自然言語処理　150－24
(2002. 7. 16)

テキストの構文的類似度の評価方法について

高橋　哲朗†　乾　健太郎†　松本　裕治†

† 奈良先端科学技術大学院大学 情報科学研究科
〒 630-0101 奈良県生駒市高山町 8916-5
E-mail: †{tetsu-ta,inui,matsu}@is.aist-nara.ac.jp

あらまし　与えられた2つの構文木の類似度を効率的に評価する方法について論じる．Collinsが提案したTree Kernelの拡張し，構文木間の一般的な類似度評価方法を与える．この評価方法は，大きく3種類の類似性尺度を提供し，計算量のオーダを $O(|T_1||T_2|)$ に抑えながら，対応ノード間の類似度の定量化やノードの飛び越えを許す照合を実現することができる．また，提案した構文的類似度評価方法が質問応答タスクに応用できる可能性についても議論する．我々の評価方法は，従来の手法に比べて計算コストのオーバヘッドを定数倍に抑えたまま，より大域的な構造の照合をより柔軟に行うことができる．

キーワード　類似度評価，木構造，依存構造，木の照合，Tree Kernel

Methods for Estimating Syntactic Similarity

TAKAHASHI Tetsuro†, INUI Kentaro†, and Yuji MATSUMOTO†

† Graduate School of Information Science, Nara Institute Science and Technology
Takayama, Ikoma, Nara, 630-0101, JAPAN
E-mail: †{tetsu-ta,inui,matsu}@is.aist-nara.ac.jp

Abstract　This paper discusses how to estimate the similarity of a given pair of syntactic trees. The proposed method, an extension of Collins' Tree Kernel, provides three options for similarity measurement and relaxes the constraints of node/tree matching with the time complexity $O(|T_1||T_2|)$. The paper also argues that the proposed method fits the question-answering task.

Key words　similarity estimation, tree structure, dependency structure, tree matching, Tree Kernel

1. はじめに

テキスト間の照合は，用例ベースの機械翻訳やテキスト間のアライメント，情報検索，テキストの自動校正，質問応答などさまざまな応用分野を持っており，自然言語処理における基本的な処理の一つと言える．照合の方法は対象とするデータ構造によっていくつかのレベルに分けられる．その中でもっとも基本的なレベルは表層文字列間での照合であり，この問題についてはオートマトンによる照合やDPマッチングなど，効率的に類似度を求める手法が存在し，多くのアプリケーションで利用されている．しかしアプリケーションによっては表層文字列レベルの照合だけでは不十分なものもあり，より複雑な構造に対する照合を必要とする場合がある．

そこで本稿では照合の対象として構文木を選んだ．構文木を用いることによりテキストの構文情報を考慮した類似度を求めることができる．構文的な照合を行うためには，テキスト間において構文的な情報を考慮した類似度(あるいは距離)を定量的に計算する方法が必要であるが，現在までに多くの先行研究で構文的類似度が扱われてきている．しかしそれら中で構文木の類似度は個別の事例毎にさまざまな手法で計算されており，構文木の類似度全体に対する体系的な整理はされてきていない．また類似度の種類にはアプリケーションに依存したさまざまな尺度が考えられるが，その違いをパラメタライズできる計算手法が望ましい．そこで本稿では類似度の分類を行ない，それぞれについてその類似度を求めるアルゴリズムを整理する．データ構造としてはより一般的な計算方法を提示するために句構造木と依存構造木の両方を対象とするが，本稿ではこれらをあわせて構文木と呼ぶ．

2. 先行研究

テキスト間の類似度計算は，すでにいくつかの分野で使われている．ここではそれらについて我々の対象とする類似度計算と比較し整理する．

2.1 編集距離

文字列に対する編集距離と同様に，木構造に対する編集距離の計算方法が提案されている [13], [16]．編集距離では，ある構造を別の構造に変換するのに必要な最短の編集操作を探索し，それを基に類似度を求める．またどのようなオペレーションを定義するかにより，得られる類似度の性質を変えられるという利点もある．しかし，多くの情報が得られる一方で計算必要なコストが非常に大きくなるという問題がある．Zhang ら [16] の手法ではノード数 N 木の深さ d_N の木と ノード数 M 木の深さ d_M の木の類似度を計算した場合，計算量は $O(NMd_Nd_M)$ となる．我々の目標はテキスト間の類似度を求めることなので，編集距離の手法は計算量の問題から適していないと言える．

2.2 アライメント

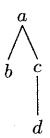
アライメントの問題においてもテキスト間の照合が用いられる．Meyers ら [8] や Watanabe ら [14] はコーパスベースの翻訳の知識獲得のために，依存構造解析されたバイリンガルコーパスのアライメントを行っている．Watanabe らは始めに単語の対応を取り，その後単語の対応を用いて句の対応を取っている．Meyers らは依存構造木間において，各ノードの子ノード間のスコアを再帰的に計算しながら各ノード間のスコア (類似度) を計算し，最適なスコアを求める．

ここで行われているアライメントは，類似する 2 つのテキストから，再利用可能な細かい対応を探し出すタスクと言える．したがって，木全体の構造を考慮した類似度を求めることは目的とはならない．質問応答システムでは質問文の木構造全体が，候補テキストの木構造全体に対してどれくらい類似しているかが焦点となる点に違いがある．

