RF-009

文節データベースを用いた日本語アナグラムの自動生成

Automatic Generation of Japanese Anagrams by Using Bunsetsu Database

鈴木啓輔† 佐藤理史† 駒谷和範†
Keisuke Suzuki, Satoshi Sato and Kazunori Komatani

1. はじめに

西欧で発達したことば遊びの一つにアナグラムがある。『図説ことばあそび遊辞苑』[1] では、アナグラムを「ある語句のつづりを一字ずつに分解し、そのすべてを使って、まったく別の語句に仕立てるもの (p103)」と説明している。

日本語は、使用する文字の数が多いため、アナグラムの作成にはあまり適していない。しかしながら、名称などの短い語句に対しては、アナグラムの作成は比較的容易であり、たとえば、本名のアナグラムを、芸名やペンネームとして用いる例も見受けられる。これに加え、「いろは歌」に代表される無同字歌 (歌俳や詞章などで同じ字音を二回使わずにつづったもの [1]) も、一種のアナグラムとみなすことができる。

本論文では、西欧アナグラムの流れを汲む日本語の「読みアナグラム」を自動生成する問題を取り上げる。日本語アナグラムの自動生成に関するこれまでの試み [2][3] は、いずれも生成能力に大きな制限があり、この問題は、ほぼ未開拓の問題であると言ってよい。

人間によるアナグラム作成が、パズル的な娯楽を目的としているように、アナグラム自動生成の研究も、娯楽的な知的興味に基づくものである。その一方で、アナグラム自動生成の問題を、「文法的に適格で、かつ、意味がとれる (通じる) 日本語表現を生成する問題」とみなせば、言語の研究、あるいは、言語処理の研究としての意義を持つ。文法的適格性・意味的整合性の研究は、チョムスキー以来、長い研究の歴史があるが、いまだに多くの問題が未解決である。本論文では、日本語アナグラム生成の基本アルゴリズム、および、そのアルゴリズムと文節データベースを組み合わせたアナグラム生成システムを示し、現時点での到達点を明らかにする。

本論文の構成は、次のとおりである。まず、2 節で日本語の「読みアナグラム」生成問題を形式的に定義する。次に、3 節でアナグラム生成の基本アルゴリズムを提案する。4 節では、このアルゴリズムと文節データベースを用いたアナグラム生成システムについて述べる。5 節ではアナグラム生成例を示し、6 節で現時点での到達点について検討する。

† 名古屋大学大学院工学研究科電子情報システム専攻

2. 問題の定式化

本節では、日本語の「読みアナグラム」生成問題を形式的に定義する。「読みアナグラム」とは、「名古屋大学 (なごやだいがく)」と「長い誤訳だ (ながいごやくだ)」のように、読み (かな表記) がアナグラムの関係となっているものを指す。

まず、日本語表記 (文字列)J に対し、読み (かな表記)K を定める [4]。日本語表記 J に対する読みは、必ずしも一意に定まるとは限らない。たとえば、「人気」には、「にんき」と「ひとけ」の 2 つの読みが存在する。このような問題を回避するために、日本語表現を、日本語表記とその読みの組 $\langle J, K\rangle$ として定義する。

次に、ある文字列 X に対して、その文字列を構成する文字の順序を並べ替えた文字列の集合 $\mathrm{perm}(X)$ を定義する。たとえば、$X = abc$ に対しては、

$$\mathrm{perm}(abc) = \{abc, acb, bac, bca, cab, cba\} \quad (1)$$

となる (自分自身を含む)。なお、文字列の長さが n で、文字列を構成する文字がすべて異なる場合、上記の集合の要素の数は $n!$ となる。

以上の準備に基づき、ある日本語表現 $\langle J_s, K_s\rangle$ のアナグラムの集合を次のように形式的に定義する。

$$\mathrm{anag}(\langle J_s, K_s\rangle) = \{\langle J_a, K_a\rangle | K_a \in \mathrm{perm}(K_s) \ \wedge \ \mathrm{valid}(\langle J_a, K_a\rangle)\} \quad (2)$$

