英文とその校正からの書き換えルール自動抽出

松崎幸太郎
東京大学工学部計数工学科

田中久美子
東京大学大学院情報理工学系研究科

概要

英語で書かれた校正前後の文書対から，編集距離を利用して編集操作を抽出し，その中から有効なものを校正ルールとして抽出する研究について述べる．英文を単語の列として処理する方法と，構文木として処理する方法の二種類を行った．抽出する校正ルールは $A \to B$ という書換規則として記述される．抽出された校正ルールは一連の編集操作であり，A の出現回数に対する $A \to B$ の出現回数の割合によりルールとしての有効性を評価する．

実際に校正ルールの抽出を行い，Precision, Recall を算出した．結果，Recall が 10% の時に，単語列による手法では Precision が 50%，構文木による手法では Precision が 80% であった．

1 序論

現在は文書が電子化されるようになり，文章校正も PC 上で行うことが多い．校正履歴が記録できるソフトウエアもあり (図 1)，校正記録付の電子文書が増えてきた．しかし，このような校正記録の活用についての研究はまだまだ発展途上である．

本研究では，校正記録の蓄積から有用なデータを抽出し，校正支援に応用することを目的とする．従来の汎用の校正支援とは異なり，著者や文書の種類に特化した校正支援が期待できる．

校正記録から抽出する校正ルールは，校正記録に現れる，A という文字列が B という文字列に直されている，という書換規則 (rewrite rule) のうち頻度の高いものであるとし，この書換規則を $A \to B$ と略記する．

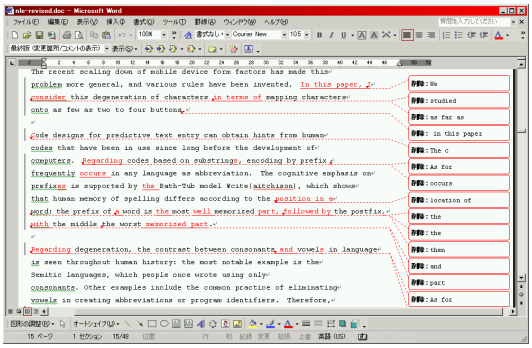

図 1　Microsoft Word による校正サンプル

2 文書処理の概観

2.1 入力と出力の概要

入力は，英語で書かれた校正前と校正後の文書を 1 対として，その集合である．出力は校正ルール $A \to B$ の集合である．本研究で元としたデータは図 1 のようなもので，単語・表現の修正など文単位の比較で処理が可能なものがほとんどである．そこで，本稿では文や段落の順序が変わる大規模な校正は考えない．

2.2 全体の処理の流れ

入力から校正ルールを得るまでの処理の流れは以下のようになる．

1. 校正前の文と校正後の文の対応を算出
2. 対応付けされた文の組から，編集操作を検出・記録
3. 連続した編集操作を統合し校正ルールを構成
4. 得られた各校正ルールを評価
5. 評価の高い校正ルールを採用

3 文を単語列と見たときのルール抽出

3.1 編集距離

校正前後の文書の対応付けには，文字単位，単語単位，文単位の 3 つのレベルがある．まずは編集距離の定義を以下に示す．

定義 1 二つの要素列に対し，一方の要素列を他方に変換するために必要な編集操作 (追加・削除・置換) 回数の最小値を編集距離という．

例えば次の 2 つの文

- My hobby is music and penpaling.
- My hobbies are music and pen paling.

に対し，編集距離を求める際に行われる編集操作を図 2 に示した．3 つの置換と 1 つの追加が行われているので，この 2 文の編集距離は 4 となる．

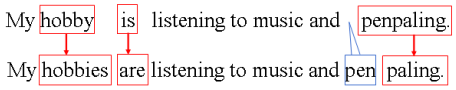

図 2　編集距離の例

さらに編集距離を次のように拡張する．要素の編集操作 $e_1 \to e_2$ のコストを $\chi(e_1 \to e_2)$ とする．コストは距離の条件を満たすものを考える．ここでは，拡張された編集距離を，要素列 A から B に移す際に行われる編集操作のコスト合計の最小値とする．また，空白の要素を λ と表記する．$e_1 \to \lambda$ は e_1