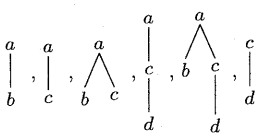
2.3 Tree Kernel

Collins ら [2] は句構造木間の類似度を与える Tree Kernel を提案している．句構造木間の内積を，それらの句構造木が共通に含む部分木の数と定義した．(1) の構文木には，(2) のような部分木が含まれている．二つの構文木の間で，このような部分木を共通に持つ数が内積となる．ここで求められる内積は，木全体を考慮した類似度とみなすことができる．この手法を用いることにより，構文構造を考慮した質問応答が行えると考えられる．

(1)



(2) a, b, c, d,



本稿では，Collins らの提案する手法を基に構文木の類似度の種類とその計算方法について議論する．

Collins らは対象となる木 T を，式 (3) のようなベクトルにより表現した．そしてその各要素 ($h_s(T)$) を，T の中で部分木 s の出現する回数と定義した．例えば (1) は (2) に示す 10 個の部分木を持っている．この部分木をベクトルの要素とし，各部分木の数をベクトルの値となる．

$$\phi(T) = h_1(T), h_2(T), \ldots, h_s(T) \tag{3}$$

$$\begin{aligned} K(T_1, T_2) &= \langle \phi(T_1), \phi(T_2) \rangle \\ &= \sum_s h_s(T_1) h_s(T_2) \end{aligned} \tag{4}$$

式 (4) の $h_s(T_1), h_s(T_2)$ はそれぞれ 式 (5) のように表すことができる．ここで $I_s(n)$ は n をルートとする部分木の集合に部分木 s が含まれるときに 1，そうでないときに 0 を返す関数である．

$$\begin{aligned} h_s(T_1) &= \sum_{n_1 \in N_1} I_s(n_1) \\ h_s(T_2) &= \sum_{n_2 \in N_2} I_s(n_2) \end{aligned} \tag{5}$$

式 (4) と式 (5) より式 (6) を導くことができる．式 (6) は 式 (7) により与えられる関数で，n_1 と n_2 をそれぞれルートとする部分木中に含まれる共通部分木の数を示す．

$$\begin{aligned} K(T_1, T_2) &= \langle \phi(T_1), \phi(T_2) \rangle \\ &= \sum_s h_s(T_1) h_s(T_2) \\ &= \sum_{n_1 \in N_1} \sum_{n_2 \in N_2} \sum_s I_s(n_1) I_s(n_2) \\ &= \sum_{n_1 \in N_1} \sum_{n_2 \in N_2} C(n_1, n_2) \end{aligned} \tag{6}$$

$$C(n_1, n_2) = \sum_s I_s(n_1) I_s(n_2) \tag{7}$$

内積を求めるためには 式 (6) を計算すればよいが，構文木の持つ次元は構文木が大きくなるにつれて指数的に増加する．したがって部分木の数をベクトルの要素と捉え，式 (7) により内積を計算することは現実的でない．そこで Collins はこの計算を効率的に行う手法を提案している．この手法では以下の規則により $C(n_1, n_2)$ を求める．

- n_1 と n_2 の子ノードを導出する規則が異なるとき，$C(n_1, n_2) = 0$
- 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とき，$C(n_1, n_2) = 1$
- 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とき

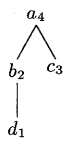
$$C(n_1, n_2) = \prod_{i=1}^{nc(n_1)} (1 + C(ch(n_1, i), ch(n_2, i))) \tag{8}$$

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 (8) では子ノード間の類似度を用いているが，各ノード間の類似度を後順序で求めることで，再計算することなく効率的にすべてのノード間の類似度を求められる．その結果計算に必要なオーダーは，比較する構文木のノードの数をそれぞれ m, n とおくと，$O(nm)$ となる．

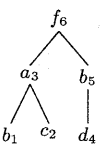
例として (9) と (10) の類似度を考える．各ノードのアルファベットはノードを示し，数字は後順序でノードを選んだ時

の順序を表している．式 (6) は対象とする 2 つの構文木の各ノード対における類似度の総和を求めている．ここで各ノード対での類似度 $C(n_1, n_2)$ は上記の計算方法により表 1 のように求められる．後順序で $C(n_1, n_2)$ を求めていけば，$C(a_4, a3)$ を計算するときに $C(b_2, b_1)$ と $C(c_3, c_2)$ はすでに求められており，その結果を用いることができる．構文木全体の類似度は表 1 のすべての要素の総和により類似度 9 が求まる．

(9)



(10)



(9)

a_4	0	0	4	0	0	0
c_3	0	1	0	0	0	0
b_2	1	0	0	0	2	0
d_1	0	0	0	1	0	0
	b_1	c_2	a_3	d_4	b_5	f_6

(10)

表 1　(9) と (10) の類似度評価における $C(n_1, n_2)$ のマトリックス

2.4 Dependency Structure Kernel

Collins ら [1] は Tree kernel と同じように依存構造木におけるカーネルを提案した．ここで対象としている依存構造は，ノードが語であり，ノード間には"sbj", "obj", "pp", "dt"などの関係が示されている構造である．

