいうと，情報がパターン中の変数に代入され，共通部分が定数文字列である．

1 ファイル中に同じ情報が複数回埋めこまれることがあり，この場合は同じ変数がパターン中に 2 回以上必要である．実際，パターン言語の学習では各変数の出現回数を制限しないパターンの族が多く研究されてきた．しかし，本論文では，抽出した情報でどれとどれが同じかを判断することは，情報抽出の次のステップであると考え，純粋に情報抽出といえば，その情報

☆ 任意の文字列 $w \in \Sigma^*$ に対し，w を w\$ より先に並べることにする．

が何であるかは分からないが，すべて抽出することと考える．その意味では，同じ種類の情報が代入されるとしても，変数の対応づけは無視してよい．よって，パターンを正則パターンに制限する．

正則パターン p の変数はたかだか 1 回しか出現しないので，p 中に連続して現れる変数の列は新たな単一の変数に置きかえてよい．よって，p は $p = w_0x_1\cdots w_{m-1}x_mw_m$ $(w_i \in \Sigma^*\ (0 \le i \le m),\ x_i \in V\ (1 \le j \le m))$ と表現できる．この表現は変数名を除いて一意に定まる．このとき，p 中に出現する定数文字列の集合 $\{w_0, \ldots, w_m\}$ を**テンプレート**と呼ぶ．ただし，少なくとも 1 つの定数文字列は空ではないと仮定する．

この定義では，定数文字列の順序については関知しないことに注意する．これは，定数文字列が順序を含めて復元できれば，サンプルを生成した正則パターンが再構成されると期待されるが，パターンを正則に限ったとしても，現実的な時間内の学習は容易ではないことが知られている[16] からである．また，情報を抽出するのみであれば，サンプル中の各文字列から定数部分以外を抽出すればよいので，順序を再構築する必要はない．

次に，変数に対する代入について考察する．情報抽出の場合は，パターンの変数部分にはコンテンツが埋め込まれている．コンテンツは単語や文章など自然な文字列からなる．そこで，変数へ代入する定数文字列に頻度に関するベキ分布を仮定する．ベキ分布とはある 2 つの値 x と y を両対数でプロットしたときに，傾きが負の直線になるという法則である．つまり，ある定数 $a\ (>0)$，b が存在して $y = bx^{-a}$ が成立する．

x として英単語の頻度 f を，y としてサンプル S 中にちょうど f 回出現する異なる部分文字列の数 $V(f)$ をとると，経験的に次式のベキ分布が成立することが知られている．

$$\log V(f) = b - a\log f \qquad (1)$$

これはジップの法則☆と呼ばれる．

以上の議論から，テンプレート発見問題を以下のように定義する．

定義 1 テンプレート発見問題とは，未知のパターン p からある f と $V(f)$ がベキ分布に従った代入により生成されたサンプル S が与えられたとき，p のテンプレートを探す問題である．

パターン言語の学習は，サンプルに対し極小であるようなパターンを探す問題である[2]．パターン p がサンプル S に対し**極小**であるとは，$S \subseteq L(p)$ かつ $S \subseteq L(q) \subset L(p)$ となるパターン q が存在しないときにいう．一般に，S を固定しても，複数の極小なパターンが存在する．たとえば，$S = \{abcabc, bcbca\}$ に対し，$p_1 = xbcybcz$ と $p_2 = xbcay$ はともに極小なパターンである．この場合，S 中の語が p_1 の p_2 どちらにより生成されたかは判断できない．仮に p_1 が S を生成したパターンであるとすると，(bc, bc) が定数文字列の列である．代入により S のどちらの要素にも，bc の後に a が代入されたため bca がすべての要素に共通となる．しかし，仮に $|S|$ と $|\Sigma|$ が十分大きければ，このような代入が起きる可能性は低くなる．

また，現実的な情報抽出の場合を考えると，定数部分はある程度の長さを持っている．そのため，偶然，定数文字列とまったく同じ文字列が変数に代入される可能性も低い．さらに，情報抽出の対象として HTML や XML を考えた場合，コンテンツ部分には “<” や “>” は含まれないので，そもそも定数文字列と同じ文字列が代入されることがない．このような理由から，上述のような短い例ではなく，現実的な問題では S を生成したパターンのテンプレートを一意に決定できると期待できる．よって，極小なものを探す問題ではなく，S を生成したパターンのテンプレートを探す問題と定義する．

4. 部分文字列増幅法によるパターン発見

本章の最終的な目的は，テンプレート発見問題を線形時間で高確率で解くことができることを示すことである．

ベキ分布に従うと仮定した $V(f)$ は，S 中の異なる部分文字列の数である．一般に，ある文字列 w 中の異なる部分文字列の数は，文字列の長さを固定しても，文字の現れ方により変化する．一方，w の長さが決まれば部分文字列の総出現数は文字の現れ方に依存せず一意に決まる．つまり，文字列が実際に与えられないと異なる部分文字列を見積もることは難しいが，総出現数は文字列の長さから見積もることができる．本章では，f 回出現する部分文字列の総出現数 $F(f)$ について考察する．

