Vol. 36 No. 1 情報処理学会論文誌 Jan. 1995

読み情報を活用した日本文エディタの作成と評価

畠山 勉[†,*] 角田博保[†]

本研究では，かな漢字変換型ワープロの利用者が漢字かな混じりの表記を入力する際に指定する読み綴りと文節の区切りの情報（あわせて読み情報と呼ぶ）を漢字かな混じりテキスト（漢字層）とともに読み層として保持し，様々な指令においてそれを活用する日本文エディタ（JEM—Japanese Editor with Multi-layered text structure）を設計，作成し，その有用性について評価した．JEMの特徴は，読み情報を保持した層を実際にもっていることにある．それを利用して漸増探索指令や動的展開を，英文に対する時と同じ感覚で実行できるとともに，仮名漢字変換のやり直し（再変換）を効率的に行うことができる．これらの機能を使用する時以外は，Emacsの基本モード（fundamental mode）とほぼ同じ編集操作をすることができる．機能評価実験を行ったところ，読み情報を利用した漸増探索は，利用者が画面上で注目している場所へカーソルを素早く移動させるための手段として有力であることが分かった．

The Implementation and Evaluation of a Japanese Editor Utilizing Pronunciation-Information

Tsutomu Hatakeyama[†,*] and Hiroyasu Kakuda[†]

We designed and implemented a Japanese editor that keeps boundaries of phrases and pronunciation spellings which are used for Kana-kanji conversion, and evaluated its advantages. The editor utilizes the boundaries and pronunciation spellings. We named the editor "JEM (Japanese Editor with Multi-layered text structure)". The characteristic of JEM is that it has layers which keep information of the boundaries and pronunciation spellings. Using this editor, a user can I-search (incremental search) and dynamic abbreviate expand in Japanese text as well as in English one, and he/she can re-convert effectively. Except for these functions, JEM can be used in the same way as the fundamental-mode's way of Emacs. JEM's I-search can be performed faster than Kana-kanji conversion type I-search. In experiments, we found that users move the editor's cursor to some point as fast as in other ways.

1. はじめに

欧米ではキーボードを介してテキストエディタで文書を作成するのはごく自然なことである．それは，(1)慣れると手書きより速い，(2)出力（印刷）がきれい，(3)修正が容易，(4)計算機可読なのでテキストの自動加工などが可能，といった理由からである．それと同じことを日本文に対して行おうとすると，英文の場合ほど快適というわけにはいかない．それは，日本文エディタが上記の(1)と(3)について劣っているので，使いにくいからである．

エディタで編集対象となる文字列のことをテキストと呼ぶことにすると，英文エディタでは打鍵したそのものがテキスト（たとえばアスキー文字列）を構成するのに対し，現在主流のかな漢字変換型日本文エディタでは，読みを入力し，変換という操作を経て得られた漢字かな混じり文字列がテキストを構成する．英文の場合は打鍵そのものがテキストとなるので，直接的であるが，日本文の場合は打ち込みたい漢字かな混じり文字列の読みを適当な意味単位（文節）で区切って打鍵し，同音語の選択作業を通してテキストに変換するので，間接的である．画面上にはテキストが見えているので，カーソル移動，削除などの編集指令はテキスト，つまり字面に対して行えばいいが，新たな漢字かな混じり文字列を挿入しようとすると，字面ではなくその読みを意識しなければならない．このように，日本文エディタの利用者は，「読み」と「テキスト（＝字面）」の2つの対象物を意識してエディタに向かうことになる．

英文エディタにおいては，テキストは単一の構造であり，読みと字面といった区別はなく，指令にもその

† 電気通信大学電気通信学部情報工学科
Department of Computer Science, Faculty of Electro-Communications, The University of Electro-Communications
* 現在，(株)日立製作所デザイン研究所
Presently with Design Center, Hitachi, Ltd.

区別はない．英文エディタに日本文の入力，表示機能を追加しただけの日本文エディタ（市販のワープロを含むほとんどのものがそうであるが）でも，テキストを単一の構造として扱うことになるので，読みから漢字かな混じり文字列への変換を前処理的に扱い，編集指令そのものは英文エディタにあるものをそのまま使うことになる．そうすると，文字単位でのカーソル移動や削除指令といった物理的な指令は字面上で行うので問題ないが，探索指令といった字面を参照する指令も字面上で行わざるをえない．つまり，探索したい漢字かな混じり文字列に対応する読みを打鍵し，変換して漢字かな混じり文字列にしてから，探索するということになる．探索される文字列はもともと，その読みを打ち込んで変換した結果できたものであるにもかかわらず，読みを入力するだけでなく，わざわざ漢字かな混じり文字列に変換してから探索をするので非常にもどかしい．隔靴掻痒の感がある．読みのまま探索した方がずっと自然である．また，英文エディタでは単語単位での編集操作（カーソル移動，削除など）は利用者の直観にも訴えるし，簡単に実現できることから良く使われるが，日本文では字面から単語を切り出すのには形態素解析が必要になるという処理の難しさから，実現されることが少ない．しかし，そもそもその字面は読みから変換作業を経過して得られたものであり，変換作業時に処理内部では形態素解析（単語の切り出し）を行うのが普通であるから，その情報を捨てずに使えば簡単に実現できるはずである．捨てている情報を活用すればよいのである．

