単語抽出法による次世代データ圧縮法の開発

Next-generation data compression by Word eXtraction method

岡野原　大輔
Daisuke Okanohara

東京大学 理学部 情報科学科
(自宅 〒166-0015 東京都杉並区成田東 2-6-6　E-mail: VZV05226@nifty.com)

ABSTRACT. I developed a new compression algorithm which exceeds the ordinary N-gram based compression algorithms such as LZ PPM BWT. It is known that data stream can be divided in variable-length data which is callled multi-gram, but it is hard to get it efficiently. I developed the fast and low-memory requeament algorithm, “Word eXtration method”, based on Suffix Array idea for multi-gram constraction. Word eXtraction method enable us to use Trigger-model and Class-model which are used in Natural Language Processing region, to consider long-range and abstracted relations between data. Based on these idea, I suppose the new Compression Algorithm, “Static PPM”, which has the following caracteristics, (1) good compression ratio and at decompression (2) low-memory requeament (3) fast- processing (4) easily implemented at decompression stage.

1．背景

現在、非歪みデータ圧縮（以下、データ圧縮は非歪みデータ圧縮を指す）は多くの分野で使われており、現在の情報技術の基盤をなしている。

そのデータ圧縮法は現在、N-gram に基づく方法が主流である。つまり隣接間の関係のみを利用してデータ圧縮を行っている。例えば、LZ 法、PPM 法[1]、BWT 法[9]がそれにあたる。しかし、実際のデータには離れたデータ間の関係、階層的なデータ構造が存在していることがあり、現在のデータ圧縮法はそれらを利用していない。

また、その一方でデータ圧縮の利用法を見ると、データ配布などで用いられる場合が多く、復元時の方が多くのユーザーにとって関係している現状がある。モバイル機器での利用を考えた場合においても、復元時に負担が少ないようなデータ圧縮法は需要が大きいと考えられる。

2．目的

次の二つの特徴

(1)N-gram より高次の相関を利用する

(2)復元時の負担が少ない

を持つデータ圧縮法を開発する。これらを実現するた

めに、次のアルゴリズムを用いた圧縮法を開発する。

(1) WX method

(2) Class Model

(3) Trigger Model

(4) Static PPM

本年度では、(1) について開発を行った。

3． 成果の概要

本プロジェクトでは、以下の２つのソフトウェア開発及び調査を行った。

1) 辞書無しでデータを抽出、分割するプログラム(wx)
2) Zip-f に基づいて圧縮を行うプログラム(sPPM)
3) wx の際に基準となる評価値の調査
4) 抽出の際の棄却条件の評価値の調査

4． 既存のデータ圧縮法

wx 法が今までのデータ圧縮と、どこが違うのかを比較するため、今までのデータ圧縮について概要を述べる。

現在、非歪みデータ圧縮方で主流である LZ 法、BWT 法、PPM 法はいずれも同様の仮定を用いている。それは、「あるデータの出現確率は、直前のデータにしか影響を受けない」という N-gram model の仮定である。

