ハイパフォーマンス
コンピューティング 93－12
（２００３．３．１２）

テキスト情報圧縮の時間的非対称性に関する考察

丹 羽 竜 哉†

急速な情報技術の発展に伴って、計算機の持つ記憶容量や処理能力も著しく向上しているものの、同時に処理すべき情報量の増加もそれに劣らず著しい。特に、ワールドワイドウェブやバイオインフォマティックスの分野では、その状況が顕著である。ここで情報圧縮技術を用いる事により、記憶容量の節約のみならず処理能力の向上も期待できる。この場合、書き込み処理（圧縮）の時間より読み出し処理（復元）の時間の節約が有効である。また、パターン照合などは圧縮イメージのまま処理が可能となる場合もある。本稿では、最近のテキスト情報圧縮技術の動向とその応用について紹介する。

A Study of Asymmetric Processes on Text Data Compression

Tatsuya NIWA†

Although the storage capacity and throughput which a computer has are also improving remarkably with development of a rapid information technology, the increase in the amount of information which should be processed simultaneously is also as remarkable as it. Especially, in the field of world-wide-web or bio-informatics, the situation is remarkable. By using data compression technology, not only saving of a storage capacity but the improvement in throughput is expectable. In this case, saving of the time of decompress processing is more effective than the time of compress processing. Moreover, processing of pattern matching may be attained with a compressed image. In this paper, I introduce about the trend of the text data compression technology.

1. はじめに

今日、磁気ディスクや半導体メモリの技術向上に伴って価格も低下し、容易に大容量記憶システムが構築できる様になった。従って記憶容量の節約はあまり需要がないと思われがちである。しかし、実際には処理すべき情報量の増加もそれに劣らず著しく、情報圧縮技術の必要性は以前にも増すものと期待できる。また、最近の情報圧縮技術は文字列パターンを利用するものが多く、それに伴って、圧縮ファイルを復元せずに圧縮イメージのままパターン照合を行い、総合的なパフォーマンスの向上を目指すという様な研究もある[1]。本稿では、最近のテキスト情報圧縮手法の動向とその応用について紹介する。

2. 情報圧縮アルゴリズムの種類

まず、最近の情報圧縮アルゴリズムを一通り紹介する。一般に圧縮アルゴリズムは、元通りに復元できる無ひずみ圧縮と、ひずみのある圧縮とに大別できる。ひずみのある圧縮は、主として画像や音声などの圧縮に用いら

† 独立行政法人 産業技術総合研究所 情報処理研究部門
† Information Technology Research Institute
National Institute of Advanced Industrial Science and Technology

れるが、本稿で扱うテキスト圧縮には適さない為、ここでは無ひずみ圧縮のみを扱う。

無ひずみ型データ圧縮アルゴリズムは、圧縮符号化に確率を陽に用いるものと用いないものとに分けられる。確率を陽に用いるものとしては、Huffman 符号、Tunstall 符号、算術符号などがあり、それぞれに事前に仮定された確率分布に従って符号化を行う静的符号と、確率分布を推定しながら符号化する動的符号がある。その他、統計的確率推定と算術符号の組み合わせとして、CTW (Context Tree Weighting) 符号, PPM (Prediction by Partial Matching)符号などがある。また、確率を陽に用いない圧縮法は、1977 年頃から辞書式 (LZ(Lempel-Ziv)77 符号, LZ78 符号, LZW (LZ-Welch)符号, LZFG (LZ-Fiala-Greene)符号など) が開発され、1994 年頃から開発されたソート式 (BS (Block Sorting)符号, CS(Context Sorting) 符号など)、および文法式 (SEQUITUR 符号, MPM(Multilevel Pattern Matching)符号, BPE(Byte Pair Encoding)符号など) が比較的新しい圧縮符号として知られている。[2]

なお、辞書式やソート式は Huffman 符号などと組み合わせて用いられる場合が多い。例えば、現在 Linux 等でよく使われている圧縮アルゴリズムとして gzip や bzip2 があるが、gzip は LZ77 と Huffman 符号を組み合わせたもの、bzip2 は BS 符号に MTF(Move To Front), RL(Run Length)の各手法を用いた後 Huffman 符号化するものである。