そこで，漢字かな混じり文字列を入力する時に利用者が与える読みと文節区切り情報を，変換後得られるテキストとともに保持することを考案した．テキストは変換される単位（＝文節）の並びからなり，各単位について，その字面（漢字かな表記）と読みなどを対応づける．テキストにおける字面のみを並べたものが漢字層，読みのみを並べたたものが読み層を構成する．このように，テキストを複数の層からなる構造を持つものと考えることにした[1]．

また，英文エディタの指令を調査し，漢字層に対して働くべきか，読み層に対して働くべきかを検討し，適切な層に対して働くようにした．つまり，画面の表示は漢字層で行い，カーソル移動などの字面に対する指令は漢字層に対して行う．英単語を１字ずつ打鍵するごとに順次探索が進むといった漸増探索（incremental search）や，単語の途中まで打鍵して残りの部分を補間するといった動的展開（dynamic abbreviate expansion）のような字面の内容に関係する操作は，読み層の上で働くことになる．挿入した字面（漢字層）上の誤り（ミススペルとか誤変換）を修正しようという場合も読み層で働く．本研究では，このようにして読み情報を活用したテキストエディタを設計，作成した．また，既存のエディタとは独立して作成すると，実際に利用されることが少なく，おもちゃとなりかねないし，読み層を持ち，それを活用するといったことの有効性を客観的に検証することも難しいので，実際に広く使われているエディタ（Emacs）の上に実用品として利用できる形に構成することにした．本エディタは GNU Emacs 系のエディタである Nepoch 上に，編集指令系ができる限り互換になるように実現した．これを JEM（Japanese Editor with Multi-layered text structure）と名付ける．

なお，かな漢字変換時に入力した読み情報を保持しているエディタ（ワープロ）はすでにいくつか実用化されているが，それらは再変換といったかな漢字変換における不具合の修正などに利用することを意図したものであり，漸増探索や動的展開といった英文エディタで使われる操作を利用者が同様の使いやすさで日本文に対して使うことを考えて作られたものではない．JEM は，英文エディタで得られている使いやすさをできるだけ自然な形で，日本文エディタに適用しようとしたものである．

以下，本文では，JEM の外部仕様，内部構成について述べ，その有用性を実験を通して検討する．

2. JEM の設計

本章では JEM の外部仕様について述べる．

2.1 設計方針

JEM は以下の方針で設計した．

（1） 読み情報を保持して，様々な指令で活用できるようにする．

（2） 実用性を考えて，現在ワークステーションで標準的に使われているエディタ Emacs の編集指令との互換を保つようにする．

（3） Emacs の各種編集マクロをなるべく書き換えることなしに JEM の機能と共存させて使うことができるようにする．

2.2 多層構造

ユーザから見ると，JEM は次のように振舞わなければいけない（**図1**）．

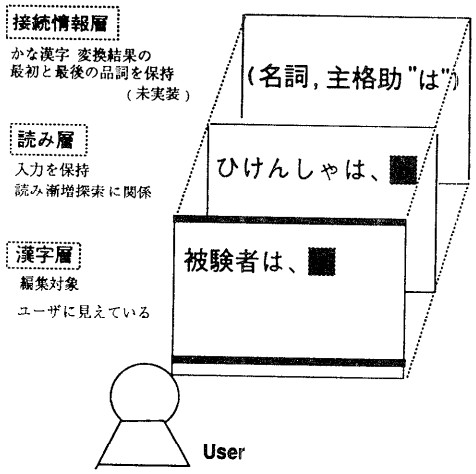


図 1 ユーザから見た JEM のイメージ
Fig. 1 What does JEM look like to a user.

（1） 各文節の入力時の読みを理解している．
（2） 各層のカーソルが同期して動く．
（3） 漢字層の変更は各層に反映される．

なお，ここで文節と呼んでいるのはかな漢字変換機構を使ってわかち入力する単位のことであり，国文法でいわれている文節と一致しない場合もある．

2.3 漢字層と読み層

Emacs でいう**バッファ**とは，編集対象とするファイルの内容を編集するために一時的に入れる場所のことである．**漢字層**は利用者が普段編集対象としているバッファのことである．漢字層の読み情報を利用できる形で保持するバッファを**読み層**とよぶ．漢字層と読み層はそれぞれ別のファイルに格納する．読み層に関連づけられたファイル（読み層ファイル）は漢字層に関連づけられたファイルの名前に拡張子 ".yomi" を付加した名前がつけられる．漢字層ファイルの読み込み時にもし読み層ファイルが存在しない場合は，仮名漢字変換システム **Wnn**[2]の逆変換を用いて漢字層ファイルから自動生成する．利用者が指定するのは漢字層のファイルだけである．

読み層と漢字層との対応づけは文節単位で行う．必要があるまでは文節内の漢字１文字ごとの対応づけはとらない．

なお，仮名漢字変換システムにある単語登録機能を使って，たとえば，「プログラミング」を「ぷろ」で入力できるようにした場合は，漢字層が「プログラミング」，読み層が「ぷろ」となる．単語登録機能を多用した文書は，他人にとってその読みが不自然なものとなってしまう．