（1） LZ 法

ＬＺ法は以前に出現したデータを特殊記号により表現するなどして、繰り返し部分の冗長な部分を無くすことにより圧縮を行う。

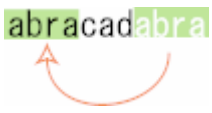
例として次のデータ

abracadabra

を LZ 法により圧縮することを考える。

abracadabra

このとき背景がついた部分列、「abra」は繰り返されているので、前回出現した部分を用いて後ろの部分列を表現することができる。



後ろのデータを前回の出現位置、部分列長などを出力すれば、元のデータより短く出力することができ、圧縮される。

（2） BWT 法

BWT 法[9]は、元データの巡回データをソートすることにより、データ圧縮がしやすいデータに元データを可逆に変形する。例として次のデータ

abracadabra

を BWT 法により変形すると

rdarcaaaabb

となる。このように BWT 法による変形後のデータは、同じデータが並びやすくなり圧縮しやすくなる。

このとき、ソート時に辞書式順序を用いているので、同じデータが並びやすくなるのに用いた情報、データの前後関係、つまり隣接しているデータである。

（3） PPM 法

PPM 法[1]はデータを逐次的に見ていき、次に出現するデータを予想することにより確率構造を変えていき圧縮を行うものである。例として次のデータ

abracadabra

を PPM 法により圧縮することを考える。

逐次的（一文字ずつ）圧縮を行っていき、最後の文字まで来たとする。

abracadabr#

そして#の部分の文字を予想する。以前のデータをみてみると abr の次は a が１回出現している。よって、a の出現する確率を高く設定し、それを算術圧縮、または RangeCoder で符号化することにより圧縮を行う。これも「直前の情報の利用」だけである。

以上(1)(2)(3)のとおり、既存のデータ圧縮は、扱い方は違うが、どれも隣接しているデータ間の関係のみを利用してデータ圧縮を行っている。

3．wx 法の意義

上記三つで述べたように既存のデータ圧縮法は直前の情報しか用いていない。wx 法は、データの構造的情報、離れたデータ間の共起関係を用いることにより、一つ上のレベルのデータ圧縮を目指す。

データの構造的情報、離れたデータ間の共起関係を調べる際にもっとも大きな問題は処理量、使用メモリ量の爆発である。

加工が何もされていないデータに対し離れている部分の共起関係を全て調べるとすると、データ長を N としたとき、$O(N^4)$ の関係を調べなければならない。

（部分列を二つとってくることは、データ長に一つ目の部分列の区切りを二箇所、二つ目の部分列の区切りを二箇所、合計四箇所区切りを入れることと同じであるので、$_nC_4$ 通りあるから）

そこで、データをあらかじめ、有意な部分列に区切っておき、1 対 1 の関係を調べることで、調べなければならない共起関係を $O(N^2)$ まで抑えることができる。

以後、この有意に分割された部分列を unit と呼ぶ。例えば、自然言語に対する単語、ゲノム情報に対するコドン、バイナリデータに対する int サイズの部分列が、それぞれに対する unit といえる。

つまり、unit は、そのデータにおいてそれ以上分けられない最小単位である。

この考えは、自然言語処理の分野において、multi-gram[14]の名で用いられているが、分割された unit を用いて、Class model, Trigger model を求めるアイディアは本手法が初であり、データ圧縮に利用する部分においても初である。また、[14]では、HMM を用いて unit 抽出を手法が提案されているが、本手法では、より決定的なアルゴリズムを用いて高速に抽出することができる。

abracadabra

を例に unit 抽出を行い、**abra** と **cad** という二種類の unit が抽出できたとする。。この unit をそれぞれ **A**、**B** と名付ける。

A = abra　B = cad

するとこのデータは

ABA

というデータに分けられる。この情報をさらに使うことにより、

- A が一度出現した後はもう一度出現しやすい
- A の後には B が出現する

といったように一つ上の段階での構造的情報が見えやすくなる。

極端な例で言えばホームページのファイルは <HTML>と</HTML>といったようにタグ同士が必ず対になっている。この対関係を利用することにより

<###> → A のようにタグの始まりを大文字のアルファベット

</###> → a のように小文字

と置くとホームページのファイルは

<html> <body> ・・・・ </body> </html>

↓

ABCDEFGgfedcba

というふうに変換でき、記号$を「前出した大文字で表された unit を小文字の unit にして、それを逆順に出力する」という記号と置くと

ABCDEFG$

と圧縮することができる。これは今までのデータ圧縮ではできなかったことである。

同様に通常のデータも何らかの構造的な特徴を持っている場合が非常に多く、それらの特徴を的確に利用することを、この wx 法が可能にすると考える。

４． wx 法のアルゴリズム

一般にデータから unit、例えば自然言語では単語を抽出する方法は、辞書を用いてそれにより形態素解析を

行うというのが主流である。しかし、この場合だと、辞書がそのデータに対応していない場合抽出ができないという大きな問題点がある。その上、人間の書いた文章などに対する辞書は存在するが、コンピューターが作成したデータ、HTML、XML、バイナリデータなどに対する辞書はない。

