

# 検索技術を用いた作文支援

高林 哲      松本 裕治

奈良先端科学技術大学院大学情報科学研究科

{satoru-t,matsu}@is.aist-nara.ac.jp

## 1 はじめに

辞書と文献の山を机に広げ、ペンを握って文章を書くという旧来のスタイルと違い、現代では計算機上で辞書や文献を参照しながら、キーボードを用いて文章を書くことができる。しかし、作文と検索の過程には依然として障害があり、理想的な作文環境にはほど遠い。作文の過程を、言葉を探すという行為の連続と捉えれば、作文と検索は密接に結びついているといえる。本研究では、この関係に注目し、検索技術を用いた作文支援の手法を提案する。

## 2 作文における検索の必要性

人間の記憶力は限られているため、作文の際にはさまざまな外部の情報を必要とする。辞典や事典は人間の記憶力を補うための外部記憶といえる。また、技術的な文書を執筆する際には、インターネット上などの情報を検索して利用することが多い。作文における情報利用の典型的な流れを次に示す。

1. 情報を検索
2. 情報をコピーアンドペースト
3. 情報を加工
4. 加工した情報を元に文章を仕上げる

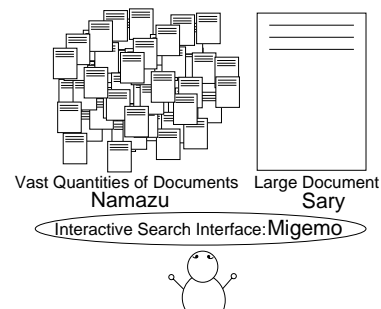
この過程ではテキスト編集と情報検索の2つの作業を要する。しかし、現在の作文環境では、これらの作業をスムーズに行うことができない。たとえば、インターネット上の情報を利用するには、テキストエディタ(またはワープロ)から離れて、Webブラウザに移動して情報収集を行い、再びテキストエディタに戻って、見つけた情報をペーストする必要がある。この一連の作業には手間がかかり、人間を疲労させる。

このような不便は検索と作文の過程が統合されていないことに起因する。我々は検索と作文をなめらかに統合することによって、よりよい作文環境が実現できると考えている。

## 3 検索技術

土台となる検索技術として、これまでに、3つの実用的な検索システム *Namazu*<sup>1</sup>, *Sary*<sup>2</sup>, *Migemo*<sup>3</sup> の実装に取り組み組んできた。いずれもインターネット上から入手可能である。

- **Namazu** 大量の文書を高速に全文検索するソフトウェア。元々は高林 [10] によって開発されたが、現在は有志による共同開発が行われている。
- **Sary** 巨大な文書を高速に全文検索するためのライブラリおよびツール。データ構造に Suffix Array[3][9] を採用している。
- **Migemo** 動的パターン展開による漸進的検索を実現するための検索インターフェイス。Emacs[5] (テキストエディタ) 用の拡張モジュールとして実装した。



<sup>1</sup><http://www.namazu.org/>

<sup>2</sup><http://sary.namazu.org/>

<sup>3</sup><http://migemo.namazu.org/>

## 4 検索技術を用いた作文支援

前節で取り上げた検索技術を用いて、我々は次のような作文支援に取り組んできた [6]。

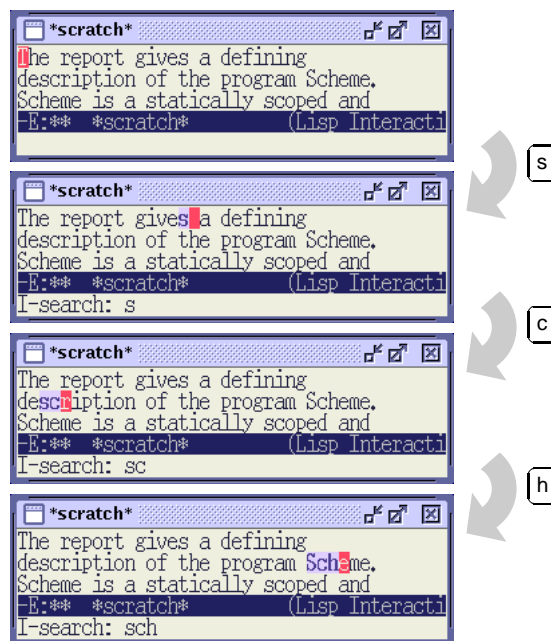
- 検索と連動したコピーアンドペースト 「検索文を選択 コピーアンドペースト」という3つのステップを、「検索しながら文を選択してそのまま貼りつける」という1つのステップに統合した。
- 入力予測 増井 [4] は利用者の過去の入力に基づいて次の単語を予測する入力方式 POBox を提案している。我々は、静的な辞書を利用する代わりに、過去に入力したすべてのテキストを動的に検索するという手法で入力予測を試みた。
- 例文検索 母語でない言語で作文をする際には、頻繁に辞書を利用する。しかしながら、辞書に載っている例文には限りがあり、特に専門用語の用例は皆無に近い。そこで、我々は専門家によって書かれた大量の論文から適切な例文を漸進的に検索するシステムを構築し、テキストエディタへ統合した。
- 文章表現の検証 利用者の入力した表現が自然かどうか、ネイティブの書き手によって書かれた大量の文書と照合して検証するシステムを構築し、例文検索とともにテキストエディタへ統合した。
- 日本語の漸進的な検索 Emacs などのテキストエディタでは、利用者が1文字入力するたびに検索を進めていく、漸進的な検索が行える。しかし、日本語では、一般に、入力にかな漢字変換を必要とするため、このような漸進的な検索は行えない。我々は日本語の漸進的な検索を実現するために、動的パターン展開という手法を考案した。