2.4 削除・挿入の扱い

文節内の読み層と漢字層の対応づけは必要な場合（削除とか挿入）に動的に行う．文節内への文字の挿入や，文節内の文字の削除が起こると，文節の単位が変更されることになる．その場合に，新たにできた文節に対して読みが付けられる．新たな読みは挿入や削除がされる前に該当文節が持っていた読みと部分一致するように選ぶ．一致しない場合は，対応する漢字層から漢字かな変換によって読みを付ける．

たとえば，「ご職業は」の「業」を削除すると，文節は「ご職」と「は」に分割され，それぞれの読みは「ごしょく」と「は」になる．全体の読みは「ごしょくは」となる．

また，「大和」（やまと）から「大」を削除すると，「和」が残り，それに対する読みは「わ」となる．

このように文節内で削除や挿入を繰り返すと文節が細かく分かれてしまうことがある．

2.5 読みによる漸増探索

Emacs で使われる漸増探索（incremental search）では，探索したい文字列の最初の文字を打鍵すると直ちに探索が開始される．探索したい文字列を順に打鍵するたびに，それまでの打鍵列に一致する文字列がどこにあるかを表示していく．探索したい文字列全部を打鍵することなく，目的の位置を特定するのに必要なだけの文字を打鍵すればよいので大変便利な指令である．

具体的にみると，C-s（制御キーを押しながら s を押す）で，漸増探索モードに入る．続いて 'a' を打鍵した場合には，テキストの最寄りの "a" の右にカーソルが移動する．続いて 'b' を打鍵するとカーソルは，探索を開始した場所以降の最初に現れる "ab" の後ろに移動するといった具合である．

JEM の読みによる漸増探索では，探索する文字列の読みを打鍵して，上記の漸増探索を行う．ローマ字で入力する場合は，自動的にローマ字が仮名に変換され，仮名が確定するたびに読み層での探索が行われる．カーソルは対応する漢字層の位置に移動する．

また，全角半角は区別しない，複数行に渡る文字列を探す時に改行を意識しないで探索できるという特徴がある．探索文字列にアルファベットを与えたいときは入力のモードをトグルするキーで切換えて入力する．なお，正規表現による漸増探索は行えない．

2.6 読みによる動的展開

Emacs で使われる**動的展開**（dynamic abbreviate

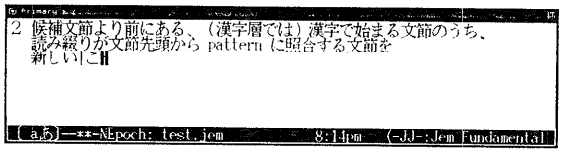


図 2 動的展開の様子（展開直前）
Fig. 2 An appearance of the dynamic abbreviation. (just before the expanding)

図 3 動的展開の様子（展開直後）
Fig. 3 An appearance of the dynamic abbreviation. (just after the expanding)

expansion）は，現在入力中の単語（文節）と同じ表記で始まる単語を現在編集中のテキストから探して，それと置き換えるといった入力方法である．動的展開はプログラムなど，同じ単語を頻繁に使うテキストの編集で特に有効である．

しかし，動的展開の対象となる文字列は字面に対応しているので，漢字を含む語を動的展開しようという場合，入力した文字列（ローマ字ないし平仮名）を漢字に変換しなければ対応づけることができない．

JEM の読みによる動的展開は読み層の上での展開を行うものである．入力した文字列は読みに変換された後，そのまま読み層において略称と展開結果の対応づけが行われ，読み層の展開結果に相当する位置の漢字層文字列が展開結果となる．このように，JEM の読みによる動的展開は，動的展開という概念を日本文に対してごく自然に実現したものである．

実行例を**図 2**，**図 3** に示す．「こ」と打ち込んだ図 2 において，動的展開指令を与えると，見つかった「候補文節」に展開される（図 3）．

2.7 再 変 換

かな漢字変換にとって誤変換はつきものである．思考の流れを中断せずに速く打ち込みたい時などに顕著である．誤変換は後で直す作業が必要になる．最も単純な修正法は誤った部分を削除して入力し直すことである．しかし，変換結果は間違っていても，その読みは正しいのであるから，読み綴りを再利用できれば，再び打ち込む手間が省ける．読みを再利用して変換することを**再変換**と呼ぶことにする．JEM では入力時に保存した読み情報を利用して再変換することができる．読み綴りや文節区切りの情報を保管していない従来の方法でも漢字かな混じり文字列を形態素解析して読み綴りに変換すれば再変換ができなくはないが，精度が高くはならない．

3. JEM の実装

複数のバッファによる多層構造を用いて，読み情報・文節区切り情報を保持する．これは，編集対象のバッファ（漢字層バッファ）を，読み情報を保持しているバッファ（読み層バッファ）に文節単位で関連づけることで行う．また，計算機資源との兼ね合いとして，プログラムサイズが大きくなったり，メモリを多く消費することがあっても，原則として**高速化と使いやすいユーザインタフェース**の方を優先する．

