誤字脱字や伏字を許容する近似辞書照合技術

齋藤 邦子　今村 賢治　松尾 義博　菊井 玄一郎
日本電信電話株式会社 NTT サイバースペース研究所
{saito.kuniko, imamura.kenji, matsuo.yoshihiro, kikui.genichiro}@lab.ntt.co.jp

1　はじめに

近年のWebの普及により、ブログやツイッターなど一般ユーザが自由な文体で記述した、多種多様なスタイルのテキストが膨大に存在するようになってきた。また、例えば電子カルテに代表されるように、従来手書きで作成されていた書類の電子化が進み、キーボード入力や音声認識など、様々な日本語入力形態で形成されたテキストも蓄積されている。

このようなテキストは、新聞記事のようにプロの記者が執筆し、更に校正チェックの過程を経て表現が統一されたものとは異なり、入力過程で混入した誤字脱字、書き手による表記の揺らぎがそのまま含まれている。また意図的に伏字で内容を隠した表記が出現することもある。このように誤字脱字や伏字などの様々な表記揺らぎを含むテキストを解析する上で、最初の障害となるのは単語検出である。即ち、予めシステムで準備している単語辞書の表記とは一部異なる表現が出現すると、従来の辞書照合技術では、辞書エントリと完全一致しない表記を検出できない。

本稿では、単語境界が自明ではないテキストからの単語検出、すなわち辞書照合において、辞書エントリと完全一致する表記だけでなく、置換・挿入・削除が発生した表記も照合可能となる、近似辞書照合技術を提案する。

2　辞書照合技術

2.1　トライを利用した大規模照合

提案手法について述べる前に、形態素解析などで利用される一般的な辞書照合技術について簡単に説明する。これは、予め準備しておいた単語辞書を利用して、入力文に出現する全ての単語を検出する技術である。日本語は単語の境界が明示的ではないため、文頭から文末までの全ての文字位置において、その文字位置から始まる文字列と一致する全ての単語を単語辞書から検索しなければならない。辞書検索には、ハッシュ、探索木、デジタル探索木、有限状態変換器(FST)など様々な手法が提案されているが、本稿ではデジタル探索木の1種であるトライ辞書に注目する。

トライは、探索キーとなる単語表記集合の、共通接頭辞を併合した木構造を有し、単語表記の各文字は、トライの枝として表現される。ある文字列についてトライ辞書と照合する場合、入力文字列を先頭から 1 文字ずつトライの枝ラベルと照合する。トライを用いると、接頭辞を共有する全てのキーを同時に照合できるため、処理速度は辞書エントリの総数に依存しない。形態素解析では数十万～場合によっては数百万語の単語辞書が利用されるため、エントリ数に依存せず高速検索できるトライはこのような用途にも十分耐えうる辞書構造と言える。

2.2　近似辞書照合への拡張アイデア

2.1節で述べた従来の辞書照合では、辞書の単語表記と完全一致するものしか照合できない。そこで本稿では辞書に収録する内容と照合時の入力文を操作して、挿入・置換・削除まで許容する辞書照合を提案する。なお、本稿では、説明を簡単にするために、1単語1文字までの挿入・置換・削除を想定する（編集距離1）。

入力文字列と辞書を照合する時に、その照合状況が、完全一致・挿入・置換・削除のどの状態であるかは次のように整理できる。ここで、辞書表記に「オーケストラ」が存在すると想定する。

まず完全一致は、入力文字列、辞書ともに何の変化も加えない状態で照合できる場合であり、入力文字列が「オーケストラ」の場合がこれに該当する。これは通常の辞書照合と同じである。

次に削除を考える。入力文字列が「オケストラ」、である時、辞書表記の2文字目「ー」をスキップさせた「オケストラ」が辞書に存在すれば入力文字列と照合できる。即ち、入力文字列と辞書スキップ表記が照合すれば、これは削除照合である。

