特集論文

コーパス検索支援のための動的同義語候補抽出

Dynamic Synonym Candidates Extraction for Searching Documents in a Corpus

吉田 稔
Minoru Yoshida

東京大学情報基盤センター
Information Technology Center, University of Tokyo
mino@r.dl.itc.u-tokyo.ac.jp, <http://www.r.dl.itc.u-tokyo.ac.jp/~mino/>

中川 裕志
Hiroshi Nakagawa

(同 上)
nakagawa@dl.itc.u-tokyo.ac.jp, <http://www.r.dl.itc.u-tokyo.ac.jp/~nakagawa/>

寺田 昭
Akira Terada

(株)日本航空
Japan Airlines Co., Ltd.
akira.terada@jal.com

keywords: synonyms, suffix arrays, query support systems

Summary

This paper proposes a method for implementing real-time synonym search systems. Our final aim is to provide users with an interface with which they can query the system for any length strings and the system returns a list of synonyms of the input string. We propose an efficient algorithm for this operation. The strategy involves indexing documents by suffix arrays and finding adjacent strings of the query by dynamically retrieving its contexts (i.e., strings around the query). The extracted contexts are in turn sent to the suffix arrays to retrieve the strings around the contexts, which are likely to contain the synonyms of the query string.

1. は じ め に

近年, WWW 上や組織内に蓄積される電子的文書の量は増大の一途を辿り, それらの文書全体を人間が把握することが困難となっている. WWW 全体における文書量の増大のみならず, 例えばその中でのサイトの限定された部分集合 (特定組織の Web 文書集合, Wikipedia 等), さらには企業内文書集合等に至るまで, 様々なコーパスにおいてもそのサイズは増大し, WWW 全体と同様に, 把握が困難となりつつある. 本稿では, そのような特定のコーパスの活用について論じる.

検索エンジンは, 文書集合を効率的に利用する有力な手段の一つであるが, その機能は基本的には「入力された文字列 (クエリ) を含む文書を返す」という範囲に留まっており, 「どのようなクエリを入力するべきか」というクエリ入力支援の問題に対しては, 研究の余地が多く残されている.

同義語抽出は, クエリ入力支援の中で, 最も一般的なタスクである. 同義語とは, 異なる語で同じ意味を持つものであり, 例えば「HDD」と「ハードディスク」「掲示板」と「BBS」等がある. 同義語は, 主に, 異なる記述者が, 同一の意味を異なる言葉で表現することにより生じる. 特に, Web 文書の集合では, 「不特定多数のユーザが」「特定の統一基準のない下で」作成した文書が多

数集まっていることが普通であり, 多くの同義語が文書集合内に存在することになる. このような場合, 例えば, 「HDD」で検索しても「ハードディスク」と書かれた文書は発見できないことになり, 検索の網羅性の点で問題が生じる. 特に, 本稿で対象とする「特定文書集合における検索」では, Web 全体に対する検索に比べ, 取得できる文書数が比較的少なく, また, 取得したい文書がすでにある程度特定されていることが多いため, 高い網羅性が要求される. 例えば, 特定組織の Web 文書内で, 特定個人の電話番号を検索したい場合に, 「電話番号」と入力しても, 目的の文書内に電話番号が「TEL:…」と記述されていればその文書は発見できないことになる.

本稿で提案するのは, ユーザの入力クエリに対し, その同義語の候補を提示するシステムである. ユーザは提示された同義語候補から検索に用いたい同義語を選択することにより, 検索の網羅性を上げることができる.*1

*1 なお, 実際には, 同義語でない言葉の中にも, 完全に違う意味ではなく, 意味の類似した語 (類義語) というものが存在する. 「周辺文脈の似た文字列を取得する」という提案手法の特徴のため, 実際の出力の中には, このような類義語も多く含まれることになる. (後述の「ワークショップ」と「シンポジウム」等, 完全に同じ意味ではないが置き換え可能なもの等.) 実際のシステムでは, この類義語も, 有用な文字列として検索支援に積極的に用いることを行っている (例えば会議全般の情報を得るため「ワークショップ」と「シンポジウム」を OR 検索する, 等.) 実際の利用において, ユーザーにとって適切な網羅

クエリ入力支援において求められるのは、「どんな文字列にも対応できる」「高速な」アルゴリズムである。ここで問題となるのは、ユーザーによって入力される文字列の多様性である。同義語抽出や言い換えに関しては従来より多くの研究が行われているが、その大部分は、予め規定された候補語どうしの類似度を測るものである。このため、本システムが想定する「ユーザがあらゆる文字列を入力する状況」において同義語を求めるためには、文書のあらゆる部分文字列どうしについての類似度を計算する必要が生じ、必要な計算時間および記憶容量は膨大なものとなる [Gorman 06]。

この問題に対し、本研究では「ユーザからクエリが与えられた際、On-Demand にそれと類似する文字列を文書から取り出す」という、新たな動的同義語抽出アルゴリズムを提案する。提案手法は、動的同義語抽出の実現に、Suffix Array による検索を用いる。Suffix Array は、任意の文字列の文書中の位置を高速に検索できる全文検索のための索引構造であり、提案手法は、この高速検索機能を、クエリの文脈抽出に用いることと、文脈に隣接する文字列の抽出に用いることで、同義語抽出のための「文脈の類似した語の取得」を、検索により実現する。提案アルゴリズムは、与えられるあらゆるクエリ文字列に対して同義語を検索することが可能なため、例えば、従来の同義語抽出システムでは困難であった、「本稿では」（「名詞＋助詞」）のような異なる品詞の接続や、「(株)」のような記号を伴った言い回し等に対しても、同義語（同義文字列）の提示が可能となる。

