類似表現検索による作文支援

吉川拓哉
東京大学学際情報学府*

田中久美子
東京大学情報理工学系研究科†

1 はじめに

本研究では近年盛んに研究が行われてきた類似表現検索技術を応用して作文者を支援するためのシステムを実現する．具体的には過去に記述された文章の中から現在記述している内容と類似した表現を検索する機能をエディタに統合し，表現の再利用を容易に行えるようにする．これにより作文者は文章の一貫性を高めたり，同一表現の繰り返し入力をコピー補完により回避することができるようになる．母国語以外での作文や特定分野における専門表現の駆使が求められる現代社会において，このようなシステムの実現は非常に重要なものとなる．

2 関連研究

類似表現の検索は [3] によくまとめられている．特に Shift-And アルゴリズムが基礎としている Bit-Parallelism という技法については詳しく説明されていて，アルゴリズムを各種分野に適用するために拡張するときに参考になる．作文支援の研究としては [5] や [4] がある．[5] では日本語の文からキーワードを抽出し，キー概念による例文検索を行うことで等価な意味を持つ英文を作成している．インタフェースに Emacs を利用している点は本研究と同じである．そして [4] は表現がどれだけ使われているかという情報を表示することで様々な英語記事を書くことを支援するシステムを研究している．何らかの指標で候補を順序付けるという点が本研究の結果の処理の仕方と共通する点である．

* 東京大学大学院学際情報学府学際情報学専攻
Yoshikawa.Takuya@iii.u-tokyo.ac.jp
† 東京大学大学院情報理工学系研究科創造情報学専攻
kumiko@i.u-tokyo.ac.jp

3 類似表現検索

本研究の類似表現検索は二つの部分よりなる．クエリーに含まれる複数の語を検索する部分と，見付かった語の組み合わせがクエリーとどの程度似ているかを判定する部分である．

3.1 英文複数語検索

複数語を検索するためのアルゴリズムの基礎として Shift-And の複数語検索版 [1] を採用する．その理由は，検索中に単語や文という単位での処理を自然に行えるように拡張できるからである．拡張結果をもとのアルゴリズムと対比する形で示すと図 1 と図 2 のようになる．Shift-And（図 1）では一文字 α を読む毎に状態 d の更新と検索している語を発見したかどうかのチェック，そして状態 d の初期化[*1]を行う．読まれた文字が何であるかは直接文字を調べないと分からない．それに対し，拡張後のアルゴリズム（図 2）では状態 d の更新で用いたテーブル B を工夫したことで，語を区切る文字や文を区切る文字が読まれた場合の処理を速度を落とさずに行えるようになっている．発見チェックや状態の初期化は区切り文字[*2]のところでのみ行う．文字による場合分けを明示的に挿入する必要はない．

3.2 類似度

類似表現検索の類似度を定めるために LCS（Longest Common Subsequence）の長さを利用する．ただし，表現の類似検索なので LCS を計算するための単位は普通の文字ではなくて単語である．例えばクエリーが more and more のような表現であったとすると，more → a，and → b のように各単語に一文字を識別子として割り当てることにより more and more → aba のような単語を表す識別子

[*1] d_i との論理和演算
[*2] スペースやカンマなどを自由に設定可能．

```
α ← 0
for each α in TEXT do
    d ← ((d >> 1) | d_i) & B[α]
    if d & d_f ≠ 0 then FOUND
end
```

図 1　Multiple Shift-And

```
d ← d_i   //初期化
for each α in TEXT do
   d ← d & B[α]
   if d ≥ 0 then    // 区切り文字
      if d > 0 then    // FOUND もしくは文末
         if d < d_eos then   // FOUND で文末でない
         else if d > d_eos then   // FOUND で文末
         else    // ただの文末
      end
      d ← d_i   // 初期化
   else   d ← d >> 1    // 算術シフト
end
```