の削除を表し，$\lambda \to e_2$ は e_2 の追加を表す．コスト χ を用いて計算する編集距離を，以下ではコスト χ の編集距離と表記する．

要素数 n の要素列 E_1 と，要素数 m の要素列 E_2 間の，コスト χ の編集距離を計算する動的計画法によるアルゴリズムが Algorithm 1 である．このアルゴリズムの計算時間は $O(mn)$ である．一連の編集操作は，二次元配列 ed をトレースバックすることで得られる．

Algorithm 1 EditDistance($E_1[1,\ldots,n], E_2[1,\ldots,m]$)

```
for i = 0 to n do
    ed[i,0] ← χ(E1[i] → λ)
end for
for j = 0 to m do
    ed[0,j] ← χ(λ → E2[j])
end for
for i = 1 to n do
    for j = 1 to m do
        delete ← ed[i−1,j] + χ(E1[i] → λ)
        insert ← ed[i,j−1] + χ(λ → E2[j])
        subs ← ed[i−1,j−1] + χ(E1[i] → E2[j])
        ed[i,j] ← minimum(delete, insert, subs)
    end for
end for
return ed[n,m]
```

3.2 文の対応付け

文の対応付けは，s_1, s_2 を文として，コスト χ_1 を，

$$\chi_1(s_1 \to s_2) = \begin{cases} s_1\text{の単語数} & (s_2 = \lambda) \\ s_2\text{の単語数} & (s_1 = \lambda) \\ s_1\text{と } s_2\text{の，単語を要素} & (s_1 \neq \lambda, \\ \quad \text{とした通常の編集距離} & s_2 \neq \lambda) \end{cases}$$

と定義し，要素を文とした文書間の，コスト χ_1 の編集距離を求めて対応を取る．コストの計算に編集距離を使用しているので，二段構えの動的計画法となる．

3.3 単語の対応付け

文の比較は，w_1, w_2 を単語として，コスト χ_2 を

$$\chi_2(w_1 \to w_2) = \begin{cases} w_1\text{のアルファベット数} & (w_2 = \lambda) \\ w_2\text{のアルファベット数} & (w_1 = \lambda) \\ w_1\text{と } w_2\text{の，アルファベットを} & (w_1 \neq \lambda, \\ \quad \text{要素とした通常の編集距離} & w_2 \neq \lambda) \end{cases}$$

と定義し，要素を単語とした文の間の，コスト χ_2 の編集距離を求めて対応を取る．3.2 と同一の処理を文レベルに適用することに相当する．

3.4 連続した編集操作の統合による校正ルールの構成

1 文中で連続した編集操作を統合し，校正ルールを構成する．例えば，次の二つの文

- 校正前: I will goto the shop.
- 校正後: I will go to the shop.

を比較する場合，go の挿入と goto → to の置換という二つの編集操作が統合され，goto → go to という校正ルールを得る．

4 構文木からのルール抽出

4.1 構文木の導入

次の校正前後の文の対を考える．

- 校正前: This is afftected so badly by noise.
- 校正後: This is strongly affected by noise.

この 2 文をそれぞれ構文木にして比較すると図 3 のようになる．“so badly”の部分が “strongly”に置き換わっていることがわかる．このように，単語の順序が前後するような入れ替えを句ごとに取り出す上で，構文木からのルール抽出は効果を発揮すると考えられる．

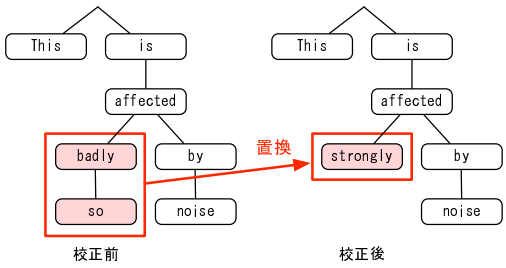

図 3　構文木同士の比較

本研究では，構文木は各ノードに単語列がラベル付けされたラベル付根付順序木 (labeled ordered rooted tree) とする．英文から構文木を作成するツールとして Apple Pie Parser[2] を用いた．

