自然言語処理 134−15
(1999. 11. 26)

文脈自由文法を用いた日本語誤り検出・訂正手法の提案

渥美 清隆

静岡大学 工学部 システム工学科

Email: kiyotaka@ka-lab.ac

概要:

誤り検出および訂正に関する研究では, n-gram またはそれを拡張した方法を用いた方法が盛んに研究されている. これらの方法はとても速く実行できるが, 欠点もある. それは現在の解析文字位置から n 文字以上前の文字の情報が利用できないためである. このため, これらの欠点をを克服する新しい誤り検出および訂正手法が期待されている.

本論文では, 日本語文章のための形態素解析および構文解析を用いた新しい誤り検出および訂正手法を提案する. この手法は, 少なくともユーザが定義した文法に適合するようないくつかの解析結果を出力し, n-gram による誤り検出および訂正手法の欠点を補うだろう.

An Error Detecting and Correcting Method for Japanese Texts Based on Context-Free Grammar

Kiyotaka ATSUMI

Department of Systems Engineering, Shizuoka University

Email: kiyotaka@ka-lab.ac

Abstract:

Error detection and correction methods using n-gram methods or their extensions have been intensively studied. These methods are very fast, but they have some disadvantages because they use information only in a 'window of a constant size', i.e., they do not consider a string read before n characters and a string which will be read from the current position. Therefore, new error detection and correction methods are desired to overcome this disadvantage.

This paper proposes a new error detection and correction methods for Japanese texts using morphological and syntactical. At the least, this method can output some results adapted a grammar defined by user, and it may complement the disadvantage of n-gram methods.

1 はじめに

コンピュータで自然言語処理を行う場合, これまでの研究では基本的に誤りを含まないことを前提に行われている. しかし, まれではあるが, 新聞記事などにも誤りが混入しており, それが理由で正しい解析が行われない場合などがある. また, 文字認識システムや音声認識システムなどで入力された文や, 一般ユーザがキーボードから入力した文にも, 誤りが含まれている. このような誤りを事前に, またはシステムに組み込んだ上で処理することが望まれているが, それが実現されているシステムは少ない.

誤り検出・訂正自体の研究では n-gram を用いた方法 [1, 2, 3] が盛んに研究されている. n-gram を用いる手法は誤り検出の速度の点では優れているが, tri-gram 程度では, 数文字連続した誤りがあった場合や, たまたま高い接続確率で誤り文字が他の文字と隣接したすると, 誤りを見逃してしまう.

このため, n-gram とは異なったアプローチによる誤り検出・訂正システムを開発することにより, 相互に欠点を補うことができるのではないかと考え, 文脈自由文法を用いた誤り検出・訂正システムの開発を行う. このシステムでは少なくとも文法的には正しい解析結果を得ることが期待できる.

本論文では, 誤り検出および訂正のための形態素解析と構文解析のアルゴリズムについて述べる.

2 諸定義

この節では, 誤り検出および訂正システムに必要な一般的な定義を行う.

2.1 誤りの定義

コンピュータに入力された文に含まれる誤りとは, コンピュータに入力される前の原文と比較して異なる部分であるとする. この中には原文にある文字が欠落したり, あるいは, 原文にない文字が挿入されている場合も含む. このような文を非文と呼ぶことにする. 非文に含まれる誤りは, 原文から表 1 の誤り生成関数の合成関数によって生成される仮定しても良いことが, 定理 1 によって証明されている [4].

表 1: 誤り生成関数

種類	誤り生成関数
置換誤り	$F_{sub}(s,i,x) = \cdots c_{i-1}xc_{i+1}\cdots$
挿入誤り	$F_{ins}(s,i,x) = \cdots c_{i-1}xc_{i}\cdots$
欠落誤り	$F_{del}(s,i) = \cdots c_{i-1}c_{i+1}\cdots$

但し, $s = c_1 \cdots c_n$ は入力文, $c_1 \cdots c_n, x$ は任意の文字, i は 1 から $n+1$ までの整数である.

