# 構文的類似度を用いた文の検索

箱田 慶太[†] 市川 宙[‡] 橋本 泰一[‡] 徳永 健伸[‡]

[†] 東京工業大学 工学部 情報工学科

[‡] 東京工業大学 大学院情報理工学研究科 計算工学専攻

{hakoda,ichikawa,taiichi,take}@cl.cs.titech.ac.jp

## 1 はじめに

類似文検索の技術は，情報検索や機械翻訳に応用するための基盤となる技術であり，様々な手法が考案されてきた．その１つに形態素など文中の語の出現頻度に基づくものがある．類似度の計算手法は様々だが，多くの語を共通に含む文を類似文とする手法が基本である．この手法は話題や内容が類似する文を検索するのに適した方法であり，情報検索などで用いられている．また，文中の品詞や助詞，助動詞の並びに注目した手法もある．このような手法は用例ベース機械翻訳などに使われている．[1, 2]

しかし，これらの品詞や助詞，助動詞などの並びをもとに検索する手法を使う場合，構文構造や係り受け関係が全く異なることがある．その場合，実際は意味が似ていない文にも関わらず，類似文と判断してしまう．これは，従来の類似文検索が語や品詞などの表層の情報のみを扱い，構文構造などのより高次の情報を扱わないためである．

そこで，構文構造に注目し類似文を検索する手法が考案されている．その中で，構文木の類似度を計る手法として，構文木間の共通する部分木の数を類似度とする Tree Kernel(TK)，構文木間の共通する導出規則の数を類似度とする Tree Overlapping(TO)，構文木間の根から葉の間で共通する部分経路の数を類似度とする Subpath Set(SS) が提案されている．TO や SS は，予めインデックス化をすることによって，類似文検索の高速化をすることができる．しかし，各アルゴリズムの評価実験 [5] は小規模なものであり，その効果を十分に評価できていない．

本研究では，大規模なコーパスを使った評価実験を行い，3 つのアルゴリズムについて比較検証した．実験で使用するコーパスは，岩波辞書の語釈文 57,982 文に構文木を自動的に付与したものを使用する．この語釈文は文の長さが比較的短く決まった言い回しも多いため，人手での類似判定に適していると考え採用した．実験の結果，検索時間の比率で比べると，TO が TK に対して 100 倍前後速く，SS が TK に対して 1,000 倍前後速いなど，TO，SS が大規模コーパスにおいても効果を示すことがわかった．

## 2 構文的類似度に関する計算アルゴリズム

### 2.1 Tree Kernel

#### 2.1.1 類似度の定義

Tree Kernel[3, 4] では，2 つの木構造の類似度をそれらの木構造が共通に含む部分木の数と定義している．ただし「部分木」は 2 個以上のノードを持ち，個々の導出規則の一部だけを含んではいけないものとする．類似度に求められる性質は応用によって異なるため，Collins の Tree Kernel がどの応用にも適しているとは限らない．そこで高橋ら [6] は，この Tree Kernel をベースとした 3 種類の類似度を提案した．本研究では，高橋らが提案した類似度 C（$K_C$）を用いる．

$$K_C(T_1, T_2) = \max_{n_1 \in N_1} \max_{n_2 \in N_2} C(n_1, n_2) \qquad (1)$$

ここで $C(n_1, n_2)$ は，ノード $n_1$ を根とする部分木としても，$n_2$ を根とする部分木としても出現する部分木の数である．

#### 2.1.2 アルゴリズム

Collins[3, 4] は，以下の規則で $C(n_1, n_2)$ を求めることで，効率的に Tree Kernel を計算する手法を提案した．

$n_1$ と $n_2$ の子ノードを導出する規則が異なる場合，

$$C(n_1, n_2) = 0 \qquad (2)$$

$n_1$ と $n_2$ の導出規則が等しく，$n_1$ と $n_2$ がともに前終端記号の場合，

$$C(n_1, n_2) = 1 \qquad (3)$$

$n_1$ と $n_2$ の導出規則が等しく，$n_1$ と $n_2$ がともに前終端記号でない場合，

$$C(n_1, n_2) = \prod_{i=1}^{nc(n_1)} (1 + C(ch(n_1, i), ch(n_2, i))) \quad (4)$$