まず，異なる頻度 f の数を見積もる．

補題 1 サンプル S 中の文字列の長さの和を n とおく．S に現れる部分文字列の異なる頻度の数はたかだか $O(n)$ である．

証明：接尾辞木の 1 つのノードが 1 つの頻度に対応

☆ 本来は単語に対して知られる法則であるが，本論文では，これを部分文字列に拡張して用いることにする．

する．ノードの数は $O(n)$ 個なので，たかだか $O(n)$ 種類の頻度しか存在しない． □

次に，$V(f)$ がベキ分布に従うとき，総出現数 $F(f)$ の変化を見積もり，漸近的に一定となることを示す．

補題 2 異なる部分文字列の数 $V(f)$ が式 (1) を満たすとき，f 回出現する部分文字列の総出現数 $F(f)$ に対し，$F(f)/F(f-1) = (1-1/f)^{a-1}$ が成り立つ．

証明：式 (1) より

$$\log V(f) = b - a\log f$$
$$\log f + \log V(f) = b + (1-a)\log f$$
$$\log fV(f) = b + (1-a)\log f$$

が得られる．ここで $F(f) = fV(f)$ であるから

$$\log F(f) = b + (1-a)\log f$$

となる．$a = 1$ の場合は，頻度 f によらず $F(f)$ が一定となるため $a \neq 1$ としてよい．

上式において $A = a-1$，$B = b$ とおくと

$$F(f) = Bf^{-A}$$

となり，頻度 f と $f-1$ における総出現数の比は以下のようになる．

$$\frac{F(f)}{F(f-1)} = \frac{(f-1)^A}{B}\cdot\frac{B}{f^A} = \left(\frac{f-1}{f}\right)^A = (1-1/f)^A$$

つまり，$V(f)$ がベキ分布に従っているときは，f の値が大きくなるつれ，比は一定に近づく． □

補題 2 は，S を生成したパターンの変数に代入される文字列中の部分文字列の総出現数に関するものである．一方，パターンの定数文字列はすべての S の要素に共通なので，定数文字列中に出現する部分文字列の出現頻度はベキ分布から乖離したものとなる．このことは，以下のように確認できる．

テンプレートを $t = \{w_0, \ldots, w_m\}$ とすると，各 w_i に出現する部分文字列の総出現数は少なくとも $O(l^2)$ である．ここで l は t 中で最も短い w_i の長さである．簡単のために w_i に現れるすべての文字は異なる場合を考えると，w_i もその部分文字列もすべてちょうど $|S|$ 回出現する．よって，$F(|S|)$ は少なくとも $O(m|S|l^2)$ となる．一方，$|S|+1$ あるいは $|S|-1$ 回出現する部分文字列は w_i の部分文字列ではないので，$F(|S|+1)$ や $F(|S|-1)$ はベキ分布に従った値となる．したがって，サンプルの個数 $|S|$ や定数文字列の個数 m，最も短い定数文字列の長さ l が十分大きければ，$F(|S|)$ の値は $F(|S|+1)$ や $F(|S|-1)$ と比べて際立って大きいものとなる．

たとえば，Web 上の新聞記事サイトから取得した

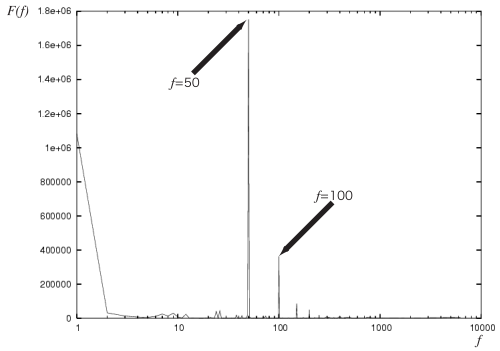


図 2 $F(f)$ におけるピーク値の存在
Fig. 2 Peaks in $F(f)$ graph.

HTML ファイル 50 個☆を S として，横軸に対数スケールで f を，縦軸に $F(f)$ をプロットすると**図 2** のようになった．実際のデータでは，定数文字列中に同じ部分文字列が存在するから，$f = |S|$ だけでなく，その定数倍のところにもピークが存在することが分かる．定数倍のところのピークの分だけ $f = |S|$ におけるピークの高さは低くなるが，それでも他の部分の $F(f)$ の値との違いは明白である．

これらの観察から，以下のようにピークを定義する．

定義 2 頻度 f に対し，$V(f)$ の変化を $G(f) = F(f)/F(f-1)$ と定義し，$G(f)$ が最大となる $f = f_p$ を**最大ピーク**，$f = cf_p$ $(c = 1, 2, \ldots)$ となる f を**ピーク**と呼ぶ．

ピークを構成する部分文字列からテンプレートが構成できるのではないかと期待できる．ピークは，同じ回数出現する文字列の全部分文字列の総出現回数を足しあわせて得られる．その意味で，$F(f)$ によるテンプレートの発見手法を**部分文字列増幅法**と呼ぶことにする．

