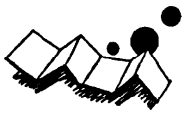
解　説



英文つづり誤りの訂正法†

伊　藤　哲　郎††

1. まえがき

事務処理の高能率・高機能化を図るオフィスオートメーションが注目を集めている．オフィスでの仕事をみてみれば，主として文書の形であつかわれる各種情報の生成，解析，複写，転送等から成りたっていることがわかる．オフィスオートメーションの大きな目的の一つは，計算機のもつ情報蓄積のためのファイル作成・編集能力，通信回線を介した情報交換能力，および，要求に応じた様式での内容表示能力を利用して，複雑で手間のかかる上記作業を自動化された形で行う，という所にあるといえよう．

文書は，本文の入力，不良箇所の修正それに読み合わせを通じて作成される．文書作成のためのシステムに必要な機能を掲げてみれば，自動改行，自動改ページ，センタリング，罫線引き，桁ぞろえ，文字列や数値の追加・削除・移動・複写，つづりの検査，ハイフン付けなどとなろう．本稿では，これらのうち，つづり誤りの発見・訂正機能を実現するためのプログラム(スペリングプログラム (spelling program) と呼ばれる[28]) について解説を行う．

英文は，英数字の羅列が区切り記号で意味ある単位ごとに分けられた，単語構造 (word structure) をもっている[34]．この観点からすれば，英文中のつづり誤りは次の3種に分けることができる[10),39)]．

(a)　単語構造誤り (word structure error)：区切り記号の脱落・挿入，区切り記号の英数字への変化およびその逆によって本来の構造がこわれる場合．

(b)　単語誤り (word error)：文中の正しい単語が，それとは異なった他の正しいつづりの単語に変わる場合．

(c)　文字誤り (character error)：通常は単語とみなされないつづりの文字列に変化する場合．

スペリングプログラムが発見・訂正の対象としている誤りは，ほとんど，3番目の場合である．これは，単語構造誤りおよび単語誤りの訂正では，検査すべき英文の構文や意味まで考慮した高度な処理が要求されるからである．文字誤りの場合には，問題としている文字列が辞書中の単語と同一か否か，もしくは，どれ程似ているかを調べるという，基本的な操作だけで済む．以下でも，つづり誤りとは文字誤りのことを指すとして話を進めてゆく．

2. 経過概観

スペリングプログラムの作成が計算機関連技術の話題として取りあげられるようになったのは，1960年頃のことであった．最初は，光学的文字読取機を通じて文字入力する際の機械的誤りや，小規模データベースへのデータ入力誤りを検査しようとして作られた[4),7),13),15)]．その他，モールスコードの認識率を高める[20)]とか，特定のオペレーティングシステムやプログラミング言語の利用時におけるキーワード・変数名誤りを正す[16),23)]ことを目的としたものもあった．

今日みられるような，一般の英文を検査対象としたスペリングプログラムの開発が盛んになったのは，ワードプロセッサの雛型ともいえるテキストエディタ等による英文の作成が行われ始めた，1970年代からのことである．なかでも，1971年，Stanford 大学の人工知能研究所で作成され，Carnegie-Mellon 大学で機能アップされた SPELL プログラム[11)]は，DEC-10 や DEC-20 システムにおいて今日でも広く使われている．また，Bell 研究所で開発された UNIX* オペレーティングシステムや IBM/360・370 システムにも，つづり誤り検査プログラムが用意された．

スペリングプログラムは，機能の違いから次のように分けることができる[28)]．

(a)　つづり検査プログラム (spelling checker)：

† Correcting Spelling Errors in English Sentences by Tetsuro ITO (University of Library and Information Science).

†† 図書館情報大学

* UNIX はベル研究所の登録商標

入力としての英文を読み，その中のつづりが正しくないものを見いだす．

（b） つづり誤り訂正プログラム (spelling corrector)：つづり誤りを発見した後，正しいと思われる単語の候補（複数あることも多い）を探し出す．

検査プログラムの作成は比較的容易である．最も単純な方法は，つづり誤りの生じる回数は低いとの仮定に基づく．すなわち，英文中で使用されている単語の頻度順表示を行い，低頻度語のみを対象につづりの正しさを調べてゆけるようにするのである[38]．この考えを発展させ，単語そのものの頻度でなく，単語中に生じる n 字組 (n-gram：n コの英数字の組）の頻度を用いる検査法も提案された[5),21),24),30),35)]．

つづり誤りの可能性を含む文字列を選び出すだけでは，真に実用的なプログラムとならない．すなわち，正しいと推定される単語を，形式的に除外できるような方策を施しておくことが必要である．このためには，コード表現した単語の表を前もって用意し，表にないコードに変換される入力つづりは誤り，とみなす方法が考え出された[2),7),15)]．また，多数の英文中での n 字組頻度を表として求めておき，これをもとに，低頻度の n 字組を含む単語を誤りつづりの候補とする，という方法も提案された[5),30)]．その他，正しい単語の集りを，スーパインポーズコード (superimposed code) で，一つの表に圧縮して収める方法も利用された[26)]．