ここで $nc(n)$ は $n$ の子ノードの数を示し，$ch(n, i)$ はノード $n$ の $i$ 番目の子ノードを示す．式 4 で子ノードの $C$ を用いるが，各ノード間の $C$ を後順序で求めれば，再計算は不要である．

市川ら [5] は検索手法として，クエリとなる木構造と検索対象の全ての木構造との間で個々に類似度 $K_C(T_1, T_2)$ を計算し，それをソートするというアルゴリズムを採用している．本研究でもこのアルゴリズムを使って実験を行う．

## 2.2 Tree Overlapping

### 2.2.1 類似度の定義

Tree Overlapping[5] では，２つの木の間でその中のあるノード同士を重ねたとき，重なり合う導出規則の数を類似度と定義する．ノード同士の「重ね方」によって類似度が変化するが，ここではその最大値を２つの木の類似度とする．

### 2.2.2 アルゴリズム

Tree Overlapping は Tree Kernel と異なり，予めインデックスを作ることで，類似度計算を高速化できる．

インデックスには表 1 のような形で，１つの導出規則 $p$ に対して「木の ID」$(T_1, T_2)$ と「木の中の導出規則 $p$ によって構造を構成するノードの ID」$(a_1^1, b_1^1, b_1^2, c_1^2)$ の組の集合 $I[p]$ を保存する．

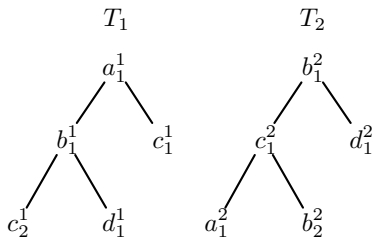


図 1: 木 $T_1$ と $T_2$

このインデックスを使ってクエリに含まれるすべての導出規則について「同じ規則がコーパス中のどこに出現するか」を探し出し，スコアを与えることができる．ここで，スコアを与える対象である「重ね方」をどう表現するかが問題となる．同じ重ね方を考慮するには，木の中の各ノード間の関係を関連付けなければならない．しかし，インデックスには同じ木の中の他の規則との関係についての情報はない．

表 1: インデックスの例

| 規則 $p$ | $I[p]$ |
|---|---|
| $a \rightarrow bc$ | $\{(T_1, a_1^1)\}$ |
| $b \rightarrow cd$ | $\{(T_1, b_1^1), (T_2, b_1^2)\}$ |
| $c \rightarrow ab$ | $\{(T_2, c_1^2)\}$ |

そこで「ある木のノード $n$ と別の木のあるノード $m$ が同じ位置に重なるように 2 つの木を重ねたときに，重なったノードの組（$L(n, m)$）の中で最も根に近いもの」を与える関数を導入する．

この関数を使って２つの木を重ねたときに一致する複数のノードの組をあらかじめ関連付けることができる．この関連付けによって２つの木のある重ね方に対しては，代表して最も根に近いノードの組にスコアを与えることで１つの「重ね方」に対する値が計算できる．その最大値を類似度とする．

## 2.3 Subpath Set

### 2.3.1 類似度の定義

テキスト間の類似度としては，テキストの単語頻度ベクトルに基づいた類似度がよく用いられる．これを木構造に応用したのが Subpath Set[5] の類似度である．Subpath Set では，テキストを構成する「語」に相当するものを，木構造の「根から葉までの経路とその一部」とする．これらを木 $T$ の**部分経路**と呼ぶ．ここでは「両方の木に共通する部分経路の数（重複はカウントしない）」という単純なものを用いる．

Tree Overlapping と同様，検索対象のインデックス化による検索の高速化が可能である．ただし，Tree Kernel や Tree Overlapping に比べると，何番目の子ノードかを区別しないなど，粗い情報に基づく検索である．そのため，アルゴリズムの単純さ故に Tree Overlapping よりさらに高速であるが，検索結果にはノイズが入りやすい．

