自然言語処理 131－7
音声言語情報処理 26－7
（1999．5．28）

Suffix array を用いた日本語単語分割

伊東秀夫
(株) リコー情報通信研究所
〒222 神奈川県横浜市港北区新横浜 3-2-3
hideo@ic.rdc.ricoh.co.jp

事例ベースの手法を用いる日本語単語分割法を提案する。本手法では、単語分割が施された事例テキストを、文字列索引法の一種である suffix array により索引づけし、入力テキストとの照合を高速化する。また、この際に発生する無駄な照合を回避するための技法、および、最長一致長を利用する効果的かつ単純な分割法を用いる。評価実験により、既存の形態素解析システムの解析結果を事例として利用することで、そのシステムの振る舞いを 99% 以上の一致精度でエミュレーションすることが可能であることを示す。また単語分割の処理速度はノート PC 上で 1GB/hour 以上の高速性を実現している。

A Method for Segmenting Japanese Text into Words by Using Suffix Array

Hideo Itoh
Information and Communication R & D Center
3-2-3, Shin'yokohama, Kohoku–ku, Yokohama, Kanagawa, 222, Japan
hideo@ic.rdc.ricoh.co.jp

We present an example-based method for segmenting Japanese texts. Suffix array is used for indexing the examples of word segmentation. Some techniques enable us to eliminate useless example retrieval, and a simple and effective segmentation strategy is adopted. Our experiments on EDR corpus demonstrate that the word segmentation program can emulate a Japanese morphological analyzer with high accuracy using it's segmentation results as examples. And it achieved high performance more than 1 GB / hour on note PC.

1 はじめに

日本語単語分割は、日本語テキストを対象とする検索・分類・分析等、幅広い応用系において第一ステップとなる重要な基本技術である。

従来より日本語単語分割に関し、規則ベース [1]、統計ベース [2][3]、事例ベース [4] 等、様々なアプローチがなされてきた。本稿では事例ベースの手法による日本語単語分割について述べる。ここでは事例ベースという言葉を次のようにかなり狭義に用いる。即ち、予め事例の集合を用意しておき、未知の入力に対して、それと類似する事例を利用して処理を行う枠組みを事例ベースと呼ぶ。

日本語単語分割に当てはめれば、単語分割例のこの集合中から、入力テキストと類似する事例を検索し、その分割のされ方に沿って入力テキストを分割する。この枠組みにおいて重要な技術要素は次の点である。

- 事例と入力間の類似度
- 事例の活用法
- 事例の検索法
- 事例の獲得

上記の各々については後節で各論するが、ここでは事例ベースの枠組みを採用する動機、および本稿の提案の位置づけについて述べておく。

辞書と接続表を用いる伝統的な形態素解析技術により、既に 99% 程度の精度を持つ単語分割は実現可能になっている。しかし、このレベルの精度をさらに向上させることを狙う場合、規則ベースの枠組みは、制御可能なファクタが限られている点、および、これらのファクタ間の依存関係が必ずしも単純ではないことが、ある種の障害になっている [5]。また、頻度を用いることを特徴とする統計ベースの枠組も、初期性能を迅速に高めることはできるが、更なる精度向上を狙う段階になると同様の欠点を持つ。

一方、事例ベースの枠組みでは、類似度の定義と事例活用法が非常に単純であれば、事例の追加による影響または効果が明確になり、適切な事例の追加による着実な精度向上を期待できる。

また近年は処理対象とするテキストが大規模化してきており、単語分割の処理速度も非常に重要になってきている [6][7]。事例ベースの枠組みでは、事例の検索を高速化することで、伝統的な形態素解析が持つ技術上の限界を上回る処理速度が得られる可能性がある。

また事例ベースの枠組みには、事例の獲得・収集という固有の問題がある。初期事例として単語分割の正例を、ある程度の量、用意する必要がある。形態素解析の結果を人手で修正する等には、かなりのコストを要するし、人手が介入することで分割の首尾一貫性が失われる問題も生じる。