4.2 順次木の編集距離

準備として，2 つの木の間のマッピングの定義を行う．

定義 2 T_1 と T_2 を順序木とする．T_1 と T_2 の間のマッピングとは，ノードの組 $(v, w) \subseteq V(T_1) \times V(T_2)$ の集合 M のことである．任意の $(v_1, w_1), (v_2, w_2) \in M$ は次の条件を満たすものとする．

- $v_1 = v_2 \Leftrightarrow w_1 = w_2$
- v_1が v_2の左側 $\Leftrightarrow$ w_1が w_2の左側
- v_1が v_2の先祖 $\Leftrightarrow$ w_1が w_2の先祖

T_1 と T_2 の間のマッピングが M であるとき，(M, T_1, T_2) をマッピングと呼ぶことにする．

マッピング (M, T_1, T_2) に対し，N_1 をマッピングされていない T_1 の頂点の集合，N_2 をマッピングされていない T_2 の頂点の集合とする．このとき M のコストを

$$\gamma(M) = \sum_{(v,w)\in M} \gamma(v \to w) + \sum_{v\in N_1} \gamma(v \to \lambda) + \sum_{w\in N_2} \gamma(\lambda \to w)$$

と定義する．ここで，$\gamma(v \to w)$ はノードの編集操作 $v \to w$ のコストで，ノード v のラベル label(v) をノード w のラベル label(w) に変換するための通常の編集距離とする．このコストは距離の条件を満たす．木の編集距離はこのマッピングのコストの最小値を求めることに等しい [1].

木の編集距離を求めるアルゴリズムは計算量が大きく，本研究のような大量の文書処理には不向きである．そこで，マッピングとしては，トップダウンの制約を加えて計算量を削減したものを考える．トップダウンの制約とは，任意の $(v, w) \in M$ に対し，それらの親ノードの組 $(\mathrm{parent}(v), \mathrm{parent}(w)) \in M$ となることである．この制約を満たすマッピングをトップダウンマッピングという．トップダウンマッピングの例を図 4 に示す．

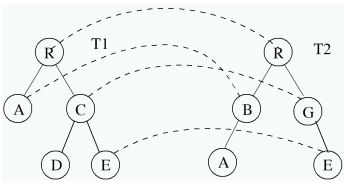

図 4　トップダウンマッピングの例

本研究では，Yang[3] のトップダウン制約付の編集距離を参考に，次のアルゴリズムを用いた．そのアルゴリズムでは，木をトップダウンにたどり，その中で前章で述べた編集距離の考え方を合わせて用いる．$v_1 \in T_1$，$v_2 \in T_2$ として，v_1, v_2 間のコスト χ_3 を，次の Algorithm 2 に定義する．そして，マッピングを求めるには，T_1, T_2 のルートに対し，コスト χ_3 の編集距離を計算する．注目点は 11 行目で，χ_3 を再帰的に呼び出している．これがトップダウンにたどる部分である．

Algorithm 2 $\chi_3(v_1, v_2)$

1: **if** $v_1 = \lambda$ **then**
2: $cost \leftarrow \Sigma_{w \in V(\mathrm{subtree}(v_2))} \gamma(\lambda \rightarrow w)$
3: **else if** $v_2 = \lambda$ **then**
4: $cost \leftarrow \Sigma_{v \in V(\mathrm{subtree}(v_1))} \gamma(v \rightarrow \lambda)$
5: **else**
6: **if** v_1が葉 **then**
7: $cost \leftarrow \Sigma_{w \in V(\mathrm{subtree}(v_2) \setminus v_2)} \gamma(\lambda \rightarrow w)$
8: **else if** v_2が葉 **then**
9: $cost \leftarrow \Sigma_{v \in V(\mathrm{subtree}(v_1) \setminus v_1)} \gamma(v \rightarrow \lambda)$
10: **else**
11: $cost \leftarrow$ subtree(v_1) と subtree(v_2) の, χ_3をコストとした編集距離
12: **end if**
13: $cost \leftarrow cost + \gamma(v_1 \rightarrow v_2)$
14: **end if**
15: **return** *cost*