定理 1 全ての非文は 3 種類の誤り生成関数の合成関数によって原文から生成することができる [4].
(証明) 原文の全ての文字を F_{del} 関数により削除し, さらに F_{ins} 関数で任意の文字を挿入すれば良い. □

さて, 置換誤り生成関数 F_{sub} は必要ないように思えるが, 置換誤りは比較的多く, これを挿入誤り生成関数と欠落誤り生成関数の合成関数で表すと, 誤りを同定する作業で効率が悪い. そのために置換誤り生成関数も独立した一つの関数として準備した.

2.2 誤り距離

誤り生成関数を利用すると, 入力された非文と原文との間の距離を定義することができる.

定理 1 で示した通り, どのような非文でも誤り生成関数の合成関数によって原文から生成可能である. しかし, その方法は一般に 1 種類ではない. その中で最も少ない誤り生成関数を用いた合成関数を取り出し, その用いられた誤り生成関数の数を 2 つの文の間の距離と定義する. この考え方はいかなる 2 文の間にも距離を定義することを許している. 当然同一の 2 文の間の距離は 0 となる.

2.3 誤り検出および訂正

誤り検出とは, 入力された非文から誤り生成関数の逆関数を組み合わせた合成関数を用いて原文を推定し, 誤り文字の位置を特定することである. また, 誤り訂正とは, 誤り検出で発見された誤り文字を正しいと推定される文字に置き直すことである.

2.4 誤り検出および訂正の性能評価方法

ある誤り検出および訂正システムが存在するとき, その性能を評価する方法を定義する. 一般にこれらの性能評価には適合率と再現率が用いられるので, それに習い, 本論文の目的に即した形で定義する.

誤り検出に関する適合率と再現率は次のように定義する. まず, コンピュータによる誤り部分の解析結果, 入力文, 原文を比較し, 次の3つのデータを数えあげる.

- D_a = 誤り検出システムが正しく誤りと検出した文字の数
- D_b = 誤り検出システムが誤りと推定し検出した文字の数
- ALL = 全体の誤り文字の数

D_a, D_b, ALL から次の適合率 P_D と再現率 R_D を計算する.

$$P_D = D_a/D_b$$

$$R_D = D_a/ALL$$

次に誤り訂正に関する適合率と再現率を次のように定義する. 誤り検出の場合と同様にコンピュータにより誤り部分を訂正した文, 入力文, 原文を比較し, 次の2つのデータを数えあげる.

- C_a = 誤り検出システムが正しく誤りを訂正した文字の数
- C_b = 誤り検出システムが誤りと推定し訂正した文字の数

C_a, C_b, ALL から次の適合率 P_C と再現率 R_C を計算する.

$$P_C = C_a/C_b$$

$$R_C = C_a/ALL$$

ここで, 適合率と再現率の性質について述べておく. 適合率と再現率はトレードオフの関係にある. 再現率の高いシステムでは多くの誤りを検出, 訂正できるだろうが, 多くの過剰な誤り検出, 訂正を含むかも知れない. また, 適合率の高いシステムでは, 過剰な誤り検出, 訂正は抑制されるが, 多くの誤り文字を見逃しているかもしれない. そのため誤り検出, 訂正システムでは, 再現率と適合率が同時に高くなるように設計されなければならない.

また, 誤り訂正に関する適合率と再現率は, 誤り検出に関する再現率と適合率とは一致しないだろう. なぜならば, 誤り文字の位置を正しく指摘したとしても, そこで適用する誤り生成関数の逆関数が正しく選択されるとは限らないからである. このため一般には誤り検出で得られる適合率, 再現率よりも誤り訂正で得られる適合率, 再現率の方が低くなるだろう.

3 誤り検出および訂正のアルゴリズム

誤り検出および訂正は2.3節で定義したように, 非文から誤り生成関数の逆関数を組み合わせて原文を推定することになるが, やみくもに関数を組み合わせても原文に近付くことは出来ない. 文献[4]は, 原文を生成できるような文法を定義し, その文法に3つの誤り生成関数を誤り生成規則として組み込み構文解析をする方法を提案している. この構文解析では, 誤り生成規則の数が最小の解析木を1つだけ出力する. しかし, その解析木が原文を復元するような解析木になっているかどうかは分からない. 文献[5]では, 誤り生成規則の数が最小のものだけでなく, 最小 $+k$ までの範囲にある全ての解析木を出力するように拡張している. 本論文では, 基本的なアイディアとして, 準最小非文訂正法

[5] のアルゴリズムを採用し実際の日本語文を解析するために必要な処理を追加変更することとする.