Dependency Structure Kernel $K(d_1, d_2)$ は Tree Kernel と同じように式 (11) により求まる．

$$K(d_1, d_2) = \sum_{n_1 \in N_1, n_2 \in N_2} C(n_1, n_2) \tag{11}$$

ここで $C(n_1, n_2)$ は以下のようにして求められる．

- n_1 と n_2 のノードが等しくないか，n_1 と n_2 のどちらかが子ノードを持っていなかったら $C(n_1, n_2) = 0$
- それ以外は以下の式により求める

$$C(n_1, n_2) = \left(\prod_{(x,y) \in sim(n_1, n_2)} (C(x, y) + 2) \right) - 1 \tag{12}$$

$sim(n_1, n_2)$ は n_1 と n_2 の子の中で，n_1, n_2 が同じ関係で同じ語を持つ子ノードのペアの集合を返す関数である．

3. 構文的類似度の評価方法

Collins らの提案している Tree Kernel や Dependency Structure Kernel は共通に持つ部分木の数を数えることで，木構造における類似度を表している．しかしこの類似度の計算方法は，どんなアプリケーションにもしているわけではない．3.1 節ではこの問題を取り上げ，類似度の種類について議論する．3.2 節では類似度の計算方法を整理し，3.3 節でその計算式に対していくつかの拡張を加える．

3.1 類似度の種類

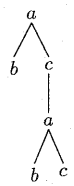
(13) に対する類似度を (14) から (18) の木について比較するという問題を考えよう．

(13)

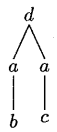
(14)

(15)

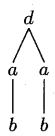
(16)



(17)



(18)



Collins らの Dependency Structure Kernel を用いると，(13) との類似度の大小関係は

$$(16) > (15) > (14) > (17) = (18)$$

となる．

しかし，この評価が適当かどうかはアプリケーションに依存する．アプリケーションによっては，類似度の大小関係を以下のように評価することが望ましい場合も考えられる．

- (14) の類似度をもっとも高く評価したい ((13) の情報を過不足なく含んでいる)
- (14) = (15) = (16) としたい．(重複して照合する場合は数えない)
- (18) よりも (17) の類似度を高くしたい ((18) よりも (17) の方が (13) の情報をより多く含んでいる)

たとえば，情報検索においてはクエリーの情報をとにかくたくさん含んでいるテキストほど類似度を高くするべきだろう．また，用例ベースの機械翻訳では，構造全体がもっとも大きくクエリーと一致したパタンを選ぶ必要があるだろうし，言い換えにおける変換パタンの選択においては，完全に一致する必要があるだろう．質問応答においては，クエリーに含まれる情報を重複は考慮せず多く持つテキストほど類似度を高くするべきである．

そこで，類似度を以下の 3 種類に分類することを考える．

〈**類似度 A**〉　共通する部分木を多く含む木ほど類似度が高い

$$K_A(T_1, T_2) = \sum_{n_1 \in N_1} \sum_{n_2 \in N_2} C(n_1, n_2) \tag{19}$$

〈**類似度 B**〉　探したい木の情報をなるべく多く含んでいる木ほど類似度が高い．このとき探したい木そのものとの類似度が最大で，それ以上の類似度は現れない

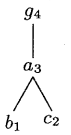
$$K_B(T_1, T_2) = \sum_{n_1 \in N_1} \max_{n_2 \in N_2} C(n_1, n_2) \tag{20}$$

〈類似度 C〉 両方の木のマッチする部分が大きいほど類似度が高い

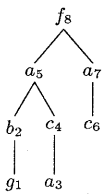
$$K_C(T_1,T_2) = \max_{n_1 \in N_1} \max_{n_2 \in N_2} C(n_1,n_2) \quad (21)$$

(22) と (23) を用いて説明しよう．両者を比較すると，各ノードの対応 $C(n_1,c_2)$ については表 2 に示す値が得られる．

(22)



(23)



(22)

g_4	1	0	0	0	0	0	0	0
a_3	0	0	1	0	4	0	2	0
c_2	0	0	0	1	0	1	0	0
b_1	0	1	0	0	0	0	0	0

(23)

	g_1	b_2	a_3	c_4	a_5	c_6	a_7	f_8

表 2 (22) と (23) の類似度評価における $C(n_1,n_2)$ のマトリックス

〈類似度 A〉 では 表 2 のすべての値を足し合わせ，類似度 11 が得られる．この値は (23) に (22) の部分木が多く含まれるほど高くなる．〈類似度 A〉の計算方法では，比較する構文木に共通に含まれる部分木の数が多いほど類似度が高くなるので情報検索のようなタスクに有効だと思われる．

〈類似度 B〉 では各行での最大値が選ばれる．この例では $(g_4,g_1),(a_3,a_5),(c_2,c_4),(b_1,b_2)$ の対応がとられる(注1)．構文木間の類似度はこれらのノードの類似度の和となり 7 が得られる．この値は (22) の部分木が不足なく (23) にあるほど，また構文情報が一致しているほど類似度は高くなる．(22) の各ノードについて最大値を選んでいるので，(23) の c_4，c_6 のように複数のノードが対応しても，重複して数えられることはない．〈類似度 B〉の類似度計算方法では，比較する構文木のうち片方の構文木に含まれる部分木を，もう片方の構文木がなるべく広くカバーしているほど類似度が高くなる．質問応答のようなタスクでは，質問文に含まれる情報を対象のテキストが不足なくカバーしているほど類似度が高くなるので，質問応答のようなタスクに有効だと思われる．