ここで、$\mathrm{valid}(\langle J_a, K_a\rangle)$ は、日本語表現 $\langle J_a, K_a\rangle$ の妥当性を判定する論理関数とする。

上記の式において、J_s は右辺には現れない。すなわち、J_s は入力として必須ではない。そこで、これを省略すると、次の式が得られる。

$$\mathrm{anag}(K_s) = \{\langle J_a, K_a\rangle | K_a \in \mathrm{perm}(K_s) \ \wedge \ \mathrm{valid}(\langle J_a, K_a\rangle)\} \quad (3)$$

この式が、我々が定義する「読みアナグラム生成問題」である。すなわち、入力はかな表記 K_s、出力は日本語表現の集合である。

さて、この式の値を求めるためには、次の 2 つの計算が必要である。

$$K_a \in \mathrm{perm}(K_s) \quad (4)$$

$$\mathrm{valid}(\langle J_a, K_a\rangle) \quad (5)$$

式 (4) の右辺を計算する素朴なアルゴリズムは単純である。ただし、$\mathrm{perm}(K_s)$ の要素数は、K_s の長さが長く

```
1  def permutation(str, suf='')
2    if str.length == 0
3      print suf, "\n"
4    else
5      used=Hash.new
6      0.upto(str.length-1) do |i|
7        c = str[i,1]
8        if !used[c]
9          used[c] = true
10         str2 = str.dup; str2[i] = ''
11         permutation(str2, c+suf)
12       end
13     end
14   end
15 end
```

図 1　並べ替え文字列を列挙する Ruby プログラム

なると急速に増加するので、実際には何らかの工夫が必要となる。

アナグラム生成研究の本質は、式 (5) をどのように実装するかという点にある。一般に、アナグラムは、語、句、文の一部、文など色々な形式を取りうるため、その文法的適格性を判定する方法は自明ではない。加えて、自然言語を対象とした現在の構文解析技術は、与えられた文に対して、その構文構造を出力することを意図しており、それをそのまま、文の適格性の判定に用いることはできない。さらに、意味的整合性の判定は、現在の言語処理技術の射程に入っていない。つまり、アナグラムの生成の研究は、これらの新しい問題への挑戦という意味合いを持つ。

3. アナグラム生成の基本アルゴリズム

3.1 アルゴリズムの骨格

式 (3) を計算するために、$\mathrm{perm}(K_s)$ をジェネレータとして用いる。図 **1** に、文字列の並べ替えをすべて列挙する Ruby プログラムを示す。このプログラムは、並べ替え文字列を、後ろから前へ伸ばす形で作成する。第 2 引数は、途中まで作成された並べ替え文字列であり、第 1 引数は、残っている (これから使用するべき) 文字列を表す。アルゴリズムの中核は、第 1 引数から 1 文字を選んで取り除き、それを作成途中の並べ替え文字列の先頭に追加して、自分自身を再帰的に呼び出す処理 (6–13 行目) である。

完成した並べ替え文字列は、3 行目で出力される。効率のことを考えないのであれば、この部分に、(1) かな表記 K(=`suf`) からの日本語表記 J の生成と、(2) それによって得られる日本語表現 $\langle J, K\rangle$ に対する妥当性の判定を追加すれば、アナグラム生成アルゴリズムは完成する。

3.2 基本単位集合を利用したアナグラム生成

図 1 のプログラムにおいて、探索における分岐は、6 行目から 13 行目の do ループにおいて発生する。分岐先に解がないことが判明すれば、その時点で枝刈りをすることができ、探索が効率化する。たとえば、「なごやだいがく」からのアナグラム生成では、「〜だご」となる日本語表現が存在しなければ、その時点で、その分岐は枝刈りできる。

このような枝刈りは、可能な日本語表現を厳密に定義することによって実現できる。日本語表現の集合 (すなわち、言語) を定義する最も単純な方法の一つは、形式言語理論に基づく次のような定義である。