3.1 アルゴリズムの概略

まず, 原文を網羅する TRIE 構造の単語辞書と日本語文法を準備する. この文法は文脈自由文法でなければならなず, 前終端記号は単語辞書に登録されている品詞の集合であるとする. この文法を文献 [4] に従って非文訂正のための文法に変換しておく. これは 3.2 節で述べる.

次にコンピュータに入力された非文を形態素解析する. 正文の入力を前提としている形態素解析では正しく単語を分割することができない. そこで, すべての部分文字列に対して辞書見出し語と比較し, 単語内の文字単位の誤りについて処理を行って, その結果を出力する. これは 3.3 節で述べる.

形態素解析の結果を入力として, 文献 [5] のアイディアを用い, 文法的な制約により, 単語単位の誤りについて処理を行う. ただし, そのままでは利用できないため, 本論文の目的に合わせて拡張を行う. ここでは与えられた日本語文法で受理可能な文に変換できる解析木を複数出力することになる. これは 3.4 節で述べる.

構文解析から得られる結果には, 入力された非文から, 最初に与えられた日本語文法で受理可能な文に変換するための方法が示されている. この解析結果の処理については 3.5 節で述べる.

3.2 非文訂正のための文法の拡張

原文を網羅する文脈自由文法 $G = (N, C, \Sigma, P, S)$, N は非終端記号の集合, C は前終端記号の集合, Σ は終端記号の集合, P は生成規則の集合, S は出発記号について考える. この文法 G に次のような手順で非終端記号と生成規則を追加し, 新しい文法 $G' = (N', C, \Sigma, P', S')$ を定義する [4].

アルゴリズム 1 非文訂正文法への変換

1. $N' = N \cup \{S', H, I\} \cup \{E_a | \forall a \in C\}$.
2. 生成規則 $S' \rightarrow S$, $S' \rightarrow SH$, $H \rightarrow HI$, $H \rightarrow I$ を P' に追加する.
3. もし生成規則 $A \rightarrow \alpha_0 b_1 \alpha_1 b_2 \alpha_2 \cdots b_m \alpha_m$, $m \geq 0$, ここで $\alpha_i \in N^*$, $b_i \in C$ が P に含まれていれば, 生成規則 $A \rightarrow \alpha_0 E_{b_1} \alpha_1 E_{b_2} \alpha_2 \cdots E_{b_m} \alpha_m$ を P' に追加する.
4. 全ての終端記号 a について, 以下の生成規則を P' に追加する.
   (a) $E_a \rightarrow a$,
   (b) $E_a \rightarrow b$ for all $b \in C$, $b \neq a$ (置換誤り生成規則),
   (c) $E_a \rightarrow Ha$,
   (d) $I \rightarrow a$ (挿入誤り生成規則),
   (e) $E_a \rightarrow \varepsilon$, ここで ε は空列 (欠落誤り生成規則).

ステップ 4b, 4d, 4e で追加された生成規則は誤り生成規則としてマークしておく.

3.3 形態素解析

ここでは, 入力された全ての部分文字列に対して, 辞書中の全ての見出し語と比較し, 一定の誤り距離以下 (本論文では 2) の見出し語を解析結果として出力する. このような比較は非常に多くの重複作業を含むため, うまく省略する必要がある. そのアルゴリズムを説明する.

まず, 与えられた辞書を 1 つの有限状態オートマトンであると考える. この有限状態オートマトンの例を図 1 に示す. 説明の都合上, この有限状態オートマトンの辺は TRIE 構造に合わせて水平方向の辺と垂直方向の辺とを区別することとする.

この有限状態オートマトンに誤り生成関数にあたる辺をアルゴリズム 2 で追加する.

アルゴリズム 2 TRIE 構造辞書から誤り訂正オートマトンへの変換

1. 全ての辺に付されている a について, $a/a/0$ と置き換える. これは入力文字が a のときに a と 0 を出力することを意味する. 垂直方向の辺に付されている ε の場合は $\varepsilon/\varepsilon/0$ と置き換えることになる.