そこで我々は事例ベースの単語分割を、辞書と接続表を用いる伝統的な形態素解析のエミュレータとして、まず位置づける。この場合、初期事例は、プレーンテキストを入手し高精度の形態素解析システムを用いて単語分割を行うことで低コストで獲得できる。また、この事例集合は人手によるものより首尾一貫性を持つであろうから、単語分割の性能を明確に把握でき、効率的に改良を進めることができる。単語分割システムが手本とする高精度の形態素解析システムを精度よくエミュレートできた後に、事例の追加により更なる精度向上を図る。

2 単語分割法

本節では我々が提案する日本語単語分割法の基本方式を説明する。単語分割には、予め用意された分割例の集合を用いる。この集合を本稿では事例データと呼ぶ。具体的に事例データとは、日本語テキストと、そのテキスト中の各文字毎にその直後が単語境界があるか否かを示す情報から構成される。

図 1 に事例データの例を示す。図中の上段は日本語テキストである。下段は各文字の直後が単語境界であるか否かを 1,0 で表現したデータである。上段を事例テキスト、下段を分割データと呼ぶ。

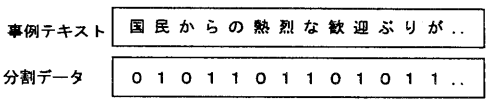

図 1: 事例データの例

事例データは、エミュレートする形態素解析システムを用いて日本語コーパスを解析することで得る。勿論、この事例データは誤りを含むかもし

れないが、我々の当初の目的は、この形態素解析システムをエミュレートすることであるから問題にはならない。

単語分割は、未知の入力テキストと事例テキストを照合することから始める。照合は入力テキストの各文字毎に、その文字を起点として最長一致する事例テキスト中の部分文字列 (LCS と呼ぶ) を求めることを処理単位とする。

事例データの量が十分であれば、複数の LCS により入力テキストの多くの部分が被覆され、それらの LCS にその分割データが対応している。

各 LCS は、それが被覆する入力テキスト部分の単語分割に関して、その入力テキストとの類似度に見合った貢献をすることを要請する。このような類似度として LCS の長さ（即ち最長一致長）を採用する。

この要請を満たした上で、入力テキストを被覆する各 LCS の分割データを基に、入力テキストの最終的な単語分割結果を得る方法を以下に説明する。

入力テキストを $c_1, c_2, .., c_n$ で表す。各文字 c_k $(k = 1, .., n-1)$ 毎に、以下の 2 つのスコア $A0(c_k)$, $A1(c_k)$ を設ける。各スコアの初期値は零とする。

A0 この文字の直後の分割を抑制するスコア

A1 この文字の直後の分割を促進するスコア

事例データとの照合により文字 c_i を起点とする長さ m の LCS $c_i, c_{i+1}, .., c_{i+m-1}$ が得られたとする。LCS 中の各文字 c_k に対応する分割データを $S(c_k)$ で表す。$S(c_k)$ が 0 のとき事例テキスト中の該当文字の直後が単語境界でないこと表し、1 のとき単語境界であることを表す。そして、$m > 1$ のとき $k = i, i+1, .., i+m-2$ に関し、次のようにスコアを与える。

- $S(c_k)$ が 0 ならば $A0(c_k)$ に $m-1$ を加える
- $S(c_k)$ が 1 ならば $A1(c_k)$ に $m-1$ を加える

入力テキストと事例データの照合において、入力テキスト中の各文字を起点とする LCS が得られる度に、その LCS が覆う範囲の文字に関して上記スコアリングを行なう。各文字 c_k $(k = 1, 2, .., N-1)$ に関するスコアリングが終了後、次のように単語分割を行う。

- $A1(c_k) > A0(c_k)$ ならば c_k の直後を単語境界とする。
- $A1(c_k) < A0(c_k)$ ならば c_k の直後を単語境界としない。
- $A1(c_k) = A0(c_k)$ ならば c_k の直後を単語境界とするか否かをデフォルト処理で定める。

