部分文字列増幅法による共通パタン発見アルゴリズム

池田, 大輔
九州大学情報基盤センター

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：情報処理学会研究報告 ： 数理モデル化と問題解決. 2003 (122), pp.45-48, 2003-12. 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ン発見アルゴリズム

池田大輔[†]　山田泰寛[††]　廣川佐千男[†]

複数の文字列に共通な部分列を見つける問題をテンプレート発見問題として定式化する．テンプレート以外の文字列の頻度分布はベキ分布に従うことを仮定する．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，(1) 問題の再定式化，(2) 部分文字列の集合によるテンプレート表現，(3) 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を平均的にほぼ入力長に線形で解くアルゴリズムを構築する．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した．

A Pattern Discovery Algorithm by Substring Amplification

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

We define the problem to find a subsequence common to given strings as the *template discovery problem*. We assume that the frequency distribution of substrings on non-template parts follows the power-law distribution. Although the longest common subsequence problem is well-known to be NP-complete, we show that the template discovery problem can be solved in almost linear in the total length 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

共通する部分列の中で最も長い部分列を探す問題は最長共通部分列問題と呼ばれ，NP 完全であることが知られている．この問題では，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が存在しているものを仮定している場合が多い．

共通でない部分は，例えば，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 [2),4),6)] 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．これらはジップの法則などベキ分布に従うことが経験的に知られている．

以上の考察から，共通な部分列を探す問題として**テンプレート発見問題**が自然に定式化できる．これは，未知のパタンから生成された複数の文字列が与えられた時に，パタンの定数部分である共通な部分列を探す問題である．変数を置きかえた部分の部分文字列の出現はベキ分布に従うものとする．

本論文では，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，テンプレートとなる文字列はある程度の長さを持っていなければならない．このとき，計算時間は $O(nm|t|^2 \log t)$ である．ここで，n は総入力長，m は定数文字列の個数，$|t|$ は最も短い定数文字列の長さである．n と比較すると m や t は非常に小さく，現実的には入力長の線形に近い．

この計算量でのテンプレート発見を実現するために，(1) 部分列をさらに短い文字列の集合で表わす手法 [3)] と，(2) 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**という手法を用いたアルゴリズムを提案する．このアルゴリズムは，代入される文字列がベキ分布に従うことを利用しており，平均的に正しくテンプレートを発見できる．

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

† 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

パタンが含まれる場合，複数の異なるパタンから生成された文字列の和集合の場合についても，テンプレートを発見できることを示した．

2. テンプレート発見問題

Σ を有限アルファベット，$V = \{x_1, x_2, \ldots,\}$ を Σ と交わらない**変数**の集合とする．$\Sigma \cup V$ 上の文字列を**パタン**と呼ぶ [1]．各変数が高々1 回しか出現しないパタンを**正則**パタンという．変数に代入される文字列ではなく，定数部分を検出することが目的なので，以後パタンを正則なものに限定する．

パタン $p = w_1 x_1 \cdots w_m x_m$ ($w_i \in \Sigma^*$, $x_i \in V$ が与えられた時，パタン中に出現する定数文字列の列 $(w_1, \ldots, w_m)$ を**テンプレート**と呼ぶ．テンプレート中の文字列 w_i を**テンプレート文字列**と呼ぶ．テンプレートの**幅**を最も短いテンプレート文字列の長さと定義し $|t(p)|$ で表す．

V から Σ^* への写像 θ を**代入**と呼ぶ．パタン p と代入 $\theta = [x_1 := w_1, \ldots, x_m := w_m]$ に対し，$p\theta$ により，パタン中の変数 x_i を w_i で置きかえて得られる文字列を表す．パタン 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．サンプル S が与えられた時，$S \subseteq L(p)$ かつ $S \subseteq L(q) \subset L(p)$ となるパタン q が存在しない時，パタン p は S に対し**極小**であるという．

パタ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 [5]．しかし，情報抽出の場合は，テンプレート文字列でない部分には単語や文章など自然な文字列が入る．そこで，変数へ代入する定数文字列に頻度に関するジップの法則などのベキ分布を仮定する．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を $V(f)$ とすると，ジップの（第二）法則とは，ある定数 $a > 1$ と b を用いて，

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

として表わされる．

定義 1 (テンプレート発見問題) テンプレート発見問題とは，未知のパタンからあるベキ分布に従った代入により生成されたサンプル S が与えられた時，サンプルに対し極小であるようなパタンのテンプレートを探す問題である．

確率分布そのものは与えられていないことに注意する．

3. 部分文字列増幅法によるパタン発見

接尾辞木のノードの個数から次の補題が示せる．

補題 1 サンプル S に現われる部分文字列の異なる頻度の数は高々 $O(n)$ である．

補題 2 異なる部分文字列の数 $V(f)$ が式 (1) を満たす時，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．