3.1 JEM のシステム構成

JEM は Emacs の X-Window 対応版である Epoch[3]の日本語対応版 Nepoch 1.1 の上に Emacs lisp によって書いたものである．X11 R5 の上で稼働している．

日本語環境としては Wnn V4[2]を，日本語フロントエンドプロセッサには EGG（たかな）を用いている．JEM は Emacs Lisp で約 9000 行である．そのうち，読みによる漸増探索の部分は約 1400 行，動的展開の部分は約 900 行である．**図 4** に JEM のシステム構成を示す．なお，実用のために Nepoch 1.1 の内部パラメータを若干変更してある．

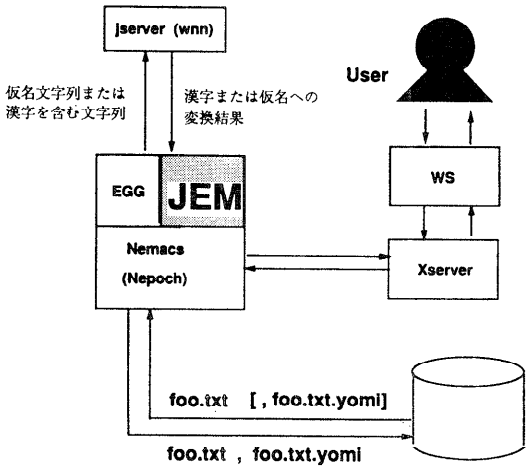


図 4 JEM のシステム構成
Fig. 4 Configuration of JEM.

3.2 読み情報の保持の実現

漢字層の第 i 行の第 j 番目の文節の読み情報を読み層の第 i 行の第 j 番目の**文節読み情報レコード**として保持するようにする．文節読み情報レコードは下の(1)から(4)を連結した文字列である．読み層の各行は0個以上の文節読み情報レコードからなる．各行は改行コードで区切られる．

(1) レコード先頭を表す文字，
(2) 漢字層での長さを表す文字列，
(3) 読み綴り文字列，
(4) レコード末尾を表す文字．

かな漢字変換をせずに直接入力した文字列とかな漢字変換で変化しない文字列はそのままの文字列を(3)の読み綴りとして保持する．通常の辞書登録では，

"a", "#", "「", "1", "^I"

などがそれにあたる．

(2)は漢字層・読み層を対応づけるために使用する．(2)が意味するものは，その文節の漢字層におけるバイト単位での長さである．現在の版においては，16種類の文字を2個用いることで長さ0から255を表現している．(1)と(4)は読み情報に関する処理を可能ないし実装しやすくするためである．また，読み層の探索を既存の関数で効率良く行えるように(1)(2)(4)は読み綴りに現れないと考えられる文字群として，現在の版ではロシア文字18文字（全角）を利用している．他の文字群を用いるようにすることも可能である．

文節読み情報レコードの例として漢字層が次の3行からなるテキストを考える：

1：彼は新しい
2：gnus.el を
3：インストールしました．

このとき，読み情報としては

1：[4]かれは[6]あたらしい
2：[5]gnus.[5]elを
3：[22]いんすとーるしました．

が保持される．“[4]”等に相当するものとして上記(2)の“漢字層での長さを表す文字列”を使う．

3.3 読み層の同期の実現

GNU Emacs 系のエディタでは，編集指令は Emacs Lisp または C で記述され，内蔵のインタプリタで評価実行される．また，Nepoch はテキストの変更（挿入・削除）を行う前後に before-change-function, after-change-function（以下 BCF, ACF と略す）という2つの変数の値（関数）を呼ぶように作られている．そこで，BCF, ACF に任意の関数を割り当てることによって，テキストの変更前後のエディタの挙動を自由に修飾することが可能である．

JEM では，BCF と ACF に漢字層の変更を読み層に反映させる lisp コード（jem-synchronous-hook）を割り当て，それが呼ばれることで読み層の同期を実現する．漢字層の変更が削除ならば読み層の対応範囲を計算して削除を行い，挿入ならば対応する読みを読み層の対応位置に挿入する．詳しくは以下のとおりである．

漢字層の何らかの文字列が削除される場合の過程：
(1) BCF が呼ばれる．削除が起こることと，漢字層の削除される範囲がわかる．削除位置に対応する読み層の位置を案分比例によって推測し，かな漢字変換してみて漢字層と対応するように読み層の位置を調整する．うまく行かない場合は漢字層から逆変換（漢字かな変換）をして読みを定める．
(2) 漢字層の削除がおこる．
(3) ACF が呼ばれる．漢字字層の文節区切り表示の更新．（これはオプション．）

漢字層に何らかの文字列が挿入される場合の過程：
(1) BCF が呼ばれる．（挿入が起こることがわかる．）
(2) 漢字層に挿入がおこる．かな漢字変換のために入力した読み綴りを一時保管しておく．
(3) ACF が呼ばれる．挿入された文字列とその位置がわかる．これをもとに一時保管しておいた読み綴りの挿入場所を計算し，読み層に挿入する．