### 2.3.2 アルゴリズム

クエリの木を $T_0$，検索対象の木の集合を $F$ とする．また，木 $T$ の部分経路の集合を $P(T)$ と表す．

予め，あらゆる部分経路 $p$ について，以下のようなインデックス $I[p]$ を作成しておく．

$$I[p] = \{T | T \in F \wedge p \in P(T)\} \quad (5)$$

以下のアルゴリズムで $S[T]$ を計算する．

$S[T] := 0$ **for all** $T$
**foreach** $p$ **in** $P(T_0)$ **do**
  **foreach** $T$ **in** $I[p]$ **do**
    $S[T] := S[T] + 1$

このようにして計算した $S[T]$ が「$T_0$ と $T$ に共通する部分経路の数」になる．これをソートして比較すればよい．

# 3 大規模コーパスを使った評価実験

Tree Kernel(TK)，Tree Overlapping(TO)，Subpath Set(SS) に関して，市川らは東工大コーパスの 2,483 文を検索対象として実験を行っている．しかし，この実験は小規模なものであり，アルゴリズムの十分な効果を示していない．そこで，大規模なコーパスを使って 3 つのアルゴリズムの比較によって有効性を検証する．

## 3.1 使用するコーパス

本実験では，岩波辞書 [7] の語釈文 57,982 文を使用した．各語釈文に対して，ChaSen で形態素解析を行い，その解析結果をもとに MSLR パーザ [8] で複数の構文木を得る．さらに PGLR モデル [9] を用いて生成確率が最大値となる構文木を選択し，この実験で使用する構文木とする．PGLR モデルの学習は，すでに人手で正解の構文木が付与している東工大コーパス [10] の 20,190 文を使用した．しかし人手で正解の構文木を付与してはいないため，構文木が必ずしも正解の構文木であるとは限らない．

市川らの使用した東工大コーパスは，新聞記事の文に構文木を付与したコーパスである．形態素解析の結果 1 文における平均形態素数は 26.1 であった．それに対して岩波辞書の語釈文の 1 文における平均形態素数は 7.4 と短くなっている．また決まった言い回しも多い．そのため人手で類似を判定するのに適していると考え，本実験のコーパスとして採用した．

## 3.2 実験と結果

TK，TO，SS の 3 つのアルゴリズムを Ruby1.8.2 で実装して実験を行う．使用するマシンは，Xserve G5 CPU 2.3GHz メモリ 2GB．クエリは 3.1 節で述べたコーパスから 1,000 文をランダムに選び，検索を行う．評価は検索時間と検索精度を各アルゴルズム間で比較する．

検索時間に関する結果は，1 クエリ当たりの平均検索時間を表 2 に，1 クエリにおける検索にかかる時間のアルゴリズムごとの平均倍率を表 3 に示す．検索精度に関する結果として，各アルゴリズムで 1 位の文が他のアルゴリズムで何位であるかを表 4 に，各アルゴリズムで上位にヒットしたものの中にどれだけ共通の文が含まれるかの平均を表 5 に示す．

## 3.3 考察

TO と SS は，TK との比較では検索時間が短いことが分かる．しかし TO に関しては，実用的な面から言うとまだ検索時間が長いと言える．

平均検索時間について市川らの実験と比較すると，TO は TK の約 84 倍短かった．本実験では約 99 倍短い．表 3 より，1 クエリごとに見ても TO は TK より 128 倍短い．同じく平均検索時間について SS は TK の約 1,126 倍短かった．本実験では約 751 倍短い．表 3 より，1 クエリごとの SS は TK より約 778 倍短い．このことから大規模なコーパスにおいても TO と SS の高速化の効果は失われていないと言える．

また，精度については表 4 より，約 4 割は 1 位が一致していて，10 位以内までには，ほとんど他のアルゴリズムの 1 位が含まれている．市川らの実験と比べても表 5 より，各アルゴリズムの上位で共通している文の割合がかなり上がっている．その原因としては，決まった言い回しが多いという語釈文の特徴が，構造的な類似性と関係しているためと考えられる．実際に人手で分析した結果，似たような表現を含む文が多く見られる．しかし，表現の類似と意味の類似が必ずしも一致するとは限らず，その点が今後の課題である．