表 1 に、従来手法と提案手法の違いを示す。既存の同義語抽出手法では、事前に何らかの規準（例えば「名詞」や「クエリログに蓄積された入力文字列」等）で候補語を決定し、候補語同士の類似度を計算する。そのため、事前に想定された候補語については、それと類似する候補語（同義語）を抽出することができるが、それ以外の候補語については同義語を返すことができない。これに対し、提案手法は、与えられたクエリに対し、動的に文脈を取得するため、コーパス中の部分文字列であれば、任意のクエリに対応できる。また、候補語は「文脈に囲まれる文字列」として自動的に切り出されるため、単語切り分け等の前処理も必要ない（すなわち、単語の境界は、単語切り分けの処理を経ずに自動的に推定されることになる）。任意のクエリに対して、事前に準備されたリストから候補を返すのではなく、その場で全部分文字列の中から同義語リストを計算する既存手法は、筆者らの知る限りでは存在せず、提案手法は、任意のクエリへ同義語を返すことのできる新しい手法である。その一方、文脈として Suffix Array の検索により抽出される接続文字列

表 1 従来手法と提案手法の位置づけ

手法	従来手法	提案手法
候補語	あらかじめ設定した集合	すべての部分文字列
前処理	必要（単語切り分け等）	不要
文脈	係り受け関係、周辺単語等	接続文字列のみ

犬だった。そのことは...
 犬との楽しい日々を...
 犬ではなく、...
 犬は嫌いです。でもドッグフードは...
 犬も歩けば棒に当たるというのは、...
 犬も歩けば棒に当たる！
 犬も猫も大好き！
 犬猫病院

図 1 Suffix Array の例

情報のみを用いるため、係り受け関係や品詞情報といった言語的情報を用いる従来手法に比較すると精度はやや低下する。

事前に候補語を限定しないことは、候補語としてあり得ないような部分文字列まで候補として考慮するということであり、同義語リストの精度を低下させる懸念があるが、我々は、後述する List Cleaning と呼ばれる処理を出力リストに適用することにより、この問題を回避している。

以下、第 2 節において記号の定義と Suffix Array の説明を、第 3 節において提案手法の説明を行う。その後第 4 節にて実際の応用例を示し、第 5 節で提案手法の精度・速度について実験を行う。第 6 節で関連研究について述べ、第 7 節で結論を述べる。

2. 準備：Suffix Array

本稿では、コーパスを S 、クエリを q と表記する。コーパス S は一つの文字列であり、入力が文書集合の場合は、これを連結したものを S とする。システムは、与えられたクエリ q に対し、同義語を発見する。コーパス S に対し、その全ての部分文字列を要素とする集合を $P(S)$ 、あるいは、 S が明らかな場合は P と表記する。

また、文字列 s の文脈を、「 S 中で s に接続する文字列」（すなわち、 $s.x \in P$ 又は $x.s \in P$ ）と定義する。ここで、 x は文字列の結合であり、 $x \in P$ は（ P の定義より） x がコーパス S の部分文字列として出現することを示す。 $s.x \in P$ のとき、 x を s の右文脈、 $x.s \in P$ のとき、 x を s の左文脈と呼ぶ。

Suffix Array [Manber 90] は、与えられた文字列の全ての接尾辞（へのポインタ）を辞書順に並べた配列であり（図 1 参照）、この配列への二分探索を行うことにより、効率的に ($|x|$ を文字列 x の長さとしたとき、 $O(|q| \log |S|)$ の計算時間で) 部分文字列の位置を取得できる（図 2 参照）。この配列は、整数を 4 バイトで表現したとき $4|S|$

性は異なり、ユーザーが候補のうちどれを選択するべきかも異なってくるが、これは「事前に同義語辞書を準備する」というアプローチでは難しいため、インタラクティブに同義語候補を返す手法の利点の一つといえる。

図2 Suffix Array の構築及び検索の例

バイトのメモリを必要とする^{*2}が、我々は、コーパス及び Suffix Array はすべてメモリ上に配置されていると仮定する。

提案手法では、 $\mathcal{A}$ と $\mathcal{A}_r$ という二つの Suffix Array を使用する。前者は S から、後者はそれを反転させた $rev(S)$ から構築した Suffix Array である。 $\mathcal{A}$ は右文脈の取得に、 $\mathcal{A}_r$ は左文脈の取得に、それぞれ使用する。

Suffix Array により実現される操作で、実際に同義語取得において用いるものは、 $nextGrams(\mathcal{A}, x)$ と $freq(\mathcal{A}, x)$ である。前者は、 x を接頭辞に持つ、長さ $|x| + 1$ の $\mathcal{A}$ 中の文字列を、後者は、 $\mathcal{A}$ 中における x の出現回数を、それぞれ返す（以降、明らかな場合は $\mathcal{A}$ を省略し、 $nextGrams(x)$ 、 $freq(x)$ 等と表記する。）例えば「東京都」「東京大学」「東京ディズニーランド」といった文字列を持つコーパスにおいて、 $nextGrams(\text{“東京”}) = \{\text{“東京都”}, \text{“東京大”}, \text{“東京デ”}\}$ などとなる。

また、順序付きリスト $cands$ と、サイズ固定の順序付きリスト $results$ ^{*3}を用意する。 $cands$ はこれから処理される文字列を優先度^{*4}の高い順に保持し、 $results$ はこれまで取得された文脈や同義語候補の上位 n 個をスコアの低い順に保持する^{*5}。 $getFirst$ 操作を行うことにより、 $cands$ からは次に処理する文字列が、 $results$ からは現在最下位の文字列が取得されることに注意されたい。

3. 同義語抽出アルゴリズム

3.1 アルゴリズム

文字列の表記にかかわらず同義語を抽出するアルゴリズムでは、一般に、「意味の似ている語は、似た文脈で使われる」という “Distributional Hypothesis” [Harris 54] に基づき言葉の意味類似度を測定する。特に、「隣接する文字列」は、文脈として非常に有用であることが報告されており [Hagiwara 06]、この「隣接する文字列」は、Suffix Array により容易に検索することができる。この