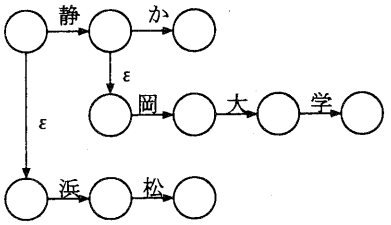

図 1: 有限状態オートマトンで表現された TRIE 構造辞書の例

2. 全ての節点について, $./\varepsilon/1$ を付したその節点自身を指す辺を追加する. これは挿入誤りに対応した辺である. "." は入力文字が任意の文字であることを意味する.
3. 全ての水平方向の辺に付されている a について, $\varepsilon/a/1$ を付した辺を追加する. これは欠落誤りに対応した辺である. 入力文字がなくても a と 1 を出力する.
4. 全ての水平方向の辺に付されている a について, $\sim a/a/1$ を付した辺を追加する. これは置換誤りに対応した辺である. $\sim a$ は a 以外の任意の文字をあらわす. 入力文字が a 以外だった場合, a と 1 を出力する.

図 1 で示した例を基にこのアルゴリズムを適用すると, 図 2 のような有限状態オートマトンとなる. これを誤り訂正オートマトンと呼ぶことにする.

この誤り訂正オートマトンに, 単純に全ての部分文字列を入力として解析させると, 入力列の長さ n, TRIE 構造辞書に登録されている文字数 m のとき, 文字の比較回数が $O(n^2m)$ となり効率が悪い. そこで, 入力文字列を部分文字列に分解しないで, 誤り訂正オートマトンに与えて, 効率良く解析するアルゴリズムを述べる.

誤り訂正オートマトンを 1 つのグラフと捉え, その中を入力列に従って探索する. まず, 次のような項を定義する.

$$[p, q, \{d_1 \cdots d_k\}]$$

ここで, p は入力文字列の位置, q は誤り訂正オートマトンの節点のポインタ, k はこの項が

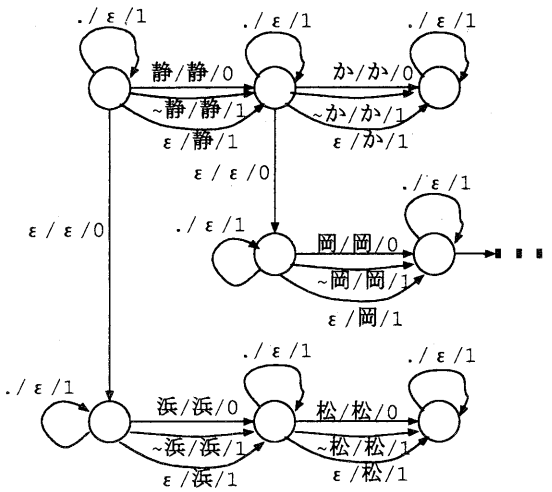

図 2: 誤り訂正オートマトンの例

表 2: $[5, q, \{2, 1, 0\}]$ が示す部分文字列

部分文字列	誤り距離
私は静岡大	2
は静岡大	1
静岡大	0

先頭から何文字分までの部分文字列を扱っているかを示し, d_i は i 文字目から始まる部分文字列のときの現在の入力文字列の位置までの誤り距離を示す.

例えば, 入力文字列が「私は静岡大学の教官です. 」であるとして, ある項が

$$[5, q, \{2, 1, 0\}]$$

であるならば, この項は誤り訂正オートマトンの状態 q において, 表 2 の部分文字列を処理中であり, それぞれの部分文字列に誤り距離が付されていることを示す.

次に具体的な探索アルゴリズムを示す. *openlist* と *closelist* は集合である. それぞれ項を追加する時点で, 入力文字列の位置とオートマトンの状態を指しているポインタが同一である項が既に *list* の中に存在するとき, *openlist*

ではより小さな誤り距離を持つ項を *openlist* の中に残すこととし, *closelist* では同一項とみなして, 追加しないこととする. また, *openlist* に項を追加する場合は一度 *closelist* に入力文字列の位置とオートマトンの状態を指しているポインタが同一な項がないことを調べてから, *openlist* に項を追加することとする.