SEQUITUR 符号

文法式圧縮符号の例として、SEQUITUR 符号について説明する。これは、入力テキスト列を順次読み込みながら、文脈自由文法に似たルール群を構成する。復元はできたルールに基づいて文字の置き換えを行うだけなので、高速化が期待できる。ルール生成の手順を以下に示す ;

Step1　データ列から 1 文字読み込み、ルール S に追加する。

Step2　既にルールに含まれている文字対 $\alpha\ \beta$ が生成されたとき、

2-1 もう一方の文字対が $A_i \rightarrow \alpha\ \beta$ の形のルールである場合は生成された方の文字対を A_i に置き換える。

2-2 それ以外の場合は、新しいルール $A_j \rightarrow \alpha\ \beta$ を作り、両方の文字対を A_j で置き換える。

2-3 2-2 の操作で一度しか使われないルールができた場合には、そのルールを削除し元の文字対に戻す。

例として、x=abcdbcabcd の場合を示す。

$\{S \rightarrow a\underline{bc}d\underline{bc}$　（初めの 6 文字）

$\begin{cases} S \rightarrow aAdA \\ A \rightarrow bc \end{cases}$

：　（3 文字追加）

$\begin{cases} S \rightarrow aAdAa\underline{bc} \\ A \rightarrow \underline{bc} \end{cases}$

$\begin{cases} S \rightarrow \underline{aA}dA\underline{aA} \\ A \rightarrow bc \end{cases}$

$\begin{cases} S \rightarrow BdAB \\ A \rightarrow bc \\ B \rightarrow aA \end{cases}$

$\begin{cases} S \rightarrow \underline{Bd}A\underline{Bd} \\ A \rightarrow bc \\ B \rightarrow aA \end{cases}$　（1 文字追加）

$\begin{cases} S \rightarrow CAC \\ A \rightarrow bc \\ \underline{B} \rightarrow aA \\ C \rightarrow \underline{B}d \end{cases}$

$\begin{cases} S \rightarrow CAC \\ A \rightarrow bc \\ C \rightarrow aAd \end{cases}$

SEQUITUR 符号は、入力テキスト長に対してほぼ比例した時間でルールを生成でき (実験による経験値)、できたルールを算術圧縮することで gzip と同程度以上の圧縮率を達

成できる。ただし、現状では処理時間は圧縮／復元とも gzip の 3～30 倍程度かかっている[3]。これには算術符号の処理時間も含まれるため、その影響が大きいかもしれない。基本的には、特に復元アルゴリズムは文字の置き換えで済むので、高速化が可能であると思われる。

BPE (Byte Pair Encoding) 符号

SEQUITUR 符号とよく似ているが、これとは独立に考案された BPE 符号を紹介する。入力テキスト全体を見渡して、最も多い文字対を、テキスト中に現れていない別の文字で置き換えるという操作を、置き換えができなくなるまで（同じ文字対がなくなる，テキスト中に現れていない文字を使い果たす）繰り返す。SEQUITUR 符号との違いは、全体を見渡す事の他、ルールの右辺は必ず 2 文字である事と、その為にルールを消さない事である。以下に入力テキストを ababcabcd とした場合の例を示す；

$$\{S = \underline{ab}\underline{ab}c\underline{ab}cd$$

$$\begin{cases} S = X\underline{Xc}\underline{Xc}d \\ X \rightarrow ab \end{cases}$$

$$\begin{cases} S = XYYd \\ X \rightarrow ab \\ Y \rightarrow Xc \end{cases}$$

この符号化だけ（Huffman 符号等を使わない）で、LZW 符号に匹敵する圧縮率で、短い展開時間の符号が実現できる。

この符号の欠点は、入力テキスト全体を見渡すために十分な記憶容量が必要で、また圧縮時間は比較的長くかかる。[4]

