日本語OCR文における英字・カタカナのスペル誤り訂正法

畑田 稔[†] 遠藤裕英[†]

OCR (Optical Character Recognition) 装置を通して入力された文は，通常，誤字，脱落あるいは誤挿入の誤りを多く含んでいる．本論文では，日本語OCR文で特に認識誤りの多い英字とカタカナを対象としたスペル誤り訂正法を述べる．この方法は，入力単語と辞書中の全単語間のハッシュ距離計算による1次候補単語選択と，入力単語と各候補単語間のレーベンシュタイン距離計算による最終候補単語選択の2つのステップからなる．ハッシュ距離は，文字列に対応付けられた整数値を用いて単語間の類似性を調べるもので，「ハッシュ距離 $\leq$ レーベンシュタイン距離」の関係が成り立つ．辞書および距離計算管理データはフラットな1次元配列構造として，スペル誤りの多い文に対応できるようにした．レーベンシュタイン距離が同じときは，辞書単語の使用頻度の大きい順に候補単語を提示することにより，スペル誤り訂正能力の向上を図った．数値例によって，提案手法の有効性を検証した．ハッシュ距離はレーベンシュタイン距離のおよそ50倍の速さで計算される．約11,700語の辞書で，スペル誤り数が1～8の場合，候補単語選択時間の平均値は11～40 msであった．誤り率40%のとき，ヒット率（正しい単語が候補の先頭）およびトップ5率（正しい単語が候補の5番目までにある）は，それぞれ73～93%および89～98%であった．

Spelling Correction Method for English and Katakana in Japanese OCR Text

MINORU HATADA[†] and HIROHIDE ENDOH[†]

Usually, OCR (Optical Character Recognition) devices introduce lots of errors of substitutions, deletions, and insertions. This paper describes a spelling correction method for English and Katakana in Japanese OCR text. The proposed method consists of two steps: selection of preliminary candidate words by calculation of the hash distance between the input word and each word in the dictionary, and selection of final candidate words by calculation of the Levenshtein distance between the input word and each word among the preliminary candidate words. The hash distance is a measure of similarity of two words, which uses a hash value mapped from string, and it satisfies the property hash-distance $\leq$ Levenshtein-distance. This method uses the dictionary of first-order array structure, which has no restriction on the number of spelling errors. If Levenshtein distances are equal each other, candidate words are sorted by their frequencies of use. This improves the ability of spelling correction. Examples are given to show the effectiveness of the proposed method. For the word-length 6 to 8 byte, the calculation time of hash distance was reduced to one fiftieth of that of Levenshtein distance. In case which the dictionary includes about 11,700 words and the numbers of spelling error per word are one to eight, the average computing time of selecting candidate words were 11 to 40 ms. For error rate 40%, hit ratio (probability that correct word is the top of candidate words) and top five ratio (probability that correct word is among the five candidate words) were respectively 73 to 93 % and 89 to 98%.

1. はじめに

スペル誤りのチェックおよび自動訂正の研究は古く，英文で30年，和文で10年以上の歴史があるが[1),2)]，今日もなお研究が続いている．それぞれの手法は用途が限られ，また誤り訂正能力に制約があり，OCR (Optical Character Recognition) などでは，スペル誤りの訂正に要する工数は今なお，多大なためである．また，近年，ディジタル図書館の機運が高まり，紙でしか存在しないもののディジタル化手段として，OCRの新たな需要が生まれた．

従来のスペルチェック・訂正の研究では，1つの単語に含まれる文字誤りは少ないこと（せいぜい2，3個）が前提になっていることが多い．しかし，英字を含む和文OCRでは，長さ10文字の英単語に4，5文

† 日立製作所システム開発研究所情報センタ
Systems Developments Laboratory, Information System Center, Hitachi, Ltd.

字のエラーが含まれていることが珍しくない．

本論文は，日本語OCR文において認識エラーの多い英字，カタカナを対象としたスペル誤りの訂正法を提案したものである．プログラムの開発にはオブジェクト指向プログラミング言語C++を使用した．

2. スペル誤りのチェックと訂正

2.1 OCRにおけるスペル誤りの特徴

近年，和文OCR技術は進んでおり，漢字，ひらがなの認識誤りは比較的少なく，エラーのほとんどが可変字間ピッチの英単語で発生する．また，カタカナでは,「パ」と「バ」,「エ」と「ェ」のような誤りが多い．カタカナと同様，ひらがなでも,「ぴ」と「び」などで誤りが起こりやすいが，カタカナに比べると，このような文字の使用が少ないため，相対的に，ひらがなの誤りは少ない．

数年前までは，パソコンとかワープロの印刷は全角，半角文字が中心であり，プロポーショナルな英字印刷が行われるようになったのは，最近のことである．このような印刷文書については，和文OCRは従来以上に難しくなり，認識エラーが増大している．

キー入力とOCRのスペル誤りの共通点としては，誤字（ある文字を別の文字に間違えている），脱落（文字が抜けている），誤挿入（余分な文字が入っている）がある．相違点としては，キー入力では，転置（連続した2つの文字が入れ替わっている）がよくあるが，元の文書自体に含まれる誤りを抜きにすれば，OCR段階では原理的に転置エラーは起こらない．キー入力では英単語の一部の文字を日本語に間違えることはほとんどないが，OCRでは頻発する．また，印字品質に依存するが，OCRでのスペル誤りは，キー入力に比較すると，はるかに多い．

2.2 専門文書で使われる単語数

スペル誤りのチェック・訂正では，通常，辞書（単語のリスト）が用いられる．著者らは，全体で約200万文字のコンピュータ関連のニュースレターでの単語の使用状況を調査した．ここでは，約11,700語が延べ約16万回使用されている．単語の長さの平均は8.9バイト，使用頻度による加重平均は7.7バイトである（カタカナ1文字は2バイト）．累積使用頻度では，僅か100語で47%に達し，1,000語では，80%を超える．また，別の専門文書（英文）の調査では，200語で50%，次の2,000語で95%，さらに20,000語で100%という報告がある[5]．