アルゴリズム 3 形態素解析

1. $openlist = \{[0, \hat{s}, \{0\}]\}$, $closelist = \{\}$ を用意する. ここで $\hat{s}$ は誤り訂正オートマトンの初期状態とする.
2. *openlist* が空になるまで, 次の処理を実行する.
   (a) *openlist* から 1 つ項を取り出し, $node = [i, s, \{d_1, d_2 \cdots d_k\}]$ とする.
   (b) オートマトンの状態 s から遷移可能な状態のリストを $\{s_1, s_2, \cdots\}$ として, 以下の処理を行う. ここで, x は任意の文字, d は 0 または 1 とする.
      i. *node* の状態の次の状態を指す辺が自状態以外 1 つもない場合, 次のフォーマットで出力し, ステップ 2c へ行く.

         $[j, i, d_j,$ 見出し語の品詞, 見出し語$]$

         この出力フォーマットは入力文字列の j 文字目から i 文字目までが誤り距離 d_j 個で見出し語となり, その品詞を示している.
      ii. *node* の状態が $s = \hat{s}$ のとき, $./\varepsilon/1$ が付されている辺に従い, $node' = [i+1, \hat{s}, \{d_1+1, d_2+1, \cdots d_k+1, 0\}]$ を *openlist* に追加する. ここで複数の部分文字列に同時に対応する.
      iii. *node* の状態が $s \neq \hat{s}$ のとき, $./\varepsilon/1$ が付されている辺に従い, $node' = [i+1, \hat{s}, \{d_1+1, d_2+1, \cdots d_k+1\}]$ を *openlist* に追加する.
      iv. 状態 s_i に遷移する辺に付されている情報が $x/x/0$ で, かつ入力文字列の i 番目の文字が x ならば, $node' = [i+1, s_i, \{d_1, d_2, \cdots d_k\}]$ を *openlist* に追加する.
      v. 状態 s_i に遷移する辺に付されている情報が $\sim x/x/1$ で, かつ入力文字列の i 番目の文字が x でないならば, $node' = [i+1, s_i, \{d_1+1, d_2+1, \cdots d_k+1\}]$ を *openlist* に追加する.
      vi. 状態 s_i に遷移する辺に付されている情報が $\varepsilon/x/d$ ならば, $node' = [i, s_i, \{d_1+d, d_2+d, \cdots d_k+d\}]$ を *openlist* に追加する.
   (c) *node* を *openlist* から削除し, *closelist* に加える.

このアルゴリズムの厳密な計算量は, 誤り距離の計算などもあるため, $O(n^2m)$ と素朴な方法と変わらないが, 文字の比較回数を $O(nm)$ にまで減らすことができたので, 大幅に処理速度が上がる.

3.4 構文解析

次に誤り訂正のために拡張した文法 $G' = (N', C, \Sigma, P', S')$ を用いて, Earley 法 [6] を誤り訂正のために拡張した方法 [5] で構文解析を行う. Earley 法と異なる点として, 入力文字列を 1 文字読み込むというステップの代わりに, 形態素解析からの出力結果を確定した部分解析木として読み込むことである.

形態素解析への入力列が $c_1 c_2 \cdots c_n$ であったとして, まず, 解析テーブル $I_0, \cdots I_n$ と呼ばれる集合に生成する項について説明する. 解析テーブルに生成する項は次のような形式である.

$$[A \to \alpha \bullet \beta, p, \{e, b_0 \cdots b_k\}] \in I_q$$

ここで, $A \to \alpha\beta$ は生成規則, $\bullet$ はそこまで解析が完了していることを示すメタ記号, p はこの項の処理が入力列のどこから開始されたかを示すポインタ, e はこの項を生成するために必要な最小の誤り生成規則の使用回数, b_i は 0 または 1 の数字で, 1 ならば, $e+i$ 個の誤り生成規則を使用してもこの項は生成可能であることを示す. k は最小+k 回までの誤り生成規則の使用を許すことを意味する.