Damerau[6)] は，つづり誤りの80%以上は（i）1文字欠落，（ii）1文字追加，（iii）1文字の入替り，あるいは（iv）隣接文字の反転のいずれかによって生じていることを指摘した．訂正法としては，この事実に基づき，二つが提案された．そのうちの一つは，上記原因を逆に考え，誤りつづりから正しいと思われる文字列を派生し，それが辞書（正しいつづりの単語，もしくは，そのコード化されたものの集り）にあるか否かを調べる完全一致 (exact match) 法である．残りの一つは，誤りつづりは正しいものの軽微な変化であるとの認識から，入力文字列とよく似たつづりの辞書中の単語を取り出す最適合 (best match) 法である．完全一致法での問題点は，入力つづりと一致する単語が収められているか否かを迅速に判断できるような辞書の構成法にある．これには，単語をコード表現とする方法，単語の集りを生起頻度に従い階層的に分類する方法[29)]，各単語を一意的な番地に記憶させる木構造法[25),33)]やハッシュ法[11),22),26)]などが考え出された．最適合法での問題点としては，任意の文字つづり同士の似ている割合を数量的にとらえる方法，および，似た割合に応じて辞書中の単語を効率よく取り出せる方法の定式化の二つがあげられる．数量的なとらえかたに関して，Levenshtein[18)] は，似たものに対しては値が小さくなる距離を取り上げ，文字列間の距離とは片方をもう一方に変換するのに要する単位操作の最小回数と定義した．この距離を動的プログラミングで求める方法[19),36)]や重みづけしたものに発展させる方法[27)]も出された．似たものに対しては値が大きくなる類似度も，両文字列が持つ共通文字部分列の割合として定義された[9),14)]．辞書の構成に関しては，パターン認識の分野における研究成果を取り入れたものが用いられた[12),14),32)]．

1970年頃までの経過は文献1),23)に簡単にまとめられている．最近の文献としては，スペリングプログラム全般に関し詳しく述べた28),29)，誤り訂正を似たつづりの文字列の発見という観点からまとめた12)がある．ここでは，辞書を用いた方法を主として述べるが，n 字組による方法は39)にわかり易く解説されている．

3. 単語の扱い

英語の文章は，単語を構成する英数字と，単語を区別したり全体を読み易くしたりする空白，コンマ，ピリオド，コロン，カッコ，引用符，改行，改ページ等の区切り記号 (delimiter) の羅列という，単語構造を成していた．スペリングプログラムでは，これら区切り記号をもとに選び出された単語を次のように扱う．（i）数字は無視あるいは区切り記号と判断する．（ii）ハイフンは，通常，区切り記号とみなす．（iii）アポストロフィについては最後の"'"，および，所有をあらわす"'s"を除去する．そして，（iv）大文字はすべて小文字に変換する．このように処理された単語は，辞書項目との一致具合判断のためさらに変形される．

つづり誤り発見・訂正を高い精度で行おうとすると，およそ5万の単語が収められた辞書を必要とする．通常の英単語は最長44文字（そのうち99%は17文字以下），平均8.1文字であることから[29)]，辞書をそのままの形で扱うと最低400 KB要する．これは小規模の処理システムの主記憶装置における量でない．

辞書のサイズを小さくする方法として，単語の短縮形であるコードを利用するものもあるが，最近では，接辞を除去し，語幹のみを辞書の項目とするものが広

表-1
(a) 接尾辞付与規則
"able" の付与も "g"flag に準じる.

```
"g" flag
  ...e:=...ing
  ...#:=...#ing (#≠e)

"o" flag
  ...e :=...ed
  ...%y:=...%ied (%≠a,e,i,o,u)
  ...%#:=...%#ed (otherwise)

"s" flag
  ...%y:=...%ies (%≠a,e,i,o,u)
  ...#  :=...#es  (#=s,x,z,h)
  ...#  :=...#s   (otherwise)

      ...  ...  ...
```

(b) 接頭辞

```
anti, bio, dis, electro, en, fore,
hyper, intra, inter, iso, kilo,
magneto, meta, micro, milli, mis,
mono, multi, non, out, over, photo,
poly, pre, pseudo, re, semi, stereo,
sub, super, thermo, ultra, under, un
```

```
In a recent survey on detecting
and corecting spelling errors, Peterson
states that, "... a hash table appraoch
would seem the most reesonable."
However, all the algorithm use it
only to restrict the search rainge
(e.g., the cited DEC-10 SPELL program)
...
```