〈類似度 C〉 では 表 2 の内，もっとも高い値 4 が選ばれる．(22) のうち (23) にもっとも大きく対応のとれる部分木のルートである (22) の a_3 と (23) の a_5 との類似度が全体の類似度となる．〈類似度 C〉の類似度計算方法では，比較する構文木中の部分木の中で共通に持つもっとも大きな部分木の類似度が結果として返されるので，両方の構文木が総括的に照合するほど類似度が高くなる．したがって用例ベースの機械翻訳などにおいてデータベースから知識を取り出すタスクなどに有効だと思われる．

これらの式の他に，「探したい木そのものに近く，余分な情報がないほど類似度が高い」という尺度も考えられる．この類似度に関しては，Collins ら [2] が提案するように以下の式を用いて正規化することによって求められる．

$$K'(T_1,T_2) = \frac{K(T_1,T_2)}{\sqrt{K(T_1,T_1)K(T_2,T_2)}} \quad (24)$$

3.2 ノード間の類似度計算

より一般的な木構造の類似度計算に拡張するために，木構造を以下のように定義する．

- 子ノードの順序は任意
- ノード自体が素性を持っている．
- ノード間の arc には関係が示されている．

3.2.1 子ノードの順序

Collins らの Tree Kernel では，同じ導出規則により導出されるノードを対象としていたので，順序木間の類似度を評価していたことになる．ここで対象とする木の兄弟間の順序を任意とすると，$C(n_1,n_2)$ の値は式 (25) によって求められる．

$$C(n_1,n_2) = \prod_{k \in ch(n_1)} \prod_{l \in ch(n_2)} (1 + C(k,l)) \quad (25)$$

ここで $ch(n)$ は n の子ノードの集合を返す関数である．Collins らの Dependency Structure Kernel では，$sim(n_1,n_2)$ という関数を定義することにより対応するノード対の集合を得ていたが，ここでは式 (25) のように子ノード間の組み合わせを考慮することで同等の処理を実現できる．式 (12) がノード間の関係のみを数えるのに対し，式 (25) は Collins の Tree Kernel の定義と同様に共通に含まれる部分木の数を数えている．単独のノードの存在も類似度に影響すると考えこのような計算方法を定義した．

3.2.2 ノードの素性

ノード自体が素性を持つことを許し，ノード間の照合を柔軟に行なえるように式 (25) を式 (26) のように拡張する．

$$C(n_1,n_2) = sim(n_1,n_2) \prod_{k \in ch(n_1)} \prod_{l \in ch(n_2)} (1 + C(k,l)) \quad (26)$$

ここで $sim(n_1,n_2)$ は n_1 と n_2 のノードの持つ素性間の類似度である．例えば，単語間の意味的な類似度や文節中の機能語の類似度などが考えられる．このようにノード間の類似度を考慮する方法は鹿島ら [6] が提案している．鹿島らと同様に，我々の評価尺度もノードの持つ素性間の類似度を考慮できるように自然に拡張することができる．

3.2.3 ノード間の関係

ノード間の関係を考慮できるように式 (26) を拡張する．関係の類似度を $relsim(rel_1,rel_2) \in [0,1]$，$n$ と k の間の関係を $rel(n,k)$ で表すと，式 (27) のように拡張できる．

$$C(n_1,n_2) = sim(n_1,n_2) \prod_{k \in ch(n_1)} \prod_{l \in ch(n_2)} relsim(rel(n_1,k), rel(n_2,l)\, C(k,l)) \quad (27)$$

(注1)：(c_2,c_4) と (c_2,c_6) の類似度はどちらも 1 であり優劣がないので，(c_2,c_6) が選ばれる可能性もあるが，同値であるためこの選択が全体の類似度に影響を与えることはない．

式 (27) を用いることにより，a) 子ノードの順序の制約なく，b) ノード自身が素性を持っており，c) ノード間の アークに関係が示されているような木構造に対して類似度を評価できる．

式 (27) ではパラメータとして，ノード間の類似度 $sim(n_1, n_2)$ や，関係間の類似度 $relsim(rel_1, rel_2)$ が使われている．これらのパラメータを調整することにより，類似度計算の振舞を変更できる．例えば $relsim(rel_1, rel_2)$ を 1 にした場合，ノード間の類似度 $sim(n_1, n_2)$ だけが使われることになる．

これまでに Collins の提案した計算手法に，子ノード間の順序の制約の緩和，ノードの素性の考慮，ノード間の関係の考慮の拡張を行なったが，計算量の増加は定数倍に抑えられるため，それぞれの木に含まれるノードの数を $|T_1|$，$|T_2|$ とするとオーダは $O(|T_1||T_2|)$ のままである．