なお，カットやペーストも正しく動作する．変更の取り消し（undo）機能も対応する取り消し用バッファの許す限りさかのぼることができる．ただし，文節が細かく区切れてしまう場合もある．

現在の実現法では削除，挿入，変更の取り消しなどで文節が細かく分かれてしまう場合がある．連接する細かな文節を統合して，文法的に正しい文節にする手順については現在検討中である．並列に処理を行うことによって実現できる見通しである．

4. JEM の評価実験

読み情報をもつことによる利点の最たるものの1つと考えられる読みによる漸増探索について，その有用性を客観的に検証するために，評価実験を行った．Emacs では，英文テキストに対する漸増探索は広く

使われている．プログラム作成中などにおいて現在画面上で注視している場所へのカーソル移動の手段としてもマウスによる移動でなく漸増探索を使うことは，筆者らもよく採る手段である．

利用者はカーソルと目標の距離が近い場合にはカーソル移動指令を使うものである[4]．カーソル移動指令を使わなくなる境界距離を d_0 とすると，d_0 よりも距離が離れている状況には次の2通りがある．

（a） 目標がどこにあるか分からない．
（b） 目標が画面内にあり，かつ視認している．

目標にむかってカーソルを移動する指令として以下の3つをとり上げる．

（1） マウスボタンによる移動
（2） 従来の漸増探索（かな漢字変換が確定するたびに探索が行われる）
（3） 読みによる漸増探索（仮名が確定するたびに探索が行われる）

上記の各状況において，3つの方法のどれが優位かを考えてみる．まず，（a）の状況を考える．マウスを使う場合は，今見ている画面内に目標があるかどうかを目でみて判定しなければならない．それには平均して数秒はかかる．その画面にない場合は次の画面に進めて，また同様に目でみて探すことになる．本をめくってある単語を探すような状況であり，探索指令を利用する方が自然かつ効率がよいことは明らかである．したがって，漸増探索（2）と（3）の比較を詳細に見ていくことにする．

例えば，“情報”という単語を漸増探索するのに（2）では “C-s C-k j o u h o u SPC RET RET ESC” となるのに対し，“じょ”という読みが目標以前に現れないならば（3）では “C-s j o ESC” という打鍵で探索することができる．（a）の状況において，（2）と（3）を比較した実験[5]によると，「従来型の漸増探索の1回の所要時間は読みによる漸増探索の1回の所要時間の 1.25 倍よりも大きい．（有意水準 95%）」という結果が得られている．ただしここで所要時間とは，「これを探せ」といった問題を獲得する時間と実際に遂行する時間の和となっているので，問題の遂行時間だけで考えるならば読みによる漸増探索はさらに速いことになる．所要時間の差は打鍵数が少ないことによっている．以上により（a）の状況では（3）つまり読みによる漸増探索が優れている．

次に（b）の状況について考える．（1）から（3）が利用されうるが，（2）は（3）より劣っているので，（1）と（3）のどちらが優位かを検討する．

4.1 仮　　説

Card らによる打鍵レベル模型[4]を用いて，カーソルを目標まで移動するというタスクに対する次の2種類の手法（メソッド）における利用者が実行（遂行）する時間を予測する．いずれも，タスク内容（どこへカーソルを移動させればよいか）の獲得はすでに終っているものとする．

（1） **手法1（マウスの場合）** 片手をマウスに移し（H），画面上のあらかじめ指示されている位置にマウスポインタが入るよう，マウスを動かして，そのマウスの左ボタンを押す（P, K）．そして，マウスを持っている手をキーボードに戻し（H），改行キーを打鍵する（K）．

（2） **手法2（読みによる漸増探索の場合）** 読み漸増探索を起動し（M, K(Ctrl), K(s)），探索の読み綴り文字列をタイプ（*nK*(*char*)），計算機の反応時間があり（R），改行キーを打鍵する（K）．なお，目標に対する読みを打鍵すれば目標に一意に到達するという状況を仮定する．そうでない状況ではまた別の手法をとると考えられるので，ここでは一意に到達する状況に限定して考える．

（1）の場合，

H＋P＋K(左ボタン)＋H＋K(RET)
＝2H＋2K＋P
＝2.30秒
(H＝0.40秒, K＝0.20秒, P＝1.10秒とした場合)

（2）の場合，たとえば文字列を “情報” とすると，

M＋2K(Ctrl, s)＋nk(j o u h o u)＋R＋K(RET)
＝M＋(n＋3)K＋R
＝M＋9K＋R （n＝6の場合）
＝M＋1.80＋R秒 （K＝0.20秒とした場合）

ここで，Mオペレータは心的準備にかかる時間であり，文献[4]では1.35秒とされているが，それは様々な状況での平均値であり，特定な作業に熟練した場合はより小さくなると言われている．また，次に打鍵するコマンド（C-s）があらかじめ予想されるので，小さく考えてもかまわない．たとえば，0.5秒と考え，Rは0秒とすると，2.3秒となり，マウスの場合の予測値と一致する．実際のところどうであるかを調べるために，実験を行った．

4.2 被 験 者

被験者は筆者らの研究室から募った，男性8名（う

ち，40歳代1名，20歳代7名）である．全員とも，WS上で日常的にプログラム開発や日本語の文書の作成を行っている．

4.3 装　　置

（1） ワークステーション SUN Sparc Station ELC.