図3 同義語抽出アルゴリズム概要

ため、提案手法は、

- STEP-1 「クエリ文字列に隣接する文字列」を検索し、
- STEP-2 得られた隣接文字列から適切な「文脈」を選択し、
- STEP-3 「文脈に隣接する文字列」を検索する

という三段階の処理により同義語を抽出する。図3に、手法の概要を示す。図中では、文字列 “president” の文脈として “vice” や “of U.S.” が検索され、さらに、“vice” や “of U.S.” に接続する文字列として、“chairman” や “minister” が取得されている。

この問題を厳密に解くことを考えると、以下のように定式化される。

入力 コーパス S 、そのすべての部分文字列集合 P 、および、入力文字列 q

出力 P の要素を、 q との類似度の高い順に並べたリスト。

ここで、 x と y の類似度は、 x の全ての右文脈の集合 $C_r(x) \subseteq P$ および x の全ての左文脈の集合 $C_l(x) \subseteq P$ を定義したとき、 $C_l(x)$ 、 $C_r(x)$ と $C_l(y)$ 、 $C_r(y)$ を引数とする関数として定義される。

P の要素をすべて出力することや、 C_l, C_r の要素をすべて列挙し類似度計算に用いることは、計算量の増大を招き、本研究の目的とする On-demand な同義語抽出には適さない。しかしながら、後述の通り、 P や C_l, C_r の要素には、それぞれ「同義語候補としての良さ」「文脈としての良さ」を表現するスコア関数が定義される。そこで、提案手法では、スコア関数の高い上位 N 個の要素のみをアルゴリズム中で扱うという近似を行うことにより、アルゴリズムの高速化を図る。長い文脈文字列を用いると高い精度を確保できる反面、取得できる同義語数が少なくなる [Pasca 05]。逆に、短い文脈文字列を用いる手法では、多くの無関係な文字列が候補として取得され、精度を上げるために、対訳テキスト等の別の文脈情報が必要となる。[Shimohata 05] 提案手法は、長さで候補を規定するのではなく、スコアの高い文脈を選択するため、これらの問題をある程度解決していると考えられる。

^{*2} Compressed Suffix Array 等、よりメモリ効率の良い実装も提案されているが、本システムでは通常の Suffix Array を使用する。

^{*3} これらは、赤黒木を実装した Java TreeSet クラスを用いて実現する。TreeSet は、順序付きリストを保持し、 $O(\log(N))$ 時間の add, remove, getFirst 操作を実現する。サイズ固定リストは、新たな要素に対する add 操作と、それに続いて最初の要素を remove する操作を行うことにより実現する。

^{*4} アルゴリズム内では、優先度が高い文字列を先に処理する。スコア関数（後述）の値が高い文字列ほど優先度が高くなる。

^{*5} すなわち、上位 n 個のうちでスコアが最低のものが最初に配置される。

この問題（文脈文字列の上位 N 個を取得する問題）は、文脈文字列の集合をトライ木（根ノードを文字列の先頭とし、文字の接続関係を枝で表現することで、各ノードを一つの文字列と対応させた木構造、例えば、図4中「シンポジウム」よりも右側の木構造）で表現し、各ノードにスコアが付いていると仮定すると、木構造中でスコア関数上位 N 個のノードを選択するという木構造探索の問題として表現することができる（図4）。ここでは、トライ木中の各ノードが文脈文字列を表現する。以下、本稿では、「トライ木」と「探索木」を同義として用いる。木構造探索は、木構造の根ノード（トライ木におけるヌル文字列）から開始され、「何らかの規準で探索済みのノードのうちの一つを選択し、新たにその子ノードのスコアを調べる」という操作を繰り返すことにより行われる。ここでは、「ある文脈文字列 s について、それを接頭辞として一文字増加させた、より長い文脈文字列を探す」ことが、子ノードの探索に相当する。ここで、「スコア関数上位 N 個の要素のみを考慮する」という方針を採用することにより、探索空間の枝狩りによる高速化が可能となる。ここで枝狩りとは、「これ以上子ノードを再帰的に辿っても、上位 N 個の解に入るノードは存在しない（すなわち、子ノード全体に対するスコアの上限が、現在の N 位の解よりも低い）」と判明したノードについては、それ以上の子ノードの探索を打ち切ることである。これは、 N -best 解に対する分枝限定法の一種と見なすことができる。なお、探索の際には、スコア関数をそのまま優先度とした Best-First 探索を行っている。

しかしながら、適切に分枝限定を行うためには、枝狩りが可能となる（すなわち、スコアの上限を定義できる）スコア関数の設計が必要となる。提案手法では、文脈スコアとして TF-IDF に類似したスコア関数を定義し、これを変形させた関数を上限関数として探索空間削減に用いることができることを示す。また、同義語候補の抽出では、リランキングを用いることにより枝狩りを可能とする。

Suffix Tree[McCreight 76] は、与えられた文字列の全ての部分文字列を含むトライ木構造^{*6}であり、Suffix Tree 上で s を検索した結果のノードが、そのままある文字列 s に接続する文脈文字列のトライ木となっている。Suffix Array は、Suffix Tree を使用メモリ量を抑えて表現する索引構造と見なすことができ [Manber 90]、Suffix Array のみを用いて、Suffix Tree 上のノード探索を、最大 $O(\log |S|)$ の追加計算コストで実現することができる。（前述の nextGrams 関数を用いることが、探索木における子ノードの取得に相当する。）このため、Suffix Array による索引付けは、単に接続文字列を取得するだけでなく、同時に、木構造の枝狩り操作も可能とする。すなわち、Suffix Array を用いることで、クエリに接続する文脈文字列のトライ木を探索することが可能となり、枝狩