そこで、与えられたデータから効率的に情報を収集し、unit に分割する方法、WX 法を開発した。

本問題は次のように定式化される。

次のような長さ n のデータ S

S = $x_0x_1x_2x_3x_4\Lambda\ x_{n-1}$

が与えられた時、この S を適当な部分列に分け、その各部分列にユニット u_i を対応させる。この

$$x_0x_1x_2x_3x_4\Lambda\ x_{n-1}$$
$$\Downarrow$$
$$[x_0x_1]\oplus[x_2x_3x_4]\oplus\Lambda\ [x_{n-2}x_{n-1}]$$
$$\Downarrow \qquad \Downarrow \qquad \Downarrow$$
$$u_0 \qquad u_1 \qquad u_{k-1}$$

ただし、各 u_i は次のような情報を持っている。

u_i.l := u_i の長さ

u_i.s := u_i が示す部分列 $a_0a_1a_2\ \Lambda\ a_{u_i.l-1}$

u_i.c := u_i のデータ中における出現回数

こうして、データは次のように変換できる

S = $u_0u_1u_2u_3u_4\Lambda\ u_{k-1}$

ただし、k は全 unit 数である。

このとき、分割の方法の有意性を判断するため、ユニットによる表現におけるデータの記述長 L を用いる。L は、

$$L = \sum_{i=0}^{k-1} -\log_2\left(\frac{u_i.c}{k}\right)$$

として求めることができる。L はそのまま、ユニットを用いて表現したときのデータ本体のデータ長といえる。

次に具体的なアルゴリズムについて述べる。

6．wx 法の実装

wx 法の実装は大きく分けて

- 全ての unit 候補を抽出する （6．1）
- unit 候補を割り当てる （6．2）

の二つに分けられる。そのままこの方法を使うと処理量は非常に大きいので、次のような高速化

- unit 候補を枝刈りする （6．3）
- WordTree による割り当ての高速化（6．4）

が必要になる。上記のように大きく４つに分けて説明する。

6－1．全ての unit 候補を抽出する

全ての unit 候補を抽出するのには様々な方法が挙げられる。

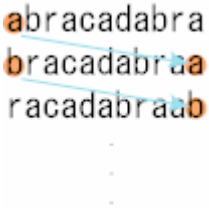
- ＬＺ法のようにデータを読みながら、一致したものを unit 候補として登録していく
- Trie 木を利用して、登録していく。
- ハッシュを用いて登録していく。
- Suffix Array の利用

最後の Suffix Array の利用は、速度的にも、メモリ的にも優れている。速度はファイルサイズNとしたときO（N)、メモリはファイルサイズの定数倍と一定に抑えられる。そこでこの方法を紹介する。

データ例として

abracadabra

を考える。これを巡回させたものを並べる



結果として次のような巡回データが得られる。

abracadabra
bracadabraa
racadabraab
acadabraabr
cadabraabra
adabraabrac
dabraabraca
abraabracad
braabracada
raabracadab
aabracadabr

このデータを辞書式順序でソートし、単語候補を抽出する。

aabracadabr
abraabracad
abracadabra
acadabraabr
adabraabrac
braabracada
bracadabraa
cadabraabra
dabraabraca
raabracadab
racadabraab