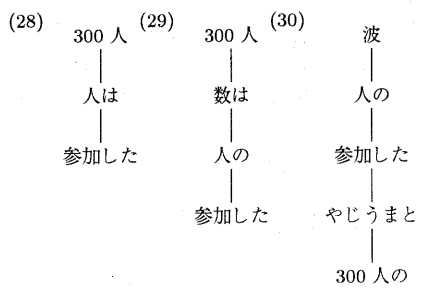
3.3 類似度計算の拡張

本稿で議論している類似度をさまざまなアプリケーションに応用するときにはいくつかの拡張が必要となると考えられる．ここではより柔軟に類似度を計算できるように，類似度の計算方法について以下の拡張を行う．

- 親子関係の条件の緩和
- 各ノード間の対応

3.3.1 親子関係の条件の緩和

(28) と (29) を比較すると，これらの依存構造木は意味的には似ているが，(29) には「数は」というノードが入っているために，上述の評価方法では「参加した–人の」と「300 人」という二つの構造が独立に対応づけられてしまう．したがって (28) と (29) の類似度は，意味的に異なる (28) と (30) の類似度とほぼ同じ値となってしまう．

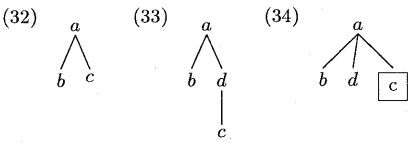


そこで，式 (27) を式 (31) のように拡張することにより，ノードを飛び越えてマッチできるようにする．

$$C(n_1, n_2) = sim(n_1, n_2) \prod_{k \in ch(n_1)} \prod_{l \in ch(n_2)} \max_{s \in ex(k)} \max_{t \in ex(l)} pen(k, s)\, pen(l, t)\, relsim(rel(n_1, k), rel(n_2, l))\, C(k, l) \tag{31}$$

ここで $pen(k, s) \in [0, 1]$ は ノード k を飛び越えて代わりにノード s を参照するときに課せられるペナルティであり，$ex(k)$ はノード k をルートとする部分木から k を飛び越えて照合可能なノードの集合を返す．

例えば (32) と (33) の類似度を計算する時に，式 (27) では (33) の c が a の子孫であるという情報を考慮することができない．しかし式 (31) を用いることで，(34) のように c を a の子ノードのように扱うことができる．このとき c にはペナルティが課せられ，a の類似度を求める時には b と c のうち大きい値が用いられる．

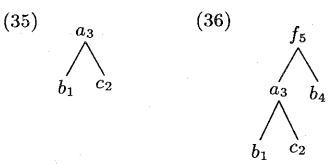


$ex(k)$ により飛び越えるノードの数を定数段にすることで，計算量の増加は定数倍に抑えられるため，両方の木に含まれるノードの数を n, m とするとオーダーは $O(|T_1||T_2|)$ となる．

親子関係の条件の緩和については，鹿島らも提案しているが，鹿島らの手法ではすべての子孫ノードも対象とするので，多く数え過ぎてしまうという問題がある．$ex(k)$ を調節することで助詞だけは飛び越え可能といったような処理を行なえる．

3.3.2 ノード間の対応の特定

これまでに示してきた計算手法では 2 つの木全体の類似度を求めることができるが，ノード間の対応関係がうまく判別できない場合が生じる．例えば (35) と (36) の類似度を評価した場合，(35) の b_1 が (36) 中の b_1 と b_4 のどちらと対応するのか局所的な情報だけでは判別できない．しかし，ここでは明らかに (36) の b_4 との対応を選択すべきである．



あるノード対の類似度は，すでに計算されたそれらの子ノードの類似度を用いて計算される．このアルゴリズムにより共通に持つ部分木を高速に数え上げることができるのだが，この計算法ではあるノードの類似度はそのノード自身とその子ノードだけに依存することになる．したがってノード間の類似度には親ノードの情報を用いることができず，特に leaf においてはノード間の照合にはノード自身の情報だけが使われる．このアルゴリズムではノード間の構文情報を十分考慮して類似度を求めることができない．

ここで問題となるのは，2 つのノード間の類似度をそれぞれのノード自身と子ノードだけの情報により求めているという点である．類似度の計算方法を拡張し，ノード間の類似度にそれぞれのノードの親の情報も使うように拡張する．

各ノードをルートとする部分木間に共通に含まれる部分木を求める式 (25) を拡張し，式 (37) を用いる．

$$C(n_1, n_2) = C_{bu}(n_1, n_2) \times C_{td}(n_1, n_2) \tag{37}$$

$$C_{bu}(n_1, n_2) = \prod_{i=1}^{nc(n_1)} \prod_{j=1}^{nc(n_2)} (1 + C(ch(n_1, i), ch(n_2, j)))$$

(38)

$$C_{td}(n_1,n_2)=\left(C_{td}(p(n_1),p(n_2))\times\frac{C_{bu}(p(n_1),p(n_2))}{C_{bu}(n_1,n_2)}\right)+1 \quad (39)$$

ここで $p(n)$ はノード n の親ノードを示す．

式 (37) で用いる 式 (38) は 式 (25) と同じ式であり，Collins らの提案するボトムアップな計算による子ノード方向への部分木の数である．この式には 3.2 節で述べたようにいくつかのパラメータにより拡張可能である．

式 (39) は，ボトムアップの計算が終了した後に，トップダウンに親ノード方向への共通部分木の数を計算する．最終的にあるノードとノード間の共通部分木の数はボトムアップに計算した子ノード方向への共通部分木の数とトップダウンに計算した親ノード方向への共通部分木の数の積により求められる．この値は，それぞれのノードを含む共通部分木の数を返すことになる．