図4 STEP-1 概要（文字列「シンポジウム」に接続する文字列の探索木の例）。スコアは一部のノードのみ表示。

りによる探索空間削減を実現することが可能となる。

なお、文脈の選択により、本来候補語として取得すべき文字列を大量に見逃すことにより著しく再現率が低下する懸念があるが、後述の実験の結果では、正解のうち8割5分以上を候補として取得できており、現状ではそれほどの再現率の低下は招いていないと考えられる。

以下、STEP-1,2 と STEP-3 について、順を追って説明する。

3.2 STEP-1,2

STEP-1,2 では、後の処理で使われる文脈の数を N_1 とし、スコア上位 N_1 個の文脈を取得する^{*7}。前述の通り、接続する文字列集合は、トライ木として表現され、探索対象の文字列のスコア付けを探索と同時に行うことによって、枝狩りによる探索空間の削減を行う（図4）。

以下、簡単のため、右側接続文脈の探索についてのみに述べる。STEP-1,2 では、クエリ q に接続する文脈 x のスコアを

$$sc_c(x) = freq(q.x) \log \frac{|S|}{freq(x)}$$

で定義する（ $freq$ は頻度、 $q.x$ は文字列 q と x の連結、 $|S|$ はコーパス全体の文字列長）。 $freq(q.x)$ は「 q の隣に x が現れる頻度」、 $\frac{|S|}{freq(x)}$ は「 x の現れる位置の数と、文書中の全ての位置の数の比率」を表しており、TF-IDF 値に類似したスコア付けとなっている。このスコアが上位の N_1 個の文脈を取得するが、その際に、

$$sc'_c(x) = freq(q.x) \log |S|$$

なる上限を用いて枝狩りを行う。 $\log |S| \geq \log \frac{|S|}{freq(x.y)}$ かつ $freq(q.x) \geq freq(q.x.y)$ であるため、 $sc'_c(x) \geq sc_c(x.y)$ が如何なる y についても必ず成り立つ。すなわち、 $sc'_c(x)$ が現在 N_1 位の候補のスコアを下回れば、それ以上の探索を打ち切ることができる。

アルゴリズムを図5に示す。 $nextGrams$ が、木構造中での子ノードの取得に相当する。ここで「」はヌル文字列（長さ0の文字列）であり、また、 s から文脈文字列 c を取り除く操作を $cut(s, c)$ と表記している。

*6 ただし、分岐数1のノードは縮退される

*7 同一スコアの場合は、先に入った要素が高いランクとなる。


```

forall( $c \in C_l$ ) {  $cands \leftarrow (c, '')$  }
while( $cands \neq \emptyset$ ) {
   $D = \emptyset$ ;
  while ( $x$  does not change) {
    ( $c, x$ ) =  $getFirst(cands)$ ;
     $Y = nextGrams(A, c, x)$ ;
    foreach( $y \in Y$ ) {
       $y' = cut(y, c)$ ;
       $D = D \cup \{(c, y')\}$ ;
       $sc_l(y') = sc_l(y') + 1$ ; } }
  forall (( $c, d$ )  $\in D$ ) {
    if ( $sc_l(d) > sc_l(getFirst(results))$ ) {
       $cands \leftarrow (c, d)$ ;
       $results \leftarrow d$ ; } } }

```

図9 同義語候補抽出アルゴリズム

先度も高くなるよう定義する．現在最も有望な（スコアの高い）候補 x の子ノードを探索し，現在保持されている最低スコアの候補を上回る候補のみを新たに $cands$ に追加する．

こうして得られた $2N_2$ 個の候補を，最終的に，左文脈・右文脈を同時に用いたスコア関数

$$sc(x) = \sum_{c \in C_l} \log \frac{freq(c, x)}{freq_{exp}(c, x)} + \sum_{c \in C_r} \log \frac{freq(x, c)}{freq_{exp}(x, c)}$$

により順位づけする（図8参照）．ここで $freq_{exp}$ は，

$$freq_{exp}(c, x) = freq(x) \cdot \frac{freq(c)}{|S|}$$

で定義される，「 c と x が無関係だった場合の x の予測頻度」をモデル化した値である．「 c と x が実際に隣接した回数」と「 c と x が独立だった場合に隣接する回数」の比を計算するという点で，相互情報量に類似したスコア関数となっている．

i. List Cleaning

文脈リストや候補語リストは，重複した文字列を多く持つ．例えば，「研究に関する」と「に関する」といった複数の文字列が同一のリストに入っている場合がある．我々は，このような冗長性を排除するため，得られた文脈リストおよび候補語リストに対し，List Cleaning という後処理を行った．List Cleaning では， n_1 番目の要素が n_2 番目の要素の部分文字列だった場合， $\max\{n_1, n_2\}$ 番目の要素（より順位の低い要素）をリストから削除する*8．例えば，[...，“on the other hand”，..., “on the”，...] というリストがあった場合，要素 “on the” が削除される．予備実験では，文脈リストの List Cleaning により，実行時間を短縮するだけでなく，得られた結果も改善することが確

認された．文脈の冗長性は，スコア計算の際に重複した文脈に関する部分を多くカウントしてしまうという不均一性を生み出すため，スコアに悪影響を与えるためだと推測される．また，候補語リストに対する List Cleaning でも，結果が改善される現象が観察された．