スコアが同点の場合のデフォルト処理は、c_k, c_{k+1} の字種の組み合わせで定める。考慮する字種として記号類、数字、英字、カタカナ、平仮名、漢字等がある。図 2 に入力テキストと事例テキストの照合の例を示す。

```
len= 5 : 国 民 か ら の
len= 3 :    民 か ら
len= 4 :       か ら の 熱
len= 2 :          ら の
len= 4 :             の 熱 烈 な
len= 5 :                熱 烈 な 歓 迎
len= 1 :                   烈
len= 1 :                      な
len= 5 :                         歓 迎 ぶ り が
len= 1 :                            迎
len= 3 :                               ぶ り が
len= 2 :                                  り が
len= 1 :                                     が
       : 国 民 か ら の 熱 烈 な 歓 迎 ぶ り が
    A0 : 4  0  9  0  0  7  7  0  8  0  6  0  0
    A1 : 0  6  0  8  6  0  0  4  0  4  0  7  0
```

図 2: 単語分割法の例

2.1 単語分割法の特徴

前述の単語分割法の特徴を以下に示す。

- LCS が長いほどその分割データは優先される。また加算により、重なり合う LCS の分割データを総合的に利用する。
- 単純な枠組みであり、各事例が分割結果にどのように影響するかが明確である。
- 分割の組合わせを求めないため非常に高速である。処理時間の大部分は次節で述べる事例照合によって占められる。

3 事例照合の高速化

単語分割法の処理時間のほとんどは、入力テキストと事例テキストの照合、つまり入力テキスト中の各文字を起点とする LCS の同定に費やされることから、この高速化は重要である。

事例照合を高速に行うために、事例テキストを予め索引づけしておく。このために文字列索引法の一

種である Suffix array [8][9] [10] を用いる。Suffix array は文字列索引としては最もコンパクトであり、また計算効率が字彙の規模に依存しない点で、日本語テキストの索引づけに適している。

図 3 にテキスト "BANANA" に対する suffix array の例を示す。

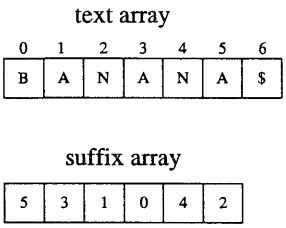

図 3: Suffix array の例

長さ N のテキスト $T = t_1, t_2, \ldots, t_N$ 中の各文字 t_k を起点としテキスト末までの文字列を suffix と呼び、k をその suffix へのポインタと呼ぶ。Suffix array は、suffix をキーとしてポインタを辞書順にソートすることで得られるポインタ配列である。

事例照合では、事例テキスト T に対する suffix array を予め構築し、入力テキストの文字毎に、その文字を起点とする入力テキストの suffix を検索キーとして suffix array を二分探索する。結果として、LCS および、その LCS を接頭とし辞書順で最も若い事例テキスト中の suffix の開始位置が得られる。この開始位置を用いて対応する分割データを求める。

3.1 スキップによる高速化

入力テキストの各文字毎に事例照合を行うことで、より多くの分割データを用いることができる。この方法を non-skip 法と呼ぶ。しかし図 4 の "×" で示した LCS のように、端点を共有する無駄な事例照合も発生する。

このことを避ける方法として、Backward-Forward 照合法 (BFM 法) を提案する。BFM 法では、二分探索における照合方向を後ろ向き (Right to Left) に行う。このため、リバースされた事例テキストに対して Suffix array を構築しておく。入力テキスト中の文字 c_i を起点とする後向きの二分探索により、最長一致文字列 LCS1 と共にその事例テキスト中の開始位置も得られる。その後、入力テキスト中の文字 c_{i+1} を起点とし、LCS1 の事例テキスト中の後続部分を、suffix array を用いず、前向きに直接照合する。この前向き照合の結果 LCS2 が得られ、LCS1 と LCS2 の連接である LCS3 を最終的に得ることになる。そして次の事例照合は、入力テキスト中の LCS3 末の次の文字から開始する。即ち、LCS2 の長さ分だけ事例照合をスキップすることになる。LCS1 は、いわば入力テキスト中に LCS2 が出現することを予測する働きを持つことと、LCS1 は最長一致文字列であることから、長い LCS2 による高速化が期待できる。図 5 に BFM 法を例示する。