図 3 に，テンプレート発見問題に対するアルゴリズムを示す．入力はサンプル S であり，出力は文字列の集合である．この出力は，後に示すように，高確率で S に対し極小なパターンのテンプレートとなる．

アルゴリズムは，まず `Count(`S`)` を呼び出す．これは S から接尾辞木を構成し，接尾辞木のすべての分岐語の頻度をカウントする．次に，頻度 f をキーに持ち，ちょうど f 回出現する分岐語の集合を値として持つハッシュ V を構成する．

次に $V(f)$ から $F(f)$ を計算する．このために $F(f) = f \cdot |V(f)|$ を計算するが，これだけだと f

☆ 空白文字やタグの削除も含めて，いっさいの前処理は行っていない．

```
function Template(var S:sample):
    set of strings
  var
    V, F: hash table;
  begin
    V:=Count(S); { 分岐語の頻度をカウント }
    for f in keys(V); { すべての頻度に対し }
        F(f):=0;
        for w in V(f);
          F(f) = F(f) + f|w|;
        end;
    end ;
    f_p:= FindPeaks(F);
    return(V(f_p));
  end ;
```

図 3 部分文字列増幅法を実装したテンプレート発見アルゴリズム
Fig. 3 Template discovery algorithm which implements the substring amplification.

回出現する分岐語の個数しかカウントされない．よって，まず $F(f) := 0$ と初期化し，各分岐語 $w \in V(f)$ に対し，分岐語と等価な接頭辞の出現回数は等しいので，$F(f) = F(f) + f \cdot |w|$ を計算する．

次に FindPeaks(F) は，すべての $f \geq 2$ について計算し，最大ピークとなる f_p を返す．最後に，S 中に f_p 回出現する分岐語の集合を $V(f_p)$ を返す．

次に，アルゴリズムの正当性を示す．アルゴリズム Template は最大ピークとなる f_p に対し，f_p 回出現する分岐語を出力する．よって，代入される文字列によってはアルゴリズム Template は誤って違う文字列を定数文字列として出力したり，本来定数文字列として出力しないといけない文字列を見落したりする可能性がある．そこで，文字ごとの出現確率が与えられたときの誤り率を見積もる．

文字ごとの出現確率として，たとえば，Σ の各文字が等しい確率で出現する場合が最も単純であるが，この場合は文字列の出現がベキ分布に従わないことが知られている．一方，英語については，経験的に文字ごとの出現もベキ分布に従うことが知られている[20]．さらに，Σ 中の文字の出現がベキ分布に従うときは，文字列の出現がベキ分布に従うので，以下の定理では，文字の出現は独立であり，かつ，ベキ分布に従うと仮定して誤り率を見積もる．

定理 1 与えられたサンプル S に対し，S を生成したパターンのテンプレートを t_0 とする．また，t_0 中の文字列は互いに異なるものと仮定する．図 3 のアルゴリズム Template は高い確率で正しく t_0 を出力する．誤り率はたかだか $1/c^{rk} + 1/c^{l_0} + 2/c^{|S|}$ 以下である．ただし，S は入力サンプル，l_0 は S を生成したパターン中の最も短い定数文字列の長さであり，r と k は $m_0|S|l_0^2/F(|S|-1) < rk^2/F(r-1)$ を満たし，$0 < 1/c < 1$ は Σ 中の文字に割り当てられた最も高い出現確率である．

証明：$t = \{w_0, \ldots, w_m\}$ をアルゴリズム Template が返す文字列の集合とする．また，$t_0 = \{w_0^0, \ldots, w_{m_0}^0\}$ とする．

Template が誤って t_0 の要素でないものを出力するのは，本来 $|S|$ が最大ピークであるべきなのに，それ以外を最大ピークと判断する場合と，最大ピークは正しく判別できたが，代入された文字列の中にちょうど $|S|$ 回出現する文字列がある場合である．後者の確率は $1/c^{|S|}$ である．

一方，最大ピークの誤判定は，代入された文字列の中に長い文字列が頻繁に出現する場合と，t_0 中の文字列に同じ文字列が出現する場合の 2 通りがある．後者の場合は，すべての i に対し $w_i^0 = uu$ となり，$|S|$ ではなく $2|S|$ を最大と誤判定する可能性がある．u の長さを k とすると，w_i^0 中の部分文字列の総出現数は $2k(2k-1)/2 = 2k^2 - k$ である．一方，u 中の部分文字列の総出現数は $k(k-1)/2$ であり，これが 2 回出現するので，トータルで u の部分文字列がなす $F(f)$ の高さは $k(k-1)$ である．よって，この場合は問題なく w_i^0 の部分文字列がなす $F(f)$ の値が大きく，ピークの誤判定は生じない．