4.3　対応付けと編集操作の統合

構文木を使用する場合も，3.2 と同様の処理を，文間のコストを構文木間の編集距離として行い文の対応を求める．また，3.4 と同様に，同じ親ノード上で連続した編集操作の統合を行う．

構文木特有の編集操作の統合として，次の処理を行う．4.1 で示した文対を構文木に変換して (図 3)，4.2 で述べた処理により，

1. badly → strongly
2. so → λ

と編集操作が抽出されるが，これを ((so) badly) → (strongly) という一つの編集操作に統合したい．そこで，ノードの編集操作 $(v \rightarrow w)$ に対し, w が葉の場合は, $(v \rightarrow w)$ を $(\mathrm{subtree}(v) \rightarrow w)$ に変更する．subtree(v) はノード v をルートとした部分木である．同様に, v が葉の場合は, $(v \rightarrow w)$ を $(v \rightarrow \mathrm{subtree}(w))$ に変更する．前の例では，w にあたる “strongly” の部分が葉なので，このような統合が行われる．

5　有効なルールの選択方法

A が B に置換された割合を $f = \frac{\#(A \rightarrow B)}{\#(A)}$ と記述し，評価関数とする．尚，$\#(A)$ は校正前の文書中に A が出現した数，$\#(A \rightarrow B)$ は $A \rightarrow B$ の置換が行われた数を表す．尤度比検定による評価関数も試みたが，その後の実験でルールの Precision を反映できておらず，採用しなかった．

6　実験

実験は 10 分割交差検定を用いて行った．使用した校正文書対は，同一の日本人著者による英語論文 18 編で，文数が約 4000，単語数が約 87000 である．出力した校正ルールの数を N とおく．N を変化させて Precision と Recall を調べた．Precision，Recall の定義を以下に示す．まず，E, T, S を以下のように定義する．

E = ファイル m に出現する全ての編集操作の数
T = 校正ルールのうちファイル m に適応可能なルールの数
S = 校正ルールのうちファイル m に適合したルールの数

そして，Precision，Recall を

$$\mathrm{Precision} = \frac{S}{T} \qquad \mathrm{Recall} = \frac{S}{E}$$

と定義する．ルール $A \rightarrow B$ がファイル m に適用可能とは，A がファイル m の校正前文書に含まれているということであり，適合するとは，ファイル m の編集操作に $A \rightarrow B$ が含まれているということである．

6.1　文書から抽出されたルール例

抽出した校正ルールの一部を表 1 に示した．f は前節で定義した，置換割合による評価関数である．得られた校正ルールを，誤りの種類により分類する．

- 語彙:2,5,8,9,11,14,17,19,20
- 文法:11
- 語順:1,3,5
- 表現:4,6,7,10,15,18,19
- 冠詞:12,13,16
- 単数・複数:16

これらは日本人による典型的な誤りで，無くすことは困難である．しかし，このように頻出する誤りを具体的に明示することで，筆者が自分の間違いの傾向を把握し，今後の執筆において注意することができる．

また，f が 1.0 より少ないルールは，それが適正であるかは文脈によるが，筆者が修正前の語 A と修正後の語 B を明確に区別していないことを表す，という点で意味がある．

また，連語によるルールに着目する．1,3,4,5,10 が単語列により抽出されたルールである．その中には，1,3 の様に語順の変更が見られる．他方，構文木では句単位の連語が抽出される．例えば，12,13,16 の様に冠詞の有無を抽出できる．このように，単語列による処理と構文木による処理は相互補完的である．