(a) 入力文

```
in a recent survey on detect
and corect spell error peterson
state that a hash table appraoch
would seem the most reeson
however all the algorithm use it
only to restrict the search rainge
e g the cite dec spell program
...
```

(b) 処理結果
図-1

く用いられている. このようにすると, 単語の長さが短くなるだけでなく, 同一の語幹をもつ単語は一つで代表できることになる. DEC-10 SPELL で取られている接尾辞付与規則の一部を表-1(a)に示す. この表を利用して接尾辞を除去する場合, 文字つづりに見合う規則を逆からたどればよい. 接頭辞も同様に扱える. 表-1(b)は UNIX spell で用いられている接頭辞リストである. 接辞解析の詳しさにもよるが, 項目が語幹のみから成る辞書を考えると, 全体的な大きさはもとのものの 1/2 から 1/5 で済む.

以上をまとめると, 各単語の扱いは以下のようになる.

(S 1)数字やハイフンを区切り記号とみなし, 単語を分解する.

(S 2)大文字を小文字に変換し, アポストロフィの処理をする.

(S 3)必要ならば, 図-1 に従って接辞を除去する.

(S 4)残りの文字列に対し, その正しさを辞書を通じて調べる. もし, 辞書になければ誤りとし訂正を試みる.

図-1(a)の英文[22]中の単語を(S 1)～(S 3)に従って処理した結果を図-1(b)に示す.

4. 辞書の構成と誤り訂正法

たとえ項目数を減らしたとしても, すぐれたスペリングプログラムを作るには, 探索を効率よく進められるような構造を辞書に持たせる必要がある. ここでは, 辞書の構成法とその利用の仕方についていくつかみていこう.

4.1 完全一致法

つづり誤り発見の後, 正しいと考えられる文字列を派生し訂正を行う, 完全一致法のための辞書として以下のようなものがある.

(1) 分散記憶表

もっとも良く知られている辞書構成法の一つはハッシュ法である. DEC-10 SPELL の方法をもとに説明しよう[11]. ハッシュ法では, 各辞書項目の分散記憶表 (scatter table)中での位置を決めるハッシュ関数 h を用意することが第一ステップとなる[17]. SPELL での関数 h は, 各単語 w に対して, 次のように与えられる.

$$h(w)=(I_1\times 26+I_2)\times 10+\min(L-2, 9)$$

ここで, I_1, I_2 は, それぞれ, 1番目と2番目の文字の数字表現 ($a=0, b=1, \cdots, z=25$, "'"$=25$)で, $L(\geq 2)$ は単語の長さ. 次に, 同一の関数値を持つ単語 (はじめの2文字と長さが同じ語) をポインタでつなげ, 連鎖 (chain) として蓄積する (図-2). 入力としての文字列 u の正しさを調べるには, $h(u)$ から, u が収められているべき連鎖を求め, この連鎖の中に u と同じつづりの単語があれば (なければ) 正しい (誤り) とする.

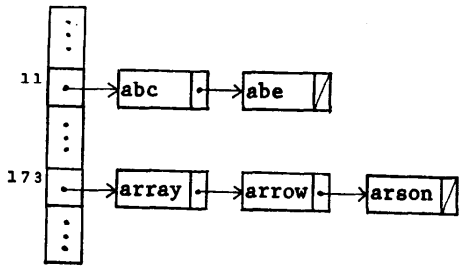


図-2 分散記憶表

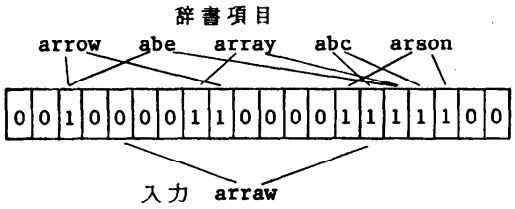


図-3 スーパインポーズコード
$h_1(w)=E_w \bmod 17$
$h_2(w)=E_w \bmod 19$
ただし,
$E_w=(I_1+I_2+\cdots+I_L)\times 10+L$

(2) スーパインポーズコード

今述べた仕方によれば，正しいとされた入力は必ず辞書中にあるとの保証が得られるが，その代り，たとえ項目を語幹と限定しても辞書の保持に大きな記憶域を要する．たとえば，平均8文字の単語25,000を収める場合，ポインタ等を除いても200 KBとなる．これに対処するには，条件を緩め，正しいとされた入力のほとんどは辞書中にある（誤りとされれば必ず辞書にない），と判断できるスーパインポーズコード[17]を利用すればよい[26]．まず，値が0と1だけの表（大きさTで，最初は0ばかりが入っている）を用意する．単語wの蓄積は，独立なm個のハッシュ関数$h_i, 1\leq i\leq m$，によって$h_i(w)=\alpha, 0\leq\alpha\leq T-1$，なる場所$\alpha$に1を置くことにより行う．そして，入力$u$のつづりの正しさを調べる時は，この表を辞書にみたて