これらの調査結果から，専門文書のスペル誤りのチェック，訂正にはそれほど大規模な辞書は必要としないことが分かる．

2.3 類似度の測定

スペル誤りの訂正法の研究では，文字列間の相違の程度を表すものとして，レーベンシュタイン距離がしばしば用いられる[2),7)]．

正しい単語を表す文字列 $X = x_1, x_2, \ldots, x_m$ とスペル誤りのある文字列 $Y = y_1, y_2, \ldots, y_n$ の間の類似度を表すレーベンシュタイン距離 $d(m,n)$ は，次の式の繰返し計算で求めることができる．

$$d(i,j) = \min\{d(i-1,j-1) + w_s(i,j),$$
$$d(i,j-1) + w_i,\ d(i-1,j) + w_d\}$$
$$d(i,0) = i \cdot w_d, d(0,j) = j \cdot w_i,$$
$$w_s(i,j) = 0 \quad (x_i = y_j)$$
$$= w_s \quad (x_i \neq y_j)$$

ここで，w_s，w_i，w_d はそれぞれ，誤字，誤挿入，脱落の重みである．

文字列 $x_1, x_2, \ldots, x_i$ と文字列 $y_1, y_2, \ldots, y_j$ を比較するとき，項「$d(i-1,j-1)+w_s(i,j)$」は，文字列 $x_1, x_2, \ldots, x_{i-1}$ と文字列 $y_1, y_2, \ldots, y_{j-1}$ が対応し，文字 x_i と文字 y_j が対応しているケースである．項「$d(i,j-1)+w_i$」は，文字列 $x_1, x_2, \ldots, x_i$ と文字列 $y_1, y_2, \ldots, y_{j-1}$ が対応しており，正しい単語には y_j に対応する文字がない，すなわち，誤った文字 y_j が挿入されているケースである．項「$d(i-1,j)+w_d$」は，文字列 $x_1, x_2, \ldots, x_{i-1}$ と文字列 $y_1, y_2, \ldots, y_j$ が対応しており，誤ったスペルには x_i に対応する文字がない，すなわち，文字 x_i が脱落しているケースである．レーベンシュタイン距離は，転置に対しても拡張されている[2)]．キー入力では，転置エラーは発生しやすいが，イメージ入力に対するOCRでは，原理的に起こらないため，本論文では転置エラーに対する項は除外している（転置エラーは誤字が2回連続したものと同等）．また，誤字，誤挿入，脱落の重み w_s，w_i，w_d はすべて1としている．

2.4 類似単語の取り出し

辞書にないスペルWに似た文字を辞書中に探すには，原理的には，以下のようにすればよい．

[候補単語選択基本アルゴリズム]

```
W: 入力文字列（誤ったスペル）
N: 辞書中の単語数
Cand=∅（空集合）: 候補単語の集合
for(d=1; |Cand|==0; d++){
   for(n=0; n<N; n++){
        if(Dist(X[n],W)==d)
            Cand=CandU{X[n]};
   }
```

```
}
```

ここで，|Cand|は集合Candの要素数，Dist(X[n],W)は2つの文字列 W と $X[n]$ の間の距離，X[n]は辞書中の第 n 番目の単語を表す．

このアルゴリズムでは，少なくとも1回は，辞書中のすべての単語について入力文字列とのレーベンシュタイン距離計算を行う．

文字列 $x_1, x_2, \cdots, x_m$ と $y_1, y_2, \cdots, y_n$ のレーベンシュタイン距離 $d(m,n)$ の計算量は $m \times n$ のオーダになるため，辞書中のすべての単語について距離計算を行うには，時間がかかりすぎる．このため，距離計算回数を低減する研究が以前から種々行われている．

キー文字セット法[3]は，単語から使用頻度が少ない順に4文字（重複なし．使用文字数が4未満のときは空白を追加）選んで，キー文字とする．入力単語と辞書中の単語を比較して，4文字中3文字が一致する単語を候補とする．あらかじめ，3文字セットごとに辞書中の単語を指す約3,000個のインデクステーブルを作成しておく．たとえば，入力文字列がexanpleならば，キー文字セットは{x, p, l, n}であり，これから1文字除いた3文字セット{x, p, l}，{x, p, n}，{x, l, n}，{p, l, n}に対する4つのインデクステーブルをたどって候補単語を得る．この場合，{x, p, l}インデクステーブルのエントリの1つがexampleを指している．キー文字セット法は，辞書中の全単語をスキャンするわけではないので，候補単語の選択は迅速に行われる．しかし，この手法はスペル誤りが少ないことが前提になっている．キー文字が2つ以上欠けた（たとえばexampleが誤ってexanbleとなった）場合，正しい候補が得られない（候補単語の中にexampleはない）．

本論文では，スペル誤り数に制限がなく，また，辞書管理テーブルのサイズが，スペル誤り数に依存しない下記のアルゴリズムを提案する．

[候補単語選択アルゴリズム]

```
W: 入力単語（誤ったスペル）
N: 辞書中の単語数
MIN: 候補単語数の下限値
LD[]: レーベンシュタイン距離配列
HD[]: ハッシュ距離配列
C[]: ハッシュ距離による候補単語番号
//Step 1: ハッシュ距離の計算
HW=Hash(W); //W のハッシュ値計算
for(n=0; n<N; n++)
  HD[n]=HashDist(H[n],HW);
//Step 2: 候補単語の選択
Cand=∅（空集合）; //候補単語の集合
m=0;
for(d=1; d<|W| && |Cand|<MIN; d++){
  //Step 2.1: ハッシュ距離による候補選出
  for(n=0; n<N; n++)
      if(HD[n]==d)
         C[m++]=n;
  //Step 2.2: C[] を単語使用頻度降順で並び替え
  qsort(C,m,compare_function);
  //Step 2.3: レーベンシュタイン距離を計算
  for(k=0; k<m && |Cand|<MIN; k++){
     n=C[k]; //単語番号を n にセット
     if(HD[n]==d)
         LD[n]=Dist(X[n],W)
     if(LD[n]==d)
         Cand=CandU{X[n]};
  }
}
```