本稿では、上に挙げた作文支援システムの中から、日本語の漸進的な検索および、その基盤となる動的パターン展開について詳しく取り上げる。

## 5 日本語の漸進的な検索

Emacs などのテキストエディタでは、利用者が1文字入力するたびに検索を進めていく、漸進的な検索 (in-

cremental search) が行える。下の図は ‘scheme’ という単語を漸進的に検索した過程である。



この例では、利用者は、 $\boxed{^S}\boxed{s}\boxed{c}\boxed{h}$  の4打鍵で目的のテキスト位置に到達している<sup>4</sup>。普通のキーワード検索では最低でも  $\boxed{s}\boxed{c}\boxed{h}\boxed{e}\boxed{m}\boxed{e}$  の6打鍵を必要とする。漸進的な検索では ‘flocinaucinihilipilification’ のような長い単語でも最初の数文字を打つだけで検索できる。

しかし、日本語で漸進的な検索を行おうとすると、かな漢字変換という壁につきあたる<sup>5</sup>。漸進的な検索では1文字入力するたびに検索を進めていくという点が重要だが、かな漢字変換には、ある程度まとまった単位の入力が必要であるため、スムーズな漸進的な検索は行えない。

そこで、我々は動的パターン展開という手法を考案し、日本語の漸進的な検索を実現した。動的パターン展開とは、利用者が1文字入力するたびに動的に正規表現のパターンを展開して検索する手法である。下の図はキーワード「活発」を動的パターン展開によって漸進的な検索を行っている過程である。

<sup>4</sup> $\boxed{^S}$ キーは検索を開始する。 $\boxed{ctrl}$ キーを押しながら $\boxed{S}$ を押す。  
<sup>5</sup>T-Code[2], TUT-Code[8] といった、漢字を直接に入力する方式もあるが、一般的ではない。



この例では、利用者は `[S][k][a][p]` という4打鍵で目的のテキスト位置に到達している。かな漢字変換を行う検索では、`[S][KANA][k][a][p][p][a][t][u][CONVERT]` と11打鍵を要したはずである<sup>6</sup>。

## 6 動的パターン展開

上の「活発」の漸進的検索に対応する動的パターン展開は、SKK[1]の辞書を用いて、次のように行われる。

**k**

```
[k  .,.;:?!"'^ _\>>>#全々〇ー
/\~|... ' '( ) [ ] { } 《 》 「 」 『 』
【 】 + - ± × ÷ = < > 。 ￥ $ ¢
£ % # & * @ $ ... (5,085 バイト)
```

**ka**

```
[ ~' " ( ) [ ] { } 《 》 「 」 『 』 【 】 × *
かくカケカケ 哀愛囲易畏一蔭
陰隠烏渦影鷗下化飯何伽伽佳加可嘉夏嫁家寡科
暇果架歌河火珂禍禾稼箇... (2,140 バイト)
```

**kap**

```
[ ] |kap| かつ |かば|かび|かぶ|かべ|かぼ
|カッ|カバ|カビ|カブ|カベ|カボ|河童|割賦
|割腹|割烹|喝破|恰幅|活潑|活発|活版|合羽
|勝平|川童|闊歩|濶歩|kap|Kappa
```

テキストエディタ上では、上のように展開された正規表現のパターンを元にして実際の検索が行われるが、

<sup>6</sup> `[KANA]` キーがかな漢字変換の開始と終了を、`[CONVERT]` キーがかな漢字変換を行うキー操作である。

利用者からは動的パターン展開の過程は隠されているため、通常の漸進的検索とまったく同じように操作できる。1文字目の「k」を展開した5,085バイトの巨大なパターンはテキストの多くの位置にマッチするが、「ka」、「kap」と入力を進めていくにつれてパターンは小さくなり、マッチする位置は限られていく。

動的パターン展開による日本語の漸進的検索では、かな漢字変換の作業に煩わされることなく、スムーズに検索が行える。「kikai」で検索をすると「機械」「機会」「奇怪」「器械」のすべてにマッチするという問題が起こり得るが、不適切なマッチは「S」で即座にスキップできるので、大きな問題にはならない。一方、通常のキーワード検索では「こんにちは」で検索をかけると「今日は」は見つからないが、動的パターン展開で「konnnitiha」で検索すれば両方とも見つかるという利点がある。