一方，前者の場合の誤り率は，代入された文字列の中に長さ k の同じ部分文字列 w^k が r 回出現する確率である．この確率は $P^r(w^k) = 1/c^{rk}$ と書ける．ただし，r が最大ピークとならないといけないので，$G(|S|) = F(|S|)/F(|S|-1) < F(r)/F(r-1) = G(r)$ となる必要がある．w^k 中の部分文字列の総出現数は少なくとも $O(rk^2)$ であるので，$F(r) = O(rk^2)$ である．よって，

$$\begin{aligned} &G(|S|) \\ &\quad = F(|S|)/F(|S|-1) < F(r)/F(r-1) \\ &\quad = G(r)m_0|S|l_0^2/F(|S|-1) < rk^2/F(r-1) \end{aligned}$$

を満たす必要がある．

次に，t_0 の要素をすべて検出できているか考える．t_0 の要素 w_i^0 を見落とす可能性がある場合は，w_i^0 そのものが代入された文字列の中に含まれる場合と，w_i^0 に隣接して同じ文字がすべての S の要素に代入される場合がある．

まず，前者の場合，$j \neq i$ に対しては w_j^0 はちょうど $|S|$ 回出現し，上述のような別のピークが存在しなければ，$|S|$ が最大ピークとなる．しかし，w_i^0 は $|S|+1$ 回出現するので，出力された t に w_i^0 は含まれない．この誤り確率はたかだか $P(w_i^0) = 1/c^{l_0}$ で

ある．

次に，長さ k の文字列 w^k が各 w_i^0 に連接して出現する場合は，$w_k = w^k w_i^0$ となる．この場合，w^k がなすピークはそのまま w_i^0 がなすピークと重なり，これが最大となる．隣接しているので，k は 1 文字でも最大ピーク（の一部）となる．この場合はすべての w_i^0 は w_j に含まれるが，$w_i^0 = w_j$ ではない．よって，この場合の誤り率は，同じ文字が w_i^0 の前（または後）に $|S|$ 回出現する確率である．この確率はたかだか $1/c^{|S|}$ である．

以上よりアルゴリズム `Template` はテンプレート発見問題を誤り確率 $1/c^{rk} + 1/c^{l_0} + 2/c^{|S|}$ 以下で解く．ただし，r と k は $m_0|S|l_0^2/F(|S|-1) < rk^2/F(r-1)$ を満たす必要がある．$|S|$，l_0，$|\Sigma|$ が十分大きければ，誤り確率は 0 に近づく． □

上記の証明では文字の出現は独立であることのみを利用しているので，実際の文字の出現確率が一様であってもベキ分布であってもかまわない．

次にアルゴリズムの計算量を見積もる．

定理 2 アルゴリズム `Template` の時間計算量は $O(n)$ 時間である．ただし，n はサンプル S 中の文字列の長さの和である．

証明：サブルーチン `Count()` は入力文字列から接尾辞木を作り，そのノードを巡回し，分岐語の総出現数を数える．接尾辞木は $O(n)$ 時間で構成でき[12]，ノード数は $O(n)$ 個であり，総出現数のカウントも $O(n)$ 時間でできる．

$F(f)$ の計算は `for` ループが二重になっているが，全部で分岐語の数だけ $F(f) = F(f) + f \cdot |w|$ を計算すればよい．よって，これも $O(n)$ 時間である．

`FindPeaks()` はすべての頻度 f に対して，その $F(f)/F(f-1)$ を計算し，最大となる f を計算する．補題 1 より，頻度の数は $O(n)$ 個であるので，このサブルーチンは $O(n)$ 時間で動く． □

5. 実　　験

本章では，まず，仮定したベキ分布の妥当性について Web 上から取得したファイルで検証する．次に，同様のデータを用いて，部分文字列増幅法がテンプレートを見つけられることを示す．

テンプレート発見問題は，単一の正則パターンから生成されたものが入力される．しかし，実際にはノイズが混入したり，また，より複雑なパターンから生成されたりしている場合もある．そこで，次に，複数のパターンの和から生成された入力，パターンとは無関係なファイルなどが大量に混入している場合でもテンプレート部分を示すピークが見つけられること示す．

より正確には，正則パターンに対し，正規表現の演算の一部である和と繰返しを導入し，この拡張されたパターンから生成されたサンプル S から定数部分を発見する．機械学習とは異なり，元のパターンそのものを見つける問題ではなく，さらに定数部分がどのような順序で現れるのかどうかにも関知しないことに注意する．

複数のパターンが許されるので，これらの実験では最大ピークのみでなく，2 番目以降のピークも用いる．最大ピークは一意に定まるが，2 番目以降のピークは，テンプレートを共有する文字列の数や，そのテンプレート内の文字列の長さによりその高さは変化する．本章の実験では，何番目までのピークを実際にピークと見なすかは筆者の判断とした．