ここで，|W|は単語 W の長さ，H[n]は辞書中の第 n 番目の単語のハッシュ値，HashDist(X,Y)は2つの文字列 X と Y のハッシュ距離を表す．実際のプログラムは，高速化対策のために，上のアルゴリズムとは少し異なっている．

開発したプログラムでは，通常のスペルチェッカと同様，候補単語をリスト表示して，ユーザが選択する．表示順序は，レーベンシュタイン距離が小さい順とし，距離が等しいものは，使用頻度が高い順とする．

ここで，ハッシュというのは，一般に可変長の文字列に整数を対応付けるものである．単語の高速検索を目的とした場合，「異なる文字列に対してハッシュ値が一致する確率を小さくする」のが，ハッシュ関数の基本条件であるが，スペル誤り訂正におけるハッシュ値は，不等式「ハッシュ距離 $\leq$ レーベンシュタイン距離」を満たすハッシュ距離関数HashDistが作れるものでなければならない[7]．本論文で用いるハッシュアルゴリズムを以下に示す．

[ハッシュアルゴリズム-1]

```
HX=0; //文字列 X のハッシュ値
for(文字列 X のすべての文字 X[i] について){
  if(X[i] が 1 バイト文字)
    HX のビット (X[i] mod 64) をセット;
  else
    HX のビット ((X[i] mod 64)+64) をセット;
}
```

ここで，X[i]は文字列 X 中の第 i 番目の文字を表す．

英字，カタカナがなるべく違うビットに写像される

ように，16バイト（#0～#127ビット）という大きなサイズのハッシュ値を用いている．

辞書中の単語に対するハッシュ値の算出は辞書のロード時に1回行うだけであるが，ハッシュ値に基づく距離計算は毎回行っている．この計算はレーベンシュタイン距離計算より桁違いに高速であるため，全体的な候補選択時間の短縮に寄与する．

ハッシュアルゴリズム-1では，英字とカタカナを別のビットに写像している．英字とカタカナの混在単語がない場合，あらかじめ，辞書のサーチ範囲を英単語あるいはカタカナ単語範囲に限定すれば，英字とカタカナを同じビットに写像してもよい．このときのハッシュアルゴリズムを以下に示す．ハッシュサイズは8バイトとなる．

[ハッシュアルゴリズム-2]

```
HX=0; //文字列Xのハッシュ値
for(文字列Xのすべての文字X[i]について)
    HXのビット(X[i] mod 64)をセット;
```

英字の大文字，小文字を区別せず，また，カタカナを写像したときの重なりが多くなることを許せば，次に示すようにハッシュサイズを4バイトに縮小することができる．

[ハッシュアルゴリズム-3]

```
HX=0; //文字列Xのハッシュ値
for(文字列Xのすべての文字X[i]について)
    HXのビット(X[i] mod 32)をセット;
```

C++言語で扱える整数の最大サイズは4バイトである．したがって，8バイトあるいは16バイトのハッシュ値はそのままでは整数扱いができない．そこで，オブジェクト指向プログラミングの特徴を活かして，新しいデータ型（クラス）HASHを導入し，演算子のオーバライド機能によりビット演算が行えるようにした．その結果，ハッシュ距離関数はハッシュ値のサイズに関係なく，次のような分かりやすい形で実現できる．

[ハッシュ距離関数]

```
HashDist(hX, hY)
{
    hZ=hX & hY; //ビットごとのAND演算
    if(hZ==0)
       return 99;
    nXZ=countBit(hX-hZ);
    nYZ=countBit(hY-hZ);
    return max(nXZ, nYZ);
}
```

ここで，hXおよびhYはそれぞれ文字列 X および文字列 Y に対するハッシュ値を表す．countBit()はビット1の個数をカウントする関数である．

(hX-hZ)は文字列 X にあって文字列 Y にない文字を探し，(hY-hZ)は逆に文字列 Y にあって文字列 X にない文字を探すものである．

hZが0というのは，文字列 X と文字列 Y に共通する文字がないことを意味している．先にあげた定義に従えば，レーベンシュタイン距離はこの場合max(|X|,|Y|)であるが，入力文字列とまったく共通する文字がない単語を候補とするのは不合理であるため，距離を99（実際上，無限大と同じ）として，無条件に候補から除外している．また，後述するように，このスキップ処理は，ハッシュ距離計算時間の大幅な高速化に寄与している．