次に挿入を考える。入力文字列が「オーケッストラ」である時、入力文字列が4文字目「ッ」をスキップさせた「オーケストラ」であれば辞書表記と照合できる。即ち、入力スキップ文字列と辞書表記が照合すれば、これは挿入照合である。

最後に置換を考える。入力文字列が「オオケストラ」である時、入力文字列の2文字目「オ」と辞書表記の2文字目「ー」をそれぞれスキップさ

せ、どちらも「オケストラ」であれば両者は照合する。即ち、入力スキップ文字列と辞書スキップ表記が照合し、かつ、両者の文字スキップ位置が一致していれば、これは置換照合である。

上記をまとめると、通常の辞書照合技術について次の 2 点の拡張を行うことにより、完全一致だけではなく挿入・置換・削除まで許容する近似辞書照合技術を実現できる。ここで照合処理は n 文字からなる入力文字列 C に対し、文字位置 i=1,…,n を先頭から末尾まで 1 字ずつ移動しながら進む。C[i]は文字列Cのi番目の文字を示す。

[拡張 1]照合実行時に利用する辞書には、照合したい単語表記エントリだけでなく、全エントリに対して全ての位置の文字をスキップさせた表記を全展開して収録する。その際、辞書に格納する値は、キーとなる表記に対し、スキップ前の元の表記と、スキップ文字位置とする。例えば「オーケストラ」という単語表記に対しては、表 1 で示すような展開を行い、全てを辞書に収録する。

[拡張 2]照合実行時は入力文字列Cの現在の文字位置 i から始まる文字列に対して[拡張 1]で準備した辞書と照合する。更に、文字位置 i からみて j 番目(j≧1)の文字をスキップさせた文字列との照合も実行する。

なお、[拡張 1]では元々の辞書エントリに対して全ての文字位置でのスキップを全展開するため、辞書サイズが増大するが、語数と平均長から最終的な辞書サイズはほぼ予測できる。また、トライとの照合では収録数は速度に影響しないため、あとはデータ構造とコンパクトな実装方法が問題となる。トライの実装については既に様々な手法が提案されており、辞書サイズや目的に応じて選択すればよい。本稿では数 10 万語規模の単語辞書を想定し、ダブル配列法に基づくトライを用いた。また、接頭辞だけを圧縮したトライではなく接尾辞側も併合した FST を採用すれば更にコンパクトな辞書になると期待できる。

[拡張 2]のスキップ位置 j は、辞書エントリの最長単語長をℓとすると 1≦j≦ℓとなる。しかし、実際には、スキップさせない入力文でのトライ辞書との照合状況、即ち、現位置 i からのトライ辞書照合で、何文字先まで照合が成功していたかを示す文字位置 s を記憶することにより、位置 i でのスキップ位置 j は 1≦j≦s を満たせばよいという条件を利用できる。s＜j の場合は、スキップ位置 j の手前の位置でトライ辞書との照合が終了することが明白であるため照合を省略できるからである。文字長 n の入力に対するトライ辞書照合の処理回数は O(n)である。もし全ての文字位置で現位置から最終文字位置まですべてのスキップを考慮すると O(nℓ)であるが、スキップ位置の上限 s を利用すれば O(ns)になる。

1 単語内 2 文字以上の置換・挿入・削除についても、この拡張の範囲で実現可能である。しかし、スキップ処理の種類が増大するため、辞書サイズや処理速度がさらに深刻な問題となる。ただし、1 単語内 1 箇所複数文字連続の同種タイプ照合であればスキップ処理も限定的であり、実装は容易である。我々は 3 文字までの連続挿入・置換・削除を許容する近似辞書照合について、数 10 万語規模の単語辞書での動作を確認した。しかし、実際に誤字脱字や伏字などを含むテキストを想定すると 1 文字挿入・置換・削除で大部分はカバーできると予想され、本稿ではこのまま 1 文字までの近似辞書照合に限定することとする。