$\forall i\ h_i(u)=1$ のときにかぎりuは正しい

と判断する（図-3）．ただ，欠点として，本来は辞書にない誤りのつづりであっても，種々の単語に対する値1の組み合わせ方から，正しいとされる可能性が生じてしまう．（この確率は，表全体としてN個の単語が収められた時，1の分布は表中で一様と仮定すると，$(Nm/T)^m$である．）実際問題としては，mとTとをうまく選び，誤りの可能性を小さくすればよい．たとえば，$N=25,000$に対し，独立なハッシュ関数を10個と大きさ50 KBの表を用意すると，誤りつづりが正しいとされる確率は1％以下となる．この表の大きさは，先のものと比較して1/4になっている．UNIXのspellでは，約24,000の単語を50 KBの表に収めるのに11個のハッシュ関数を用意してつづり検査を行っている．

(3) 頻度分類

辞書の階層構成も有効である[28]．これには，英文中では各単語の使用頻度は一様でなく，大きく偏りがあることを利用する．まず，単語を

(a) よく使われる単語（100～200語）．

(b) 各専門分野で使用される単語（1,000～2,000語）．

(c) 残りの単語（10,000～100,000語）．

と分類する．そして，木構造法の一種としてのトライ(trie)[17]，上で述べた表，および，索引順次編成ファイル，それぞれを，辞書構成時に用いるのである．

(4) 一致法による訂正

誤りと判断された入力を訂正する方法を求めよう．完全一致法による訂正では，典型的なつづり誤りが，1文字の入替り，1文字の追加，1文字の欠落，あるいは隣接文字の反転のいずれかであることに注目する[6]．すなわち，誤り入力から逆に正しいと思われる文字列を派生し，これが辞書項目か否かを判断していくという発見的手法がとられる[11]．具体的には長さLの入力$u=u_1\cdots u_i\cdots u_L$からの正しいつづりは，

(i) 1文字の入替りとの仮定：位置$i, 1\leq i\leq L$，にある文字u_iをそれ以外のものu^*と取り換える．

(ii) 1文字の追加との仮定：位置$i, 1\leq i\leq L$，にある文字u_iを除去する．

(iii) 1文字の欠落との仮定：位置iと$i+1, 0\leq i\leq L$，の間に仮想的な文字“#”を挿入した文字列を新しく形成し，これに対して(i)と同様の処理を施す．

あるいは

(iv) 隣接文字の反転との仮定：位置iと$i+1$，$1\leq i\leq L-1$，とにある文字u_i, u_{i+1}を交換する．

と得られると考えるのである（表-2参照）．作り出されるつづりの数は文字の種類を26とすると，それぞれ，$25L, L, 26(L+1)$および$L-1$である．訂正過程では，これらすべての可能性を調べ，辞書項目と一致する文字列のみを正しい単語の候補として取り出す．

表-2 文字列の派生
u_0, u_{L+1} は，それぞれ，文字列の先頭および終端を示す仮の文字．

u :	$(u_0)u_1\cdots u_i u_{i+1}\cdots u_L(u_{L+1})$
(i)	$u_1u_2\cdots u_{i-1}u^*u_{i+1}\cdots u_L$ $u^*(\neq u_i)\varepsilon\{a,b,\ldots,z\}$
(ii)	$u_1u_2\cdots u_{i-1}u_{i+1}\cdots u_L$
(iii)	$u_1u_2\cdots u_i{}^{\#}u_{i+1}\cdots u_L$ $\#\varepsilon\{a,b,\ldots,z\}$
(iv)	$u_1u_2\cdots u_{i+1}u_i\cdots u_L$

表-3 典型的でないつづり誤り

Incorr. Spell.	Corr. Spell.
aquiese	acquiesce
catagorey	category
develope	develop
fullfil	fulfill
hinderence	hindrance
manuveur	maneuver
phamplet	pamphlet
raphsody	rhapsody

4.2 最適合法

完全一致法に従い，前記4種の原因の組み合わせから成る典型的でないつづり誤りを訂正しようとすると，非常に効率の悪いものとなってしまう．表-3は文献6)に載せられた誤りつづりのうち，4.1節の方法では訂正できなかった例である．

このような誤りの訂正にも対処できる方法として，入力と（もっとも）良く似たつづりを示す辞書中の単語を正しい候補として取り出す，（最）適合法がある．最適合法のための辞書を求める際には，文字列対の一致度の数量化が必要である．これには，反射律，対称律および三角不等式を満たし，よく似ている対には小さな値を与える距離 d，あるいは，反射律および対称律を満たし，よく似ている対には大きな値を与える類似度 s が用いられる．

