Web上の多言語テキストデータからのラッパー自動生成

池田, 大輔
九州大学情報基盤センター

山田, 泰寛
九州大学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

https://doi.org/10.15017/4784011

出版情報：九州大学情報基盤センター年報. 3, pp.7-14, 2003-03. 九州大学情報基盤センター
バージョン：
権利関係：

Web上の多言語テキストデータからのラッパー自動生成
Automatic Wrapper Generation for Multilingual Web Resources

池田 大輔† 山田 泰寛‡ 廣川 佐千男*
Daisuke Ikeda† Yasuhiro Yamada‡ Sachio Hirokawa†

† … 九州大学情報基盤センター
† … Computing and Communications Center, Kyushu University
‡ … 九州大学システム情報科学府
‡ … Graduate School of Information Science and Electrical Engineering, Kyushu University

要旨 本稿では、半構造化テキストデータからコンテンツ部分を抽出するラッパーを自動生成するシステムを提案する。入力として、テキストデータ以外にコンテンツを囲む区切り文字の最初と最後に現われ得る文字の集合を与えるものとする。入力テキストに対して同種のコンテンツ(レコード)が複数回現われるものと仮定するほかは、特に背景知識等は不要であり、入力に対し全自動でラッパーの生成を行なう。システムは、コンテンツの種類(フィールド)ごとに左と右区切り文字のペアを出力する。半構造化テキストデータを単なる文字列として扱うので、入力は任意のマークアップ言語や自然言語で書かれていて構わない。様々な言語で書かれたWebページを対象とした実験によりその有効性を示す。マークアップ言語はXMLとHTMLで、4つの自然言語で書かれており、検索機能により動的に生成されたものもあれば、静的なページもある。本システムでは、XMLやHTMLのコメントやタグの属性なども通常の文字として扱うが、ある実験では、通常のコンテンツ部分だけでなく、コメントやタグの内部から有用な情報を抽出することもできた。また、タグにマルチバイト文字が含まれているようなデータでも問題なく扱える。

Abstract We present a wrapper generation system to extract contents of semi-structured documents. In addition to input documents, our system receives a set of symbols with which a delimiter string must begin or end. We assume that input documents contain instances of a record. Wrapper generation is done automatically. The system outputs a set of pairs of left and right delimiters each of which surrounds instances of a field. Our system treats semi-structured documents just as strings so that it does not depend on markup and natural languages. We show experimental results on text files markuped in HTML or XML. Contents of them are written in four natural languages. Some of them are dynamic pages, that is, produced by a search facility, and the others are static pages. In addition to usual contents, some generated wrappers extract useful information hidden in comments or tags which are ignored by other wrapper generation algorithms. Some generated delimiters contain whitespaces or multibyte characters.

1 はじめに

Web上には大量のテキストデータが存在し、その数は日々増加している。しかし、個々のファイルを見てみると、広告やナビゲーションのためのリンク(「戻る」や「トップページへ」など)や著作権に関するページへのリンクなどもあり、必ずしもデータすべてが有用とは限らない。また、同種のデータを保持する別のサイト間では、ブラウザで見た時に同じように見えたとしても、内部的に同じ構造を持つことは稀である。このため、複数の異なるサイト間に、統一的に検索を行なうということはできない。

個々のファイルの内部構造とは別に、サイトごとの同種のファイルや、検索機能つきサイトから生成され

るファイルをまとめて見ると、同じ構造を持つ項目 (**レコード**と呼ぶ) があることが多い。例えば、検索結果のページにおいては、検索結果1件が1つのレコードである。標準的な検索エンジンでは、検索にマッチしたページのタイトル、簡単なページの要旨、URL など (これらを**フィールド**と呼ぶ) が1レコードを構成していて、複数のレコードで1ページを成している。ニュース記事を提供するオンラインニュースサイトでは、各記事のファイルが1つのレコードであり、各レコードは一般に見出し、記者名、日付、本文などからなる。この場合は1レコードが1ファイルであることが多い。