5.1 ベキ分布の妥当性

テンプレート発見問題では，変数への代入はベキ分布に従うと仮定した．本節では，テンプレートを持たない文書を対象に，頻度 f と $V(f)$ の関係を調べる[18]．**図 4** が頻度 f を持つ異なる部分文字列の数 $V(f)$ のグラフである．(a) が日本語の小説で，(b) が英語の小説である．使用言語に関係なく $\log V(f)$ と $\log f$ はほぼ線形の関係となり，ベキ分布に従うことが分かる．

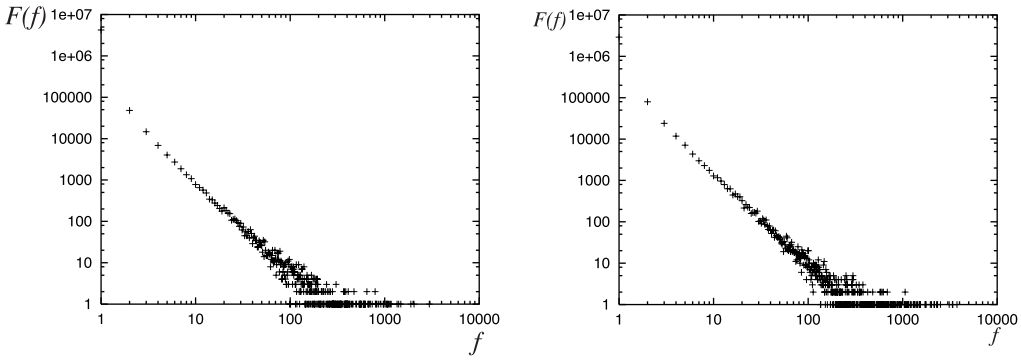
対象を英単語に限った場合のベキ分布はジップの法則として知られるが，図 4 では長さ 50 までのすべての部分文字列分を足しこんでいる．このように長さに関して $V(f)$ を足し合わせても，ベキ分布は保存されていることが分かる．

5.2 単一フォーマット

本節では，1 つのテンプレートから作成されたと考えられるファイルを対象とした実験を示す．入力は，産経新聞のサイト☆から取得した新聞記事の HTML ファイル 50 個であり，全部で 526 Kbyte であった．これを与えたときの $F(f)$ のグラフは図 2 のようになった．

$f = 50$，100，150，200 と 50 おきにピークがあることが分かる．また，最大ピークは $f = 50$ のときである．これより，入力として与えた 50 ファイルの各ファイルにはテンプレートがあり，このテンプレートを構成する部分文字列が 50 回出現していることが分かる．実際，ちょうど 50 回出現する部分文字列，つまり，最大ピークを構成している分岐語がどのように使われるか調べてみると，ほぼテンプレート部分と一

☆ http://www.sankei.co.jp/



(a) 日本語の小説 (b) 英語の小説

図 4 頻度が f である部分文字列の異なり数 $V(f)$ のグラフ
Fig. 4 Graphs of $V(f)$ for the number of different substrings with frequency f.

致することが分かった．

単なるタグの部分をテンプレートとしているわけではなく，タグ以外の部分でもテンプレートとなりうる．産経新聞では，記事のカテゴリに応じて，title タグに “Sankei-itimen” や “Sankei-international” などと表示される．この中のハイフン（“-”）より左側はすべての記事に共通であり，右側が記事によって多少違う．そのため，“Sankei-” は共通な定数文字列と判断されたが，右側はそうではなかった．

$f = 50$ の最大ピーク以外に，50 おきに小さくなりながらピークが出現しているのは，テンプレート部分に同じ文字列が含まれているからである．たとえば，title タグの場合 “<title>” や “</title>” という文字列は，各ファイルに一度しか現れないが，“title” という文字列は各ファイルに 2 回ずつ現れている．そのため，“title” はちょうど 100 回出現することになるが，このような文字列は，50 回出現する文字列より短いため，より小さなピークとなって現れている．

5.3 出現範囲が異なる複数フォーマット

次に，Yahoo!☆の検索結果を対象に実験を行った．Yahoo! に限らず，検索結果のページが複数ある場合は，意味の異なる（少なくとも）2 種類の共通な定数文字列が存在すると予測される．1 つは各ファイルごとに共通であり，もう 1 つは各検索結果ごとに共通である．後者は 1 ファイル中に複数回現れる．この実験は，このような入力に対して，どのようなグラフが得られるかを確かめるものである．

図 5 が Yahoo! の検索結果 46 ファイル（1,212 Kbyte）を入力とした $F(f)$ のグラフである．検索結果を取得するために，検索語として Yahoo! のディレクトリのカテゴリ名を使った．すべてのカテゴリ名からランダムに 50 個抜きだし，1 つのカテゴリ名を検索語として与え，検索結果の最初のページを取得した．ただし，検索結果のうち 4 つは取得に失敗したため，結果的に 46 ファイルが集まった．各ファイルには，基本的に 20 件の検索結果が存在するが，14 件しかないものと 19 件しかないものが存在したため，検索結果の総数は 913 件であった．

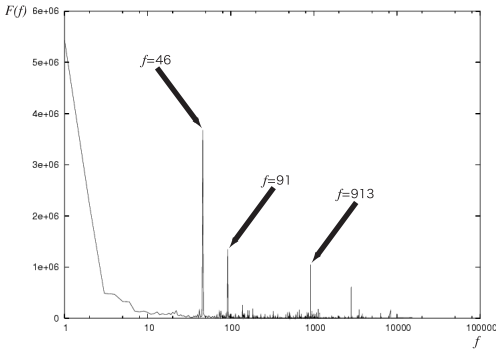


図 5 Yahoo! の検索結果の $F(f)$ グラフ
Fig. 5 f vs. $F(f)$ graph for search result pages of Yahoo!.