4. 応 用 例

本研究のアルゴリズムを実際に応用したいいくつかのシステムの例を示す．いずれのシステムも，各クエリに対して数秒の応答時間で同義語候補を返し，実用上問題のない程度の実行時間となっている（例えば，図10の例は約3秒，図11の例は約2秒で得られた）．後述の航空分野のレポートと比較して，文書サイズが大きいにもかかわらずそれと遜色のない速度が得られているが，これは，アルゴリズムが使用する文脈数（この場合，最大 N_2 個）および文脈の頻度（この場合，最大 F_1 ）を限定しており，また，Suffix Array の一回の検索時間は文書長 $|S|$ に対し $\log |S|$ のオーダーであるため，コーパスサイズが増加しても，パラメータが変化しなければ抽出時間には大きな影響がないためである（実行時間の多くは同義語候補の探索（STEP-3）によって占められており，STEP-3の実行時間は主に文脈数と文脈頻度に依存するため）．なお，パラメータ F_1 は，頻度に対する閾値であるため，コーパスのサイズが大きくなるにつれ，適切な値も増加する．しかしながら，各文字列頻度がコーパスサイズに比例して増加するということには必ずしもならない．何故なら，コーパスのサイズが増加し新たな文書が追加された場合，新たな文書の中には，既存の文書にはない新たな言葉が現れるからである．また，同一のサイズのコーパスであっても，コーパス内文書の一貫性（すなわち，コーパス内で用いられる文字列の重複しやすさ）により，適切な F_1 の設定も変化することになり，実際には F_1 等のパラメータ値はコーパスによってその都度変化させることが望ましい．このことから，本システムでは， F_1 等の値を，コーパスサイズ等を指数とした関数としてではなく，ユーザーが設定可能な固定値として定義している．この場合，後述の航空分野レポートが分野の限定されたコーパスであったのに対し，Wikipedia や東京大学 Web サイトがそれよりも遥かに多様なトピックを含むため，各文字列の頻度がそれほど大きくなり， N_1 や F_1 の値が航空分野レポートの際の値と比べそれほど大きくなくても同義語・類義語が返せっていると推測される．

4.1 Wikipedia 検索支援システム

本研究のアルゴリズムを用い，Wikipedia の検索を支援するシステムを試作した．データとしては，Wikipedia の2007年5月29日版のスナップショットから，<text>タグで囲まれた部分を取得した．取得したテキストはそのままコーパスとして用い，それ以上の後処理（タグの除

*8 List Cleaning により，実際に得られるリストのサイズが，パラメータ N_1 や N_2 よりも小さくなることに注意されたい．

図 10 Wikipedia 検索支援システム

図 11 東京大学 Web ページ検索支援システム

去等)は行っていない。これは、Wikipediaのタグやレイアウト情報が、同義語・類義語を発見するうえでの重要な文脈情報として機能するとの観察に基づいた措置である。取得されたテキストファイルは926MBとなった。実行は、AMD Opteron 248 (2.2GHz) + 13GB メモリのマシン上でっており、パラメータは、 $F_1 = N_1 = N_2 = 1000$ と設定している。

図10にスクリーンショットを示す。検索窓にクエリを入力すると、リアルタイムで同義語を検索し、類似度の順に並べて提示する。図では「マンション」というクエリを入力した際、その同義語として「住宅」や「アパート」という文字列が用いられやすいということを提示している。ユーザは、提示された同義語候補をクリックすることで、クエリを変更することができる。変更された結果は、そのままGoogleへのクエリとして、検索に用いることができる。

4.2 東京大学 Web ページ検索支援システム

同様に、特定組織のWebページ集合の例として、東京大学のWebサイトから収集したWeb文書(HTMLタグ除去後、727.0MBのテキストファイル)に対し、検索支援を行うシステムを作成した。実行環境は、Xeon プロセッサ(2GHz)2個 + 16GB メモリのマシンである(パラメータ: $F_1 = N_1 = N_2 = 1000$)。図11に実行例を示す。「シンポジウム」等、大学で頻繁に使われる言葉に対し、多くの同義語が取得できることが確認できる。これにより、例えば「シンポジウム OR ワークショップ」のような、検索の網羅性向上に役立てることができる。

4.3 同義語辞書作成支援ツールと青空文庫での実行例

また、同義語辞書作成支援ツールとして、任意の文書集合を与えることにより、提案手法を用いてクエリの同義語を検索できるようにするツールを開発した。例として、青空文庫の全文書(128.9MBのテキストファイル)

図 12 青空文庫を対象とした同義語辞書作成支援。左右の窓は、使用された文脈文字列を示す。

を利用した例を図12に示す。実行環境は、Intel Core Solo U1300 (1.06GHz) プロセッサ + 2GB メモリである。(パラメータ: $F_1 = N_1 = N_2 = 500$)。「自動車」に対して「馬車」「人力車」「テーブル」に対し「デエブル」「卓子」等、文学作品に特徴的な言い回しが取得できている。

4.4 英語での実行例

提案手法は、隣接文字列のみを文脈情報として用いるため、特別な前処理を行わずに、そのまま様々な言語へ適用できる。英語クエリに対する抽出例として、Reuterコーパスから一部(487 MBytes)を取り出し、コーパスとして用いた。Suffix Arrayの索引付けのみを用いるという提案手法の方針から、空白除去や単語境界認定等の前処理は行っていない。(実行環境は、Wikipedia 検索支援システムと同様。)図13に、いくつかの抽出例を示す。略語“U.S.”から“United States”が、縮約形“doesn't”から“does not”が取得できる等、文法カテゴリに依らない抽出ができることが確認できる。

5. 実験

本研究で提案する同義語抽出アルゴリズムに関し、実験を行った。実験は(株)日本航空インターナショナルの所蔵する航空安全情報文書(7Mbytes)と、それに付随する同義語辞書を用い、「辞書に登録された語に対し、同義語を検索する」という問題設定により行った。同義語辞書は、見出し語 t と、その同義語集合 $S(t)$ のペア $(t, S(t))$ の集合であり、例えば (CAPT, {Captain}), (T/O, {Takeoff})

Query: likely to
Results: 1.will / 2.expected to / 3.going to / 4.ould / 5.able to

Query: U.S.
Results: 1.US / 2.Australian / 3.United States / 4.American / 5.Canadian