サイトや検索機能ごとにレコードの記述方法は異なっているが、レコードのデータを自動的に抽出できれば、関係のないサイト間で統一的なデータ利用が可能になるであろう。サイトごとにデータを抽出するための手続きを**ラッパー**と呼ぶ。例えば、個々の自動車メーカーのサイトのラッパーを生成し、同種のデータをメーカー間で統合できれば、容易に比較検討できる。さらに、ディーラーのページから車種とその値段を含んだフィールドを抽出できれば、スペックに加え販売価格も同時に見ることができる。

Web 上のデータの膨大さを考えると、ラッパーを手動で作ることは現実的ではなく、自動的に生成することが課題となる。ラッパー自動生成の研究は盛んに行なわれており、これらの手法は主に3つに分類される。最初の手法は、トレーニングサンプルを与えて、抽出すべきコンテンツの前後に現われる規則を学習する機械学習によるものである [6, 7]。いくつかの規則の表現形式があるが、コンテンツの前後を囲む文字列のペアで規則を表わすものや、HTML の木構造のパスで規則を表わすものもある。この場合、学習アルゴリズムに抽出すべきデータはどこであるか指示した例を与える必要があるが、これは人手で作るのでコストが高い。

次の手法は、入力ファイルを HTML に固定し、HTML 文法の知識を利用する [2, 4]。[4] では、いくつかのヒューリスティクスを組み合わせて、高精度なアルゴリズムを構築している。ヒューリスティクスは、例えば、(1) 繰り返し現われるタグは区切り目である、(2)<hr>, <td>, <tr>, <a>, <p>,
 など特定のタグの直後に区切り目があるなどである。この手法ではトレーニングサンプルを用いないが、HTML 以外のマークアップ言語には適用しにくい。特に、XML のように自由にタグを定義できる場合、特定のタグの意味を考えなければならない点は致命的である。

最後の手法は、一部はさきほどの手法にも使われていたものだが、入力ファイルの規則性を探そうとするものである。例えば IEPAD [3] は、文字列の最大反復を見つけるアルゴリズムを用いてレコードの区切り目を探している。また、HTML ファイル中のリストの規則性を見つけ、データ抽出を可能にするアルゴリズムも提案されている [8]。上述した3種類のどの手法においても、HTML ファイルを入力として受けとり、HTML の文法に関する知識を利用して構文木を生成したりコメントを削除したりすることは普通に行なわれている。

本稿では、筆者らが構築したラッパー自動生成システム [9, 10] を用いて、実際のデータを用いたシステムの実験結果を紹介する。提案するシステムは、複数のレコードインスタンスを含むテキストデータが与えられると、各フィールドのコンテンツがある位置を推定し、これを囲む左右の区切り文字を出力する。本システムにおいては、区切り文字の最初と最後になりうる文字の集合は、入力として与えるものとする。例えば、区切り文字は必ずタグで始まるようにしたい場合は、この文字の集合として {<, >} を与える。これを除けば、フィールド位置の推定は、データやその周囲の文字列の規則性のみを用いるので、完全に自動的に行なうことができる。また、入力されるテキストデータを単なる文字列として扱うので、任意のマークアップ言語と自然言語で記述されていて構わない。

このシステムを用いて、マークアップ言語として HTML ファイルと XML ファイルで、コンテンツを示す言語として英語、日本語、ドイツ語、中国語を用いたファイルを入力データとした実験結果を紹介する。これらのファイルは、動的に生成されたものもあれば、そうでないものも含まれる。システムは、通常のコンテンツ部分からフィールド部分を抜きだすことに成功した。さらに、コメントやタグの内部にあり、ブラウザには表示されない部分からも、有用な情報を取り出すことができた。コメントやタグの内部は、他のラッパー生成アルゴリズムでは無視される部分である。また、XML のタグが日本語で定義されているような場合でも、問題なくラッパー生成ができることも示された。