文字列中の個々の文字は，ハッシュ値の1つのビットに写像されている．ビットが1であるということは，文字列中に，このビットに写像される文字が少なくとも1つ存在することを意味している．ビットが0ということは，このビットに写像される文字は含まれていないことを意味する．したがって，(hX-hZ)のあるビットの値が1であれば，このビットに写像される文字が文字列 X には少なくとも1つ存在するが，文字列 Y には含まれていない．すなわち，誤字か脱落と判断できる．文字列 X の2つ以上の文字が同じビットに写像されるケースがあり，また同じ文字が使われていても，順序が異なるケースをハッシュ値では検知できないため，(hX-hZ)のビット1の個数 nXZ は誤字または脱落の個数に等しいかまたはそれよりも小さい．したがって

$$nXZ \leq \text{誤字，脱落数} \leq \text{誤字，脱落，誤挿入数} = \text{レーベンシュタイン距離}$$

となる．同様に

$$nYZ \leq \text{レーベンシュタイン距離}$$

が得られる．以上のことから，

$$\text{ハッシュ距離} \leq \text{レーベンシュタイン距離}$$

が成り立つ．

ビット1の個数の高速カウントのために，2バイトデータのビット1の個数をテーブル参照で得ている．まず，4バイト演算命令で0か否かを判定して，0でないとき，ビット1の個数カウントを2回のテーブル参照と加算で行う．16バイトサイズのハッシュ値に対しては，この操作を4回繰り返す．

次に，前述のアルゴリズムに基づく，レーベンシュタイン距離計算関数を示す．

[レーベンシュタイン距離関数]

```
Dist(X,Y)
{
  wX[]; //字種・文字コード格納2バイト配列
  wY[];
  for(文字列Xのすべての文字X[i]について){
    wX[i]=X[i]; //下位バイトに文字コード
    if(X[i]がカタカナの第2バイト)
        wX[i]の上位バイトに2をセット;
  }
  文字列Yからwyを作成; //上記と同様
  I=文字列Xの長さ;
  J=文字列Yの長さ;
  for(i=0; i<I; i++)
    for(j=0; j<J; j++)
      D[i+1][j+1]=
            min(D[i][j]+Comp(wX[i],wY[j]),
              D[i+1][j]+1,D[i][j+1]+1);
  return D[I][J];
}
```

ここで，関数 Comp(x,y) は $x=y$ のとき0，そうでなければ1を返す．

パソコンでは，カタカナは2バイトで表現されるが，第2バイト目は英字と一致することがある．このため，レーベンシュタイン距離計算の前に，カタカナの第2バイト目と英字が区別できるデータを作成している．

2.5 データ構造

システムで対応できるスペル誤り数に制約が生まれないように，辞書データ構造および候補単語選択で使う管理データは，シンプルなフラットテーブル構造とした．この結果，データの並び替えおよび検索にC++プログラミングシステムの持つANSI規格のライブラリ関数bsearch，qsort[6]がそのまま利用でき，高性能で，コンパクトなスペルチェッカを開発できた．

メモリ上での辞書データ構造を**図1**に示す．単語は可変長のため，単語文字列格納エリアに順に詰め，そのポインタを単語リスト管理テーブルに持っている．単語ポインタの代わりにオフセットを持つ考え方もあるが，その場合，単語リスト中の単語の検索，並び替えに必要な情報が単語リストでクローズしていないため，ライブラリにあるバイナリサーチ関数bsearch，クイックソート関数qsortが簡単には使えなくなる．

2.6 スペルチェッカのインプリメント

対話型のスペルチェッカでは，ユーザインタフェースが重要な鍵となる．ベースとして，テキストエディタと同様の機能が必要となる．開発システムでは，Microsoft Win32 API（Application Program Interface: Windows NT，Windows 95☆でサポート）のエディット・コントロールを用いて，テキストエディタを実現した．

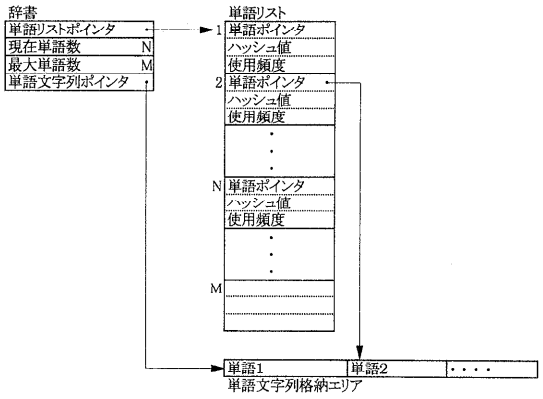


図1 辞書データ構造
Fig. 1 Data structure of dictionary.

ファイルの入出力および文字列の検索・置換のユーザインタフェースは，Win32 APIで用意されたカスタム・ダイアログボックスを使用した．スペルチェッカの基本となる修正候補を提示するダイアログボックスのみ独自設計した．

このようなアプローチにより，土台となるテキストエディタ自体は，約500行で実現できた．その上に，スペルチェッカの基本機能が900行，評価機能が500行程度であり，総計，約1,900行である．

図2にスペルチェッカの画面例を示している．この例では，RISCのI（アイ）が1（エル）に誤っているため，誤りを含む単語として，検出された．検出された単語が上から4行目に表示されるように，自動的にスクロールする．修正候補として，リストボックスに，レーベンシュタイン距離1のRISC，距離2のRS，RSA，RAS，CSCが表示されている．

①第一候補に修正する場合，「修正」ボタンをクリックする．

②第二候補以下を選ぶときは，リストボックスの文字列をクリックして，修正候補とする（ラベルの下のボックスに選んだ語が表示される）．次に「修正」ボタンをクリックする．

③辞書に未登録の正しい単語であった場合には，「辞書に追加」ボタンをクリックする．

④変換もしないし，辞書にも登録しないときは，「無

☆ Windows NT，Windows 95は米国Microsoft Corp.の登録商標である．

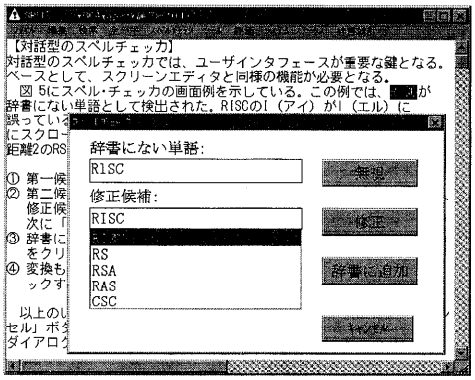


図2 スペルチェッカの画面例
Fig. 2 Display screen of spell checker.