(1) 日本語の基本単位の集合 $\mathcal{U}$ を列挙により定義する。この集合の要素を $u = \langle J, K\rangle$ と表すことにする。

(2) 上記の集合の要素の列を、可能な日本語表現と定義する。すなわち、

$$\langle J', K'\rangle = u_1 u_2 \cdots u_n \quad (u_i \in \mathcal{U}) \tag{6}$$
$$= \langle J_1, K_1\rangle\langle J_2, K_2\rangle \cdots \langle J_n, K_n\rangle \tag{7}$$
$$= \langle J_1 J_2 \cdots J_n, K_1 K_2 \cdots K_n\rangle \tag{8}$$

次節において、我々は、この基本単位として「文節」を採用するが、「形態素」や「語」などの他の単位に対しても、本節の議論は成り立つ。

集合 $\mathcal{U}$ に対して、次の 2 つの関数を実現する。

(1) members(S,$\mathcal{U}$): 集合 $\mathcal{U}$ 中に含まれる、文字列 S がかな表記と一致する要素を返す。

(2) suf_member(S,$\mathcal{U}$): 集合 $\mathcal{U}$ 中に、文字列 S がかな表記のサフィックスとなっている要素が存在するかどうかを調べる。

これらの関数は、次のように形式的に定義できる。

$$\mathrm{members}(S,\mathcal{U}) = \{u | u = \langle *, S\rangle \in \mathcal{U}\} \tag{9}$$

$$\mathrm{suf\_member}(S,\mathcal{U}) = \begin{cases} \text{true} & \text{if } \langle *, PS\rangle \in \mathcal{U} \\ \text{false} & \text{otherwise} \end{cases} \tag{10}$$

ここで、$*$ は任意の文字列、P は 1 文字以上の任意の文字列を表すものとする。

サフィックスとは、ある文字列から先頭の n 文字を削除することによって得られる文字列をいう。一般には、文字列それ自身も、その文字列のサフィックスに含める (つまり、$n \geq 0$) のが普通であるが、ここでは、文字列自身は、その文字列のサフィックスとはみなさない (つまり、$n > 0$) こととする。たとえば、文字列 abc のサフィックスは、bc と c のみとする。

これらの 2 つの関数を用いて、図 1 のプログラムを図 **2** のように変更する。このプログラムは、アナグラムを、基本単位の列 `ul` として生成する。これを実現するために、再帰呼び出しを 2 つに分けている。1 つ目の再帰呼び出し (11.3 行目) は、基本単位が新たに完成した場合に実行する。この呼び出しに必要な基本単位 `u` は、関数 members が供給する。もう 1 つの再帰呼び出し (11.6 行目) は、`suf2` が基本単位のサフィックスとなっていた場合に実行する。いずれの再帰呼び出しも、条件を満たす場合にのみ実行され、見込みのない分岐は生成しない (枝刈りされる)。なお、このプログラムでは、基本単位集合 $\mathcal{U}$ を `dic` で表している。

```
 1  def anagram(dic, str, suf='', ul=[])
 2    if str.length == 0
 3      anagram_output(ul) if suf == ''
 4    else
 5      used=Hash.new
 6      0.upto(str.length-1) do |i|
 7        c = str[i,1]
 8        if !used[c]
 9          used[c] = true
10          str2 = str.dup; str2[i] = ''
11.1        suf2 = c+suf
11.2        members(suf2, dic).each do |u|
11.3          anagram(dic, str2, '', [u]+ul)
11.4        end
11.5        if suf_member(suf2, dic)
11.6          anagram(dic, str2, suf2, ul)
11.7        end
12        end
13      end
14    end
15  end
```

図 **2**　基本単位集合を利用したアナグラム生成

3.3　付加的な制約の導入

自然言語では、一般に、基本単位のあらゆる連接が、文法的に許容されるわけではない。特に、基本単位として語を採用した場合は、このことは顕著となる。たとえば、動詞の未然形の直後に接尾辞「ない」は接続するが、動詞の基本形の直後には「ない」は接続しない。このような制約を組み込むことにより、文法的に許容されない表現の生成を抑制することができる。