2 アーキテクチャ

提案するラッパー自動生成アルゴリズムは、あるレコードのインスタンスを複数含む半構造化テキストデータの集合を入力として受けとり、レコードを構成するフィールドインスタンスを抽出するようなルール

(左区切り文字と右区切り文字のペア) の集合を出力する。半構造化テキストは単なる文字列として扱われるので、以下では単に文字列と呼ぶ。また、文字列の集合と共に後述する文字の集合 E_l と E_r も受けとる。図 1 がアルゴリズムの擬似コードである。

```
procedure CreateRules (var D:  文字列集合、
      E = (E_l, E_r): 区切り文字集合);
  var i,n,a:  integer;
  var R:  ルール集合;
  var r:  テキスト集合;
  begin
    n,a := FindOptimal(D); {2.2節}
    r := D の (n,a) におけるレンジの集合;
    for i := 1 to |D| do
      r[i] := D[i] の (n,a) におけるレンジ;
    end;
    R := rule(D,r,E); {2.3節}
    R 中の不要なルール候補を削除; {2.4節}
    ルールを統合;{2.4節}
    R を出力;
  end;
```

図 1: ルールを生成するアルゴリズム

アルゴリズムは、コンテンツ検出、ルール抽出、ルール統合の 3 つのステップからなる。コンテンツ検出では、各入力文字列を頻度の高い部分とそうでない部分にわける。新聞記事の HTML ファイルに対して、頻度の低い部分がコンテンツ部分とほぼ一致することが知られている [5]。このことを新聞記事以外のファイルについても利用して、以後のステップでルールとなる文字列を探す。

ルールとは**区切り文字**のペア (l, r) のことであり、それぞれ**左区切り文字**と**右区切り文字**と呼ぶ。このようなルールの集合からなるラッパーを LR ラッパーと呼ぶ [6]。ただし、各入力文字列において l と r の出現回数は等しく、左区切り文字は E_l 中の文字で終り、右区切り文字は E_r で始まるものとする。ルール抽出ステップでは、検出したコンテンツ部分を囲み、かつ、上の制約を満す文字列のペアをルール候補として生成する。

ルール候補として生成された文字列のペアは、単に左と右の出現回数が等しければ候補となりうるので、ほんの数回しか現われないものもありうる。このようなものは、本来コンテンツとは無関係かもしれないし、関係あるかもしれないが、その判断には意味的な処理が必要となる。ここでは、ルール候補が現われる回数にしきい値を設け、この回数以上あればルールとみなすことにする。つまり、例えば入力文字列の半数以上に現われるルール候補のみをルールとする。最後のステップでは、この条件を満さないルール候補を削除し、また、同じ文字列を抽出するルールを統合する。

以下、ステップ毎に説明していくが、最初にコンテンツ検出で用いる重要な概念であるレンジ文字列と交代数について述べる。

2.1 レンジ文字列と交代数

v を文字列 x の部分文字列とする。v の x における**出現**とは $x[i]\cdots x[i+|v|-1] = v$ を満たすある正の整数 i のことである。ただし、$x[i]$ で x の i 番目の文字を表わす。v の出現と長さを用いて、v を $x[i..i+|v|-1]$ と表わすこともある。文字 a に対し、$[x]_a$ で文字列 x 中の文字 a の数を表わす。

x を文字列とし、$W = \{v_1, \ldots, v_n\}$ を x の部分文字列の集合とする。x 上の W による**レンジ文字列**(単に**レンジ**とも呼び $r_x(W)$ と表記する) は以下の条件を満たす長さ $|x|$ のバイナリ文字列 ($\{0,1\}$ 上の文字列) である：v_k $(1 \le k \le n)$ の各出現 i に対し、$i \le j \le i+|v_k|-1$ を満たす j 番目の位置は 0 ($r_x(W)[j] = 0$)、それ以外の j に対しては $r_x(W)[j] = 1$ である。