一方，テンプレート文字列はすべての S の要素に共通なので，テンプレート文字列の部分文字列の出現頻度はベキ分布の直線から乖離したものとなる．テンプレートを $t = (w_1, \ldots, w_m)$ とすると，テンプレート文字列中に存在する部分文字列の総出現数は少なくとも $O(|t|^2)$ である．簡単のためにテンプレート文字列に現われる文字は全て異なる場合を考えると，テンプレート文字列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 $O(m|S||t|^2)$ となる．従って，サンプルの個数 $|S|$ やテンプレート文字列の個数 m，テンプレートの幅 $|t|$ 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．

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，$F(f) > (1-1/f)^{a-1}$ となる f を**ピーク**という．

$F(f)$ のピークを構成する部分文字列から共通部分列を構成すれば，高速にテンプレート発見問題が解けるものと期待できる．ピークはすべての部分文字列の数を足しあわせているという意味で，この $F(f)$ によるテンプレート文字列の発見手法を**部分文字列増幅法**と呼ぶことにする．

図 1 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 S を生成する極小なパタンのテンプレートである．

アルゴリズムは 3 つのサブルーチン `Count(S)`, `FindPeaks(F)`, `Reconstruct(V)` を呼び出す．まず始めに `Count(S)` により S 中に現われる全ての部分文字列の頻度をカウントし，頻度 f をキーに持ち，ちょうど f 回出現する異なる部分文字列の集合を値として持つハッシュ V を構成する．

`FindPeaks(F)` は $F(f)/F(f-1)$ をすべての $f \geq 2$ について計算し，ピークとなる f を P に追加していき，最終的に P を返す．実際にはベキ分布からはずれる語が存在する．そのため，実際にはなんらからのしきい値を設定する必要があると考えられるが，ここではそのしきい値は設定しない．よって，平均的にはピークをうまく見つけられるが，そうではない場合も存在することに注意する．なお，このしきい値を低くとればピークとなる頻度が増えるが，多項式時間という意味では計算量に影響しない（補題 1 参照）．

サブルーチン `Reconstruct(V, P)` はピークを構成

```
function Template (var S: sample): template
  var
    t: テンプレート;
    V,F: ハッシュ表;
  begin
    V :=Count(S);
    for f ∈ keys(V); { すべての頻度に対し }
        F(f) := f · |V(f)|;
    end ;
    P := FindPeaks(F); { ピークを取る頻度を計算 }
    t :=Reconstruct(V,P);
    return t を出力;
  end ;
```

図 1 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. 1 Template discovery algorithm which implements the substring amplification

する部分文字列をつなぎ合せて，テンプレート文字列を再構成する．与えられた全てのピーク $f_p \in P$ に対し，S に f_p 回出現する部分文字列の集合を $W := V(f_p)$ とおく．W の要素の長いものから順に，その要素 w が S の各要素に同じ回数現われているか調べる．現われていない場合は W から削除して，次の W の要素に移る．現われていたら，テンプレート候補 t の列に追加する．この時，すでに追加されているテンプレート文字列の出現位置から，追加する順序を決める．

サブルーチン Reconstruct(V,P) の動作はピークのある f_p に対してだけに行なわれる．次の補題 3 に示すように，ピークの個数は少ないので，高速な共通部分列検出が可能になる．

補題 3 テンプレート $t = (w_1, \ldots, w_m)$ を持つパタンから生成されたサンプル S が与えられた時，$F(f)$ のピーク数は高々$O(m \log |t|)$ 個である．

アルゴリズムの正当性と計算量を次の二つの定理で示す．

定理 1 与えられたサンプル S に対し，S を含む極小なパタンのテンプレートを t とする．$|S|$ や $|t|$ が十分大きければ，図 1 のアルゴリズム Template は平均的に正しく t を出力する．

定理 2 アルゴリズム Template の時間計算量は $O(nm|t|^2 \log t)$ 時間である．ピークの判別をしない場合は $O(n^2|t|^2)$ 時間である．

実際のデータにおいては，n が非常に大きく，m や $|t|$ はこれに比べると小さい．よって，実際には，Template は入力長の線形に近い速度で動く．

4. 実　　験

本節では，Web 上から取得したデータを用いて，部分文字列増幅法が単一のパタンから生成された入力だけでなく，複数のパタンの和から生成された入力，パタ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．

まず，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 (526Kbyte) を用いた実験では，f = 50, 100, 150, 200 と 50 おきにピークがあることが確認され，最も高いピークは f = 50 の時であった．ピークを構成している文字列は，ほぼテンプレート部分と一致した．

次に，Yahoo!の検索結果 46 ファイル (1212Kbyte) を対象に実験を行なった．検索結果の総数は 913 件であった．**図 2** の (a) が $F(f)$ のグラフである．