次章では、この拡張アイデアに基づく近似辞書照合システムの詳細を説明する。

表 1. 辞書の文字スキップ全展開

キー	値(元表記, スキップ位置)
オーケストラ	オーケストラ,0
ーケストラ	オーケストラ,1
オケストラ	オーケストラ,2
オーストラ	オーケストラ,3
オーケトラ	オーケストラ,4
オーケスラ	オーケストラ,5
オーケスト	オーケストラ,6

3 提案手法

提案手法の処理の流れを説明する。まず、予め照合させたい単語辞書に対し、単語表記の文字スキップ全展開をしてからトライ辞書を構築する。トライ辞書は元々の単語表記と、文字スキップ全展開した表記をキーとし、値には元表記とスキップ位置を格納する。この処理は 2.2 節で述べた[拡張 1]に該当し、事前にオフラインで実施する。

続いて準備したトライ辞書を参照しながらオンラインで照合処理を実行する。処理のアルゴリズムを図 1 に示す。照合は入力文字をそのまま利用する完全照合（図 2: 1-2, 7-15 行目）と、現文字 C[i]から見て j 文字目、即ち文字 C[i+j-1]を順次スキップさせてから照合するスキップ照合を実行する（図 2: 3-4, 17- 22 行目）。2.2 節で述べたとおり、完全照合時に位置 i でのトライ探索到達位置 s を記憶し、スキップ位置 j の上限とする（図 2:3 行目）。ここで s は、トライの終端に到達して照合成功した場合だけでなく、s+1 文字目でトライと照合失敗し、照合結果なしであった場合も含む。この処理は 2.2 節で述べた[拡張 2]に該当する。

得られた照合結果に対し、タイプ判定を実施し

```
T: スキップ展開済みトライ辞書, C: 入力文字列
i,j : 文字位置,   w:照合結果

1:  for (i=1; i≦n; i++) {    # 完全照合
2:    w, s  ←  trie_get_all_list(T,C,i)
3:    for (j=1; j≦s; j++) {# s までスキップ照合
4:      w, s ← trie_get_all_list_skip(T,C,i,j)
5:  }}
6:
7:  sub trie_get_all_list {
8:    s ← 0        # トライ内最終到達文字位置
9:    node ←  0  # トライ初期ノード
10   for (k=i; k≦n; k++) {
11:     c ← C[k]
12:     c で通常のトライ照合実施
13:     遷移可である限り s++;
14:     トライ終端に達したら val を w に追加
15:  } return w, s }
16:
17: sub trie_get_all_list_skip {
18:  s ← 0,   node ← 0
19:  for (k=i; k≦n; k++) {
20:    c ← C[k]
21:    if (k≧ i+j-1) { c ← C[k+1] } #入力スキップ
22:  12-15 と同じ }
```

図 1. 照合アルゴリズム

て、照合結果がどの照合条件から由来するものかによって、その照合が完全一致・挿入・置換・削除のいずれであるかを判断する。表 2 に具体的な照合条件と照合タイプの分類をまとめる。照合条件は、照合時の辞書および入力文字列が完全表記なのかスキップ表記なのかの組み合わせで 4 通りに分類され、各照合タイプが判定される。

最後に重複フィルタリングを実施する。これはある照合が発生した場合に必ず付随的に発生する照合を取り除く処理である。具体的には、完全一致照合が存在すると、その表記に対して

- 先頭と末尾に任意の文字を挿入した表記
- 先頭と末尾の文字を削除した表記
- 表記内の任意の文字を置換した表記

は必ず発生する。そのため、完全一致の照合が得られた場合、入力における照合開始文字位置を記憶しておき、上記の条件に一致するものは重複照合としてフィルタリングを行う。