レンジ上の連続する 0 は、文字列 x のどの部分が W に含まれているかを表わしている。例えば $x = accbaacbc$ と $W = \{cb, ba\}$ とした場合、$r_x(W) = 110001001$ となる。レンジの 3 文字目から 5 文字目は $r_x(W)[3..5] = 000$ となっており、このことは $x[3..5] = cba$ が W の要素である cb と ba によって覆われていることを示している。

ある文字列 x と x の部分文字列の集合 W に対する交代数とは、$r_x(W)$ において 0 と 1 が切り換る場所の数である。例えば、上述した例の $r_x(W) = 110001001$ の場合、交代数は 4 である。また、単一の文字列 x を文字列の集合 S に拡張した場合、交代数は S の各要素に対する交代数の和と定義する。

2.2 コンテンツ検出

このステップでは、サブルーチン FindOptimal [5] を用いて、各入力文字列を高頻度部分とそうでない部分に分ける。FindOptimal はラッパー自動生成システ

ムの入力文字列をそのまま受けとる。文献 [5] において、新聞記事の HTML ファイルを FindOptimal に与えた時、その中で高頻度でないと判定された部分は、コンテンツ部分の大部分と重複することが実験的に明らかになっている。

ただし、この分類は完全にコンテンツ部分と一致するわけではなく、また、入力データの種類によっては一致部分が大幅に少なくなることもある。例えば、検索結果などの動的なページを入力とした場合、FindOptimal はうまく動かないことが分かっている [5]。しかし、本稿で提案するアルゴリズムは、このような不完全な分類であっても、次のステップ以降でルール抽出が可能となっている。

また、オリジナルの FindOptimal は、入力の半構造化テキストを単なる文字列として扱うが、前処理として連続する空白文字 (半角空白、タブ、改行文字) を 1 つの半角空白に置換していた。本稿ではこの前処理も行なわず、入力されたままの半構造化テキストを単なる文字列として扱うことにする。

ここで、簡単に FindOptimal の動作を説明する。FindOptimal は入力文字列の集合を受けとり、2 つの正の整数のペア (n, a) を出力する。(n, a) はカットポイントと呼ばれ、n は文字列の長さ、a はパーセントで $1 \leq a \leq 100$ である。カットポイントは、各文字列の高頻度部分を表わしている。つまり、カットポイント (n, a) に対し、文字列の集合 S の**高頻度部分**とは、S の要素から長さ n のすべての部分文字列の頻度を数え、頻度の高いもの上位 a パーセントにはいっている部分文字列が S の要素に占める部分である。このように考えると、カットポイント (n, a) は部分文字列の集合を規定することになるので、W の代りに (n, a) を使ってレンジを考えることができる。つまり、文字列 x に対し、W を (n, a) によって決まる高頻度部分を成す部分文字列の集合とする。以後、$r_x(W)$ の代りに $r_x(n, a)$ と書くことにする。

次の問題は、与えられた文字列の集合のコンテンツ部分とうまく重複するように (n, a) を決めることである。このステップで、アルゴリズムはまず初期値 $(2, 1)$ から始め[1]、n または a のどちらか一方の値を 1 だけ増やす。現在のカットポイントを (n, a) とした場合、FindOptimal は (n, a), $(n+1, a)$, $(n, a+1)$ の 3 点における交代数を求め、最も小さい交代数を与えるカットポイントへ遷移する。もし、現在のカットポイントが最小であれば、そこで停止し現在のカットポイント (n, a) を出力する。

[1] 初期値として $(1, 1)$ という選択肢もあるが、長さが 1 の場合は単に文字の頻度になるので、少なくとも長さが 2 の部分文字列から始めることにする。

2.3 ルール抽出

本節では、ルール抽出を行なうサブルーチン rule(D, r, E) の動きを説明する。サブルーチンは、文字列の集合 D、高頻度とそうでない部分に分けられた状態であるレンジ集合 r と、区切り文字の開始と終了文字の集合 $E = (E_l, E_r)$ を受けとり、ルール候補の集合を生成する。