付加的な制約のチェックは、アナグラム生成の最終段階 (図 2 の 3 行目) のみで行なうよりも、基本単位が新たに生じた時点 (11.3 行目) で行なう方が、効率の観点から好ましい。これを実現するために、次のような関数 connectable を導入する。

connectable(u, ul): 基本単位の列 ul の左側に、
基本単位 u が接続可能かどうかを判定する。

この関数を用いて、図 2 のプログラムの一部を図 **3** のように変更する。

さて、ここまでで、論理関数 valid がどのように組み込まれたかを整理しておこう。

(1) 出力される日本語表現 $\langle J_a, K_a \rangle$ は、基本単位 $\mathcal{U}$ の要素の列である。すなわち、

$$\langle J_a, K_a \rangle = u_1 u_2 \cdots u_n \text{ where } u_i \in \mathcal{U} \quad (11)$$

(2) 基本単位 u_i は、後続する基本単位列に対して接続可能である。つまり、$i = 1, 2, \cdots, n$ に対し、

$$\text{connectable}(u_i, u_{i+1} \cdots u_{n+1}) = \text{true} \quad (12)$$

ただし、u_{n+1} は仮想的な終了記号とする。

4. 文節データベースを用いたアナグラム生成

前節のアルゴリズムにおいて定まってない部分は、何を基本単位として採用するか、制約 connectable をどのように実装するか、という 2 点である。我々は、アナグラム生成においては、基本単位として、形態素や語ではなく、文節を採用することが重要と考える。その理由は、次のとおりである。

```
11.2    members(suf2, dic).each do |u|
11.31     if connectable(u, ul)
11.32       anagram(dic, str2, '', [u]+ul)
11.33     end
11.4    end
```

図 **3**　付加的な制約の導入

(1) 日本語の文法的制約において、文節内における形態素の連接制約は相対的に強い。つまり、文節として認めうる形態素列は、文法的に限定されるため、あらかじめ可能な文節集合を作成しておくことができれば、事前計算の効果が期待できる。

(2) 日本語の文節間の係り受け制約は相対的に弱い。つまり、アナグラム生成時には、複雑な制約チェックを行なわなくてよい。

(3) より大域的な文法制約や意味的整合性のチェックを導入する場合、文節という単位を認定することが不可欠である。文節を基本単位とすれば、文節認定は不要 (自明) となる。

本節では、我々が作成したアナグラム作成システムで用いる文節データベースと、文節列に対する制約の実装について述べる。このうち、前者は、我々が回文の自動生成 [5] で用いた文節データベースと同一である。

4.1　文節データベース

可能な文節集合を列挙することは、自明でもなければ容易でもない。そこで、我々は、非常に簡単な文節モデルと形態素解析用辞書を用いて、暫定的な文節集合を生成し、文節データベースとして使用する。

表 1 に、文節データベースの概要を示す。データベースに格納された各文節は、日本語表記のリスト、読み (かな表記)、文節タイプの 3 つの要素から構成される。この文節タイプは、文節の文法的性格を抽象化したもので、36 種類存在する。