図 2　英文用 Multiple Shift-And

の列として扱うことになる．

まず普通に LCS を使用することを考えると，クエリーが三つの語の列 abc で与えられるときに二つの語の列 asbttcd と abcd は両方ともクエリーと同じ長さの LCS を持つので区別できない．実際 LCS の長さを μ で表すことにすると

$$\mu(\mathrm{abc}, \mathrm{asbttcd}) = \mu(\mathrm{abc}, \mathrm{abcd}) = 3, \quad (1)$$

となる．この点を改善するため，本研究では隣接関係を表す特殊な文字をクエリー内の各隣接毎に挿入してから LCS の長さを計算する．

$$\mu(\mathrm{a}\xi\mathrm{b}\eta\mathrm{c}, \mathrm{asbttcd}) = 3, \quad (2)$$
$$\mu(\mathrm{a}\xi\mathrm{b}\eta\mathrm{c}, \mathrm{a}\xi\mathrm{b}\eta\mathrm{cd}) = 5. \quad (3)$$

ここで ξ はクエリー内の a と b の隣接に対応し，η は b と c の隣接に対応している．μ の第二引数である二つの表現に対しても，もしクエリー内と同じ隣接が存在する場合には同じ隣接記号が挿入してある．[*3] これにより隣接関係がクエリー内と同様に成り立っている表現の類似度の方が高くなる．

[*3] abcd では ab と bc の間にそれぞれ ξ と η が挿入されている．対して asbttcd の方は a と b や b と c の間に関係ない文字が存在するので隣接関係はないとして何も挿入していない．

その他に，実際の検索において有用な任意の語 $*$ に対応するマッチングも考慮に入れる．例えば in the end の the を任意語で置き換えた in $*$ end のようなクエリーが考えられる．この場合，in と end の間に一語他の語が存在する表現に高い類似度を与えたい．このための拡張は簡単で，記号 $*$ を語の識別子と同様にクエリーに挿入し，対象とする表現に対応する語が存在したときにその語も $*$ に変更して LCS を考えればいい．詳細は省略して例だけ示すと，単語 a と b の間に任意の語 $*$ があるクエリーは aξ*b とエンコードし，sttacbvw と sttabvw のような二つの単語列で表される表現に対しては，前者だけ $*$ に対応する単語 c が a と b の間に存在するから c を $*$ で置き換えて LCS を考える．

$$\mu(\mathrm{a}\xi{*}\mathrm{b}, \mathrm{stta}\xi{*}\mathrm{bvw}) = 4, \quad (4)$$
$$\mu(\mathrm{a}\xi{*}\mathrm{b}, \mathrm{sttabvw}) = 2. \quad (5)$$

ここで ξ も一緒に挿入したのはクエリーで挿入される $*$ の数が 0 のときや 1 より多いときも統一的に扱えるようにするためである．

3.3　語検索と類似度計算の統合

語を検索するためのアルゴリズムと類似度の計算手法が決まったのでそれらを統合する．基本的には英文複数語検索のアルゴリズムで文を区切る記号を読んだときの処理において類似度計算を行えばいい．[*4] ただし各語が見付かったときにそれらに対する識別子の割り当てなどを行う必要がある．アルゴリズムを上記のように英文検索用に拡張してあるのでこれらの処理はごく自然に行える．

4　作文支援

類似表現検索を作文支援に応用するため，文単位での類似表現検索機能が付いたエディタを作成し検索結果を入力に使用するためのインタフェースを付け，作文者がコーパスにある表現を参考にしたり入力に利用したりできるようにする．