視」ボタンをクリックする.

以上のいずれかの操作を行うと，次の単語の抽出処理に移る.「キャンセル」ボタンをクリックした場合，変換もしないし，辞書にも登録せず，ダイアログボックスをクローズし，スペルチェック処理を完了する.

なお，単語の切り出しは字種（英数字，カタカナ，およびその他の日本語コード）の変わり目の検出を基本としている．英字が漢字に化けているケースなどへの対応から，いくつかの例外処理も行っている.

たとえば,「Info 血 ation」のように，英字の次にひらがな以外の全角文字（日本語コード）が続き，その直後が英字のときは，誤字と判断し，全角文字を含んだ全体を単語として切り出す．ひらがなを除外したのは「Internet と Intranet」のように，2つの英単語にひらがな（助詞）が挟まれるケースが多いためである.

また,「l」（エル）を「1」（数字）に誤るなど，英字から数字への認識誤りに対処するため，単語の切り出しでは，英字と数字は同一字種としている.

3. 実験結果および考察

3.1 実験条件

実験に使用した辞書は，2.1節で述べたコンピュータ関連のニュースレターから抽出し，誤りを正して作成した．単語数は約11,700語（英単語：約5,000語，カタカナ単語：約6,700語）である．また，性能測定に使用したマシンはPentium Pro☆パソコン（180 MHz, 32 MB）である.

訂正精度の評価には，辞書作成に使用したニュースレターの一部（約400 KB）に使われている延べ約9,800語の英単語を用いた．この英単語に挿入，誤字，脱落を等しい確率（各々1/3）でランダムに発生させた文字列を入力単語として，候補単語を選択した.

また，本章の最後にOCRによって得られたデータに対する実験結果を示す.

☆ Pentium Proは米国Intel Corp.の商標である.

表1 ハッシュ距離計算時間
Table 1 Calculating time of hash distance.

ハッシュサイズ（byte）	4	8	16
ハッシュ距離計算時間（ms）	2.3	2.8	7.5
辞書の計算範囲	英字	英字	全体
単語あたりの計算時間（μs）	0.47	0.56	0.64

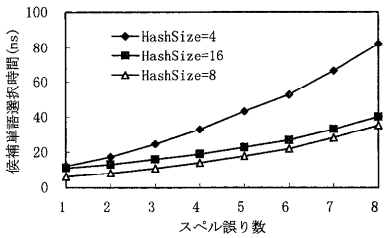


図3 候補単語選択時間とハッシュサイズの関係
Fig. 3 Relationship between candidate-word selection time and hash size.

3.2 ハッシュサイズ

ハッシュ距離計算時間は入力単語によって多少異なる．入力単語Evlnrbaと辞書中の単語との間の実測結果を**表1**に示す．単語あたりの計算時間は，ハッシュサイズが大きいほど大きくなるが，サイズに比例しているわけではなく，差は小さい．これは，ハッシュ距離計算で，2つの文字列の間に共通する文字がないことが明らかになったときは，それ以降の処理をスキップして，処理時間の短縮を図っているが，ハッシュサイズが大きいほどこの効果は大きいためである．ハッシュサイズ4では，大文字と小文字が区別されないため，ハッシュサイズ8よりも効率が低下する．ハッシュサイズ16では，英字とカタカナ間のハッシュ距離計算はすべて処理時間の短いスキップルートをとるため，ハッシュサイズ8よりも高速化の効果が大きい．スキップ処理を除くと，たとえばハッシュサイズ16のときのハッシュ距離計算時間は48%増の11.1 ms（0.95 μs/語）となる.

ハッシュサイズと候補単語選択時間の関係を**図3**に示している（候補単語数下限値 MIN = 5）.

従来の研究では，通常，英字の大文字と小文字を区別していないが，本論文の対象とする文書では，略語，固有名詞が多いため，通常の英文に比較すると，大文字の使用頻度がきわめて高い．このため，大文字と小文字を区別しないハッシュサイズ4のアルゴリズムで

は，ハッシュ距離による候補単語の絞り込みが十分に行われず，レーベンシュタイン距離の計算回数が増える．この結果，ハッシュサイズ8に比べて，候補単語選択時間がおよそ2倍となった．

ハッシュサイズ8とハッシュサイズ16では，レーベンシュタイン距離計算回数は同じであるが，ハッシュ距離計算時間が大きい分，ハッシュサイズ16での候補単語選択時間の方が大きくなっている．その差は誤り数に依存せず，4.9 msである（表1での差4.7 msと多少異なるのは，ハッシュ距離計算時間は入力単語に多少依存するためである）．

以上のことから，英字とカタカナの混在単語を対象にするときは，ハッシュサイズ16バイトが最適で，混在がないときは8バイトが最適といえる．

3.3 候補単語数

前述したように，開発プログラムは，通常のスペルチェッカと同様，候補単語を提示して，ユーザが正しいスペルを選択する（正しいスペルが候補になければキー入力して訂正する）ものであるから，候補単語の先頭でなくても，上位に位置すれば，即座に選択できる．

ヒット率（正しい単語が候補単語の先頭），トップ5率（正しい単語が候補単語の5番目までにある）およびトップ10率（正しい単語が候補単語の10番目までにある）の実測結果を**図4**に示す．ヒット率とトップ5率の差は7～12%と大きいが，トップ5率とトップ10率の差は1～4%と小さい．5単語に比べて10単語となると，目で追って，瞬時に正しい単語を見つけるのもかなり難しくなる．6～10番目に正しい単語がある率は1～4%であり，無駄骨になる率は96～99%ときわめて高い．したがって，アルゴリズムの評価指標としては，ヒット率およびトップ5率が妥当と考えられる．このため，候補単語数の下限値MINの値は5に設定している（トップ10率を測定するときにはMIN = 10とした）．MINの値を5より大きくしても，トップ5率は向上しない．

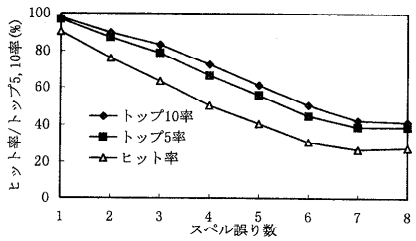


図4 ヒット率とトップ5，10率の比較
Fig. 4 Comparison among hit ratio, top five, top ten ratios.