## 動的パターン展開にかかるコスト

漸進的検索では、利用者にストレスを与えない高速な処理が不可欠である。Migemoでは、利用者の1文字単位の入力に追従するために、「a」、「k」、「s」などの短い文字列に対しては、辞書から動的に正規表現を生成する代わりに、あらかじめ生成した正規表現を用いて、実用的な速度を実現している。しかし、動的パターン展開にかかるコストは計算機が高速になるにつれて無視できるようになると予想できる。増井[7]は計算機資源をふんだんに活用して、優れたソフトウェアを開発する手法「富豪的プログラミング」を提唱している。

ところで、日本語を漸進的に検索する別の方針として、検索するたびにテキストの内容全体をローマ字に変換し、得られたローマ字列に対して検索するという手法が考えられる。しかし、この方法は、テキストが大きくなるとすぐに破綻する。

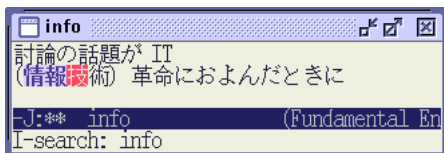
## 動的パターン展開の問題点

動的パターン展開では、辞書を利用して正規表現を生成するため、辞書に載っていない言葉は検索できないという問題がある。また、現在の実装では単語単位の検索にしか対応していないため、「genzainojissou」で

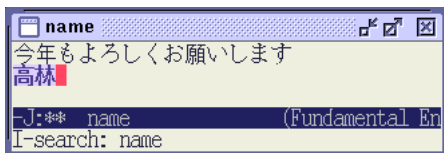
「現在の実装」を検索するような、複数の単語にまたがる検索は行えない。この問題は連文節によるかな漢字変換を内部的に行うことで解決できる。通常の日本語入力に用いるかな漢字変換では適切な変換候補を利用者に提示する必要があるが、漸進的検索のための内部的かな漢字変換では候補の順序を最適化する必要がないため、簡単に実現できると考えている。

## 動的パターン展開の応用

動的パターン展開による漸進的検索は日本語に限らず、中国語などの、文字入力に変換を要する言語一般に適用できる。また、入力の読みと検索対象が一致しないような一風変わった応用も考えられる。次の例では、英和辞書を使って、「information」という英単語で「情報」を検索している。



これは、日本語を学習している英語圏の利用者にとって便利なだけでなく、日本語を母語とする者にとっても役立つ。たとえば、「ソフトウェア」を検索するときは、カタカナ読みの 'sofutouxea' を入力するよりも、元の英単語の綴りである 'software' を入力の方が自然である。実際、Migemo で利用している SKK の辞書には両方の読みが含まれているため、どちらの読みでも検索できる。他にも、人名辞書や地名辞書を使って人名、地名にだけマッチする漸進的検索などの応用が考えられる。



このように、動的パターン展開は、これまで漸進的な検索が困難であった検索対象に対して、幅広く適用できる。さらに、テキストエディタ上の文字列検索だけでなく、一般的な情報検索システムのフロントエンドとしての応用も考えられる。従来の情報検索システムでは、検索質問の全体を入力してから検索を実行するものが多かったが、計算機の性能が向上していくに

したがって、検索質問を入力しているそばから漸進的に検索を進めていく手法が適用可能になりつつある。

## 7 おわりに

本稿では、我々が取り組んできた、検索技術による作文支援を概説し、その1つである、動的パターン展開による日本語の漸進的検索を詳しく取り上げた。今後は、高度な文字列検索とともに、品詞情報などの言語知識を活用した、より柔軟な作文支援を追究していくつもりである。

## 参考文献

- [1] SKK openlab. <http://openlab.ring.gr.jp/skk/>.
- [2] T-code laboratory. <http://openlab.ring.gr.jp/tcode/>.
- [3] Udi Manber and Gene Myers. Suffix arrays: A new method for on-line string searches. In *1st ACM-SIAM Symposium on Discrete Algorithms*, pp. 319–327, 1990.
- [4] Toshiyuki Masui. POBox: An efficient text input method for handheld and ubiquitous computers. In *Proceedings of the International Symposium on Handheld and Ubiquitous Computing*, pp. 289–300, September 1999.
- [5] Richard Stallman. *GNU Emacs Manual*. 2000.
- [6] Satoru Takabayashi. Writing assistance through search techniques. Master's thesis, Nara Institute of Science and Technology, 2001. <http://namazu.org/~satoru/pub/mthesis.ps.gz>.
- [7] 増井俊之. 富豪的プログラミング. *bit*, Vol. 29, No. 1, pp. 36–37, 1997.
- [8] 大岩千穂子. Tut-code homepage. <http://www.crew.sfc.keio.ac.jp/~chk/>.
- [9] 山下達雄. 用語解説「suffix array」. *人工知能学会誌*, Vol. 15, No. 6, p. 1142, 2000.
- [10] 高林哲. Namazu: 全文検索で文書の山に立ち向かう. *情報処理*, Vol. 41, No. 11, pp. 1227–1232, 2000.