アルゴリズムは、高頻度ではない部分 (以下、低頻度部分と呼ぶ) をコンテンツとみなし、その前後で区切り文字のペアを探す。各低頻度部分に対し、この低頻度部分全体を囲み、かつ、この部分に最も近い区切り文字候補 l と r を探す。ただし、l は E_l の要素で終り、r は E_r の要素で始まる。

l と r が入力文字列全体で違う回数の出現を持つ場合、アルゴリズムはより頻度の高い区切り文字候補の長さを伸ばし、新たな候補とする。長さを伸ばす時は、1 文字ずつではなく、次の E_l または E_r の要素まで一度に伸ばす。これを出現回数が同じになるまで続ける。回数が同じにならなければ、次の低頻度部分の処理に移る。

区切り文字の候補 l と r が同じ文字列であるか、対応する開きタグと閉じタグ[2]の場合、アルゴリズムはどちらの長さも伸ばす。

2.4 ルール統合

R を前のステップで生成されたルール候補の集合とする。R の要素 (l, r) は、単に出現回数が同じというだけであり、複数の候補が同じインスタンスを抜きだしたり、ごく少数のインスタンスしか抜きださなかったりするものも含んでいる。そこで、このステップではこのようなルールを統合したり、削除したりする。

いくつかのレコードには、ある種類のデータが含まれない場合もありうる。例えば、名簿データの場合、何人かはメールアドレスの欄が空欄かもしれない。そこで、我々は一部の入力文字列に対して何も抜きださないルールも認め、この問題に対応することにした。しかし、このことがルール候補の削除を困難なものにしている。

削除するには何らかの基準が必要である。すべての

[2] 例えば、l = <tagA><tagB>, r = </tagB></tagA> となっている時。

入力文字列からインスタンスを抜きだせる場合は、問題なくこのルール候補はルールとしてよいだろう。しかし、上述したようにいくつかの入力文字列から抜きだされなくても問題ないとしている。そこで、ルールがどれだけの文字列に現われていないといけないかを、入力として与えることにし、このしきい値を越えないルール候補を削除する。

最後に、各入力文字列から同じ文字列を抽出するルール候補を統合し、ルールの集合を出力する。

3 実験結果

前節のアルゴリズムをプログラミング言語 python を用いて実装し、HTML と XML ファイルを使った実験を行なった。中身は日本語、英語、中国語、ドイツ語のいずれかで記述されている。これらのファイルは 13 サイトから集めたもので、総数は 1197 である (表 1 参照)。4 番目の列は収集したファイル数であり、合計は 1197 である。タイプ列の「検索」と「メール」は、これらのページが検索エンジンまたはメールアーカイブのページであることを示している。同様に、「ニュース」はオンラインニュースサイトのページ、「マニュアル」はオンラインマニュアル、「データベース」は公開されている何らかのデータベースのページ、「教官データ」は九州大学教官データベースのデータである。ただし、教官データベースの XML ソースファイルは公開されていない。

実験では E_l を ">" と空白文字 (半角スペース、タブ、改行文字) と、E_r を "<" と空白文字と定義した。また、ルール候補をルールとして認めるしきい値を 50%として実験を行なった。

提案するアルゴリズムでは、あらかじめ抽出すべきコンテンツを定義しているわけではない。したがって、アルゴリズムにより自動生成されたラッパーによって抽出された「コンテンツ」が、本当に抜きだすべきものなのかを評価する必要がある。そのため、あらかじめ入力ファイルを人手で解析し、どの部分を抜きだすべきかを定義し、この人手による定義と自動生成されたラッパーを比較することにした。