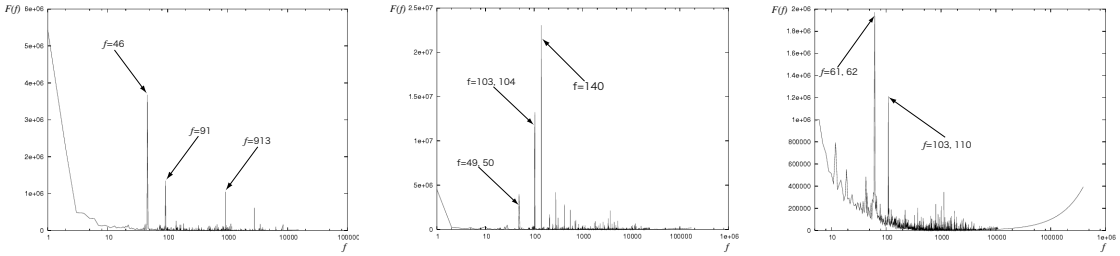
ファイル数と同じ f = 46 の時に一番大きなピークが存在することが分かる．f = 91 の時にピークは，検索語にマッチした Yahoo!のカテゴリ名の総数が 91 であったためである．

さらに，検索結果の数と同じ f = 913 の時にもピークが存在する．このように出現範囲が異なるテンプレートが複数存在していても，すべてのテンプレートを見つけられることが分かる．このことは，部分文字列増幅法が，パタン言語に正規表現でいう繰り返し演算 $(\cdot)^+$ や和の演算 $\cdot|\cdot$ にも対応可能であることを示している．

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なった．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 (3412Kbyte)，読売新聞 140 ファイル (2324Kbyte) を同時に与えて行なった．

これに対する $F(f)$ のグラフが図 2 の (b) 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．パタ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ピークが発見できることを示している．この時，異なるパタンの数をあらかじめ知っておく必要はない．

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 段階辿って取得したファイル 598 個 (5584Kbyte) を対象にした．図 2 の (c) が実験結果である．高いピークは，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 割程度しかなくとも，十分ピークが確認できた．



(a) Yahoo!の検索結果　(b) 3 社混合　(c) 九州大学 ($f \geq 5$)

図 2　実験結果の $F(f)$ グラフ
Fig. 2　f vs. $F(f)$ graph

同じサイトであれば，同じディレクトリにあ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つけても，見つけたものが同一テンプレートを持つ**すべて**という保証はない．しかし，部分文字列増幅法は同一頻度を持つ文字列からテンプレート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．

5. ま と め

未知のパタンから生成された文字列が複数与えられた時に，共通な部分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．

テンプレート発見問題を解く手法として，共通部分列とそうでない部分に現われる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．本論文では部分文字列増幅法と呼ぶ全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 $O(nm|t|^2 \log |t|)$ で解けることを示した．実際には入力長の線形時間に近い速度で動くといえる．

実際のデータでの有効性を示すために，Web 上のテンプレートを持つ入力ファイルに対する実験も行なった．これにより，テンプレートを持たないファイルが 9 割程度あったとしても，問題なくテンプレートを発見することができた．さらに，繰り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．

テンプレート文字列以外の部分の頻度分布がベキ分布に従っていると仮定したが，誤り率などは算出していない．そのため，確率的な解析によりアルゴリズムの誤り率を見積り，入力によってどのように誤り率が変化するかを調べることは今後の重要な課題である．同様に，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も重要な課題である．

提案手法が探すパタンは部分列である．部分列中の各文字列に対し，不一致を許すようにを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) Angluin, D.: Finding Patterns Common to a Set of Strings, *Journal of Computer and System Sciences*, Vol. 21, pp. 46–62 (1980).
2) Arasu, A. and Garcia-Molina, H.: Extracting Structured Data from Web Pages, *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pp. 337–348 (2003).
3) Ikeda, D., Yamada, Y. and Hirokawa, S.: Eliminating Useless Parts in Semi-structured Documents using Alternation Counts, *Proceedings of the 4th International Conference on Discovery Science*, Lecture Notes in Artificial Intelligence 2226, Springer-Verlag, pp. 113–127 (2001).
4) Kushmerick, N., Weld, D. S. and Doorenbos, R. B.: Wrapper Induction for Information Extraction, *Proceedings of the 15th Intenational Joint Conference on Artificial Intelligence*, pp. 729–737 (1997).
5) Shinohara, T.: Polynomial Time Inference of Extended Regular Pattern Languages, *RIMS Symposium on Software Science and Engineering (1982)*, Lecture Notes in Computer Science 147, Splinger-Verlag, pp. 115–127 (1983).
6) Yamada, Y., Ikeda, D. and Hirokawa, S.: Automatic Wrapper Generation for Multilingual Web Resources, *Proceedings of the 5th International Conference on Discovery Science*, Lecture Notes in Computer Science 2534, Splinger-Verlag, pp. 332–339 (2002).