（1） 距　離

提案されている距離として，よく知られた Hamming 距離，最長共有部分列 (longest common sequence) による距離のほか，Levenshtein 距離[18]がある．このうち Levenshtein 距離は，つづり誤り訂正のために考えだされたもので，文字の入替え，追加，削除（および隣接文字の反転）操作によって，片方の文字列をもう一方に変換するのに要する最小の操作回数と定義される．これは，動的プログラミングによって求めることができる[12),19),36)]．

辞書中の任意の項目間に距離が定義されると，辞書全体は距離空間を成していると考えてよい．すると，つづり誤り訂正とは，距離空間上で入力ともっとも近い，あるいは，十分近くにある単語を効率良く見い出すことと言い換えられる．

ここで距離空間の性質をみてみよう．適当な単語 c_i と実数値 $\varepsilon_i(\geq 0)$ を選び，$R_i=\{w: d(c_i,w)\leq\varepsilon_i\}$ とおく．（R_i はクラスタ，c_i は中心と呼ばれる．）いま，入力 u と近接した単語 w^* が求められたとする．集合 R_i の各 w について

$$d(u,w)+d(w,c_i)\geq d(u,c_i)$$

および

$$d(c_i,w)\leq\varepsilon_i$$

が成り立つ．これより，もし $d(u,c_i)-\varepsilon_i\geq d(u,w^*)$ ならば，$d(u,w)\geq d(u,w^*)$ となり，R_i 中には w^* より u と近接した単語は存在しないこととなる[3)]．

この距離空間の性質を辞書探索の効率を高めるために利用する．辞書は以下の手順で求める[32)]．

（S 1）適当な数の中心 c_i を選び，辞書全体をいくつかのクラスタ R_i に分ける．このとき，ε_i は R_i の大きさが定められた値 k 以下となるようにする．

（S 2）どの R_i にも属さないものをゴミと名づけて集める．

（S 3）もし選ばれた c_i の数が k より大きければ，これらについてステップ（S 1）と（S 2）をくり返す．すなわち（S 1），（S 2）をくり返すごとにレベル数が1増加した木構造とするのである．つづり検査・誤り訂正過程は次となる（簡単のため，入力 u ともっとも近接した単語 w^* を，正しい語候補として取り出すことを考える）．

（S 1）注目しているクラスタが最下位レベルのものでない限り，（S 2）を行う．

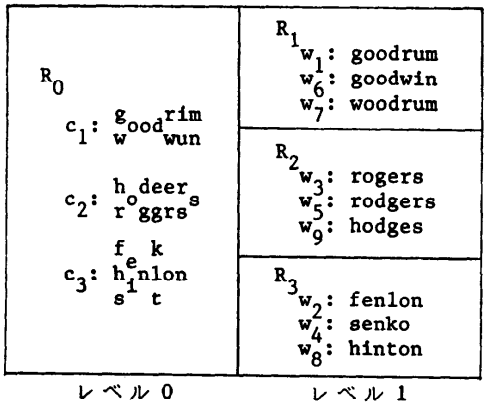
（S 2）クラスタを探し，未処理で，入力 u ともっとも小さな距離を示す，中心 c_i を求める．そして，対応するクラスタ R_i へ探索を進める．このとき，c_i が存在しないか，あるいは，$d(u,c_i)-\varepsilon_i\geq d(u,w^*)$ ならば，一つ上位レベルのクラスタへ戻る（上位レベルのクラスタがなければ終了）．ただし，ゴミのクラスタへは，常に探索を進める．

(S 3)入力 u ともっとも小さな距離を示す単語 w' をみつける．もし，$d(u,w')=0$ ならば，u は正しいとして終了．また，もし，$d(u,w')<d(u,w^*)$ ならば，$w^*=w'$ と更新し，上位レベルのクラスタへ戻り，(S 2)をくり返す．

w_1:goodrum, w_2:fenlon, w_3:rogers
w_4:senko, w_5:rodgers, w_6:goodwin
w_7:woodrum, w_8:hinton, w_9:hodges

	w_1	w_2	...	w_5	...	w_9
w_1	100	0 12	0	26 52	84	0 25
w_2		100	0 31	0 5	0 31	0
w_3			100	0 73	6 12	0 37
w_4				100 0	0 12	0
w_5				100	23 28	0 50
w_6					100 36	5 25
w_7		*			100	0 25
w_8						100 6
w_9						100

(a) 文字列間類似度



(b) 辞書の構造

ステップ	探索されたクラスタ	hoodgus との類似度	候補
1	R_0	c_2; 0.57 c_1; 0.42 c_3; 0.05	
2	R_2	w_9; 0.57 w_5; 0.42 w_3; 0.10	w_9: hodges
3	R_0	c_1; 0.42 c_3; 0.05	(終了)