表 2 に、静的ページと動的ページを用いた実験結果を示す。「フィールド (精度)」列は、人手で作成したラッパーのフィールドの属性名がどれだけの割合で抜きだされているかを示している。Ev_1 は、人手でラッパー生成する際には見落していたが、自動生成されたラッパーの出力を見て必要と判断したフィールドの数である。一方、Ev_2 は抜きだす必要のない部分をフィールドとして抜きだした数である。例えば "reuters" データセットでは、筆者らは見出し、日付け、本文を抜きだすべきであると判断し、アルゴリズムは完全にこれらのフィールドインスタンスを抜きだすことに成功している。さらに 4 つの有用なフィールドを見つけているが、3 つのフィールド (とアルゴリズムがみなしたもの) を間違えて出力したことを示している。

3.1 静的ページ

ほとんどの他のラッパー生成アルゴリズムでは、入力ファイルは動的に生成されたものを仮定している。このようなページは、適当なテンプレートを検索結果等のコンテンツで埋めた形式をしているため、ページ間で共通な部分はほぼ例外なく等しい。つまり、ラッパー生成問題としては簡単であると言える。一方、本稿では静的なページも対象にラッパー生成を行なった。表 2 が実験結果の概略である。Ev_1 の中には、メタタグの中やコメント部分から有用なコンテンツが抜きだせたフィールドも含まれている。"ftd" データセットの記事には、メタタグの中に記事の要約が、コメントタグの中に日付が書かれているが、これを抽出するルールを生成することができた。他のラッパー生成アルゴリズムは、タグの中やコメント部分を無視するので、このような情報抽出は不可能である。

表 2 を見ると、いくつかのフィールドでは精度は高いものの完璧ではないものが存在する。この原因は、主に入力ファイルのフォーマットと用いたラッパーの形式にある。アルゴリズムは、同じ種類のコンテンツは同じペアの文字列に囲まれていると仮定している。しかし、同じ種類のコンテンツにもかかわらず、時々違う文字列によって囲まれているインスタンスが存在する。例えば、あるフィールドの大部分のインスタンスは "<abc" という文字が後にあるが、いくつかは "<ABC" となっている場合である。本稿で提案するアルゴリズムは、入力に対して知識をほとんど要求しないため、大文字と小文字の同一視すら行なっていない。そのために、このような場合は同じ文字列とみなさない。

以下に、いくつか特徴的なルールが生成されたデータセットを示す。表 3 に "mainichi" データセット (XML ファイル) の結果を示す。1 行が 1 つのフィールドに対応したルールで、上段が左区切り文字、下段が右区切り文字である。生成されたルールは明らかに HTML のタグとは異なるタグを含んでいることが分かる。また、ルールの中には、XML のソースファイル

表 1: 実験に用いた 13 サイトのリスト。

ID	URL	言語	#	タイプ
HTML				
altavista	http://www.altavista.com/	英語	17	検索
freebsd	http://docs.freebsd.org/mail/	英語	49	メール
ftd	http://www.ftd.de/	ドイツ語	101	ニュース
java	http://java.sun.com/j2se/1.3/docs/	英語	30	マニュアル
lycos	http://www.lycos.com/	英語	50	検索
peopledaily	http://www.peopledaily.co.jp/	中国語	127	ニュース
redhat	http://www.redhat.com/mailing-lists/	英語	50	メール
reuters	http://www.reuters.de/	ドイツ語	50	ニュース
sankei	http://www.sankei.co.jp/main.htm	日本語	108	ニュース
yahoo	http://www.yahoo.com/	英語	45	検索
XML				
kyushu-u	–	日本語	50	データベース
mainichi	http://www.mainichi.co.jp/digital/newsml/	日本語	470	ニュース
sigmod	http://www.acm.org/sigmod/record/xml/	英語	50	データベース