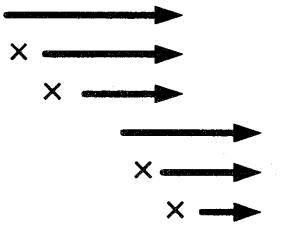

図 4: 無駄な事例照合の例

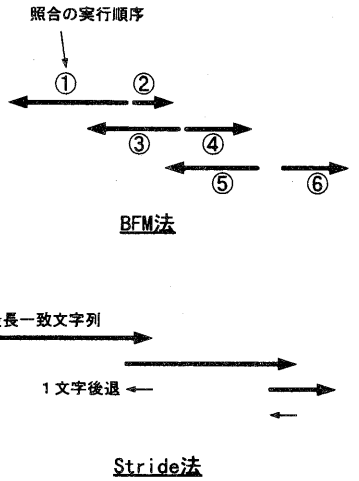

図 5: BMF 法と Stride 法

また、より単純なスキップ法として stride 法と呼ぶ方法も考えられる。これは、通常の前向きの二分探索で LCS を得て、次の事例照合は、LCS 末の一文字前の文字を起点として行う方法である (図 5)。つまり LCS 長 - 1 文字分のスキップがなされる。BFM 法よりもスキップ長が長く、より顕著な

高速化が期待できるが、その反面、分割に用いるLCSの数が減り、分割精度が若干低下することが予想される。

4 辞書の利用

事例ベースの手法を用いても、統計ベース同様、データスパースネスの問題は避けられない。事例を補完し分割精度を向上させるために、エミュレート対象である形態素解析システムの辞書を用いる。辞書の利用により、必要な事例データの量が減少し、その結果、事例照合の高速化とコンパクト化の効果も期待できる。

ここで用いる辞書とは、見出し語の語形（活用語の場合、その全ての活用変化形）に相当する文字列の集合である。品詞などの属性情報は用いない。

辞書と事例データの違いの第一は、辞書の場合は最長一致長が、高々、語形長であり、比較的短いという点である。短いLCSは分割精度を低下させる要因になる。そこで辞書に関しては、最長一致ではなく完全一致を要請する。即ち、語形全体が入力テキストに出現した場合にのみ、その語形に対応する分割データを用いる。

4.1 MDAWGによる辞書照合

Suffix arrayは最長一致を求める場合は非常に効率的であるが、完全一致を求めるのには不適である。そこで、辞書照合にはSuffix arrayではなくMDAWG (Minimal Directed Acyclic Word Graph) と呼ぶデータ構造を用いる [11]。

MDAWGはTrie構造と同等の辞書検索の高速性を保ちながら、非常にコンパクトに辞書を表現できるグラフ構造である。通常の辞書検索の場合、各語形に対応するレコード情報を取得する枠組み (順位配列) も必要となるが、本単語分割では不要である。よってMDAWGはさらにコンパクトになる。

次節の評価実験では、茶筌 [1] に付属している辞書を用いた。活用語については活用型に応じて活用語尾を連結し、全ての変化形を辞書に入れた。表1に概要を示しておく。

辞書照合は、入力テキストの各文字毎を起点として行い、完全一致語形が得られる度に、LCSと同様のスコアリングを行う。ただし、ある文字を起点とする完全一致語形が複数存在する場合は、最長のものについてのみスコア加算する。辞書照合が終了した後、事例照合を行い、さらにスコア加算を行う。単語分割位置の判断は、これら辞書照合と事例照合によるスコアリングの総和を基に、前述した方法で行う。

語数	辞書容量	MDAWGの容量
484475	5.27MB	1.34 MB

表 1: 茶筌辞書の語形情報

4.2 数字列と英字列の扱い

数字列は無限の表現の可能性があり、本手法のように有限の事例との最長一致では上手く扱うことができない。そこで、数字の連続については、事例照合による分割結果の如何に関わらず、一つの語として分割出力するための特別な処理を設けた。英字（キリル文字、ギリシャ文字）に対しても同様の処理を設けた。これら以外は全て事例照合の結果により分割を行う。

4.3 実装法について

分割データは一文字当たり1-bitで表現できることから、事例テキストを記憶する文字配列に畳み込んだ。これにより分割データの参照も高速になる。

事例テキストTの長さをNとしLCSの長をmとすると、事例照合には最悪で$m \log N$回の文字比較を要する。これをManber–Myersらは$m + \log N$回にする高速化技法を提案している [8]。この技法では二分探索の過程で現われ得る区間の端点に対応するする2つのsuffix L, Rと、その区間の中間点に対応するsuffix Mとの共通接頭長$Lcp(L, M), Lcp(R, M)$という2つの数値を予め計算しておき、実際の二分探索時に利用することで文字比較の回数を削減する。