表 3 は式 (25) で各ノードの類似度を求めた結果である．ここで，(35) の b_1 に対応する部分は，表 3 を見ると，(36) の b_1 と b_4 の二つのノードがあり，それぞれ同じ類似度となっている ($C(b_1,b_1)=C(b_1,b_4)$)．これに対し，式 (37) により求めた表 4 では，$C(b_1,b_1)>C(b_1,b_4)$ であり，(35) の b_1 に対応するノードとして (36) の b_1 を正しく選択できる．このように式 (37) を用いることにより，ノードのまわりの構文情報を考慮した共通部分木の数を求めることが可能となり，ノード間の対応をより正確に評価することができる．

(35)

a_3	0	0	4	0	0
c_2	0	1	0	0	0
b_1	1	0	0	1	0
	b_1	c_2	a_3	b_4	f_5

(36)

表 3 (35) と (36) の各ノードの対応 (式 (25))

(35)

a_3	0	0	4	0	0
c_2	0	3	0	0	0
b_1	3	0	0	1	0
	b_1	c_2	a_3	b_4	f_5

(36)

表 4 (35) と (36) の各ノードの対応 (式 (37))

4. 質問応答タスクへの応用

冒頭で述べたように，構文的類似度評価は質問応答タスクに応用することができる．本節では，まず既存の質問応答システムにおける構文情報を利用方法の例を紹介し，上で提案した構文的類似度評価法がそれらに比べていくつかの良い性質を持っていることを示す．さらに，解答探索における言い換え処理の役割について述べ，残された課題に言及する．

4.1 構文情報を利用した質問応答システム

(40) のような単純な場合を考えてみよう．(40a) のような質問文が与えられた場合，(40b) のように疑問詞を変数 (以下，「**質問変数**」と呼ぶ) に置き換え，ある種の「標準形」に変換することができれば，あとはこれとうまく照合する文をテキストデータベースから探し出せばよい．たとえば (40c) のような文が見つかれば，答えは「太郎」であることがわかる．

(40) a. ［質問文］ 誰が公園で泣いていたか？

b. ［質問文の肯定形］ X_{hum} が公園で泣いていた．

c. ［解答候補テキスト］ 花子は公園で太郎が泣いているのを見かけた．

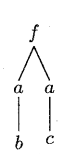
ただし，一般には，質問文の肯定形 (以下，単に「**質問文**」と呼ぶ) が解答候補テキスト (以下，「**情報源**」) と完全に一致することはあまり期待できないので，両者の照合の度合を適切に評価する何らかの手立てが必要になる．質問応答に構文情報を利用する方法については，すでにいくつかの提案がある．

たとえば，村田らの手法 [9] では，質問文と情報源をおのおの (a) 文節を要素とする集合と (b) 文節間の係り受け関係を要素とする集合で表現し，(a) 文節集合間の類似度と (b) 係り受け関係集合間の類似度の和によって全体の類似度を評価する．(b) の係り受け関係集合間の類似度は構文的類似度の近似を与えていると言えるが，2 つの文節間の係り受け関係しか扱わないので，より大域的な構造的類似度を反映させることは難しい．たとえば，村田らの尺度によると，(41) に対して (42) と (43) を比較したい場合に，「(42) だけが b と c の姉妹関係を保持している」という情報が無視されてしまう．また，(28) から (30) の例で指摘したような，ノードを飛び越えた祖母・孫間の依存関係を適切に扱うことができない．文献 [9] では，3 つ以上のノードを含む依存関係の類似性も反映するように拡張できるとしているが，それを効率的に計算する方法については言及していない．

(41)

(42)

(43)



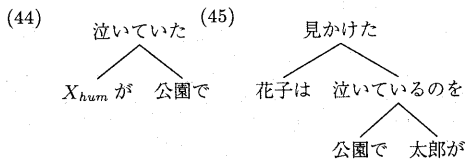
同様の議論は，清田ら [7] の質問応答システムにおいても成り立つ．彼らの類似度評価法もやはり一段の係り受け関係しか扱わないので，村田らの手法と同様に大域的な構造的類似度を評価し損う可能性がある．

依存構造間の類似度評価を質問応答に利用する試みは，Harabagiu ら [3] や Ittycheriah ら [5] など，TREC の QA Track でもいくつか報告されている．これらの報告でも，質問文や情報源の依存構造を考慮することが正解率の向上に寄与することが示されているが，依存構造の類似度については本稿で論じたような粒度の細い議論は見られない．

4.2 質問応答のための構文的類似度評価法

質問文と情報源の「照合の度合」の評価尺度として本稿で

提案した構文的類似度を用いることを考えよう．まず，(44)，(45) のような文節をノードとする依存構造木を類似度の評価対象とする(注2)．このとき，質問文と情報源の類似度は以下のように推定できる．



4.2.1 類似度の種類

村田らや Harabagiu らを含む多くの先行研究が指摘しているように，質問応答では質問文をできるだけ不足なく覆う情報源を選択するのが良い．したがって，質問文の依存構造木を T_1，情報源の依存構造木を T_2 とするとき，式 (20) で与えられる〈類似度 B〉で両者の類似度を評価するのが適当であると考えられる．3.1 節で述べたように，情報源に同じ情報 (同じ部分木) が重複して存在しても，情報源の情報は増えるわけではないので，式 (19) で与えられる〈類似度 A〉は適当ではない．