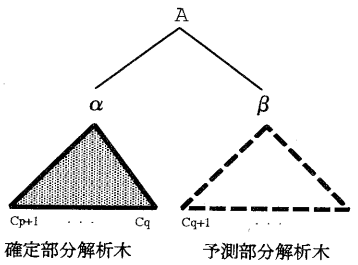

図 3: $[A \to \alpha \bullet \beta, p, \{e, b_0 \cdots b_k\}] \in I_q$ が示す部分解析木

この項は, $\{e, b_0 \cdots b_k\}$ で示された誤り生成規則数を使用して図3に示すような部分解析木を生成したことを意味する.

次に具体的なアルゴリズムについて述べる.

アルゴリズム 4 構文解析

1. 形態素解析に入力した文字列の長さ n に合わせて, 解析テーブル $I_0 \cdots I_n$ を準備する.
2. 形態素解析の出力結果 $[l, m, d, a,$ 見出し語 $]$ のそれぞれについて対応する項 $[a \to$ 見出し語$\bullet, l-1, \{d, 10 \cdots 0\}]$ を解析テーブル I_m に生成する.
3. $[S' \to \bullet S, 0, \{0, 10 \cdots 0\}]$ と $[S' \to \bullet SH, 0, \{0, 10 \cdots 0\}]$ を解析テーブル I_0 に生成する.
4. 以下の処理を I_0 から I_n まで順番に行う. 各解析テーブルでは, 新しい項が生成できなくなるまで処理を続け, 次の解析テーブルの移る.
   (a) 現在処理中の解析テーブル I_i に存在する $[B \to \gamma\bullet, p, \{e, b_0 b_1 \cdots b_k\}]$ のような全ての項について, それぞれの項に対応する $[A \to \alpha \bullet B\beta, q, \{e', b'_0 b'_1 \cdots b'_k\}]$ のような項が解析テーブル I_p に存在するならば, $[A \to \alpha B \bullet \gamma, q, \{e'', b''_0 b''_1 \cdots b''_k\}]$ を I_i に生成する. ここで, $B \to \gamma \in P'$, $\gamma \in \{N' \cup C\}^*$. このとき $\{e'', b''_0 b''_1 \cdots b''_k\}$ の計算方法はアルゴリズム 5 で述べる. ただし, $B \to \gamma$ が誤り生成規則ならば, 誤り距離を示すリストはいつでも $\{1, 10 \cdots 0\}$ であるとする.
   (b) 現在処理中の解析テーブル I_i に存在する $[A \to \alpha \bullet B\beta, q, \{e, b_0 b_1 \cdots b_k\}]$ のような全ての項について, それぞれの項に対する $[B \to \bullet\gamma, i, \{0, 10 \cdots 0\}]$, $\forall B \to \gamma \in P'$ を解析テーブル I_i に生成する.

2つの項を組み合わせるときの, 誤り距離の計算方法は次のアルゴリズムに従う.

アルゴリズム 5 誤り距離の計算方法

1. $e'' = min\{e,\ e'\}$,
2. $e < e' - k$ の場合, $b''_1 = b_1, \cdots,\ b''_k = b_k$
3. $e > e' + k$ の場合, $b''_1 = b'_1, \cdots,\ b''_k = b'_k$
4. $e < e'$ の場合, $s = e' - e,\ b''_1 = b_1, \cdots,$
   $b''_s = b_s \vee 1,\ b''_{s+1} = b_{s+1} \vee b'_1,$
   $b''_{s+2} = b_{s+2} \vee b'_2, \cdots, b''_k = b_k \vee b'_{k-s}$
5. $e > e'$ の場合, $s = e - e',\ b''_1 = b'_1, \cdots,$
   $b''_s = b'_s \vee 1,\ b''_{s+1} = b'_{s+1} \vee b_1,$
   $b''_{s+2} = b'_{s+2} \vee b_2, \cdots,\ b''_k = b'_k \vee b_{k-s}$
6. $e = e'$ の場合, $b''_1 = b_1 \vee b'_1,$
   $b''_2 = b_2 \vee b'_2, \cdots,\ b''_k = b_k \vee b'_k$

3.5 解析結果の後処理

解析結果は $[S' \to \gamma\bullet, 0, \{e, d_0 \cdots d_k\}] \in I_n$ から解析木を辿ることで得ることができる. この解析木は誤り生成規則の使用回数によって優先度が決まっている. しかしながら, 唯一の解析結果に絞り込めるわけではないので, ここで扱う手法以外の別の観点からこれらの解析結果を絞り込む必要がある.