これを実装する際に、有効な手段が Radix ソート（基数ソート）である。

Radix ソートは、ソートするキーが m 桁の場合、処理量がO（nm）となる。

各桁をソートする場合には、

（１）文字のカウント

（２）積み上げ

（３）ソート

の三段階に分かれるが、この BWT の場合には少しの工夫で、各桁の処理の（１）（２）を省略することができる。その工夫とはソートする桁数が m 桁の時、ソートする文字の最初の m-1 文字を後ろにつけておくと、常にソートする際の出現数は一定となることを利用することである。データ例が abracadabra　でソートする文字が４文字の場合

abracadabraabr （下線をつけた部分が付け足した部分）

それぞれの４桁目をソートすると出現数は

abra

brac

raca

acad

cada

adab

dabr

abra

braa

raab

aabr

影付きの部分をカウントすると、a:5 b:2 c:1 d:1 r:1 個ずつある。同様に３桁目も同じ数、２桁目、１桁目も同じ数である。よって、カウント、積み上げを一度行えば、３桁目、２桁目、１桁目は、カウント、積み上げを行わなくてもソートが可能である。よって、次のようなアルゴリズムとなる。

```
//　まず各文字の出現数をカウントする。
    for (int i = 0; i < size; i++) count[buf[i]]++;
// 次に積み上げを行う
    for (int i = 1; i < size + 1; i++) count[i] += count[i – 1];
// 奇数回目は(A)、偶数回目は(B)の方法によりソートを行う
    pos[--count[buf[work[i]]] = work[i];   //(A)
    work[count[buf[pos[i]]]++] = pos[i];   //(B)
```

これで m 桁の Radix ソートが O(n)で行えることがわ

かる。これによって、（２）の状態まで来た。

次に、このソートされている状態から、unit を抽出するところを説明する。

これは単純に前の列と何文字一致しているかを調べていき、一致数が前回の一致数より多かったら、何列目から何文字一致しているかを保存しておき、一致数が前回の一致数より少なかったら、その保存しておいた情報を用いて unit を登録すればよい。

↓前の列との一致数

0: aabracadabr 0

1: abraabracad 1

2: abracadabra 4

3: acadabraabr 1

4: adabraabrac 1

5: braabracada 0

6: bracadabraa 3

7: cadabraabra 0

8: dabraabraca 0

9: raabracadab 0

10: racadabraab 1

0

start[]というデータは、start[N] = M は、N 文字一致している列の最初は、M 列目であるということを意味している。例えば 1:では前の列との一致数が 1 である。この場合、一致数が０から１へと多くなっているので、start[1]に 0 を代入する。次に 2:では前の列との一致数が 4 であり、この場合も一致数は多くなっているので start[2]、start[3]、start[4]にそれぞれ 1 を代入する。次に 3:は一致数は１なので、一致数は少なくなっている。よってこの場合は unit を登録する。この場合は 2,3,4 文字目が登録されることになり、その登録の開始場所は start[2],start[3],start[4]に入っている数であり、終了場所は現在の位置である。

つまり(unit の始まりの場所,unit の終わりの場所,unit 長)の組み合わせで考えると

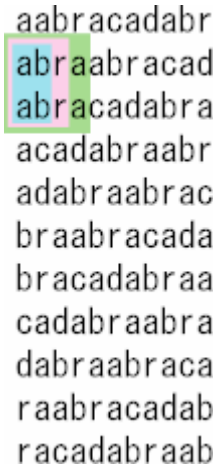
(1,3,2)

(1,3,3)

(1,3,4)

が登録されることになる。unit の出現回数は、unit の終わりの場所－unit の始まりの場所として計算でき、上の三つの場合は全て（3－1）＝ 2 で、出現回数は２回だとわかる。

この段階で抽出されている unit は下図のようになる。違う色で囲まれている領域は違う unit であることを示す。



この方法を適用することにより、全ての unit がもれなく抽出できる。

６－２． unit 候補を割り当てる

ここでは、抽出された unit を実際に、データに割り当てていく。

今抽出した unit はまだ重複した部分がたくさんあり、どの unit を使用するかは決まっていない。この割り当てはどの unit をどれだけ採択するかを決めることである。

最も単純な方法は、unit を評価値の高い順に並べていき、高い順から割り当てられるだけ割り当てていく方法である。この場合にどのような評価値を選ぶかが、unit 抽出の際の唯一の問題となる。しかしこの方法は問題が