図 5 から，ファイル数と同じ $f = 46$ のときに最大ピークが存在することが分かる．また，ちょうど 2 倍の $f = 92$ ではなく $f = 91$ のときにピークが見つかった．これは，ほぼすべてのファイルに同じ文字列が 2 回ずつ出現したが，たまたま 1 つのファイルではそうではなかったためと予想した．しかし，検出された共通な定数文字列を確認すると，当初は予想しなかったまったく別の共通な定数文字列が存在することが分かった．Yahoo! の検索結果には，検索語にマッチした Web ページだけでなく，検索語にマッチした Yahoo! のカテゴリ名も表示される．このマッチした

☆ http://www.yahoo.com/

カテゴリの総数が 91 であり，この定数文字列がピークをなしていた．

さらに，図 5 には検索結果の数と同じ $f = 913$ のときにもピークが存在する．このように出現周期が異なる定数文字列が複数存在していても，すべての定数文字列を見つけられることが分かる．これらの定数文字列の数は任意でよい．このことは，部分文字列増幅法が，パターン言語に繰返し演算 $(\cdot)^+$ が含まれていたとしても対応可能であることを示している．

5.4 複数フォーマット

次は，3 つの異なるテンプレートから独立に生成されたファイルが混在する場合について実験を行った．実験は，先の産経新聞 50 ファイルに加え，朝日新聞☆104 ファイル（3,412 Kbyte），読売新聞☆☆140 ファイル（2,324 Kbyte）を同時に与えて行った．産経新聞のファイル数に対し，朝日新聞のファイル数はおよそ 2 倍，読売新聞のそれはおよそ 3 倍になっていて，全体で見ると産経新聞のファイル数は 1/6 程度である．

これは，異なるパターンから生成されたサンプルの和集合を与えた場合にどうなるかを調べるための実験である．これに対する $F(f)$ のグラフが **図 6** である．各新聞社のファイル数に応じて，3 つのピークが存在する．したがって，複数のパターンから生成された語が混在していても，さらに，ファイル数が大きく異なっていても，すべての定数文字列を見つけることができることが分かる．このことは，パターン言語の和集合からサンプルが与えられた場合でも，部分文字列増幅法によりすべてのテンプレートが発見できることを示している．このとき，サンプルを生成したパターンの数をあらかじめ知っておく必要はない．

さらに，図 6 では，予想していなかったピークも観測された．$f = 49$ と $f = 103$ のピークである．当初，これらはそれぞれ産経新聞と朝日新聞のファイル数 50 と 104 に近いため，これらの新聞社のテンプレートで，1 ファイルにだけ存在しないものが見つかったのかと予想した．しかし，これらのピークをなす分岐語を見てみると，たとえば $f = 49$ のほうは，たまたま 49 回出現する文字列が朝日新聞と読売新聞に独立にあり，これらがある程度大きなピークを形成していた．つまり，偶然朝日新聞と読売新聞で同じ回数ではあったが，それぞれに内容が異なるテンプレートを予期せず発見することができた．

5.5 ノイズを含んだデータ

最後の実験では，九州大学のトップページからリンクを 3 段階たどって取得したファイル 598 個（5,584 Kbyte）を対象にした．大学の Web ページは，それぞれの学部や学科，あるいは研究室や個人が独立に作っており，特に共通なテンプレートが存在しないことが多い．仮に，Web ページの制作単位で共通なテンプレートを使用したとしても，大学内のページ数からいうと少数にとどまり，大きなピークは見つからないだろうと予測していた．

図 7 が実験結果である．図には，予想に反して高いピークが存在する．これは，いくつかの部署において，大学のトップページをコピーして使用しているところがあったためである．テンプレートを持つファイル数は，入力ファイルの 1 割程度しかなくても，十分ピークが確認できる．

同じサイトであれば，URL の一部が同じであったり，同じディレクトリにあったりするファイルが同じテンプレートで作成してあることが多い．しかし，このような手法で同一テンプレートを持つファイルを見

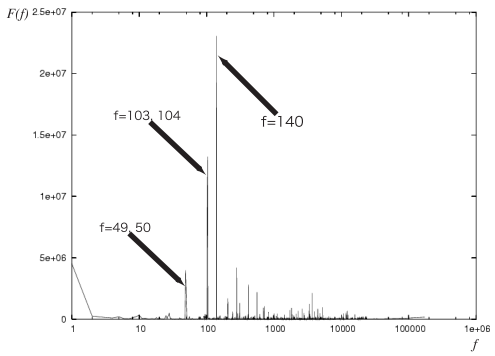


図 6 3 社混合の $F(f)$ グラフ
Fig. 6 f vs. $F(f)$ graph for Web pages from 3 online news sites.