表 2: 実験結果

ID	フィールド (精度)	Ev_1	Ev_2
静的ページ			
ftd	タイトル (0%)、見出し (100%)、記事要約 (100%), 本文 (100%)	6	5
java	クラス名 (90%)、日付 (100%)、戻り値 (100%)、本文 (100%)	1	2
mainichi	見出し (100%)、日付 (100%)、キーワード (100%)、本文 (0%)、関連語 (100%)、(別の) 見出し (100%)	1	0
peopledaily	タイトル (100%)、日付 (100%)、見出し (98.4%)、本文 (99.2%)	4	2
reuters	見出し (100%)、日付け (100%)、本文 (100%)	4	3
sankei	見出し (100%)、サブ見出し (100%)、本文 (100%)	0	0
動的ページ			
altavista	タイトル (100%)、要旨 (100%), URL (100%)	7	4
lycos	タイトル (100%)、要旨 (95.5%), URL (100%)	5	3
yahoo	タイトル (100%)、要旨 (100%), URL (100%)	7	11
freebsd	Date (100%), From (100%), To (93.9%), Subject (100%), Message-ID (100%), 本文 (100%)	0	2
redhat	Subject (100%), 本文 (98%)	3	8

表 3: “mainichi” データセットから生成されたラッパー

フィールド	区切り文字 (上段が左、下段が右区切り文字)
日付 1	</HeadLine>\n\t\t\t\t<DateLine>
	</DateLine>\n\t\t\t\t</CreditLine_xml:lang=”ja”>
日付 2	␣(␣
	␣)
キーワード 1	\n\t\t\t\t<KeywordLine>
	</KeywordLine>\n\t\t\t\t</NewsLines>
キーワード 2	<midasi>\n\t\t\t\t\t\t</kanrenmidasi>
	</kanrenmidashi>\n\t\t\t\t\t\t</midasibun>

をみやすくするために使われている空白文字が含まれていることが分かる。このようなルールは、ラッパー生成システム SCOOP [9] も含めて、他のラッパー生成アルゴリズムでは作ることができない。

これらのルールは 2 つの日付、2 つのキーワードフィールドを抜きだすが、本文を抽出するルールを生成できていない。これは “mainichi” の本文を囲む文字列と、我々が採用したラッパーの形式に問題がある。このデータセットでは、本文は “\n ” と “\n ” に囲まれている。これらは同じ文字列であるため、アルゴリズムはより長い区切り文字を見つけようとする。しかし、右区切り文字を長くしようとすると、可変な部分[3]が続くため、適当な区切り文字を見つけることができない。静的ページの結果 (表 2) において、精度が 0 パーセントとなっているところは同じような原因である。

表 4 は “sankei” データセットから生成されたラッパーである。“sankei” データセットは新聞記事からなり、各記事は通常は見出しと本文からなる。ただし、サブ見出しがあるものもないものもある。26 ファイルはサブ見出しを持たない。この場合、前後を囲む区切り文字も存在していないので、単なる LR ラッパーではこのような入力を扱うことができない。またサブ見出しがある場合でも、個数に違いがある。21 ファイルは 2 つのサブ見出しがあり、2 ファイルは 3 つのサブ見出しがあり、残りのファイルは 1 つのサブ見出しがある。アルゴリズムは、サブ見出しを囲むルールを見つけることができている。よって、インスタンスの個数に依存せず、インスタンスがない場合も問題なくルールの生成ができることが分かる。

“■” はマルチバイト文字であり、見出し文の行頭文字として使われている。このように、タグの部分だけでなく、コンテンツ部分の文字であっても、決まって出現する定型の文字列はルールの一部になりうる。提案システムは、言語に依存せずにコンテンツ部分の文字列の一部をルールとして扱うことができる。

[3]実際には日付データ。

3.2 動的ページ

動的なページとして 3 つの有名検索エンジンと 2 つのメールアーカイブのサイトを用いて実験を行なった (表 2 参照)。検索エンジンの結果には、見つけたページのタイトル、URL、ページの簡単なサマリが含まれる。また、メールアーカイブの各ページにはメールの本文、サブジェクト、他の (メールの) ヘッダが含まれる。表 2 に、これらの動的ページを対象とした実験の結果を示す。完全ではないものの、どれも高い精度を示している。