最終的に残った結果が近似照合の出力であり、照合開始文字位置、辞書表記、照合タイプ（完全一致・置換・挿入・削除）の情報が得られる。

表 2. 照合タイプ判定

		入力文字列	
		完全	スキップ(位置 j)
辞書	完全	完全一致	挿入
	スキップ(位置 h)	削除	h=j ならば置換

4 実験

4.1 トライ辞書サイズと照合処理時間

提案手法の照合処理時間を評価するため、wikipedia から抽出した固有名詞（TV・ラジオ番組、車名、薬品名、人名、店舗）145,674 語を単語辞書として、辞書サイズを 5 万語ずつ段階的に変えて処理時間測定を行った。入力文には 2010 年 12 月のブログデータ約 67MB を利用した。この入力文は 1 行 1 記事相当で 76167 行からなる。1 行を 1 つの入力文としており、平均長は 349 文字であった。全入力文データの照合処理に要した時間を time コマンドで測定した（XeonX5560 2.8GHz*2 メモリ 64GB）。照合条件は完全一致(EM)のみ照合と、完全一致・挿入・置換・削除全て(ALL)の照合を比較するために、以下の 3 つの設定で測定した。

[EM1]スキップ全展開しない辞書で EM 照合
[EM2]スキップ全展開した辞書で EM 照合
[ALL]スキップ全展開した辞書で ALL 照合

[EM1]は、通常のトライ辞書を利用した単語検出と同等である。[EM2]は、スキップ辞書を利用するが照合自体は EM のみで、入力のスキップ処理や照合タイプ判定などは実施しない。照合処理は[EM1]と同等だが、トライ辞書は[ALL]と同等であり、照合時にはトライ辞書に無駄な状態が多数存在する条件である。当然ながら[EM1]と[EM2]の照合結果は完全に一致する。[ALL]は、本稿で提案する近似辞書照合である。

それぞれの照合処理時間をトライエントリ数に対してプロットした結果を図 2 に示す。[EM1]はシンプルなトライ辞書で高速に処理が進む。トライは照合時間にエントリ数が依存しない手法ではあるが、[EM2]ではトライエントリ数が増大した分、照合時間が約 2 倍に増加していた。これは、トライエントリ数の増加に伴い、トライ状態数も増加し、トライ探索中に到達した距離 s が増加する傾向にあるためと推測している。[ALL]では[EM2]に対して約 3 倍処理時間が増大しており、

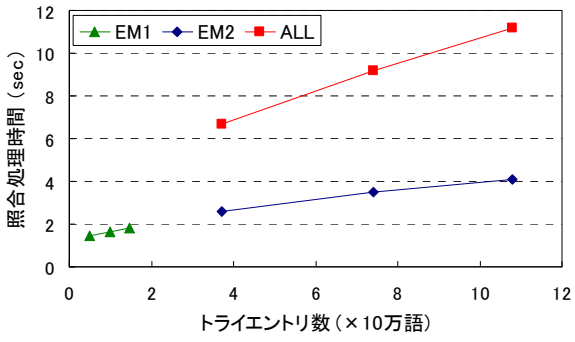

図 2. 照合処理時間

照合時のスキップ処理・重複フィルタリングなど周辺の処理によるものと解釈できる。従来の完全一致のみの辞書照合[EM1]に対しては最終的に本手法では約 6 倍の処理時間となったが、数万語以上の大規模な辞書を利用して、大規模な入力文に対する近似辞書照合を実用時間で実現できることが確認できた。なお、図 1 で示した完全照合・スキップ照合ともに Aho-Corasick 法を導入すると更に高速化が期待できる。

4.2 ブログからの伏字表記検出

本手法の応用例として、ブログからの伏字収集を行った。4.1 の辞書にさらに独自に収集した固有名詞（組織）を追加した 151,163 語を単語辞書とし、置換照合から入力側の出現文字が「○」「●」であるものを抜き出した。これらは伏字である可能性が高く、通常の辞書照合では検出できない。ただし本手法はあくまでも漏れなく全ての照合結果を出力することが目的のため、単語単位として正しいか（カタカナの途中から照合する等）、書き手の意図とあった結果となっているか（多義解消）は議論しない。これらの問題は、照合範囲を文字種で制御するか、形態素解析も併用して前後の単語接続を考慮するなどして対処することが望ましい。表 3 に抽出結果の上位を紹介する。文字長 3 では多義や単語単位の問題が深刻だが、文字長が 4 を超えると概ね正しい結果となり、5 文字以上では殆ど問題にならなかった。そのため 4 文字以上の長い単語での伏字検出は本手法だけでも十分活用できる。