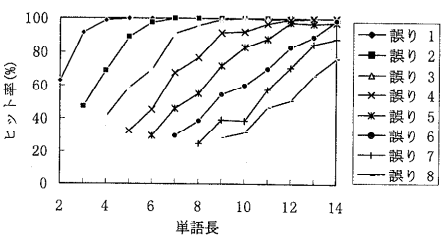
3.4 スペル誤り訂正能力の評価

図5は，スペル誤り数をパラメータとして，グラフ表示したものである．ここで横軸は元の単語の長さであり，スペル誤りを含んだ語の長さではない．一見すると，訂正能力が低いと感じるが，これは誤りがきわめて多い状態から測定しているためである．たとえば，誤り数4，単語長5の場合，ヒット率は21%，トップ5率は33%という低い値であるが，これは5文字の単語に4つのエラーが混入したケースであるから，当然の結果である．

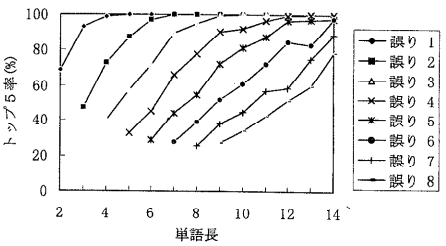
そこで，同じ実験結果について，誤り率（エラー数を元の単語の長さで割ったパーセンテージ）25%，33%，40%，50%をパラメータとしてグラフ表示したのが**図6**である．

誤り率50%では，単語の長さ2～14の範囲で，ヒット率は49～67%，トップ5率は63～88%であり，正しい単語が候補に含まれないことがかなりある．誤り率40%ではヒット率はおよそ73～93%，トップ5率は89～98%であり，実用上十分なスペル誤り訂正能力といえる．

なお，スペル誤り語が，たまたま辞書に存在すると，その誤りが検出できない．今回の辞書にはA，B，Cなど1文字も単語として持っている．したがって，たとえば，2つの大文字からなる略語で，1文字脱落エラーが起こると，検出不能エラーとなる．ランダムに

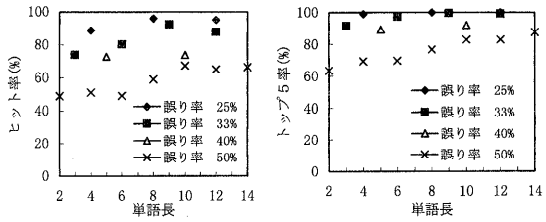


(a) ヒット率



(b) トップ5率

図5 ヒット率およびトップ5率
Fig. 5 Hit ratio and top five ratio.



(a) ヒット率　　(b) トップ5率

図6 ヒット/トップ5率とスペル誤り率との関係
Fig. 6 Relation between hit ratio/top five ratio and spell error ratio.

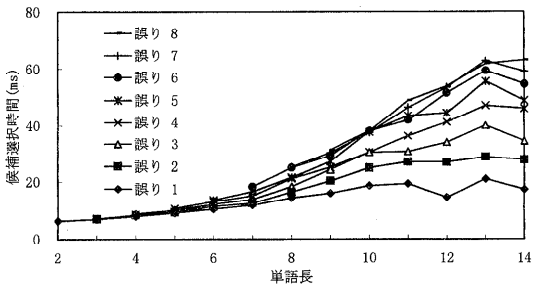


図8 候補単語選択時間
Fig. 8 Candidate-word selection time.

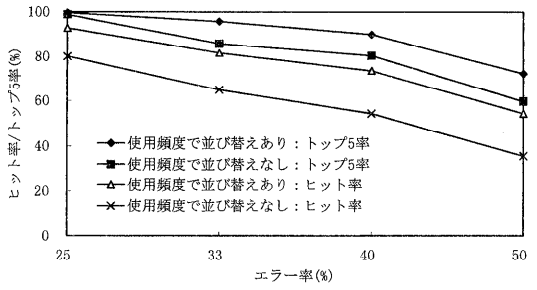


図7 並び替えの効果
Fig. 7 Effect of sorting.

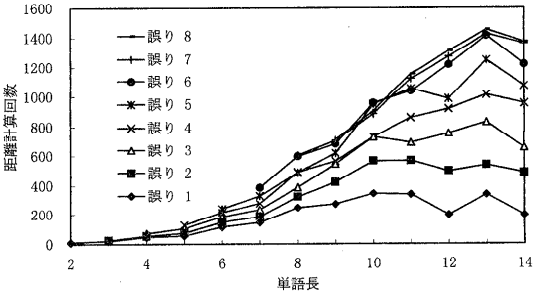


図9 レーベンシュタイン距離計算回数
Fig. 9 The number of calculations of Levenshtein distance.