この文節データベースは、次の手順で作成した。

(1) 基本集合の定義：内容語の集合と付属語の集合を定める。

(2) 活用展開：それぞれの集合に含まれる活用語に対し、すべての活用形を生成し、集合を拡大する。

(3) 付属語を持たない文節集合の作成：単独で文節を構成する内容語を抜き出し、文節タイプを付与する。

(4) 付属語を持つ文節集合の作成：それぞれの内容語に対して、接続可能な付属語を接続した後、文節タイプを付与する。

(5) コーパスフィルタリング：作成した文節集合と解析済コーパスと突き合わせ、コーパスに出現する文節のみを残す。

(6) 終助詞付与：それぞれの文節に対して、接続可能な終助詞を接続した文節を作成し、集合に追加する。

(7) 表記のグループ化：読み (かな表記) と文節タイプ

表 1　文節データベースの概要

タイプ名	文節数	文節の構造	例
サ変名詞	4669	サ変名詞	運動
サ変名詞–文末	27920	サ変名詞＋終助詞	運動よ
格–サ変	38038	サ変名詞＋助詞	運動が
格–普通	110462	普通名詞＋助詞	リンゴが
接続	93	接続詞	しかし
独立	93	感動詞	ありゃ
普通名詞	13026	普通名詞	リンゴ
普通名詞–文末	77338	普通名詞＋終助詞	リンゴよ
並列格–サ変	8710	サ変名詞＋並列助詞	運動と
並列格–普通	24702	普通名詞＋並列助詞	リンゴと
用–形–テ形	1617	形容詞のテ系	かわいくて
用–形–基本	4826	イ形容詞の基本形、形容詞のタ形	かわいい
用–形–接続	736	形容詞の条件系	かわいいなら
用–形–文末	105585	形容詞＋終助詞、ナ形容詞の基本形	かわいいわ
用–形–連体	5637	形容詞の連体形	きれいな
用–形–連用	5086	形容詞の連用形	かわいく
用–動–テ形	10808	動詞型のテ形	走って
用–動–テ形–判定–サ変	194	サ変名詞＋判定詞のテ形	運動であって
用–動–テ形–判定–普通	688	普通名詞＋判定詞のテ形	リンゴであって
用–動–基本	20148	動詞型の基本形、タ形	走る
用–動–基本–判定–サ変	570	サ変名詞＋判定詞のタ形	運動であった
用–動–基本–判定–普通	1785	普通名詞＋判定詞のタ形	リンゴであった
用–動–接続	6842	動詞型の条件形	走れば
用–動–接続–判定–サ変	98	サ変名詞＋判定詞の条件形	運動だったら
用–動–接続–判定–普通	507	普通名詞＋判定詞の条件形	リンゴだったら
用–動–文末	291647	動詞型の命令形、動詞型＋終助詞	走れ
用–動–文末–判定–サ変	44171	サ変名詞＋判定詞の基本形 (+終助詞)	運動だ
用–動–文末–判定–普通	132661	普通名詞＋判定詞の基本形 (+終助詞)	リンゴだ
用–動–連体	188	動詞型の連体形	歓迎せざる
用–動–連体–判定–サ変	4468	サ変名詞＋判定詞の連体形	運動の
用–動–連体–判定–普通	13476	普通名詞＋判定詞の連体形	リンゴの
用–動–連用	14369	動詞型の連用形	走り
用–動–連用–判定–サ変	925	サ変名詞＋判定詞の連用形	運動であり
用–動–連用–判定–普通	2298	普通名詞＋判定詞の連用形	リンゴであり
連体	77	連体詞	確たる
連用–副詞	1102	副詞	格段

が同一のものをグループ化する (つまり、表記のみが異なる文節を一つにまとめる)。

具体的には、内容語の集合には、JUMAN5.1 の ContentW.dic から語構成要素 286 語を除いた 30,369 語を、付属語の集合には、助詞 26 語と助動詞等 19 語を、終助詞には 11 語を、活用体系には JUMAN 体系を用いた。コーパスには、毎日新聞データ 1991–2005 年と日本語均衡コーパスの 2009 年度モニター公開版を用いた。終助詞付与が終った段階の文節数は 1,064,415、表記のグループ化後は 975,560 となった。アナグラムの生成においては、この文節データベースを利用し、アナグラムが完成した後、異なる表記を展開して出力する。

4.2　文節列に対する制約条件

文節列 $u_1u_2\cdots u_n$ が満たすべき制約として、「仮想的な終了記号 u_{n+1} を根とする文節係り受け木を構成できる」という条件を採用する。つまり、文節列の背後に、文節係り受け木の存在を仮定する。ここで、文節係り受け木は、次の制約条件を満たすものとする。