我々は$Lcp(L, M), Lcp(R, M)$の値を4bitで表現可能な範囲内に制限し、suffix array上に畳み込んで格納することにした。この結果、事例テキストは$32 - 4 - 4 = 24$ bitで表現できるサイズに制限されるが後述する実験にみるように実用上は十

分なサイズである。

また別の高速化として suffix array を suffix の先頭文字の異同により bucket に細分し、これらをバケット表で管理することで、二分探索の範囲を狭めることにした。

5 評価実験

5.1 茶筅に対するエミュレーション精度

本単語分割法により日本語形態素解析システム『茶筅』version 1.5 の解析を、どの程度エミュレートできるかを評価した。

実験にはEDR コーパスを用いた。EDR コーパスを以下の表 2 に示すように、事例テキストと評価用テキストに分割した。事例テキストは容量の違う 10 種類を用意した。

データ	テキスト容量
事例テキスト 1	1.45 MB
事例テキスト 2	2.90 MB
事例テキスト 3	4.35 MB
事例テキスト 4	5.80 MB
事例テキスト 5	7.26 MB
事例テキスト 6	8.71 MB
事例テキスト 7	10.15MB
事例テキスト 8	11.60 MB
事例テキスト 9	13.05 MB
事例テキスト 10	14.51 MB
評価用テキスト	1.62 MB

表 2: EDR コーパスの利用内訳

事例テキスト 1 ～ 10 を茶筅によって形態素解析し、単語分割例（即ち事例データ）を得た。これらの事例データ 1 ～ 10 および、前述した茶筅の辞書を用いて、評価用テキスト (open data, 20780 文) を単語分割する。

茶筅による評価用テキストの単語分割結果も求め、本単語分割法による結果と照合することで、図 6 に示すグラフを得た。このグラフで、横軸は事例データのサイズ（表 2 の事例テキスト番号に対応）、縦軸はエミュレーションの精度、すなわち、茶筅と分割の有無が一致した文字間位置の全体に対する割合である。

単語分割プログラムとしては、no-skip 法, BFM 法, stride 法を用意した。各々について、辞書を用いる場合と用いない場合について測定したエミュレーション精度がプロットしてある。

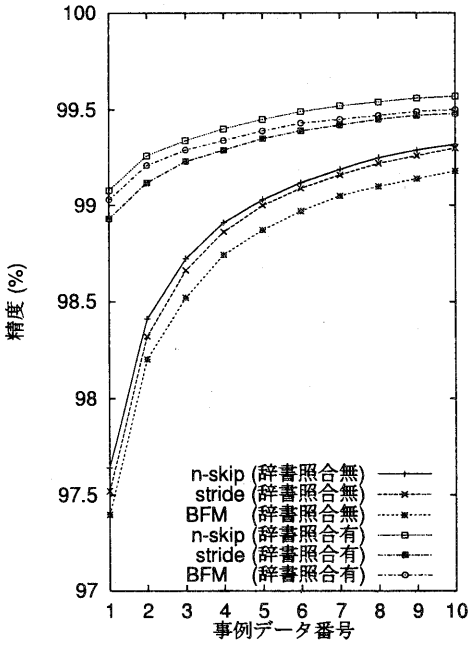

図 6: 事例データ量と精度

精度は、non-skip 法、BMF 法、stride 法の順で低下するが、その差は 0.2%程度である。

また、このグラフから、辞書利用の効果が大きいことがわかる。辞書を用いると、事例データ 1 のような小規模な事例集合でも 99%以上の精度が選られる（即ち 100 文字に一度程度、茶筅とは異なる位置での分割が発生する）。

5.2 処理速度

次に単語分割の処理速度の測定結果を示す。測定には モバイル Pentium (366MHz) を搭載したノート PC 上で行った。搭載メモリ量は 192MB である。測定にもちいたデータは表 2 の事例データと評価用テキストである。事例データについては表 2 中の番号で表している。全て辞書照合する場合の処理時間を測定した。処理時間は Linux の time コマンドの user 及び system 時間の合計から算出した。