(c) 訂正過程

図-4

(2) 類 似 度

次に，類似度に基づく最適合法をみてみよう．類似度としては，たとえば，

$$s(w,u)=\min(s_R(w,u),\ s_R(u,w)),$$

ただし，

$$s_R(w,u)=\frac{\sum_{\gamma(k)}\min(p(w,\gamma(k)),\ p(u,\gamma(k)))\,|\gamma(k)|}{\sum_{\gamma(k)}p(w,\gamma(k))\,|\gamma(k)|}$$

を考えることができる[14]．ここで，$\gamma(k)$ は，w あるいは u に含まれる長さ $|\gamma(k)|$ の任意の文字部分列．また，$p(w,\gamma(k))$ は，w が，位置 $k-1, k$ あるいは $k+1$ で始まり，長さ $|\gamma(k)|$ の文字部分列 $\gamma(k)$ を含んでいるときに限り 1，それ以外は 0，となる関数である．9個の文字列（人名）[12] に対する類似度を行列の形とした例を図-4(a)に示す．

類似度の場合は三角不等式の成立をみることができないため，上とは別の探索の効率化を計る手段が必要である．これにはクラスタ代表という概念を導入する．クラスタ代表 c_i とは，クラスタ R_i* に1対1対応して定められた一種の文字列で，具体的には，R_i 中の各文字列の位置 $l(\geq 1)$ に生じる文字すべてが，ひとまとめにされ，位置 l に置かれた形をしている．たとえば，$R_i=$ {goodrum, goodwin} に対するクラスタ代表は，good$^{\text{rim}}_{\text{wun}}$ となる．クラスタ代表同士の類似度は上式で測るが，任意の文字列 u とクラスタ代表 c_i との類似度は $s_R(u,c_i)$ で測る．

明らかに，

$$s_R(u,c_i)\geq s(u,w)\ \forall_w w\in R_i$$

より，u と似た単語 w^* が求められたとき，

$$s(u,w^*)\geq s_R(u,c_i)$$

なる R_i 中には，w^* より良く似た単語は存在しないこととなる．

この性質を辞書探索時の余分なクラスタ参照を避けるのに利用する．辞書の構成は，最初に適当な大きさのクラスタを定め，その後代表 c_i を定めることと，ゴミが生じないことを除けば，距離空間の場合と同じである．また，つづり検査・誤り訂正も同様に行われる．図-4(b)に類似度を用いた場合の辞書の構造例を，また，図-4(c)に入力 u=hoodgus としたときの訂正過程を示す．約 24,000 語が収められた5レベルの辞書を対象に，典型的なつづり誤りの訂正を行った実験では，すぐれた結果が得られた

* この場合のクラスタとは，互いに大きな類似度を示す，単語あるいは文字列の集りを指す．

と報告されている[14].

最適合法は，先に述べた利点を持ちながら，実際的なプログラムに採用されていない．その理由として，以下のような事柄が考えられる．(a)入力と辞書中の単語との距離や類似度を求めるのに手間がかかる．(b)辞書項目としての単語以外にクラスタ中心あるいはクラスタ代表を扱う必要がある．(c)探索過程で木構造を後戻りする必要があり探索の効率を減じる．現在のところ，これらを解決できる今後の研究に期待がかけられている．

5. ま と め

スペリングプログラムは，すでに述べたようなアプリケーションプログラムとしてだけでなく，近頃では，ワードプロセッサやマイクロコンピュータによる英文の処理にまで採用されてきている．また，コマンド言語を基本とする電子郵便システムに組み込まれ[8]，その有効性を発揮している．

スペリングプログラムの標準的な利用法としては二つある．文中の異なり単語をリストとしてまとめ，その各要素についてのみつづり検査・誤り訂正を行うバッチ (batch) 方式 (例：UNIX spell) が，そのうちの一つで，残りは，単語を文の先頭から順に求め，その検査・訂正を行ってゆく会話型 (interactive) 方式 (例：DEC-10 SPELL[11] やｚエディタの spell[37]) である[29]．バッチ方式によると，同一の語についての検査・訂正は1度だけで済むが，長い文章では，リストの用意や検査に要する時間が利用者にとって十分に長い待ちとなってしまう．また，誤りとされた単語がもとの文のどこにあるかを，別の手段で探し出さなければならないという欠点がある．一方，会話型方式によれば，利用者は逐次正しさを調べられることから，心理的な待ち時間は短くて済み，また，誤りつづりの英文中での場所を特定化できるという利点が生まれる．バッチ方式よりも使い易い．ただし，同一の単語に対してそれが現れる回数だけ検査・訂正がくり返されるため，十分高速の辞書探索プログラムを作り上げておくことが必要である．