（2） ディスプレイ ディスプレイは17インチ，モノクローム．

（3） フォント フォントはミニバッファの部分にはa 24，kanji 24を，それ以外の部分にはa 18，kanji 18を使用した．

（4） 画面配置 問題文が属するウインドウ（スクリーン）を中央に置く．このウインドウの縦方向は問題文が36行表示されるサイズである．ミニバッファのウインドウを画面下方に置く．

（5） 問題文 問題文はヒューマンインタフェースを題材とする論文3編をもとに作成した．長さは750行で，空白以外の記号を削除し，改行コード以外のすべての半角文字を全角に変換した．各行は長さ35文字で折り返すように編集した．

（6） 移動先目標 被験者はこの場所にカーソルを移動させる．各目標は必ず文節先頭から始まり，大きさは4文字分である．目標は全部で61件あり，次のようにして選んだ．

i番目の目標からd行先の，c番目の文節の先頭から4文字分を$i+1$番目の目標とする．dは1から35の，cは1からその行の文節数までの一様分布の乱数を発生させて決めた．dの最大値を35としたのは，目標が必ず画面に見えるdの最大値だからである．目標が複数行に渡った場合と行頭・行末の5文字分にかかった場合はcを選び直した．1番目の目標は0番目の目標がテキスト先頭にあると考えて選ぶ．テキスト末尾までいったらテキスト先頭から繰り返す．これを適当な回数繰り返した（76件）．

76件から，視点（カーソルの指す文字と直前の文字との間の位置を表す）の初期位置から目標の直前までの間に目標と読みが一致する文字列が存在する場合は手法2が使えないので除いた．これは76件中9件だった．この実験で使用した文書では1割強の場合，目標に対応する読みを打鍵してもその目標に達しないので，探索を継続する，ないしは，目標に達するまでより長く読みを打鍵し続けるといった手法2とは別の手法を使う

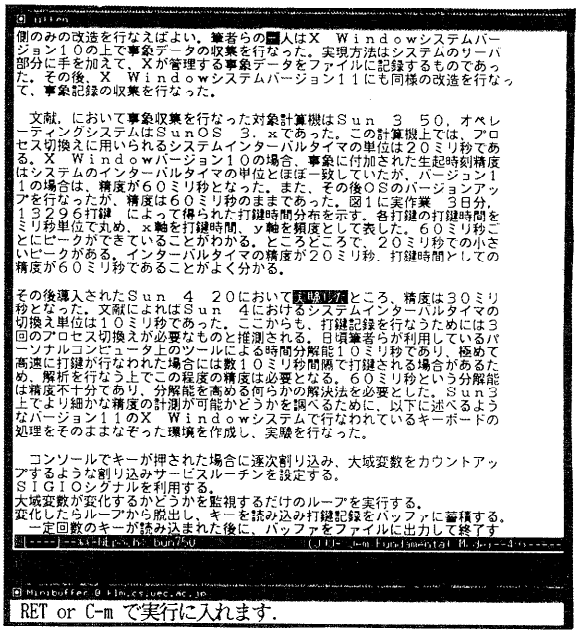


図5 実験での漸増探索（問題獲得画面）
Fig. 5 An appearance of the incremental search in the experiment. (acquiring the problem)

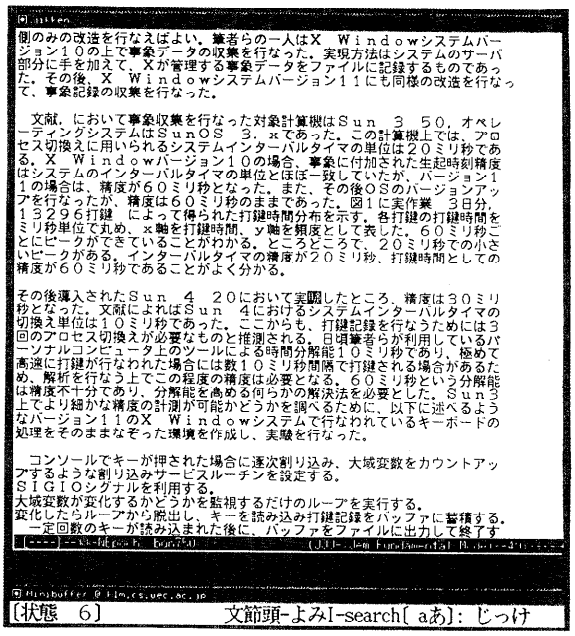


図6 実験での漸増探索（探索成功状態）
Fig. 6 An appearance of the incremental search in the experiment. (searched successfully)

ことになる．その場合は手法2よりもより時間がかかると考えられる．この点については考察で述べる．

手法2が使えない場合に加え，折り返し直後の1件と，被験者のウォームアップ用としての最初の5件を除いて，残った61件を結果のデータとなる移動先目標とした．

(7) データの収集 打鍵やマウスボタンの押し下げがあるごとに，その種類と時刻（精度は約30ミリ秒），呼ばれた指令があれば指令名，などを採集するようにJEMを改造して用いた．