(1)　文節 u_i の係り先は、かならず、それより後方の文節 $u_j(j > i)$ である。

(2)　係り受け関係は交差しない。

(3)　係り元文節 u_i と係り先文節 u_j は、あらかじめ定義されている係り受け制約を満たす。

(4)　文節係り受け木における姉妹文節は、あらかじめ定義されている制約を満たす。

(5)　その他の例外的制約☆を満たす。

これらの制約条件のうち、最初の 3 つの条件は、文節列から文節係り受け木を生成する際に一般的に用いられる制約条件である。1 番目と 2 番目が原則であるのに対し、3 番目は個別規則となる。具体的には、すべての文節タイプの係り受けの組に対し、それが可能であるかどうかを定義する。アナグラムでは、文法的適格性はあまり厳密には問われないため、我々は、規範的な日本語文に対する係り受け制約よりは、少し緩めの制約を採用している。

これに対して 4 番目の条件は、たとえば、「一つの動詞には、格助詞 (または係助詞) が省略された要素が同時に複数係ることはない」のような、親子関係を越えた制約

☆ 感動詞および接続詞に対する制約が存在する。

条件を導入するために必要となる。現実に存在する文を解析する場合には、そのような文節列は現れないので不要であるが、自由に文節列を生成する場合には、このような制約条件が必要となる。なお、これらの制約条件も、すべて、文節タイプを用いて定義している。

文節列から文節係り受け木を生成する方法は、通常の文節列の係り受け解析と同様である。事実、connectable の判定は、係り受け解析の 1 ステップ (係り受け解析木に、新たな文節を追加する) に相当する。図 2 のプログラムで、文節列を後ろから作成するのは、このような係り受け解析を可能とするためである。現在の connectable の実装は単純であり、可能な係り先のうち、最も近い係り先を選択する方法を採用している。

4.3 順位付け規則の導入

本システムは、条件を満たすアナグラムをすべて出力する。その出力数は膨大な数となることがあるため、暫定的に次のような順位付け規則を実装した。なお、以下の順番は、規則の優先順位を表す。

(1) 文節数が少ないものを優先する。
(2) 文節内の内容語 (のかな表記) の長さの平均長が長いものを優先する。
(3) 入力かな表記 K_s と出力かな表記 K_a の編集距離が大きいものを優先する。

5. アナグラム生成例

表 2 に、作成したアナグラム生成システムの実行例を示す。この表の一番左の欄がシステムへの入力、次の欄が入力の元となった表記である (システムには入力しない)。これらは、日経サイエンス 2011 年 5 月号の 4 頁 (ダイジェストページ) に出現する表現から無作為に選んだ。4–6 番目の欄は、それぞれ、アナグラム生成に要した時間、生成したアナグラムの読みの異なり数、アナグラムの出力数 (表記展開後の数) を表す。最後の欄は、システムの出力を先頭から最大 100 件調べた中で、意味の通るアナグラムと認定したものの 1 例を示す。この結果より、我々のアナグラム生成システムは、意味の通るアナグラムを生成する能力を有していることが確認できる。

6. 検　　討

6.1 生成例に対する考察

表 2 の結果から、次のことが観察できる。

(1) 入力かな表記 K_s の文字数が長くなれば、生成されるアナグラムの読みの異なり数、出力数、生成時間が増加する傾向にある。しかしながら、これらは、一般的な傾向に過ぎず、実際の数および生成時間は、個々の入力によって大きく異なる。

アナグラム生成の入力である K_s は、いわば、使用できる音のリストである。ある音のリストからは、多数の日本語表現を作ることができるのに対し、別の音のリストからは、限られた日本語表現しか作ることができない。つまり、アナグラム生成が容易な K_s とそうでない K_s が存在する。

(2) 読みに対して表記を展開すると、数倍から数十倍となる。

たとえば、最後の「社会的な対策」では、読みの異なりが 20 万に対し、表記を展開すると 610 万となる。これより、比較的長いアナグラムの生成 (探索) において、表記のみ異なる文節を一つにまとめて処理することは、効率化のために欠かせないことがわかる。