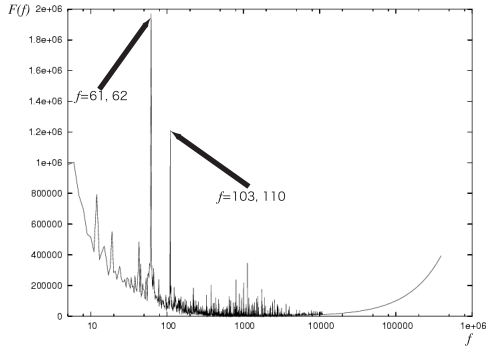


図 7 九州大学内の Web ページの $F(f)$ グラフ（ただし $f \geq 5$）
Fig. 7 f vs. $F(f)$ graph for Web pages in Kyushu Univ., where $f \geq 5$.

☆ http://www.asahi.com/
☆☆ http://www.yomiuri.co.jp/

つけた場合，見つけたものが同一テンプレートを持つ**すべて**であるかどうかは保証できない．しかし，部分文字列増幅法は同一頻度を持つ文字列から定数文字列を構成するので，もれなくすべての同一テンプレートを持つファイルが検出可能である．

6. まとめ

本論文では，未知のパターンから生成された文字列が複数与えられたときに，パターンの定数文字列を発見する問題をテンプレート発見問題として定式化した．与えられる入力はベキ分布に従う自然な代入がなされるものと仮定した．また，日本語と英語の小説を用いて，この仮定が妥当なものであることを示した．

テンプレート発見問題を解く手法として，定数文字列と代入された文字列中の部分文字列の頻度分布の差を利用して，高速に共通部分を特定する手法を提案した．単に頻度を数えるだけでは，5 章の各グラフで示したような高いピークが得られない．このことは，現実のデータでノイズが混入した場合などに，ピークがノイズに埋もれてしまうことを意味する．また，単に頻度を数える場合は，どの長さの文字列を数えるかが問題となる．本論文では部分文字列増幅法と呼ぶすべての長さの部分文字列の頻度を足し合わせる手法で，高いピークを構成可能とした．また，長さを固定しなくてよいので，データに関する事前の知識が不要である．

この手法を用いて，テンプレート発見問題が $O(n)$ 時間で解けることを示した．しかし，定数文字列に現れる部分文字列が偶然コンテンツ部分に代入されたら誤検出が生じる．コンテンツ部分を抽出すべき箇所として，適合率と再現率などで抽出の精度を実験的に評価することは今後の重要な課題である．

Web 上のテンプレートを持つ入力ファイルに対する実験を行い，実際のデータでの有効性を示した．これにより，テンプレートを持たない多数のファイルに埋もれたテンプレートであっても，部分文字列増幅法により問題なく発見可能であることが実証された．

さらに，繰返しや和集合といった正規表現の一部が用いられている場合にも，部分文字列増幅法はテンプレートを発見できることが実証された．正則パターン言語のクラスは正規言語のクラスの真の部分クラスであり，言語の表現能力は正規表現より小さい[2),16)]．XML の理論的な背景の 1 つとして正規表現は重要である[6),13)] ことを考えると，部分文字列増幅法が繰返しや和集合で表される入力も扱えることは，今後の応用面で非常に重要である．しかし，和や繰返しの場合には複数のピークを見つける必要があり，どの高さまでをピークと考えるかの厳密な定義が必要である．このことは，定理 1 において，入力文字列を生成したパターン中の定数文字列は互いに異なるものと仮定したこととも関係する．仮に同じ文字列がある場合は，（ノイズがなければ）最大ピークとなる f_p の定数倍にピークが現れるので，これらを調べればよい．しかし，本論文では，実際に何倍までをピークと判断するかの基準は示していない．

提案手法が探す共通な部分は部分文字列の集合で表される．部分文字列に対し，不一致を許すように拡張することは今後の重要な課題である．これにより，ゲノム情報処理の分野などでよく用いられる $[AC]$ のように複数の文字選択が可能になる．