4.1　ユーザインタフェース

編集用のエディタとして Emacs[*5]を利用する．検索するときはバッファ上，該当するクエリーが記

[*4] クエリーとの類似度を計算するのはコーパステキスト中の各文に対してである．

[*5] www.gnu.org

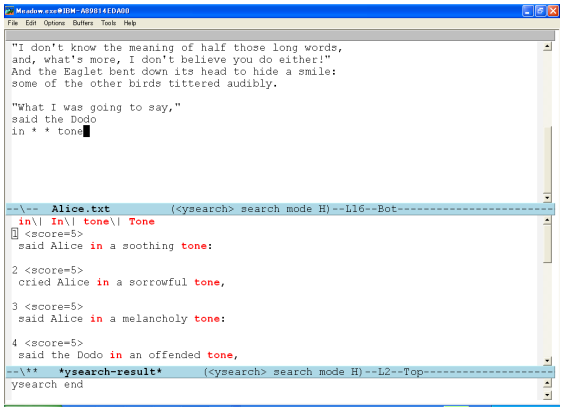

図 3　Emacs UI

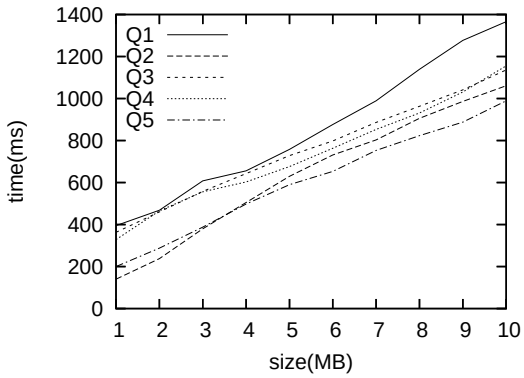

図 4　速度実験結果

述されている行において検索開始シグナルを送るためのキーをタイプする（図 3）．シグナルを受けると Emacs Lisp で記述されたイベント処理ルーチンは検索モジュールにその行に書いてあった内容をクエリーとして渡し，検索モジュールが独自のプロセスとして動き出す．その間他の編集作業は続けることができる．検索対象コーパス中に見付かった文は類似度の順にスコア付けされ，結果が上位から順に並べられて表示される．見付かった表現を利用するときは使用したい部分を切り張りできるので，入力効率を高めたりスペルミスを避けるのに役立つ．

5　評価

英文検索用に拡張した検索アルゴリズムと類似度計算が実用的速度で動くことを速度実験で検証し，表現の再利用が入力支援でどの程度利用できるのかについて入力実験により評価する．

5.1　速度評価

検索速度に関する実験として 1987 年の WSJ (Wall Street Journal) から無作為に記事を選び 1MB から 10MB まで 1MB 単位刻みでコーパスデータを作成し，検索，スコア付けまでの全処理に要する時間を計測した．[*6] クエリーはコーパスと同じ 1987 年の WSJ の記事中の 5 つの文を無作為に抽出して作成した．コーパスと同じ WSJ から選んだ理由は，実際の検索においても作文しようとしている内容と同種類の文章をコーパスとして使用することを念頭においているからである．

5.1.1　結果と考察

結果をグラフで表示すると図 4 のようになる．横軸はコーパスのサイズで縦軸はミリ秒単位で測った時間である．各曲線はそれぞれのクエリーに対応している．これからまず全てのクエリーが 10MB 程度のコーパスに対し 1 秒程度の時間で処理されることが分かる．検索の結果を待つ時間としては十分な速さである．次に，グラフの形状からコーパスサイズの増加に対して検索に必要な時間が線形に増加することが観察される．これはアルゴリズムの計算量がテキストの長さに関して線形のオーダーであることに対応している．そして各クエリーの処理時間の違いはコーパスサイズが小さいときにしか目立たないことが分かる．

5.2　入力評価

個人が過去の文章を参考にしながら入力を行うときに単語の直接入力がどの程度削減されるのかに関する評価実験を行う．実際 [2] では文章内での単語の再利用率が極めて高いことが報告されているので，表現の再利用率も高いことが期待される．

5.2.1　実験内容

個人の作文活動に注目するために，何人かの作家に関してそれぞれの作品を集めた．[*7] 実験は作家毎に行う．その概要は以下のようになる．

[*6] プログラムは IBM Think Pad 上で走らせた．プロセッサーは x86 Family で 1.70GHz，メモリは 512MB RAM であり，計算機のワード長は 32 ビットである．