(3) 最後の 2 例を除き、アナグラム生成は 1 分以内に完了している。

アナグラム生成に使用した計算機は、MacPro(2 x 2.26GHz Quad-Core Intel Xeon/Memory 16GB) で、取り立てて高速なマシンではない。10 文字程度のアナグラムの網羅的生成は、極端に出力数が多い場合を除いて、現実的な時間で完了する。

6.2 日本語表現の妥当性の判定

現在のシステムにおいて、妥当な日本語表現を生成する仕組みは、(1) 妥当な文節の生成と、(2) 妥当な文節列かどうかの判定 (connectable) の 2 つが担っている。これは、表現の妥当性を、文節内と文節列 (文節を越える範囲) に分けて考えることを意味する。

前者は、可能な (妥当な) 文節をすべて列挙することができれば解決できる。すでに述べたように、可能な文節の列挙は自明でもなければ容易でもない。しかし、

(1) 内容語の集合を定めれば、可能な文節の数は有限に収まる、
(2) 文節の妥当性は、文法的適格性によってほぼ定まると考えられるため、文節モデル (文節内文法) を精緻化していけば、理想的な文節集合に近いものを作成できると考えられる。

一方、後者の問題には、文法的適格性の他に、意味的整合性が大きく関与する。

文法的適格性、すなわち、文節の係り受けについては、精緻化が可能である (省略や倒置といった、非規範的な形式をどの程度許容するかとについては、別途、方針を定める必要がある)。文の係り受け解析では、親子関係の係り受け制約が中心となっているが、許容できない文節列を積極的に排除するためには、より広範囲の要素を考慮した制約が必要となる。それに加え、表現として完結しているかどうかの判定も必要となる。たとえば、「開く姪の顔」においてそれぞれの係り受けは可能であるが、表現としては完結しない。「口開く姪の顔」であれば、表現として完結する。このような完結性の判定は、現在の connectable の枠組では扱えず、アナグラムが完成した後に、表現全体の完結性をチェックする機構を追加する必要がある。その枠組の導入は容易であるが、実際に表現の完結性を判定するプログラムを実現するのは、これまで研究がほとんどないため、かなりの困難が予想される。