表 1　抽出したルール例

単語列		
No.	ルール	f
1	only by using → by using only	1.0
2	exhaustive → extensive	1.0
3	currently freely downloadable → freely available for download	1.0
4	further ahead → below	1.0
5	be decided straightforwardly → be straightforwardly determined	1.0
6	Firstly → First	0.62
7	is different → differs	0.50
8	raise → increase	0.50
9	industrial→ commercial	0.43
10	that are → such as	0.10
構文木		
No.	ルール	f
11	(novice view ideograms) → (beginners view ideograms)	1.0
12	(this predictive text entry) → (predictive text entry)	1.0
13	(WWW) → (the WWW)	1.0
14	(small mobiles machines) → (small mobile devices)	1.0
15	(a small evaluation) → (a small scale evaluation)	1.0
16	(the counts) → (the count)	0.75
17	(The oldest expression) → (The earliest expression)	0.67
18	(Additionally) → (In (addition))	0.67
19	((this viewpoint)(too)) → ((this respect)(also))	0.67
20	(a computer) → (a device)	0.44

6.2　Precision と Recall

単語列の場合のルール数 N と Precision の関係を図 5 に示した．Precision の減少は N に対し直線的である．従って，評価関数 f は正しくルールの Precison を反映しているといえる．

単語列の場合の，Precision と Recall の関係を図 6 に示した．Precision と Recall が直線的に変化していることが特徴的である．数値的には，例えば Precision = 0.5 のとき，Recall = 0.1 であった．従って，人手を介した校正支援を行うには有効となりえるデータが得られていると考えられる．

一方、構文木の場合の Precision と Recall の関係を図 7 に示した．Precision が常に 0.8 以上であることが特徴的である．これは単語列の結果に比べて高い数値である．しかしグラフにはノイズが見られ，得られたルールの信頼性には更なる検討を要する．

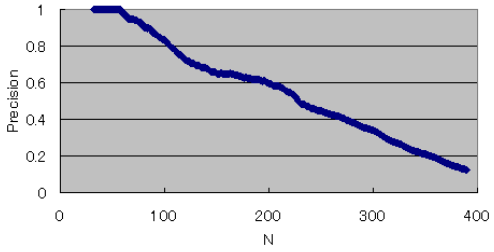

図 5　単語列における，ルール数 N と Precision の関係

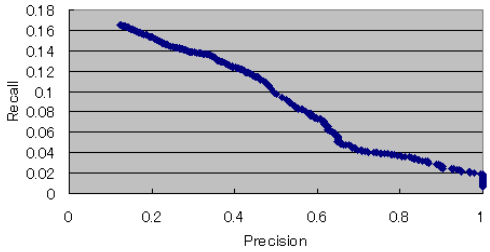

図 6　単語列における Precision と Recall の関係

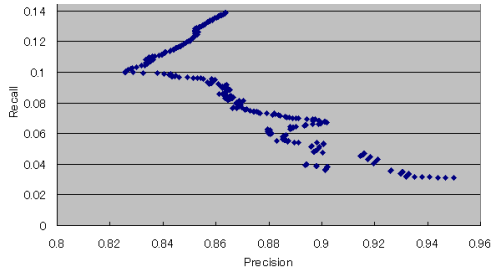

図 7　構文木における Precision と Recall の関係

7　まとめと今後の課題

英語の校正前後の文書対から編集操作を抽出し，有効な操作を校正ルールとして抽出する方法を述べた．英文を単語の列として処理する方法と，構文木にする方法の二種類を考え，編集距離を用いた動的計画法によるアルゴリズムを示した．有効なルールを判定する評価関数として，校正ルールの適用可能な割合を採用した．実際に校正ルールを抽出した結果，Recall が 10% の時に，単語列による手法では Precision が 50%，構文木による手法では Precision が 80% であった．

今後の課題としては，校正ルールの統合処理による精度の向上，単語の品詞情報を使用した解析，ルールの評価を他の文書で行う，などが挙げられる．

参考文献

[1] K.-C.Tai. The tree-to-tree editing problem. *J.ACM*, Vol. 26, No. 3, pp. 61–74, 1979.

[2] Satoshi Sekine. Apple pie parser, visited 2006. Available at `http://nlp.cs.nyu.edu/app/`.

[3] W. Yang. Identifying syntactic differences between two programs. *Software Practice and Experience*, pp. 739–755, 1991.