ある。評価値が低くても、ある場所では採択した方がよい unit も存在する。the を採択した後に they は採択できない。なぜならすでに they の unit がある場所は the がとっているからである。

the ・・・ the ・・・ they ・・・
the ・・・ the ・・・ they ・・・
↑ここの unit は(the + y)

と認識されてしまった。

またもう一つの問題がある。評価値は unit 抽出前に正確には求められないということである。評価値の所で述べた方法はいずれも抽出後の unit 数等をパラメーターとしているが、それは unit 抽出前にはわからない。

これを解決するために、実際に一度割り当ててみるという方法がある。

unit を抽出する順番を待ち行列のように並べておき、順番が来たら、その段階で unit を割り当ててみて、それを評価値とした上で、待ち行列に登録する。もし、正しい評価値でも、待ち行列の先頭にある場合には他の unit の正しい評価値が、それを超えることはない。なぜなら、正しい評価値は必ず、推定された評価値より低いから、いくら他の unit の評価値が高くても、先頭の unit よりは正しい評価値は低くなるからである。この方法は、unit 数が多いと時間がかかってしまうので、あまり効率的な方法ではない。

他に遺伝的アルゴリズム、SimE、StockE などの最適近似アルゴリズム[4]を使用して、unit を推定する方法もあるが、割り当てのコストが低くないため、時間がかかりすぎてしまう。割り当てのコストが低くなればこれらの方法は有効だと思われる。

６－３．　unit 候補を枝刈りする

（６．１)、(６．２）の方法を用いれば unit は抽出できるが、実用レベルのコストとしては処理量が大きすぎる。そこで、使用メモリ、割り当て時の負担を軽減させるため、抽出すべき unit を枝刈りする方法をここで述べる。

unit を枝刈りする際に重要なのは

- 抽出されるはずの unit は枝刈りしない（＃）
- 多く枝刈りを行う　　（＃＃）

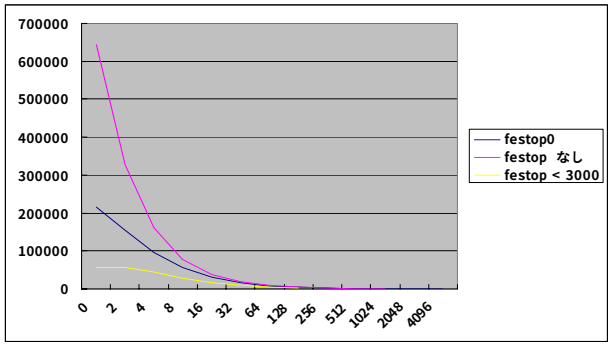
の二つのトレードオフの関係が成り立つ。両者を同時に成立させるため、様々なアプローチをした。

(i)評価値の低いものに枝刈りを行う
(ii)前後の文字との相関関係を調べて枝刈りを行う
(iii)unit の出現数、unit 長によって枝刈りを行う

これらの方法を組み合わせた上で、残された unit 数、及びデータ圧縮率を元に比較を行った。

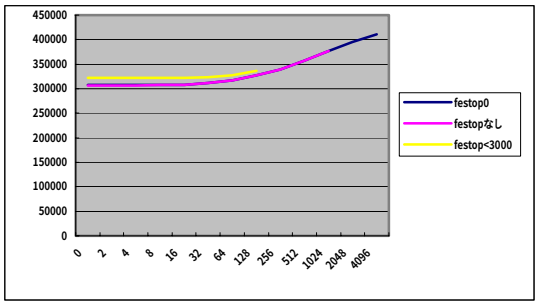
（A）抽出された unit の数と、前後の文字との相関及び unit の出現数による枝刈りの関係

festop0:前後の文字のエントロピーが０のものは枝刈り
festop なし:前後の文字の相関関係は無視
festop<3000 前後の文字のエントロピーが３より低いものは枝刈り