4.2.2 ノード間の類似度

ノード間の類似度 $C(n_1, n_2)$ は式 (31) で与えられる．各パラメタは以下を考慮して設定する．

- **ノードの持つ素性間の類似度** $sim(n_1, n_2)$：文節 n_1，n_2 の主辞の品詞の類似度 (e.g. 名詞どうしの類似度の方が名詞と動詞の類似度よりも大きい)，主辞の意味的類似度，機能語の役割の類似度 (e.g.「は」と「が」は「が」と「を」よりも近い) などに基づいて決める．主辞の意味的類似度の計算には，情報検索で使われるタームの重みづけの方法や単語間のシソーラス上の距離で近似する方法など，いくつかの選択肢が考えられる．また，質問応答では，(44) の X_{hum} のような質問変数に対応する表現を情報源から見つけることが目的であるので，n_1 がそうした変数である場合は，特殊な扱いが必要である．これについては，n_2 が変数 n_1 と同じ意味タイプのノードのときに特別に加点し，そうでないときに特別に減点するようにパラメタを設定すればよい．
- **ノード間の関係の関係の類似度** $relsim(rel_1, rel_2)$：ノード間の関係を表現する機能語列はそれが属する文節の素性として扱うので，ここでは考えなくてよい．すなわち，$relsim$ の値を常に 1 とする．
- **ノードの飛び越しによるペナルティ** $pen(k, s)$：(28)，(28) の例からも示唆されるように，「数」や「人」，「事件」，「理由」など，抽象度の高い意味を表す内容語を主辞とするノードは情報量が少く，欠落しても文の意味を大きくは変えないことが多い．したがって，そのようなノードについては，飛び越しによるペナルティを低くしてよいだろう．

(注2)：質問文と情報源はともに一文とは限らず，一般には複数の文からなるテキストである．複数の文からなる場合は，根ノードに各文の依存構造木がぶら下がる形の木で表現するものとする．

4.2.3 答えの特定

質問の答えは，質問文中の質問変数に対応する情報源中の部分木で与えることができる．多くの場合，変数は依存構造木の葉の位置に来るので，これと対応する情報源の部分木を特定するには，3.3.2 で述べた方法が有効である．

4.2.4 利　　点

上述の構文的類似度評価方法には既存の方法に比べて以下のような利点がある．

第 1 に，(a) 姉妹間や祖母・孫間の依存関係を含む大域的な依存構造の類似度と (b) 質問文・情報源を bag-of-words に近い局所的要素の集合と見なしたときの類似度を自然に融合することができる．質問文と情報源が大きな部分木を共通に持つ場合はそれが全体の類似度に大きく貢献するが，そうした共通部分木がない場合は共通する (あるいは類似の) 単語の数で情報源を比較することになる．

第 2 に，ノードの飛び越しを許すので柔軟な木の照合が可能である．(28) と (29) の例からもわかるように，言語には同様の内容を表す多様な表現が用意されているので，依存構造木の照合を行う上でこうした柔軟性は重要である．

第 3 に，上述のように大域的で柔軟な木の照合を実現しているにもかかわらず，計算量のオーダは $O(|T_1||T_2|)$ で，bag-of-words による類似度評価の計算量の高々定数倍に抑えることができる (3. 節参照)．

大域的で柔軟な木の照合，すなわち上述のような各種パラメタが質問応答の性能にどのように寄与するかは今後明らかにすべき課題である．上で提案した類似度評価方法は，そうした課題に取り組むための現実的な枠組みを与えると期待できる．

4.3 課　　題

4.3.1 構文的照合と言い換え

構文的類似度は，あくまで質問文と情報源の構文的照合を実現するためのもので，意味的類似性の近似にすぎない．質問文と情報源の言語表現の差を吸収するためには，質問文や情報源をそれぞれ (標準的な言い回しの) 同義表現に言い換えて比較する (e.g. [10])，あるいは意味表現に抽象化して比較する (e.g. [3], [11]) など，何らかの工夫が必要である．我々現在は，言い換えを利用するアプローチを取り上げ，その実現可能性と効果を調べる研究を進めている．

言い換えを用いるアプローチでは，たとえば (46c) のような言い換えを施して質問文と情報源の構文的あるいは語彙的な差を吸収する．

(46) a.　［質問文］ 公園で泣いていたのは誰ですか？

b.　［質問文の肯定形］ 公園で泣いていたのは X_{hum} です．

c.　［質問文の言い換え］ X_{hum} が公園で泣いていた．

課題は主として，(a) 多様な言語表現の同義性を十分に吸収できる多様で頑健な言い換えをどうやって実現するか，(b) テキストのどの部分にどのような言い換えを施すべきかをどうやって制御するか，の 2 点である．(a) の言い換えの工学的実現は近年研究者の関心が高まっているテーマであり [4]，我々も構文

的・語彙的言い換えシステム KURA [12](注3) の開発を進めきた．多様な言い換えの実現が質問応答にどのように寄与するかを分析するのはまさにこれからの課題である．