発見された誤りつづりの訂正は，利用者との合意のうえで実行されるのが一般的な形式である[11),31]．利用者の取れる選択枝として，

(a) 修正の操作を施さず，そのままとする．ただし，誤りとの記録は残す．

(b) この場合に限り正しいとする．ただし，誤りとの記録は残す．

(c) 正しいとみなし，これを個人用の小さな辞書に追加する．以後，この辞書にある単語は正しいとする．

(d) 誤りとみなし，利用者がこれに代る単語を入力する．以後，同様の誤りに遭遇すると入力したものに置き換える．

(e) 誤りとみなし，誤り訂正プログラムの手助けによって正しい単語を選ぶ

などが用意されている．

現在普及しているワードプロセッシングシステムに備えられている標準的な機能をみてみると，行書式設定，ページ書式設定，マージン設定，ヘッダ・フッタ・ページ番号付け，センタリング，右ぞろえ等がある．しかしながら，本稿で解説してきたつづり検査・誤り訂正機能は，オプションとして補足的に追加されることが多い．ところで，計算機による文書処理が広く社会に行きわたりつつある今日，システムの利用形態をみてみると，利用者のほとんどは専門家以外の一般の人々である．このことを考慮すると，つづり検査・誤り訂正機能は補足的でなく，標準的なものとしてシステムに組み入れられるのが望ましい．すなわち，単語が入力されれば即時検査，という上記会話型をさらに押し進めた形でスペリングプログラムが働くようにするのである．同様の方式は，日本語ワードプロセッサでのカナつづりを漢字カナ混じりつづりへ変換する場合にみられる．

最後に，つづり検査・誤り訂正機能を含む英文処理のためのディスプレイ画面を示す (図-5 参照)．画面の左半分は本文表示のための領域で，右半分は誤り訂

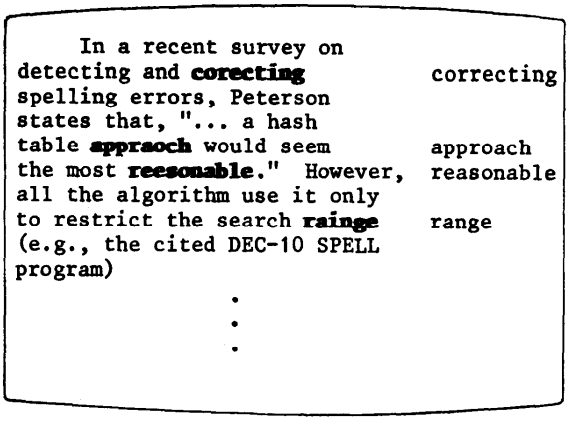


図-5 ディスプレイ画面

正のための領域である．誤りつづりの文字列は本文中に，また正しい単語の候補は誤りが生じている行の丁度右側に写し出されている．置き換えられるべき候補を日本文の処理のように本文に埋め込んでしまっていないのは，キー入力したそのままの情報がもっとも重要であるとの考えによる．