4.4 手　順

原稿修正タスクによく現れるカーソル移動操作を題材とした．現実の原稿修正タスクは紙（赤入れなどをしたハードコピー）を元にすることが多いと思われる．しかし，それでは紙のページをめくる作業が必要な場合とそうでない場合が生じるため条件が一定でなくなる．そこで実験では，若干マウス移動に有利となるが，計測の誤差が少なくなるように，目標を紙によって示すのではなく，画面上で単語を反転させることで与える（**図5**）．問題獲得後は，反転でなく下線表示にする（**図6**）．読みによる漸増探索では，その文字列に対する読みを打鍵する．マウス移動では，その位置にマウスカーソルを移動させる．したがって，実験の手順では動作実行時間について，マウス移動の方が，原稿で与えた場合に比べて位置読み取りの手間がなくなっている分だけ有利である．また，獲得時間と動作実行時間を分離して測定するために，その移り変わりで改行キーを打鍵しなければいけないようにした．

カーソルを移動させる方法の違いで以下に述べる2つの実験（1Yと1M）を行う．被験者8名を2つのグループに分け，sub. 1からsub. 4は実験1Mを先に，他の被験者は実験1Yを先に行った．問題数は61問であり，実験1Yと1Mで共通の問題を使用した．被験者は2つの実験の間に数分間休憩する．

4.4.1 実験1Y：読みによる漸増探索を使う場合の手順

実験1Yの手順の状態遷移を**図7**に示す．

（1） 休憩画面状態

この状態では問題文は隠れている．被験者はここでは何秒待っても構わない．被験者がRETを打鍵すると次の(2)の状態に移る．

（2） 問題獲得画面状態

(2)では現在視点を前回の正解の位置に置いたまま，そこが最上段に来るよう画面をスクロールさせ，マウスポインタを現在視点と同じ位置に置く．これは，初期位置を偏らせないためである．そして，目標の部分を反転表示する（図5）．被験者は問題を理解したらRETを打鍵する．すると次の(3)の状態に移る．

（3） 動作実行画面状態

(2)で反転表示されていた目標部分はここでは下線表示に切り替わる．被験者は読みによる漸増探索を使って目標内へ現在視点を移動させ（図6），移動させたらRETを打鍵する．そのとき，目標の先頭から照合していれば正解とみなされて(1)の休憩画面状態に切り替わる．

4.4.2 実験1M：マウスによる移動を使う場合の手順

（1） 休憩画面状態

実験1Yに同じ．

（2） 問題獲得画面状態

実験1Yに同じ．

（3） 動作実行画面状態

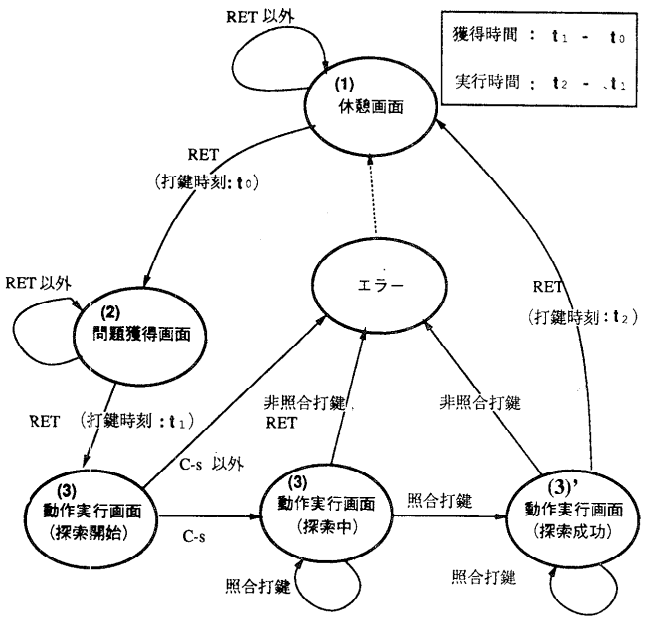


図7 実験1Yの状態遷移図
Fig. 7 The state transition diagram of the experiment 1Y.

(2)で反転表示されていた目標部分は(3)では下線表示になる．被験者は目標位置までマウスポインタを動かし，マウスの左ボタンを押すとカーソルがマウスポインタ位置に移動する．そこで RET を打鍵し，もしマウスポインタが目標内に入っていれば正解とみなされて(1)の休憩画面状態に切り替わる．

4.5 結　果

以下のデータは除いた．

(1) エラーが起こったもの．

(2) 探索文字列の入力中で，2.0 秒以上かかった打鍵が存在するもの．

これは被験者が何か別のことを考えていた．ボーっとしていた場合を除くためである．被験者が動作実行画面状態に移るために RET を打鍵してから，カーソルを移動させて RET を打鍵するまでの時間を「実行に要した時間」と考えて，それを集計した結果を**表1**に示す．各セルの上段がマウス，下段が読みによる漸増探索である．

4.6 考　察

表1をみると，sub.3 は読みによる漸増探索が優り，sub.4 はほぼ同等であるが，他の被験者では平均的にマウスの方が速い．マウスの場合の実測値は仮説の値(2.3秒)とほぼ一致しており，標準偏差も小さいことから熟練状態にあると考えられる．読みによる漸増探索の場合は，標準偏差が大きい．実際に sub.3, sub.4 は若干トレーニングをしているが，他の被験者は熟達しているという状況ではなかった．このことから，トレーニングをつむことによってまだまだ時間が短くなることが予想できる．

表1 実験結果
Table 1 The result of the experiments.