[*7] Lewis Carroll と Arthur Conan Doyle，Mark Twain の三人．それぞれの代表作を作品群として集めて利用した．作品群のサイズは最大の Mark Twain で 2.93MB であり，先に速度実験で瞬時に結果を得られると実証したサイズの範囲に含まれる．

1. 作家の作品群の中からランダムに一塊の部分を抽出し，それを入力テストデータとする[*8]
2. 入力テストデータを本研究の検索機能付きエディタで入力する[*9]
3. 結果をそのまま入力したときと比較する[*10]

例えば今入力すべき表現 was too much puzzeled to の代わりにクエリー too * * to を入力して検索すると，他の場所で使われている同じ表現 Alice [was] **too** [much] [puzzled] **to**... が見付かるので大括弧で囲われた部分をカットアンドペーストにより補完できる．

5.2.2 結果

表 1 が結果である．各行の左端は入力テストの名前である．[*11] 検索回数とは全文中補完に利用できるような検索を行うことができた文の数，補完単語数は全単語中補完によって入力された単語の数，総補完単語長とは補完された単語の長さの合計である．

表 1　入力実験結果

	検索回数	補完単語数	総補完単語長
LC1	11/31	32/167	127/726
LC2	12/22	27/175	112/709
CD1	17/34	25/207	113/892
CD2	19/28	39/291	142/1145
MT1	10/32	14/216	64/842
MT2	13/32	35/191	110/762

5.2.3 結果の評価

作家ごとに多少の差はあるが，入力すべき文の中の 3 割から 5 割は補完入力の可能性があることが分かる．また単語数のレベルでみると全体の 1 割から 2 割程度に補完入力の可能性があることが分かる．よって高い頻度での補完入力という訳にはいかなくても，一定レベルでの補完可能性はあると考えられる．

[*8] 残った部分をコーパスデータとする．

[*9] 一文毎に擬似最適な検索を行う．結果はカットアンドペーストして入力補完に利用する．

[*10] 全体の文の中で補完入力が有効でものの割合や単語がどの程度補完されたかを調べる

[*11] Lewis Caroll の入力テスト 1 と 2，Conan Doyle の入力テスト 1 と 2，Mark Twain の入力テスト 1 と 2 を意味している．

6 まとめ

Shift-And アルゴリズムを英文検索用に拡張し，LCS を基本として独自に定義した類似度と合わせ類似表現検索を実現した．またそれらを作文支援に応用し，速度面と入力補完面での評価実験を行った．今後は入力補完を自動で行うインタフェースを構築することで表現の再利用をより簡単に行うことができるようにし，表現単位での補完という新しい入力方法を実現したいと考えている．

参考文献

[1] Ricardo Baeza-Yates and Gaston H. Gonnet. A new approach to text searching. *Communications of the ACM, Vol 35, Issue 10*, pp. 74–82, 1992.

[2] Tanaka-Ishii Kumiko, Hayakawa Daichi, and Takeichi Masato. Acquiring vocabulary for predictive text entry through dynamic reuse of a small user corpus. *Annual Meeting for Association of Computational Linguistics*, pp. 407–414, 2003.

[3] Gonzalo Navarro and Mathieu Raffinot. *FLEXIBLE PATTERN MATCHING IN STRINGS*. Cambridge University Press, Cambridge, 2002.

[4] Takashi Yamanoue, Toshiro Minami, Ian Ruxton, and Wataru Sakurai. Learning Usage of English KWICly with WebLEAP/DSR. *Proceedings of the 2nd International Conference on Information Technology and Applications*, 2004.

[5] 武田朋子, 古郡廷治. 例文をもとにした英文書作成支援システム．情報処理学会論文誌，Vol 35, No. 1, pp. 53–61, 1994.