エラーを発生させた場合，このような検出不能エラーが，誤り数1，2，3に対して，4%，3%，1%発生した．誤りが多いほど検出不能エラーが少なくなり，誤り数が4以上では，1%未満である．しかし，大文字の認識エラーは少ないため，実際のOCRデータでは検出不能エラーはそれほど起こらない．

3.5 使用頻度による並び替えの効果

本アルゴリズムでは，候補単語の提示順序は，レーベンシュタイン距離が小さい順とし，同一距離では使用頻度が高い順としている．使用頻度による並び替えの有無によるヒット率およびトップ5率の違いを**図7**に示している．使用頻度による並び替えにより，ヒット率がおよそ13～20%向上している．エラー率が25%のときは，トップ5率はともに99%を超えているため，並び替えの効果は僅少であるが，エラー率が33%以上では，並び替えによりトップ5率が10～12% 向上している．なお，並び替え前の候補の順序は文字コード順，すなわち，アルファベット順，アイウエオ順である．また，使用頻度による並び替えで大きな効果が得られたのは，レーベンシュタイン距離が等しい単語が多数存在するためである．

3.6 応答時間の評価

スペルチェック・訂正時の応答時間（ユーザの思考時間，操作時間は除く）は，単語の切り出し時間 T_{Seek}，その単語が辞書に含まれる語かどうかの判定時間 T_{IsWord}，および，辞書に含まれなかったとき，近い単語（候補単語）を選び出す時間 T_{Select} からなる．

T_{Seek} および T_{IsWord} の実測結果は，それぞれ0.83 ms および 0.011 ms である．前者は，評価に使用したニュースレターでの平均値であり，後者は，辞書からランダムに1,000語を選んで，それらの単語の検索を100回繰り返して求めた平均値である．

応答時間の7，8割を占めるのが候補単語選択時間 T_{Select} である．ハッシュサイズ16，候補単語数下限値 MIN = 5 のときの実測結果を**図8**に示す．一般に，誤りが多いほど時間がかかり，誤り数2では6～19 ms（語長2の場合，表1のハッシュ距離計算時間よりも小さくなっているが，これはハッシュ距離計算時間は多少入力単語に依存し，単語長が短い場合，表1の値より小さくなることによる），誤り数4では11～47 ms，誤り数8では31～63 msある．また，このときのレーベンシュタイン距離計算回数を**図9**に示している．

候補単語選択時間は，ハッシュ距離計算時間，候補単語の並び替え時間およびレーベンシュタイン距離計算時間からなる．候補単語選択時間の実測では，内訳

として，候補単語の並び替え時間も計測した（**表2**）．誤りが多いほど，エントリ数が増えるため，並び替えに時間がかかり，その値は0.6～3.4 msとなった．候補単語選択時間からハッシュ距離計算時間と候補単語の並び替え時間を除き，この値を図9の距離計算回数で除算した1回あたりのレーベンシュタイン距離計算時間は，語長に依存するが，平均では約30 μsであり，ハッシュ距離計算時間のおよそ50倍である．

ハッシュ距離は，1つの単語で同じ文字が何回使われているかには依存せず，また，文字の順序にも依存しない．このため，単語が長いほど，ハッシュ距離とレーベンシュタイン距離の差が大きくなり，レーベンシュタイン距離の計算回数が多くなる．また，1回あたりのレーベンシュタイン距離計算時間自体も単語が長いほど大きくなる．これらの原因から，単語が長いほど，候補単語選択時間が大きくなっている．

今回開発したスペルチェッカでは，対話形式でスペル誤りを含む単語を1つずつ確認し，正しい単語を選択（候補単語になければキー入力により修正）する方式をとっている．今後，修正時間をさらに短縮するために，次のような一括修正方式の採用を計画している．

誤り単語が検出されると，色を変えて（あるいは下線を引いて）本文中に第一候補を表示して，元の誤り単語は行間に表示する．これが全テキストに対して，自動的に行われる．ユーザは第一候補が正しい単語でないもののみを選んで操作する．すなわち，該当単語をクリックすると，プルダウン式に候補単語がリスト表示される．いずれかの候補単語をクリックするとこれがデフォルトで表示されている第一候補に替わる．候補単語にない場合には，キー入力で直接修正する．すべてOKとなったら，修正アイコンをクリックすることにより，一括修正される．デフォルトで表示されている第一候補でよいときが大半であるため，チェック・修正作業時間の大幅な短縮が期待できる．

このような一括修正方式では，候補単語のすばやい選択が求められるが，本論文で提案するスペル誤り修正方式は，この要求に応えられる性能を有している．画面に20語の認識誤り語があった場合，0.5秒程度でこれらの認識誤り語に対する候補単語が表示でき，滑らかなスクロールが可能である．

3.7 OCRデータによる評価

21ページ（日本語約1万7千字，英数字・記号約1万字）の文書に対するOCRデータによる評価を行った．

本スペル・チェッカでは，英字がひらがなに変わったとか，英単語の先頭，末尾が全角文字に変わった場合，単語の切り出しエラーが発生するため，事前にエラー文字を削除した（順に16語，7語，27語）．数はかなり多いが，単純に文字を削除するだけのため，正しいスペルを入力することに比べれば，時間はかからない．また，「PC用OS」のように，正しい漢字を認識誤りと見なすような単語切り出しエラーは皆無で

表2 候補単語の並び替え時間
Table 2 Sorting time of candidate-words.

誤り数	1	2	3	4	5	6	7	8
並び替え時間（ms）	0.6	0.9	1.4	1.8	2.1	2.3	2.8	3.4

表3 OCRデータに対する実験結果
Table 3 Experimental results for OCR data.