4.3.2 構文的類似度と意味的含意

意味的類似度の近似として構文的類似度を使う場合，次のような注意も必要である．(47) の例を考えよう．(47a) を質問文，(47b) から (47d) を情報源の候補とすると，答えは「ソバ」であって，「ステーキ」や「ヤサイ」ではない．(47c) は否定の「ない」を含んでいるし，(47d) は仮定であって事実ではないからである．

(47) a. 太郎は $X_{physobj}$ を食べた．

b. 太郎が 昨日の昼に 食べ たのはソバだった．

c. 太郎は ずっとステーキ を食べ ていない．

d. 太郎が ヤサイ を食べ てくれれば，おじいさんも喜ぶのに．

この例からもわかるように，構文的類似度はあくまで意味的類似度の近似であり，構文的類似度が高いからと言って情報源が質問文の内容を「意味的に含意している」とは限らない．

この問題に対して考えられる解決策の一つは，構文的類似度が十分に大きくなるまで質問文と情報源を言い換えるという処理のあとに，情報源が質問の答えであることをモダリティ情報などに基づいて確認する処理を追加することである．Harabagiu らの質問応答システムは質問文の内容が情報源と背景知識によって論理的に被覆されるかを近似的に調べる機能を持っているが，これに近い処理が必要になる可能性もあり，興味深い問題である．

5. ま と め

本稿では構文木間の一般的な類似度評価方法を提案し，性質の異なる様々な類似性尺度がいくつかのパラメタの組み合わせによって表現できることを示した．本評価方法では，第 1 に，類似度の種類を ⟨ 類似度 A⟩，⟨ 類似度 B⟩，⟨ 類似度 C⟩ の 3 種類から選択できる．第 2 に，対応ノード間の類似度の定量化，ノードの飛び越えを許す．第 3 に，計算量のオーダを $O(|T_1||T_2|)$ に抑えることができる．

また，提案した構文的類似度評価方法が質問応答タスクに応用できる可能性について議論した．我々の評価方法は，従来の手法に比べてより大域的な構造の照合をより柔軟に行うことができ，計算コストのオーバヘッドも定数倍に抑えることができる．実際の質問応答タスクに適用した結果については別の機会に報告する予定である．

文 献

[1] 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.

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

TREC 2001

[3] Harabagiu, S., Moldovan, D., Pasca, M., Surdeanu, M., Mihalcea, R., Girju, R., Rus, V., Lactusu, F., Morarescu, P. and Bunescu, R. Answering Complex, List and Context Questions with LCC's Question-Answering Server. In *Proceedings of the 2001 edition of the Text REtrieval Conference (TREC)*, 2001.

[4] 乾健太郎. 言語表現を言い換える技術. 言語処理学会第 8 回年次大会チュートリアル, 2002.

[5] Ittycheriah, A., Franz, M. and Roukos, S. IBM's Statistical Question Answering System–TREC-10. In *Proceedings of the 2001 edition of the Text REtrieval Conference (TREC)*, 2001.

[6] 鹿島 久嗣, 小柳 光生. 半構造データへのサポートベクターマシンの適用. 第 48 回人工知能基礎論研究会 (SIG-FAI48), 2002.

[7] 清田陽司. 大規模テキスト知識ベースに基づく自動質問応答. 言語処理学会第 8 回年次大会発表論文集, 2002.

[8] Meyers, A., Yangarber, R., Grishman, R., Macleod, C. and Moreno-Sandoval, A. Deriving Transfer Rules from Dominance-Preserving Alignments. In *Proceedings of the Conference, Vol.2 *36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics (COLING-ACL)*, 1998.

[9] 村田 真樹, 内山 将夫, 井佐原 均. 類似度に基づく推論を用いた質問応答システム. 自然言語処理研究会報告書, NL-135-24, 2000.

[10] Murata, M. and Isahara, H. Nuniversal Model for Paraphrasing — Using Transformation Based on a Defined Criteria. In *NLPRS-2001 Workshop on Automatic Paraphrasing: Theories and Applications*, 2001.

[11] Scott, S. and Gaizauskas, R. University of Sheffield TREC-9 Q&A System. In *the Proceedings of Text REtrieval Conference (TREC-9)*, page 635, 2000.

[12] Takahashi, T., Iwakura, T., Iida, R. and Inui, K. Kura: A Revision-Based Lexico-Structural Paraphrasing Engine. In *NLPRS-2001 Workshop on Automatic Paraphrasing: Theories and Applications*, 2001.

[13] Tai, K. The tree-to-tree correction problem. *Journal of the ACM, 26(3):422-423*, 1979.

[14] Watanabe, H., Kurohashi, S. and Aramaki, E. Finding Structural Correspondences from Bilingual Parsed Corpus for Corpus-based Translation. In *Proceeding of the Conference * The 18th International conference on Computational Linguistics Vol.2 (COLING)*, 2000.

[15] Woods, W.A., Green, S., Martin, P. and Houston, A. Halfway to Question Answering. In *the Proceedings of Text REtrieval Conference (TREC-9)*, page 489, 2000.

[16] Zhang, K. and Shasha, D. Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal of Computing, 18(6):1245-1262*, 1989.

(注3)：http://cl.aist-nara.ac.jp/lab/kura/doc/