ある解析結果を選ぶことができたと仮定して, その解析結果には誤りの位置や誤りを訂正する方法が全て含まれている.

まず, 前終端記号の生成に誤り生成規則が使われていない場合は, その終端記号が指示する形態素解析の結果に従って文字列を修正すればよい.

前終端記号の生成に誤り生成規則が使われている場合は, 具体的な文字列の修正は困難である. 特定の品詞を持つ見出し語はたいてい複数個存在するにもかかわらず, 形態素解析の結果では, それらの見出し語を指示しなかったの

で, その中から選択することが出来ないためである. この論文では誤り位置とそこにあるべき品詞を指示するのみにとどめ, 訂正は行わないこととする.

4 今後の課題

現在ruby[1] という言語を用いて, コーディングを行っている. 文法についてはIPALの日本語辞書[2] とICOTの形態素辞書[3] の品詞情報を参考に導出規則数1000前後を目安に構築予定である.

実際にプログラムを動作させてみないと, 2.4節で述べた適合率, 再現率について述べることは出来ないが, 最初に準備する日本語文法の曖昧性が影響を与えることになる. もし, 曖昧性の非常に少ない文法を準備できれば, 高水準の適合率, 再現率が実現できるのではないかと思う.

本論文で提案した手法の問題点として次の3点が考えられる.

第1点は, 文法を文脈自由文法の書式に制限したことである. 日本語は文節単位での位置の入れ替わりは比較的自由に行われるため, 拡張格文法で定義した方が現実的であるかもしれない.

第2点は, 動作速度である. Earley法は構文解析手法の中では遅い方である. 文法の定義に合わせて一般化LR構文解析などを検討する必要がある. また, 形態素解析についても, 正文を前提とした形態素解析に比べてかなり時間がかかる. この部分についても高速化のために別の方策が必要である.

第3点は, 第2点にも関係しているが, 常に誤りを仮定することが現実的かどうかということである. 本論文の手法ではどの位置に誤りがあっても処理できるが, そのために動作速度を犠牲にしている. 誤り位置の同定に, もっと計算量の少ない別の手法, 例えば *tri*-gram などを用い, その手法で発見した誤り位置付近のみを誤り検出, 訂正するように変更することにより, 全体としてより現実的な速度で動作する誤り訂正システムを構築する必要がある.

[1] http://www.netlab.co.jp/ruby/jp/

[2] http://www.ipa.go.jp/STC/NIHONGO/IPAL/ipal.html

[3] http://www.icot.or.jp/ARCHIVE/HomePage-J.html

参考文献

[1] Araki, T., Ikehara, S., Tsukahara, N. and Komatsu, Y.: An Evaluation to Detect and Correct Erroneous Characters Wrongly Substituted, Deleted and Inserted in Japanese and English sentence using Markov Models, in *COLING 94 Proceedings Vol.II*, pp. 187–193 (1994).

[2] 竹内孔一, 松本裕治 : 統計的言語モデルを用いたOCR誤り訂正システムの構築, 情報処理学会論文誌, Vol. 40, No. 6, pp. 2679–2689 (1999).

[3] 新納浩幸 : 平仮名N-gramによる平仮名列の誤り検出とその修正, 情報処理学会論文誌, Vol. 40, No. 6, pp. 2690–2698 (1999).

[4] Aho, A. V. and Peterson, T. G.: A Minimum Distance Error-Correcting Parser for Context-Free Languages, *SIAM J.Comput*, pp. 305–312 (1972).

[5] 渥美清隆, 増山繁:構文解析上の自由度をもった非文訂正法の一提案, 電子情報通信学会論文誌, Vol. J76-D-I, pp. 686–688 (1993).

[6] Earley, J.: An Efficient Context-Free Parsing Algorithm, *Comm. ACM*, pp. 94–102 (1970).