参考のため、研究用に一般に公開されている形

態素解析プログラム Sumomo (Ver 1.3) [6] および茶筅の処理速度も一緒に示す。

処理系	事例データ	速度 (MB/hour)
stride	1	2394
stride	10	1671
BFM	1	1496
BFM	10	1208
non-skip	1	1463
non-skip	10	933
sumomo	–	1648
chasen	–	210

表 3: 単語分割の処理時間

処理速度は stride 法、BFM 法、non-skip 法の順に遅くなる。stride 法を事例データ１０と共に用いれば茶筅の単語分割結果を 99.5%のエミュレーション精度で約 8 倍の速く得ることができる。この速度は、高速な形態素解析として知られている sumomo とほぼ同等の速度である。

5.3 使用記憶量

単語分割プログラムは、事例テキスト、Suffix array、及び MDAWG 辞書を内部記憶に読み込んで利用する。Suffix array は事例テキスト中の語頭位置についてのみ索引づけすることで事例テキストとほぼ同じサイズになる。これらの使用記憶量の総計は、事例テキスト 10 を用いる場合で、約 35 MB になる。

6 おわりに

事例ベースによる日本語単語分割法について述べた。事例照合によって得られる最長一致文字列の分割データを高速に利用するための単純な枠組みと、事例データを suffix array によって索引づけしスキップ技法を用いることで事例照合の高速化を図った。評価実験によれば、これらの枠組みにより、辞書を用いる形態素解析システムの振る舞いを高速かつ高精度にエミュレートできることを示した。特にその形態素解析システムが用いる辞書の利用がキーになる。

また一般に、形態素解析システムは高精度になるほど処理速度は低下する傾向にある。このような形態素解析システムをエミュレートできれば、本単語分割システムにより、高精度の単語分割結果を高速に得ることができる。

今後は、本手法による誤解析の分析、および未知語抽出等により、事例データの改良を行うことで、エミュレート対象とする形態素解析の精度を超える単語分割性能を実現する試みに取り組む。

参考文献

[1] 松本, 北内, 山下, 平野, 今一, 今村. 日本語形態素解析システム『茶筅』version1.5, 1997. NAIST Technical Report, NAIST-IS-TR97007.

[2] 伊東伸泰, 西村雅史. N-gram を用いた日本語テキストの単語単位への分割. 自然言語処理研究会報告, No. 122.

[3] 山本幹雄, 増山正和. 品詞・区切り情報を含む拡張文字の連鎖確率を用いた日本語形態素解析. 言語処理学会第三回年次大会予稿集, pp. 421–424, 1997.

[4] 山下達雄, 松本祐治. 品詞タグ付きコーパスを直接利用した形態素解析. 言語処理学会第四回年次大会予稿集, pp. 524–527, 1998.

[5] 渕武志, 松岡浩司, 高木伸一郎. 保守性を考慮した日本語形態素解析システム. 自然言語処理研究会報告, No. 117, pp. 59–66, 1997.

[6] 鷲坂ら. 情報検索のための高速日本語形態素解析システムすもも. 情報処理学会第 54 回全国大会, pp. 59–60, 1997.

[7] 多田智之, 金岡秀信. 高速日本語形態素解析ソフト SuperMorpho-J. 言語処理学会第四回年次大会予稿集, pp. 379–381, 1998.

[8] U. Manber and G.Myers. Suffix arrays: a new method for on-line string searches. *SIAM Journal of Computing*, Vol. 22, No. 5, pp. 935–948, 1993.

[9] 伊東秀夫. 大規模テキストに対する suffix array の効率的な構築法. 自然言語処理研究会報告, No. 129.

[10] 山下達雄, 熊谷俊高, 米沢恵司, 松本祐治. Suffix Array を用いた高速文字列検索システム SUFARY, 1997. http//cactus.aist-nara.ac.jp/lab/nlt/ss.html.

[11] 伊東秀夫. 辞書検索に用いる有限オートマトンの構成と実装. 言語処理学会第四回年次大会予稿集, pp. 47–50, 1998.