参考文献

1) Agrawal, R., Imielinski, T. and Swam, A.: Mining Association Rules between Sets of Items in Large Databases, *Proc. 1993 ACM SIGMOD International Conference on Management of Data*, pp.207–216 (1993).
2) Angluin, D.: Finding Patterns Common to a Set of Strings, *Journal of Computer and System Sciences*, Vol.21, pp.46–62 (1980).
3) Arasu, A. and Garcia-Molina, H.: Extracting Structured Data from Web Pages, *Proc. 2003 ACM SIGMOD International Conference on Management of Data*, pp.337–348 (2003).
4) Asai, T., Arimura, H., Uno, T. and Nakano, S.: Discovering Frequent Substructures in Large Unordered Trees, *Proc. 6th International Conference on Discovery Science*, Lecture Notes in Artificial Intelligence 2843, pp.47–61, Springer-Verlag (2003).
5) Han, J., Pei, J. and Yin, Y.: Mining Frequent Patterns without Candidate Generation, *Proc. 2000 ACM SIGMOD International Conference on Management of Data*, pp.1–12 (2000).
6) Hosoya, H. and Pierce, B.C.: Regular Expression Pattern Matching for XML, *Proc. 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pp.67–80 (2001).
7) Ikeda, D., Yamada, Y. and Hirokawa, S.: Eliminating Useless Parts in Semi-structured Documents Using Alternation Counts, *Proc. 4th International Conference on Discovery Science*, Lecture Notes in Artificial Intelligence 2226, pp.113–127, Springer-Verlag (2001).
8) Kearns, M. and Pitt, L.: A Polynomial-Time Algorithm for Learning k-variable Pattern Lan-

guages from Examples, *Proc. 2nd Annual Workshop on Computational Learning Theory*, pp.57–71 (1989).
9) Kushmerick, N., Weld, D.S. and Doorenbos, R.B.: Wrapper Induction for Information Extraction, *Proc. 15th Intenational Joint Conference on Artificial Intelligence*, pp.729–737 (1997).
10) Kushmerick, N.: Wrapper Induction: Efficiency and Expressiveness, *Artificial Intelligence*, Vol.118, pp.15–68 (2000).
11) Maier, D.: The Complexity of Some Problems on Subsequences and Supersequences, *J. ACM*, Vol.25, pp.322–336 (1978).
12) McCreight, E.M.: A Space-Economical Suffix Tree Construction Algorithm, *J. ACM*, Vol.23, No.2, pp.262–272 (1976).
13) Murata, M.: Extended Path Expressions for XML, *Proc. 20th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pp.126–137 (2001).
14) Nagao, M. and Mori, S.: A New Method of N-gram Statistics for Large Number of n and Automatic Extraction of Words and Phrases from Large Text Data of Japanese, *Proc. 15th International Conference on Computational Linguistics*, pp.611–615 (1994).
15) Schapire, R.E.: Pattern Languages Are Not Learnable, *Proc. 3rd Annual Workshop on Computational Learning Theory*, pp.122–129 (1990).
16) Shinohara, T.: Polynomial Time Inference of Extended Regular Pattern Languages, *RIMS Symposium on Software Science and Engineering (1982)*, Lecture Notes in Computer Science 147, pp.115–127, Springer-Verlag (1983).
17) Yamada, Y., Ikeda, D. and Hirokawa, S.: Automatic Wrapper Generation for Multilingual Web Resources, *Proc. 5th International Conference on Discovery Science*, Lecture Notes in Computer Science 2534, pp.332–339, Springer-Verlag (2002).
18) 山田泰寛，池田大輔，廣川佐千男：構造的類似性を持つ半構造化文書における頻度分析，第 2 回情報科学技術フォーラム一般講演論文集 2 分冊，pp.59–60 (2003).
19) 中村貞吾：着手記号列の出現頻度に基づく囲碁棋譜からの定型手順獲得，情報処理学会論文誌，Vol.43, No.10, pp.3030–3036 (2002).
20) 長尾　眞，佐藤理史，黒橋禎夫，角田達彦：自然言語処理，15，岩波書店 (1996).

(平成 15 年 11 月 14 日受付)
(平成 16 年 1 月 9 日再受付)
(平成 16 年 1 月 22 日採録)

池田　大輔（正会員）

昭和 46 年生．平成 8 年九州大学大学院総合理工学研究科情報システム学専攻修士課程修了．平成 9 年九州大学大学院システム情報科学研究科情報理学専攻博士後期課程中退．同年 4 月より九州大学大型計算機センター助手．九州大学情報基盤センター助教授を経て，現在，九州大学附属図書館助教授．ウェブマイニングの研究に従事．ACM，EATCS 各会員．博士（理学）．

山田　泰寛（学生会員）

昭和 53 年生．平成 15 年九州大学大学院システム情報科学府情報工学専攻修士課程修了．現在，九州大学大学院システム情報科学府情報理学専攻博士後期課程在学中．ウェブマイニングの研究に従事．日本学術振興会特別研究員，修士（工学）．

廣川佐千男（正会員）

昭和 29 年生．昭和 54 年九州大学大学院理学研究科修士課程修了．静岡大学工学部情報工学科助手，九州大学教養部助教授，九州大学理学部助教授，九州大学大学院システム情報科学研究科教授を経て，現在，九州大学情報基盤センター教授．リンク情報による WWW 空間の解析，半構造化データからのデータマイニング，および型理論の研究に従事．電子情報通信学会，人工知能学会，ソフトウェア科学会，数学会，ASL，EATCS 各会員．博士（理学）．