もう一方の意味的整合性の判定は、さらなる困難が予

表 2　アナグラム生成例

読み K_s	表記	$\|K_s\|$	時間 (s)	読み数	出力数	出力例 $\langle J_a, K_a \rangle$	
まんぐろーぶ	マングローブ	6	0.005	0	0	–	
てんねんぎょ	天然魚	6	0.005	0	0	–	
うらづけられた	裏付けられた	7	0.012	1	1	–	
ぎょかくだか	漁獲高	6	0.012	0	0	–	
はやりやまい	流行病	6	0.021	3	5	鞠速いや	まりはやいや
せんしんこく	先進国	6	0.036	23	57	–	
よそくもある	予測も有る	6	0.047	26	93	祖も歩くよ	そもあるくよ
ゆうこうだった	有効だった	7	0.048	10	25	–	
いっけんじみだ	一見地味だ	7	0.060	1	1	–	
かがくてきに	科学的に	6	0.069	18	96	–	
せいさんせい	生産性	6	0.078	28	170	遺産制せ	いさんせいせ
ぼうだいなみず	膨大な水	7	0.080	10	70	–	
たんぱくしつ	タンパク質	6	0.100	33	157	パンツ支度	ぱんつしたく
にひゃくかいり	200 海里	7	0.100	3	5	–	
はたらきざかり	働き盛り	7	0.130	16	51	–	
せきりやこれら	赤痢やコレラ	7	0.140	31	177	–	
じょうげすいどう	上下水道	8	0.160	35	113	–	
ひまんしゃかい	肥満社会	7	0.200	14	22	–	
しさんもある	試算もある	6	0.220	67	545	猿も思案	さるもしあん
いしゃにかかる	医者にかかる	7	0.290	62	452	釈迦に怒る	しゃかにいかる
かんきょうづくり	環境づくり	8	0.360	4	58	気付く官僚	きづくかんりょう
かちくをそだてる	家畜を育てる	8	0.430	149	273	–	
こうだいなとち	広大な土地	7	0.520	163	608	稲田と耕地	いなだとこうち
ひまんにかんする	肥満に関する	8	0.610	153	700	–	
それだけのひとびと	それだけの人々	9	0.670	71	436	–	
あおのかくめい	青の革命	7	0.740	221	1358	悪の負い目か	あくのおいめか
かせきねんりょう	化石燃料	8	1.160	157	759	鐘売り占拠	かねうりせんきょ
せかいじんこう	世界人口	7	1.630	378	2941	時効静観	じこうせいかん
みつかっていない	見つかっていない	8	2.090	194	1080	詰み一定かな	つみいっていかな
たいしゃやいでんし	代謝や遺伝子	9	2.920	576	2364	社員嫌でした	しゃいんいやでした
ひまんかいしょう	肥満解消	8	3.100	204	748	新米評価	しんまいひょうか
だいえっとしやすい	ダイエットしやすい	9	3.990	657	2519	やっと言い出す絵師	やっといいだすえし
ひまんよくせい	肥満抑制	7	13.330	164	517	–	
かえてみたらどうか	変えてみたらどうか	9	14.860	1971	23668	恨み抱えた土手	うらみかかえたどて
よんじゅうだいのひと	40 代の人	10	22.760	2088	14914	順応と肥大よ	じゅんのうとひだいよ
しゃかいてきそんしつ	社会的損失	10	34.620	3664	27447	感謝して付添	かんしゃしてつきそい
やくななじゅうおくにん	約 70 億人	11	37.180	3459	19865	柔軟な肉屋置く	じゅうなんなにくやおく
しんけいかがくてき	神経科学的	9	388.200	21126	560733	計画して祈願	けいかくしてきがん
しゃかいてきなたいさく	社会的な対策	11	12795.760	200511	6102086	最適な社宅かい	さいてきなしゃたくかい

想される。我々が知る限り、与えられた文字列が、意味の取れる日本語表現となっているかどうかを機械的に判定するシステムは存在しない。アナグラムはさらにやっかいであり、「恨み抱えた土手」や「柔軟な肉屋置く」のように少し変な表現が面白いアナグラムとみなされることもある。現在の言語処理技術は、言語表現として成立している文字列を解析することに主眼を置いているが、このような問題に対しては、これまでとは異なる問題設定で研究を進める必要があろう。

6.3　現時点の到達点

現時点の到達点をまとめると、次のようになる。

(1) 日本語アナグラムの生成の基本的な枠組は、明らかとなった。

(2) 現在のシステムで使用している文節集合、および、文法的適格性の判定のための制約は不十分である。しかし、文節内文法および文節係り受け文法の精緻化は可能であり、改良の余地は十分にある。

(3) 意味的整合性の判定は、手付かずのまま残されている。

謝辞　本研究では、CD-毎日新聞データ集 (1991 版–2005 版)、および、国立国語研究所が開発した「現代日本語書き言葉均衡コーパス」モニター公開データ (2009 年度版) を利用した。

参　考　文　献

1) 荻生待也 (編): 図説 ことばあそび遊辞苑, 遊子館 (2007).
2) stabucky: アナグラム自動生成, http://tool.stabucky.com/anagram.php.
3) 村山大介, 小野智司, 中山茂: 統計的文章評価と遺伝的アルゴリズムに基づくアナグラム文の生成, 情報処理学会火の国情報シンポジウム 2009, C-5-1 (2009).
4) 佐藤理史:「意外や意外」回文だ－文章中に知られざる回文はあるか－, FIT-2010（第 9 回情報科学技術フォーラム), RE-003, 第 2 分冊, pp. 37–40 (2010).
5) 鈴木啓輔, 佐藤理史, 駒谷和範: 文節固定法による効率的な回文生成, 言語処理学会第 17 回年次大会発表論文集, pp. 826–829 (2011).