(a) OCRエラーの発生状況

語長	～2	3	4	5	6	7	8	9	10～15	合計
OCR 1文字エラー	0	2	25	4	8	10	10	23	6	88
OCR 2文字エラー	0	1	6	24	22	32	45	43	11	184
OCR 3文字エラー	0	0	0	2	2	4	22	21	6	57
OCR 4文字エラー	0	0	0	0	1	1	8	5	4	19
OCR 5文字エラー	0	0	0	0	0	0	0	0	1	1
OCRエラー語数	0	3	31	30	33	47	85	92	28	349
全語数	140	107	108	181	65	171	152	202	39	1,165
OCRエラー率（%）	0	3	29	17	51	27	56	46	72	30

(b) 誤り数と訂正能力の関係

	誤り1	誤り2	誤り3	誤り4	誤り5	合計
語数	88	184	57	19	1	349
ヒット率（%）	99	93	88	74	0	92
トップ5率（%）	99	96	91	79	0	95
ヒット率（%，予測値）	95	89	83	74	67	89
トップ5率（%，予測値）	100	97	95	89	83	98

あった.

上述の前処理を施したOCRデータでは，英単語(1,165語）の認識誤りは349語（30%）であり，そのうち120語に漢字への認識誤りが含まれていた．各単語に何文字の認識エラーがあったか，その分布を**表3**(a)に示している．短い単語は大文字だけの略語が多く，認識誤りはほとんど発生していない（大文字は小文字に比べて，認識誤りが少ないことによる）．原文と照らし合わせた結果，認識誤り語が辞書中の他の語に一致しているケースはなかった．誤り数と訂正能力の関係を表3(b)に示している．ここで，予測値は図5のヒット率，トップ5率に，表3(a)に示したエラーの発生分布を乗じて算出したものである．該当語の少ない「誤り4」,「誤り5」では差異があるが，合計では3%の差であり，ランダムにエラーを発生させて得た結果とおおむね一致している．

カタカナ用語（525語）の認識誤りは29語（6%）であり，28語が第一候補で正しく修正できた（ヒット率97%）．残る1語はトップ5候補にもなく，キー入力による修正となった．

4. おわりに

本論文では，日本語OCR文において認識エラーの多い英字，カタカナを対象としてスペル誤りの訂正方式を提案した．2つの単語間の類似性の評価尺度としては，従来から知られているレーベンシュタイン距離を用いた．辞書のすべての単語に対して，レーベンシュタイン距離を計算するには，時間がかかるため，あらかじめ固定長のハッシュ値を求めておき，ハッシュ距離（≤レーベンシュタイン距離）で，候補単語を絞り込むことにより，大幅な時間短縮を図った．ハッシュ距離計算時間は，レーベンシュタイン距離計算時間（使用頻度での並び替え時間を含む）のわずか50分の1であることから，大きな効果が得られた．

エラー率の高いケースに対応できるように，辞書および候補単語選択用データは，フラットなテーブル構造とした．また，これにより，単語検索，レコード並び替えにC++プログラミングシステムの標準ライブラリがそのまま利用でき，コンパクトで効率の良いシステムを実現できた．

従来の研究では，英大文字，小文字を区別しないことが多いが，技術文書では大文字の略語が多用されるため，大文字，小文字を区別したハッシュ値を採用することにより，候補単語選択時間を約半分に短縮できた．日本文での英単語は，人名，会社名，製品名など大文字で始まる語が多いため，大文字で始まる単語と小文字で始まる単語を異なる語としたことによる辞書の増大は小さい（今回の辞書では，英単語5,000語中，約400語）．

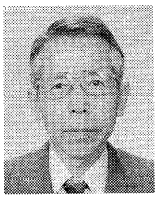
候補単語の提示順序は，レーベンシュタイン距離が同じときは，その単語の使用頻度順とした．これによって，誤り訂正能力（ヒット率およびトップ5率）が10%から10数%向上した．

今後の課題として，一括修正方式の開発，単語切り出しアルゴリズム能力の向上，スペル誤り率が高いケースでの誤り訂正精度の向上などがあげられる．

参考文献

1) Webster, R.G., 中川：英語と日本語を対象にした文章誤り検出・訂正の共通点と相違点，情報処理，Vol.37，No.9，pp.865–871 (1996).
2) Hall, P.A.V. and Dowling, G.R.: Approximate String Matching, *Computing Surveys*, Vol.12, No.4, pp.381–402 (1980).
3) Takahashi, H., Itoh, N., Amano, T. and Yamashita, A.: A Spelling Correction Method and Its Application to an OCR System, *Pattern Recognition*, Vol.23, No.3/4, pp.363–377 (1990).
4) Du, M.W. and Chang, S.C.: An Approach to Designing Very Fast Approximate String Matching Algorithms, *IEEE Trans. Knowledge and Data Eng.*, Vol.6, No.4, pp.620–633 (1994).
5) Peterson, J.L.: Computer Programs for Detecting and Correcting Spelling Errors, *Comm. ACM*, Vol.23, No.12, pp.676–687 (1980).
6) プログラム言語の標準化に関する調査，日本電子工業振興協会 (1991).
7) 高城，田中：綴り誤りの高速訂正法，信学技報，PRU95-198 (1996).

(平成8年11月11日受付)
(平成9年5月8日採録)



畑田　稔（正会員）

1942年生．1967年姫路工業大学電気工学科卒業．1972年京都大学大学院工学研究科博士課程修了．同年日立製作所入社，現在，システム開発研究所に勤務．制御系の安定問題，大規模システムの解析と評価などの研究を経て，技術情報サービスシステムの研究開発に従事．1971年電気学会論文賞受賞．工学博士．システム制御情報学会会員．

遠藤 裕英（正会員）

1941 年生．1965 年京都大学工学部電子工学科卒業．同年日立製作所入社．中央研究所主任研究員，マイクロエレクトロニクス機器開発研究所部長を経て，現在システム開発研究所情報センタ長．この間，制御用計算機，パターン認識装置，ワープロ，技術情報サービスなどの研究開発に従事．