値が低いほど、候補 unit 数が少なく、優れている

（B）圧縮率と、前後の文字との相関及び unit の出現数の枝刈りしたものの関係



値が低いほど、圧縮率が高い、すなわち不必要な unit だけを枝刈りしていることを示す。

（B）を先に見てみると、festop0 と festop なしがほぼ同じ曲線を描いている。これは、festop０で除かれた unit が、最終的な採択には影響は無いということを示している。festop<3000 は他の二つに比べて上にあるため、最終的な採択で選ばれるはずだった unit が除かれてしまったことを示している。横軸、unit の出現回数による枝刈りは 32 ぐらいまで影響が全く無い。

（A）を見てみると、festop なし＜festop0＜festop3000 の順に除かれている unit の数は多い。しかし注目すべきなのは横軸、unit の出現回数による枝刈りの影響である。この横軸には、ほぼ反比例して、必要な unit 数は減っている。これは、ジップの法則「unit の出現回数と、unit の出現順位は反比例する」の応用だといえる。

この二つを考慮すると、このテストデータの場合は festop0、unit の出現回数による枝刈りは３２前後が良いと考えられる。

７．抽出された unit の妥当性評価

unit を抽出するのは、離れたデータ間の共起関係、及び構造モデルを考えることができるようなモデルにデータを変形することが目的である。よって、こうした離れたデータ間の共起関係、及び構造モデルを、誤った unit を抽出しないようにするのが、unit 抽出における条件である。正しい unit が抽出されているかどうかを判断するために、BWT 法を用いて、圧縮したものと、元のデータを BWT 法によって圧縮したものとを比較した。

BWT 法は、隣接しているデータ同士の相関を利用してデータ圧縮を行う。もし、unit 抽出によって、保存しておくべきデータの性質を破壊してしまった場合は、圧縮後のデータサイズが、unit 抽出を行わず BWT 法によって圧縮したデータサイズに比べて大きいものとなる。もし、保存しておくべきデータの性質を残した場合は、圧縮したデータサイズは、unit 抽出を行わなかった場合に比べて圧縮率は同じくなるはずである。

(1) 正しい評価値と思われるものを用いて unit 抽出を行ったもの
(2) 適当でない評価値を用いて unit 抽出をしたもの
(3) unit 抽出をせずそのままのデータ

上のそれぞれに対して BWT を行いデータ圧縮を行った。テストデータは Canturbury Courpus の book1 ファイルである。

データ圧縮後のサイズ

length*log(count)	261,874 (1)
length	410,886 (2)
（log(log(count)) * length	251,372）(4)
unit 抽出を行わない	267,206 (3)

(Bytes)

確かに(1) (3)であり、(1)の方法は、保存しておくべきモデルが、破壊されず残っていると考えられ、(2)の場合は、保存しておくべきモデルが破壊されてしまったと考えられる。（(2)の方法は LZ 法と実質的に同じ）ここで、注目すべきなのは(4)の方法である。これは、(1)(3)よりも良い圧縮率を出している。これは、より良い評価値を用いることにより、全く unit 抽出を行わない場合よりも、BWT の性能を上げるようなデータに変換できたことを意味する。

8．Static PPM

ここでは、これらのアルゴリズムがどのように利用できるか、そして現在作成中である新しい圧縮アルゴリズムについて述べる。

抽出された unit を用いて Class model、Trigger model を求められることは 3 章で述べ、また、平成 15 年度未踏プロジェクト「汎用的データにおける確率的言語モデルの抽出およびその利用」において実装、検証中である。

それらのアイディアを用いて復元時に負担の軽いようなデータ圧縮の概要を述べる。

まず、圧縮時に、データを unit に分割した状態にしておく。次に unit 間の相関を class model によって抽象化し、また trigger model によって離れた相関も検出しておく。そして、この class model、trigger model によって得られた静的（ここでの静的とは、モデルが、データ全体の処理を通して一定であることを意味する）なモデルに基づいて PPM を適応して圧縮する。