Query: doesn't
Results: 1.does not / 2.didn't / 3.would / 4.did not / 5.could

図 13 Reuter コーパス上での同義語取得例

のように定義される．同義語辞書の作成は，コーパス中に 100 回以上出現する名詞から選んだ 404 個の単語を，人手でクラスタリングすることにより作成した．見出し語の数は 404，平均同義語数は 1.92 であった．システム実行環境は，Intel Core Solo U1300 (1.06GHz) プロセッサと 2GB メモリである．

結果の評価には，平均精度 [Chakrabarti 02] を用いた．クエリ t に対するシステムの出力リストを $\langle c_1, c_2, \dots, c_n \rangle$ としたとき，平均精度は，

$$\frac{1}{|S(t)|} \sum_{1 \leq k \leq n} r_k \cdot \text{precision}(k),$$

と計算される ($\text{precision}(k)$ は，上位 k 個を見た時の正解率．また， r_k は， $c_k \in S(t)$ (c_k が正解) なら 1，そうでなければ 0 をとる変数)．これは，各正解語の出現において，それより上の順位で出力された語を全出力と見なしたときの正解率 (精度) を測定し，その平均をとったものであり，最高値は 1.0 となる．

5.1 パラメータの変動による速度・精度の変化

まず，パラメータの値の変動による速度と精度の変化について調査した． N_1 に関しては，予備実験で値による変化が小さかったため，1000 に固定し， N_2 と F_1 の値の変動について調査した．表 2 と表 3 に結果を示す．文脈文字列頻度の上限値 F_1 を設定することにより，上限値を設定しなかった場合 ($F_1 = \infty$) に比べ，速度だけでなく，精度も向上することがわかった．それ以外の場合は，どちらのパラメータに関しても，1,000 までは値が大きくなるほど精度が上がり，その分速度が下がっているが， N_2 を 3,000 まで増やすと精度は低下した． N_2 を必要以上に増加させると，スコアの低い，あまり有用でない文脈が利用されるようになり，そのことが却って

表 2 パラメータの変動による実行時間 (/ミリ秒) の変化

Execution Time				
$F_1 \setminus N_2$	100	300	1,000	3,000
100	725.5	847.0	1070.0	2555.3
300	863.7	1034.1	1359.0	2499.6
1,000	1098.6	1403.4	1963.4	3223.6
∞	2635.9	3849.4	6550.6	12211.2

精度を落としてしまっていると推測される．最高精度は， $F_1 = N_2 = 1000$ の設定で得られ，クエリ毎の平均実行時間は 1.96 秒であった．

5.2 一般的手法との比較

提案手法で得られる同義語リストの精度を，一般的に用いられる同義語抽出手法による出力リストと比較する．一般的に，同義語抽出は，あらかじめ与えられた同義語候補リストの各ペアの間で類似度を測定し，与えられた単語に対して，他の単語を，単語間の類似度の順に並べることで行われる．我々は，各単語ごとに，単語の文脈の抽出を，「周辺単語」[Hagiwara 06] と「構文構造」[Murakami 04] の抽出により行い，各文脈が一つの次元に対応するベクトルとして表現した (ベクトル空間モデル，以下 VSM)．本実験では，単語類似度の測定は，単語ベクトル同士の余弦を計算することにより行う (「コサイン類似度」)，標準的な手法を用いた．また，ベクトルの各次元の重みづけについては，頻度 (TF)，tf-idf 値 (TF-IDF) という一般的な手法と，頻度の対数をとる手法 ($\log \text{TF}$)*⁹ を用いた．

VSM の場合，事前に類似度を測定するための同義語候補を与えなければならない．このため，精度の比較は，各クエリ毎に，「同義語候補リストが与えられたとき，それを『同義語らしさ』の順に並べ替える」という「候補語ランキング」のタスクを用いて行った．本実験では，同義語候補を，同義語辞書の見出し語 (404 個) とした．提案手法のパラメータ設定は，最も精度の良かった $F_1 = 1000, N_2 = 1000$ を採用した．また，本比較実験においては，提案手法の出力するリスト*¹⁰のうち，候補リスト

表 3 パラメータの変動による平均精度の変化

Average Precision (%)				
$F_1 \setminus N_2$	100	300	1,000	3,000
100	32.02	31.71	30.60	29.31
300	34.52	37.18	36.31	35.22
1,000	33.69	38.73	40.32	38.64
∞	22.01	28.47	31.40	31.74

*9 実際には，頻度 1 の場合を考慮し， $\log(1 + tf)$ (tf は頻度) という式で重みを定義する．

*10 候補語ランキングタスクにおいては，候補語の List Clearing は行っていない．

に入っているもののみをランキングとして出力する（出力されなかった候補語は本来はリストの最後に入ることになるが、本実験では入れていない．平均精度の定義により、これにより提案手法に有利な値となることはなく、また、リストの下位の結果が精度に与える影響は非常に小さいことに注意されたい．）このため、提案手法の値は、表 3 よりも上昇することに注意されたい．

表 4 に結果を示す．VSM に関しては、[Hagiwara 06] にも述べられている通り、周辺単語の情報が有用であり、それと構文構造を併用することにより、さらに精度が向上するという結果が得られた．また、重みづけ手法の中では、logTF が他の 2 つにくらべ非常に高い精度を示した．これは、同義語抽出タスクにおいて、文脈の頻度よりも、文脈の種類（「多くの種類の文脈が共有されているほど、同義語の可能性が高い」）が重要であるということを示唆している．VSM の間の最高精度が 57.35 ポイント（最低で 23.25 ポイント）だったのに対し、提案手法の精度は 52.20 ポイントであった．提案手法は、精度的に標準的な手法には及ばないものの、最高精度に近い値を得ていることにより、ある程度の性能を達成していると言える．前述の通り、提案手法は、接続文字列のみを同義語抽出のための文脈として用いているのに対し、VSM では、接続しない周辺単語や、構文構造まで文脈として用いているため、より高い精度が出ると考えられ、比較結果はこれを裏付けるものとなっている．また、前節の結果より（表 3）、これはパラメータの値を増加させることによって解決できるわけではなく、より精度を上昇させるためには、接続文字列以外の文脈を Suffix Array で検索できるようにする必要があると考えられる．