表 3. 伏字検出例

3 文字	●コー、○ボタ、●コン、○ック、ジャ○
4 文字	○ボタン、●コード、ジャ○コ、ユニ○ロ、○ックス、●セット、ダ○ソー、ユ○クロ
5 文字以上	ヤ○ダ電機、ドラ○もん、ジャ○ーズ、●について、アニ○イト、ブック○フ

6 関連研究

2 つの文字列の類似度を編集距離や文字 ngram などの指標で評価し、一定範囲以内の近さにあるものを高速に検出する近似照合の研究は大きく 2 つのタスクに分類できる。1 つ目は、近似性を評価する 2 つの文字列の単語境界が自明なタスクであり、既に 1 単語対大規模辞書の高速な近似照合が実現している[1]。2 つ目は、全文検索や grep などテキストからの文字列検出における近似照合であり、単語境界の不明なテキストからの網羅的な文字列検出タスクである[2][3][4]。後者は入力文字列のあらゆる単語範囲を走査する必要があるため、数語～1000 語程度の小規模辞書での近似照合にとどまっていた。

本手法は後者のタスクに対して辞書を数万語以上の大規模辞書に拡張する提案であり、大規模辞書の全エントリとテキスト中のあらゆる単語候補との近似性を高速に効率よく評価することを実現した。

Mihov らは前者のタスクの延長で、約 100 万語の大規模な辞書に対して入力文字列・辞書ともに 1 文字スキップしたものも照合する近似照合を提案しており、本提案に非常に近い[5]。しかし、本手法が全展開した辞書を 1 つのトライに収録したのに対し、Mihov らの手法は全展開した辞書でスキップ位置が同じ単語集合ごとにオートマトンを複数に分割する（辞書エントリの最長単語長が ℓ の時、ℓ 個に分割される）。そして、入力文字列のスキップ位置を変えながら同じスキップ位置に対応するオートマトンと逐一照合する。これは本手法での照合条件判定、特に入力文字列・辞書のスキップ位置の一致を判定する処理を省略できるメリットがあるが、辞書照合処理が必ず $\ell+1$ 回発生し $O(n\ell)$ である。2.2 節で述べたとおり、提案手法ではスキップ位置上限 s を利用するため $O(ns)$ と効率よく照合処理が実行できる。

7 まとめ

本稿では、単語境界が自明ではないテキストに対して、挿入・置換・削除を許容しながら大規模辞書と照合する技術を提案した。今回は、1 単語 1 箇所の近似性に限定することで、数万語以上の辞書に対しても実時間で動作する高速な近似照合技術を実現した。今後は、近似性のバリエーションを増やしたり、本技術を活用して誤字脱字や伏字を許容する形態素解析技術の検討を進めたい。

8 参考文献

[1] Okazaki, N. and Tsujii, J.: Simple and Efficient Algorithm for Approximate Dictionary Matching, *Proc. of COLING2010*, pp. 851-859, 2010.

[2] 内山将夫,井佐原均: 近似文字列照合による全文検索のための接尾辞配列の高速走査法, 情報処理学会論文誌, vol.43, no.SIG9 (TOD15), pp. 1-14, 2002.

[3] Baeza-Yates, R. and Navarro, G.: Multiple Approximate String Matching, *Proc. of WADS'97*, pp. 174-184, 1997.

[4] Muth, R. and Manber, U.: Approximate multiple string search, *Proc. of CPM'96*, pp. 75-86, 1996.

[5] Mihov, S. and Schulz, K. U.: Fast Approximate Search in Large Dictionaries, *Computational Linguistics*, vol.30, no.4, pp. 451-477, 2004.