# 4 まとめ

構文的類似度を用いた文の検索において提案された Tree Kernel，Tree Overlapping，Subpath Set について，その効果を大規模コーパスを用いて検証した．その結果，Tree Overlapping と Subpath Set は，Tree Kernel よりもそれぞれ約 100 倍，約 1000 倍ほど高速に文の検索を行うことができる．また精度の評価においては，市川らの実験では各アルゴリズム間で 1 位が共通している割合は 2 割前後だったのに対して，本実験では 7 割前後共通している．表 5 のその他の結果からも，大規模コーパスにおいて Tree Overlapping と Subpath Set が，Tree Kernel と比べてもそれほど精度を損なわずに高速化に成功していると言える．

しかしながら，いずれのアルゴリズムでも人間が見

表 2: 1 クエリ当たりの平均検索時間

| アルゴリズム | 平均検索時間 [s] |
|---|---|
| TK | 3796.1 |
| TO | 38.3 |
| SS | 5.1 |

表 3: 各アルゴリズム間の検索時間の平均倍率

| | 平均倍率 |
|---|---|
| TK/TO | 128.8 |
| TK/SS | 778.4 |
| TO/SS | 7.2 |

表 4: アルゴリズム A での 1 位の文が他のアルゴリズム B での $n$ 位までに入る割合 [%]

| A - B | 1 位 | 5 位まで | 10 位まで |
|---|---|---|---|
| TK - TO | 42.2 | 86.1 | 91.1 |
| TK - SS | 43.0 | 77.1 | 83.3 |
| TO - TK | 43.2 | 79.6 | 84.3 |
| TO - SS | 43.3 | 84.9 | 89.4 |
| SS - TK | 42.1 | 73.8 | 79.2 |
| SS - TO | 41.3 | 86.5 | 91.3 |

表 5: アルゴリズム A の $n$ 位までと他のアルゴリズム B の $n$ 位までに共通する文数の平均

| A - B | 1 位 | 5 位まで | 10 位まで | 50 位まで |
|---|---|---|---|---|
| TK - TO | 0.76 | 3.90 | 7.78 | 38.61 |
| TK - SS | 0.67 | 3.38 | 6.61 | 32.44 |
| TO - SS | 0.78 | 3.95 | 7.87 | 38.51 |

ると似ていないと感じる文が，上位に来ていることが少なくない．今後は，これらのアルゴリズムをベースにしたさらに新しい構造的アプローチや，あるいは表層の語も利用した検索が必要である．

## 参考文献


[1] Somers H, I McLean, D Jones. Experiments in multilingual example-based generation. CSNLP 1994: 3rd conference on the Cognitive Science of Natural Language Processing, Dublin, 1994.

[2] Nagao, Makoto. A framework of a mechanical translation between Japanese and English by analogy principle. In Alick Elithorn and Ranan Banerji, editors, Artifical and Human Intelligence, pages 173-180. Amsterdam, 1984.

[3] Collins, M. and Duffy, N. Parsing with a Single Neuron: Convolution Kernels for Natural Language Problems. Technical report UCSC-CRL-01-01, University of California at Santa Cruz, 2001.

[4] Collins, M. and Duffy, N. Convolution Kernels for Natural Language. In Proceedings of NIPS 2001.

[5] 市川宙, 橋本泰一, 徳永健伸, 田中穂積. テキスト構文構造類似度を用いた類似文検索手法. 情報処理学会情報基礎研究会, 79. 2005. May

[6] 高橋哲朗, 乾健太郎, 松本裕治. テキストの構文的類似度の評価方法について. 情報処理学会自然言語処理研究会, NL-150-7, 2002.

[7] 西尾実, 岩淵悦太郎, 水谷静夫（編）. 岩波国語辞典, 岩波書店, 第５版, 1994.

[8] 白井清昭, 植木正裕, 橋本泰一, 徳永健伸, 田中 穂積. 自然言語解析のための MSLR パーザ・ツールキット. 自然言語処理, Vol. 7, No. 5, pp. 93-112, 2000.

[9] 乾健太郎, 白井清昭, 徳永健伸, 田中穂積. 種々の制約を統合した統計的日本語文解析. 情報処理学会自然言語処理研究会, Vol. 96, No. 114, 1996.

[10] 野呂智哉, 橋本泰一, 徳永健伸, 田中穂積. 大規模日本語文法の開発. 自然言語処理, Vol.12, No.1, pp.3-32, 2005.