復元時には、既に求められている class model、trigger model の情報に基づいてデータを復元する。

この方法の利点として、

・復元時には、既に求められている class model、trigger model の情報を利用するだけなので、複雑な計算はほとんど必要とせず、高速である。

・復元時に、必要なメモリーは class model、trigger model、unit 情報のみであるので、大きなデータに対しても非常に小さく済む。これは BWT や通常の PPM 法が、データ長に比例するメモリーを必要とするのに対し、Static PPM 法では、例えば class 数が 64 であるとき、1kB 程度で抑えられる。unit 情報についても、単語情報はデータ長に対し無視できるほど非常に小さい。

・圧縮率は通常の BWT、PPM 法に匹敵するレベルである。

このアルゴリズムは特にモバイル機器など、少ない資源を利用しなければならない状況において、非常に有効である。

この方法の詳細については、平成 15 年度未踏ソフトウェアの成果発表で述べる予定である。

9．まとめ

本プロジェクトでは、一切の前提知識も用いずにデータを unit に高速に分解する WX 法を開発し、その効果を確認した。また、データ圧縮法として使えることも確認した。

実際に WX 法を利用して高度なデータ圧縮を行うには、Class Model、Trigger Model といった更に高度な解析が必要であり、その時に初めて従来の圧縮法を超えたと言える。この高度な解析を用いたデータ圧縮は、2003 年度未踏プロジェクト「汎用的データにおける確率的言語モデルの抽出及びその利用」で開発中である。また、この開発中のプロジェクトでは、WX 法の理論的実証、及び性能の向上も同時に進められている。

10．参加企業及び機関

プロジェクト管理組織　（株）三菱マテリアル

11．参考文献

[1] Bell T., Witten I.H., Cleary J.G., “Modeling for Text Compression”. ACM Computing Surveys, 1989, Vol.21, No.4, pp. 557-591.

[2] 北　研二　“言語と計算－４　確率的言語モデル” 東京大学出版会,1999, ISBN4-13-065404-7

[3] Shkarin D “PPM: one step to practicality”. http://datacompression.info/Miscellaneous/PPMII_DCC02.pdf

[4] Sadiq M. Sait, Habib Youssef, 白石　洋一　訳,”

組み合わせ最適化アルゴリズムの最新手法” 丸善株式会社, 2002, ISBN4-621-04998-4

[5] 情報理論とその応用学会編 “情報理論とその応用シリーズ 1-Ⅰ 情報源符号化 無歪みデータ圧縮” 培風館, 1998, ISBN4-563-01350-8

[6] 情報理論とその応用学会編 “情報理論とその応用シリーズ 1-Ⅱ 情報源符号化 歪みのあるデータ圧縮” 培風館, 2000,ISBN4-563-01451-6

[7] E.S.Ristad, Robert G. Thomas Non-Emitting Markov Models” , Technical Report CS-TR-544-96, Department of Computer Science, Princeton University. 1997

[8] Charles Bloom, “Solving the Problems of Context Modeling” ,1998 http://www.cbloom.com/papers/ppmz.zip

[9] M. Burrows, D.J. Wheeler, “A Block-sorting Lossless Data Compression Algorithm.” Digital Systems Research Center, Research Report 124.May 1994

[10] 小田 裕樹,北 研二 “PPM*言語モデルを用いた日本語単語分割” 情報処理学会論文誌 Vol. 41 No. 3 pp. 689-700, 2000

[11] 酒井 邦嘉 “言語の脳科学” 中公新書 2002 ISBN4-12-101647-5

[13] 東京大学教養学部統計学部教室編 “統計学入門” 東京大学出版会,1991,ISBN4-13-042065-8

[14] Sabine DeLIGNE and Frederic BIMBOT “LANGUAGE MODELING BY VARIABLE LENGTH SEQUENCE : THEORETICAL FORMULATION AND EVALUATION OF MULTIGRAMS” In Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)