また、提案手法で獲得した同義語候補リストの中に全く含まれない（順位を問わず、リスト中にまったく存在しない）正解がどれくらいの割合で存在するかを調査した．全正解同義語 774 個中、提案手法で取得できたのは、List Cleaning を経なかった場合 680 個であり、同義語候補リストへの List Cleaning を行うと 550 個となった．順位を無視した再現率（同義語が、出力リストの中に含まれていた割合）は、List Cleaning 無しの場合 87.9 %、List Cleaning 有りの場合は、71.1 %であった．提案手法の「予め同義語候補が与えられていない状態で、同義語候補を検索する」というアプローチを用いても、8 割 5 分以上の同義語候補を取得できているが、List Cleaning により、

表 4 候補ランキングタスクによる平均精度の比較 (%)

アルゴリズム			
提案手法	52.20		
	構文構造	周辺単語	組み合わせ
VSM (logTF)	34.43	55.61	57.35
VSM (TF-IDF)	24.72	37.85	39.19
VSM (TF)	23.25	40.12	40.87

図 14 文脈数と候補語取得時間の関係．横軸が文脈数，縦軸が候補語取得時間 (/ミリ秒)

図 15 文脈数と平均精度の関係．横軸が文脈数，縦軸が平均精度．

その率が 7 割にまで低下していることがわかった．候補語の List Cleaning により精度は改善されるが（実験では、List Cleaning 有りで 40.32 だった精度が、List Cleaning 無しでは 26.71 に低下した）、その再現率に関してはまだ改善の余地があることがわかる．

5.3 文脈数と、その実行時間との関係

文脈文字列リストに対する List Cleaning の効果を見るため、List Cleaning 後の各文脈文字列の種類数を測定した．パラメータは $N_1 = 1000$ の場合であり、List Cleaning 前は左文脈、右文脈とも 1000 個となっている．表 5 に結果を示す．左文脈数、右文脈数ともに平均 60 強に減っており、このことは、実際の文脈検索においては、本来必要な文脈の他に、それに類似した多くの文脈が取得されていることを示している．図 14 に、文脈数（合計）と、同義語候補取得 (STEP-3) にかかる時間の関係を示す．文脈数と実行時間がほぼ比例し、文脈数 100 以下では 1 秒以下の実行時間であるのに対し、文脈数が 400 以

表 5 List Cleaning 後の平均文脈数

左文脈	右文脈	合計	実行時間
60.89	64.13	125.0	1554.5

上となると最大で 8 秒の検索時間を要していることがわかる。このことより、実用に耐えうる応答時間の実現のためには、実際に使われる文脈数を 200 個程度に抑えることが必要であると言える。また、図 15 に、文脈数と平均精度（候補語ランキングタスク時）の関係を示す。相関係数は-0.088 であり、明確な相関関係は見られなかった。すなわち、List Cleaning によって大幅に文脈数が減った場合（重複する文脈が多かった場合）と、それ程文脈数が減らなかった場合（重複する文脈が少なかった場合）でも、前者の精度が明確に低いということはなかった。このことから、文脈数は必ずしも多い必要はなく、現在文脈数が多いクエリに対しても、適切な文脈数を設定できれば、それ程精度を下げずにより平均的な実行時間を上げることができると考えられる。このことを確認するため、List Cleaning 後の文脈数に制限を加えた場合の、精度と実行時間について実験を行った。結果を表 6 に示す。結果として、文脈数を 50 程度にまで制限した場合でも、それ程精度は低下せず、実行時間を約 74 % に削減できた。

6. 関 連 研 究

同義語抽出の先行研究には、依存構造解析を行い、単語の依存関係を文脈として用いる手法 [Yamamoto 02] や、動詞-目的語関係を抽出し、それを文脈として用いる手法 [Gasperin 01, Murakami 04] 等がある。また、文書外のリソースを用いる手法としては、辞書や対訳コーパスを文脈として用いる手法 [Wu 03, Plas 06] がある。これらを文脈情報として利用することにより、より精度の向上が期待できるため、これらの情報を我々の Suffix Array ベースの同義語抽出にどのように取り入れるかは今後の重要な研究課題である。また、柴田ら [柴田 09] は、ウェブコーパスに対して同義語抽出を行い、コーパスサイズを大規模にすることによって大きく精度が向上することを報告している。提案手法の現状の実装では、Suffix Array のメモリ効率については特に工夫を行っていないが、Compressed Suffix Array [Grossi 00] のようなメモリ効率の高い実装方法を用いて現実的な抽出速度を達成し、より大規模なコーパスに対しても本手法を適用可能にすることも重要な将来課題である。

与えられたクエリに対し、事前に候補を指定せず（全

文字列を候補として）、動的に同義語を取得する先行研究は、筆者らの知る限りでは存在しないが、近年、萩原らは、bootstrap 法による同カテゴリ語の動的獲得を、単語分かち書きを用いず文字列文脈のみを用いて行う手法を提案している [萩原 09, Hagiwara 08]。萩原らの手法はオフラインでの情報抽出を想定しており、実行速度に関してそのまま検索支援に用いることはできないが、ここで用いられているグラフカーネルを用いた同カテゴリ語発見手法を応用することで、我々の手法を、同カテゴリ語発見のために拡張できる可能性がある。

7. お わ り に

特定の文書集合を対象とした検索支援を主な目的とした、動的同義語抽出アルゴリズムを提案した。提案アルゴリズムは、任意のクエリ入力に対し同義語を抽出できることが特徴である。7Mbytes のコーパスに対する同義語抽出実験では、1 クエリ当たりの応答時間は約 2 秒であり、抽出精度は、一般的に用いられるアルゴリズムと比較し、やや劣る値であった。今後は、より大規模なコーパスでの実験および高速化を図る予定である。