被験者	最小値(秒)	平均値(秒)	最大値(秒)	標準偏差(秒)	有効問題数	読みによる漸増探索が速かった回数
sub. 1	1.47 1.86	2.01 3.38(1.37)	2.90 6.69	0.34 1.20	39	2
sub. 2	1.44 1.51	1.95 2.45(0.50)	2.73 4.92	0.32 0.66	45	12
sub. 3	1.82 1.44	2.52 2.47(−0.05)	3.48 3.45	0.36 0.50	44	23
sub. 4	1.71 1.68	2.33 2.49(0.16)	2.96 4.29	0.32 0.57	40	17
sub. 5	1.89 2.85	2.21 4.39(2.18)	2.52 6.66	0.18 1.02	38	0
sub. 6	1.58 1.47	2.01 2.52(0.51)	2.67 4.77	0.30 0.66	26	6
sub. 7	1.34 2.10	2.17 2.85(0.68)	3.44 3.75	0.43 0.48	30	3
sub. 8	1.52 1.62	1.97 2.63(0.66)	2.56 4.80	0.31 0.81	25	5

上段：マウス，下段読みによる漸増探索．カッコ内は差．

読みによる漸増探索が速かった回数を見てみると，sub.5 以外は読みによる方が速かった場合があることがわかる．また，最小値を見ると，被験者 sub.5 と sub.7 以外は2つの方法に大差がなく，少ない打鍵で所望の位置に移動できる時は（つまり，うまい状況では）読み漸増探索がマウスに匹敵する（あるいは勝る）ことがわかる．

この実験では問題を画面で与えたが，原稿で与えるというより一般的な場合には，マウスによる移動は位置読み取りの手間が必要になる分だけ遅くなる．また，手法2で扱えない1割強の場合には読みによる漸増探索の方が時間がさらにかかることになる．

これらを考え合わせると，ほとんどの被験者にとっては，読みによる漸増探索の方がマウスよりも速くできる状況がかなりあることが分かる．時間的にはマウスによる移動はそんなに速いわけではなく，読みによる漸増探索がカーソル移動手段としても十分使えるといえる．もちろん，以上の議論は目標が同一画面にある場合のことであり，同一画面にない場合には読みによる漸増探索が圧倒的に速いことは明白である．

5. おわりに

読み情報（読み綴り，文節区切り）を保持することで，日本文の編集がより容易になることを JEM により示した．JEM は GNU Emacs 用に書かれた，ほとんどの既存の編集マクロを書き換えなしで使用することが可能である．たとえば Emacs Lisp の simple.el というファイルに属する会話型関数群では9割強は無修正のままで利用できた．

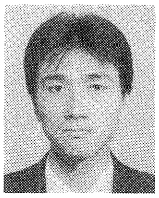
今後の課題としては，カーソル移動の方法としての，読み漸増探索とマウスのより精密な実験と評価，さらに動的展開に対する実験，評価があげられる．

参考文献

1) 角田博保：多層テキスト構造を持つ日本語エディタ，情報処理学会第27回プログラミングシンポジウム報告集，pp. 75-84 (1986. 1).
2) KABA：Wnn＋GMW 入門 Wnn 解説編，岩波書店 (1990).
3) Kaplan, S., Carroll, A. M., Love, C. and Laliberte, D. M.: *Epoch*, University of Illinois, Urbana-Champaign (1990).

4) Card, S.K., Moran, T.P. and Newell, A.: *The Psychology of Human Computer Interaction*, Lawrence Erlbaum Associates (1983).
5) 畠山 勉，角田博保：読み情報をもつ日本語エディタの作成と評価，情報処理学会ヒューマンインタフェース研究会 93-HI-47, pp. 125-132 (1993. 3).

(平成6年1月15日受付)
(平成6年10月13日採録)



畠山 勉（正会員）

1965年生まれ．1991年電気通信大学電気通信学部計算機科学科卒業．1993年同大学院博士前期課程修了．同年(株)日立製作所デザイン研究所入所．人間と機械のインタラクションなどに興味をもつ．

角田 博保（正会員）

昭和25年生まれ．同49年東京工業大学理学部情報科学科卒業．同51年同大学院修士課程修了．同57年同大学院博士課程修了．理学博士．同年電気通信大学計算機科学科助手，平成2年同大学情報工学科講師，同4年助教授，現在に至る．文字列処理，プログラミング方法論，ヒューマンインタフェース等に興味を持つ．ACM，日本ソフトウェア科学会，日本認知科学会各会員．