他に “sigmod” と “kyushu-u” という 2 つのデータセットを用いた。これらはどちらも XML ファイルである。“sigmod” データセットは “ACM SIGMOD Record: XML Version” から収集した。1 レコードは、論文のタイトル、著者名 (複数の場合もある)、巻号、年、開始及び終了ページ番号である。提案アルゴリズムは、これらすべてのフィールドを抽出するルールを生成できた。また、管理用と思われる ID 番号が XML のタグの中に、また、ファイルの生成時間がコメントの中に書いてあったが、これらも問題なく抽出できた。

“kyushu-u” データセットは九州大学の教官データベースの XML ファイルである。1 人分 (1 レコード) が 1 ファイルであり、名前、所属、学位、メールアドレス、業績などのデータが含まれる。このファイルの特徴として、タグの名前が日本語を含んでいることが挙げられる。XML では、このように自由にタグを定義できるので、仮にタグしか扱わないようなラッパー

表 4: “sankei” データセットから生成されたラッパー

フィールド	区切り文字
見出し	 ■
	
サブ見出し	\n
本文	<BLOCKQUOTE>\n\n
	<p>\n\n\n

生成アルゴリズムでも多言語化は重要である。

4 おわりに

本稿では、ラッパー生成アルゴリズムを提案した。生成するラッパーは、最も簡単なLRラッパーと呼ばれるもので、左と右の区切り文字でコンテンツを囲む形式である。区切り文字の最初と最後の文字以外には、入力テキストに関する知識は不要で、任意の半構造化テキストに適用可能である。本稿では、XMLとHTMLに関する実験結果を紹介した。

生成されたラッパーにより、動的なページだけではなく、他のラッパー生成アルゴリズムが苦手とする静的なページからもコンテンツの抽出が行なうことができた。さらに、コメントやタグの内部からも情報抽出が可能であった。

参考文献

[1] N. Ashish and C. Knoblock, Wrapper Generation for Semi-structured Internet Sources, Proc. of *Workshop on Management of Semistructured Data*, 1997.

[2] D. Buttler, L. Liu and C. Pu, A Fully Automated Object Extraction System for the World Wide Web, International Conference on Distributed Computing Systems, 2001.

[3] C.-H. Chang and S.-C. Lui, IEPAD: Information Extraction Based on Pattern Discovery, Proc. of *the Tenth International Conference of World Wide Web (WWW2001)*, pp. 4–15, 2001.

[4] D. W. Embley, Y. Jiang and Y. -K. Ng, Record-Boundary Discovery in Web Documents, Proc. of *ACM SIGMOD Conference*, pp. 467–478, 1999.

[5] D. Ikeda, Y. Yamada and S. Hirokawa, Eliminating Useless Parts in Semi-structured Documents using Alternation Counts, Proc. of *the Fourth International Conference on Discovery Science*, Lecture Notes in Artificial Intelligence, Vol. 2226, pp. 113–127, 2001.

[6] N. Kushmerick, D. S. Weld and R. B. Doorenbos, Wrapper Induction for Information Extraction, Intl. Joint Conference on Artificial Intelligence, pp. 729–737, 1997.

[7] N. Kushmerick, Wrapper Induction: Efficiency and Expressiveness, Artificial Intelligence, Vol. 118, pp. 15–68, 2000.

[8] K. Lerman, C. A. Knoblock and S Minton, Automatic Data Extraction from Lists and Tables in Web Sources, Adaptive Text Extraction and Mining Workshop, 2001.

[9] Y. Yamada, D. Ikeda and S. Hirokawa, SCOOP: A Record Extractor without Knowledge on Input, Proc. of *the Fourth International Conference on Discovery Science*, Lecture Notes in Artificial Intelligence, Vol. 2226, pp. 428–487, 2001.

[10] Y. Yamada, D. Ikeda and S. Hirokawa, Automatic Wrapper Generation for Multilingual Web Resources, Proc. of *the 5th International Conference on Discovery Science*, Lecture Notes in Computer Science, Vol. 2534, pp. 332–339, 2002.