謝 辞

本研究の一部は、文部科学省科学研究費補助金特定領域研究「情報爆発時代に向けた新しい IT 基盤技術の研究」および、経済産業省「情報大航海プロジェクト」の支援を受けて行われた。

◇ 参 考 文 献 ◇

- [Chakrabarti 02] Chakrabarti, S.: *Mining the Web : Discovering Knowledge from Hypertext Data*, Morgan-Kaufmann Publishers (2002)
- [Collins 00] Collins, M.: Discriminative Reranking for Natural Language Parsing, in *Proceedings of the Seventeenth International Conference on Machine Learning*, pp. 175–182 (2000)
- [Gasperin 01] Gasperin, C., Gamallo, P., Agustini, A., Lopes, G. P., and Lima, de V.: Using Syntactic Contexts for Measuring Word Similarity, in *Proceedings of the ESSLLI'01 Workshop on Semantic Knowledge Acquisition and Categorization* (2001)
- [Gorman 06] Gorman, J. and Curran, J. R.: Scaling distributional similarity to large corpora, in *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pp. 361–368 (2006)
- [Grossi 00] Grossi, R. and Vitter, J.: Compressed Suffix Arrays and Suffix Trees with Applications to Text Indexing and String Matching, in *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing*, pp. 397–406 (2000)
- [Hagiwara 06] Hagiwara, M., Ogawa, Y., and Toyama, K.: Selection of Effective Contextual Information for Automatic Synonym Acquisition, in *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics (COLING/ACL 2006)*, pp. 353–360 (2006)
- [Hagiwara 08] Hagiwara, M., Ogawa, Y., and Toyama, K.: Bootstrapping-based Extraction of Dictionary Terms from Unsegmented Legal Text, in *Proceedings of the Second International Workshop on Juris-informatics (JURISIN 2008)*, pp. 63–72 (2008)

表 6 List Cleaning 後の文脈数制限による精度と速度の変化

文脈数	実行時間（ミリ秒）	平均精度（%）
10	871.46	30.94
30	1224.1	38.69
50	1445.7	40.06
100	1724.3	40.43
300	1947.3	40.31
1000（無制限）	1963.4	40.32

- [Harris 54] Harris, Z.: Distributional structure, *Word*, Vol. 10(23), pp. 146–162 (1954)
- [Manber 90] Manber, U. and Myers, G.: Suffix Arrays: A New Method for On-line String Searches, in *Proceedings of the first ACM-SIAM Symposium on Discrete Algorithms*, pp. 319–327 (1990)
- [McCreight 76] McCreight, E. M.: A Space-Economical Suffix Tree Construction Algorithm, *Journal of the ACM (JACM)*, Vol. 23(2), pp. 262–272 (1976)
- [Murakami 04] Murakami, A. and Nasukawa, T.: Term aggregation: mining synonymous expressions using personal stylistic variations, in *Proceedings of the 20th International Conference on Computational Linguistics (COLING-04)* (2004)
- [Pasca 05] Pasca, M. and Dienes, P.: Aligning Needles in a Haystack: Paraphrase Acquisition Across the Web, in *Proceedings of the 2nd International Joint Conference on Natural Language Processing (IJCNLP-05)*, pp. 119–130 (2005)
- [Plas 06] Plas, van der L. and Tiedemann, J.: Finding synonyms using automatic word alignment and measures of distributional similarity, in *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistic (COLING/ACL 2006) Main conference poster sessions*, pp. 866–873 (2006)
- [Shimohata 05] Shimohata, M. and Sumita, E.: Acquiring Synonyms from Monolingual Comparable Texts, in *Proceedings of the 2nd International Joint Conference on Natural Language Processing (IJCNLP-05)*, pp. 233–244 (2005)
- [Wu 03] Wu, H. and Zhou, M.: Optimizing synonym extraction using monolingual and bilingual resources, in *Proceedings of the Second International Workshop on Paraphrasing: Paraphrase Acquisition and Applications (IWP 2003)* (2003)
- [Yamamoto 02] Yamamoto, K.: Acquisition of lexical paraphrases from texts, in *Proceedings of the 2nd International Workshop on Computational Terminology (COMPUTERM 2002)*, pp. 1–7 (2002)
- [柴田 09] 柴田 知秀, 黒橋 禎夫: 超大規模ウェブコーパスを用いた分布類似度計算, 言語処理学会第 15 回年次大会発表論文集, pp. 705–708 (2009)
- [萩原 09] 萩原 正人, 小川 泰弘, 外山 勝彦: グラフカーネルに基づく非分かち書き文からの意味的語彙カテゴリの抽出, 言語処理学会第 15 回年次大会発表論文集 (2009)

〔担当委員: 小林 一郎〕

2009 年 4 月 30 日 受理

著者紹介

吉田 稔 (正会員)

1998 年東京大学理学部情報科学科卒業。2003 年東京大学大学院理学系研究科情報科学専攻博士課程修了。博士(理学)。2003 年より東京大学情報基盤センター図書館電子化研究部門助手。2007 年より同助教。自然言語処理, Web 文書解析の研究に従事。

中川 裕志 (正会員)

1980 年東京大学大学院工学系研究科博士課程修了。工学博士。横浜国立大学を経て, 1999 年より東京大学・情報基盤センター・教授(情報理工学系研究科数理情報学専攻を兼任)。人工知能, 自然言語処理, 統計的機械学習, 情報ナビゲーションの研究に従事。

寺田 昭

1976 年京都大学工学部電気工学第二学科卒業。1978 年同大学大学院工学研究科修士課程電気工学第二専攻修了。2003 年東京工業大学大学院情報理工学研究科後期博士課程修了。博士(工学)。現在(株)日本航空インターナショナル勤務。自然言語処理, テキストマイニング, 航空安全に興味を持つ。言語処理学会会員。