参 考 文 献

1) Alberga, C. N.: String Similarity and Misspellings, Comm. ACM, Vol. 10, No. 5, pp. 302-313 (1967).
2) Blair, C. R.: A Program for Correcting Spelling Errors, Inf. Contol, Vol. 3, No. 1, pp. 60-67 (1960).
3) Burkhard, W. A. and Keller, R. M.: Some Approaches to Best-Match File Searching, Comm. ACM, Vol. 16, No. 4, pp. 230-236 (1973).
4) Carlson, G.: Techniques for Replacing Characters that are Garbled on Input, in Proc. SJCC 1966, AFIPS Press, Arlington (1966).
5) Cornew, R. W.: A Statistical Method of Spelling Correction, Inf. Control, Vol. 12, No. 2, pp. 79-93 (1968).
6) Damerau, F. J.: A Technique for Computer Detection and Correction of Spelling Errors, Comm. ACM, Vol. 7, No. 3, pp. 171-176 (1964).
7) Davidson, L.: Retrieval of Misspelled Names in an Airlines Passenger Record System, Comm. ACM, Vol. 5, No. 3, pp. 169-171 (1962).
8) Durham, I., Lamb, D. A. and Saxe, J. B.: Spelling Correction in User Interfaces, Comm. ACM, Vol. 26, No. 10, pp. 764-773 (1983).
9) Findler, N. V. and Leeuwen, J. V.: A Family of Similarity Measures Between Two Strings, IEEE Trans. Patt. Anal. Machine Intell., Vol. PAMI-1, No. 1, pp. 116-118 (1979).
10) Galli, E. J. and Yamada, H. M.: Experimental Studies in Computer-Assisted Correction of Unorthographic Text, IEEE Trans. Eng. Writing and Speech, Vol. ESW-11, No. 2, pp. 75-84 (1968).
11) Gorin, R. E.: SPELL: Spelling Check and Correction Program, Documentation Online in File DOC: SPELL. DOC at CMU-CS-A. ARPA (1971).
12) Hall, P. A. V. and Dowling, G. R.: Approximate String Matching, Comput. Surv., Vol. 12, No. 4, pp. 381-402 (1980).
13) Harmon, L. D.: Automatic Recognition of Print and Script, Proc. IEEE, Vol. 60, No. 10, pp. 1165-1176 (1972).
14) Ito, T. and Kizawa, M.: Hierarchical File Organization and Its Application to Similar-String Matching, ACM Trans. Database Syst., Vol. 8, No. 3, pp. 410-433 (1983).
15) Jackson, M.: Mnemonics, Datamation, Vol. 13, pp. 26-29 (1967).
16) James, E. B. and Partridge, D. P.: Tolerance to Inaccuracy in Computer Programs, Comput. J., Vol. 19, No. 3, pp. 207-212 (1976).
17) Kunth, D. E.: The Art of Computer Programming, Vol. 3: Sorting and Searching, Addison-Wesley, Reading, Mass (1973).
18) Levenshtein, V. I.: Binary Codes Capable of Correcting Deletions, Insertions and Reversals, Sov. Phys. Dokl., Vol. 10, No. 8, pp. 707-710 (1966).
19) Lowrance, R. and Wagner, R. A.: An Extension of the String-to-String Correction Problem, J. ACM, Vol. 22, No. 2, pp. 177-183 (1975).
20) McElwain, C. K. and Evens, M. B.: The Degarbler: A Program for Correcting Machine-Read Morse Code, Inf. Control, Vol. 5, No. 4 pp. 368-384 (1962).
21) McMahon, L. E., Cherry, L. L. and Morris, R.: Statistical Text Processing, Bell Syst. Tech. J., Vol. 57, No. 6, pp. 2137-2154 (1978).
22) Mor, M. and Fraenkel, A. S.: A Hash Code Method for Detecting and Correcting Spelling Errors, Comm. ACM, Vol. 25, No. 12, pp. 935-938 (1982).
23) Morgan, H. L.: Spelling Correction in Systems Programs, Comm. ACM, Vol. 13, No. 2, pp. 90-94 (1970).
24) Morris, R. and Cherry, L. L.: Computer Detection of Typographical Errors, IEEE Trans. Professional Commun., Vol. PC-18, No. 1, pp. 54-64 (1975).
25) Muth, F. E., Jr. and Tharp, A. L.: Correcting Human Error in Alphanumeric Terminal Input, Inf. Process. Manage., Vol. 13, No. 6, pp. 329-337 (1977).
26) Nix, R.: Experience With a Space Efficient Way to Store a Dictionary, Comm. ACM, Vol. 24, No. 5, pp. 297-298 (1981).
27) Okuda, T., Tanaka, E. and Kasai, T.: A Method for the Correction of Garbled Words Based on the Levenshtein Metric, IEEE Trans. Comput., Vol. C-25, No. 2, pp. 172-178 (1976).
28) Peterson, J. L.: Computer Programs for Detecting and Correcting Spelling Errors, Comm. ACM, Vol. 23, No. 12, pp. 676-687 (1980).
29) Peterson, J. L.: Computer Programs for Spelling Correction: An Experiment in Program Design, Lecture Notes in Computer Science 96, Springer-Verlag, New York (1980).

30) Riseman, E. M. and Hanson, A. R.: A Contextual Postprocessing System for Error Correction Using Binary n-Grams, IEEE Trans. Comput., Vol. C-23, No. 5, pp. 480-493 (1974).
31) Robinson, P. and Singer, D.: Another Spelling Correction Program, Comm. ACM, Vol. 24, No. 5, pp. 296-297 (1981).
32) Salton, G. and Wong, A.: Generation and Search of Clustered Files, ACM Trans. Database Syst., Vol. 3, No. 4, pp. 321-346 (1978).
33) Szanser, A. J.: Bracketing Technique in Elastic Matching, Comput. J., Vol. 16, No. 2, pp. 132-134 (1973).
34) Thorelli, L. E.: Automatic Correction of Errors in Text, Bit, Vol. 2, No. 1, pp. 45-52 (1962).
35) Ullmann, J. R.: A Binary n-Gram Technique for Automatic Correction of Substitution, Deletion, Insertion and Reversal Errors in Words, Comput. J., Vol. 20, No. 2, pp. 141-147 (1977).
36) Wanger, R. A. and Fischer, M. J.: The String-to-String Correction Problem, J. ACM, Vol. 21, No. 1, pp. 168-173 (1974).
37) Wood, S. R.: Z: The 95% Program Editor, in Proc. ACM SIGPLAN/OA Symp. on Text Manipulation, Portland, OR, June, pp. 1-7 (1981).
38) Zamora, A.: Automatic Detection and Correction of Spelling Errors in a Large Data Base, J. Am. Soc. Inf. Sci., Vol. 31, No. 1, pp. 51-57 (1980).
39) 川合：英文綴り検査法，情報処理，Vol. 24, No. 4, pp. 507-513 (1983).

(昭和59年1月12日受付)