3. 圧縮時間と復元時間の考察

圧縮／復元アルゴリズム側の特徴

一部の例外を除き、圧縮過程に比べて復元過程の方が処理が簡易であり、処理に要する時間も少なくてすむ。一般に、圧縮過程は情報源のモデル推定と符号化の二種類の処理が必要である。それぞれの圧縮法によって、モデル推定と符号化の二回のパスに分かれているものや、二種類の処理が明確に分かれていないものなど様々であるが、モデル推定にはパターンマッチングや頻度（数値）計算などの複雑な処理を伴う。それに対し復元の過程では、既に計算されているモデルに基づいて符号を書き戻すだけの簡単な書き換え処理だけですむ場合もある。LZ 等の辞書式符号や本稿で注目している文法式圧縮符号は、これに該当する。

実際に利用されているいくつかの PC 用圧縮ツールについて実験してみると、圧縮時間の復元時間に対する比は 3～10 倍程度であるものが多いが、中には圧縮時間の方が短いものや、復元時間の方が 20 倍以上速いものもある。[5]

記憶システム側の要求

実際にデータ圧縮が用いられる状況を考えてみよう。Shannon に始まる情報理論も元は通信の理論であったことをみてもわかる通り、圧縮技術を通信に用いることは古くから考えられ、多くの研究がある。しかし、本稿では記憶システムに対する容量の節約とそれに伴うアクセス時間の短縮を目的とし、あえて通信については考察しないでおく。

一般に記憶システムへのアクセスは、読み出しフェーズよりも書き込みフェーズの方が時間的に余裕がある。理由は、書き込みフェーズにおいてはアドレスとデータの両方を記憶システムに対して一方向に送り込むだけであるが、読み出しフェーズではアドレスを送った後に返事（データ）を待たなければならず、この応答時間は計算機システム全体の性能に大きく影響する為である。

実際の応用例

まず、計算機上のデータをテープデバイス等のストレージに納める場合を考えてみると、データの転送速度は計算機の性能に比べてかなり遅いので、特に速さを求められる場面は少ない。このため、テープ駆動装置内に圧縮機能を持っているものも多い。圧縮アルゴリズムは、ALDC(Adoptive Lossless Data Compression)という LZ77 法を変形したものが多く使われている。速さに関しては、ALDC 専用の LSI を用いたものが多く、と

くに問題にはなっていない。もちろん速いに越したことはないが、速さよりは圧縮率の向上の方が重要である。

次に磁気ディスク装置等の二次記憶に用いる場合を考える。一般の PC 上には、アクセス頻度の少ないファイルを自動的に圧縮する機能を持つ OS もあるが、Windows に使われているものは LZ77 法の変形である。しかし、現状での復元のコストを考えると、磁気ディスク上に収められる全てのファイルを圧縮する事は得策ではない。

主記憶に対して圧縮イメージを保存しアクセスするシステムは、IBM 社より製品化されたものがある（MXT (Memory Expansion Technology)）。これは、3 次キャッシュ（共有キャッシュ）メモリと主記憶の間で圧縮を行うもので、LZ77 符号および Huffman 符号を用い、専用のチップ(ASIC)上に実装してハードウェアで並列動作させている。CAM (Content Addressable Memory) を用い、圧縮／復元とも実時間で処理させている。復元にかかるオーバヘッドについては、共有キャッシュにヒットした場合は構造上オーバヘッドはないが、キャッシュミスの場合は圧縮なしの場合に比べ 1.5～4.3 倍程度の時間的オーバヘッドがある[6][7]。この値は決して小さいとは言えず、ハードウェアのコストに加え、このオーバヘッドがこの技術の普及を阻んでいるものと思われる。逆に、この点を解決できれば、実用になるものとして普及が期待できる。

以上、3 種類の記憶システムに対する圧縮技術の応用例を見たが、全て LZ77 法をベースとした圧縮を行っている。これは、圧縮手法の提案から 20 年以上が過ぎ、比較的安定したハードウェアやライブラリがある（しかし、ライブラリ zlib にセキュリティホールが昨年発見され、商用／非商用を問わず様々な OS やツールに影響を与えた事件は記憶に新しい）事と、圧縮率、速さ共かなり満足できる性能を達成している事が主な理由であろう。しかしながら、記憶システムに対する圧縮手法と見る場合、ファイルの始めの部分が後の部分に影響を与える可能性が常に残り、圧縮／展開ともファイルの始めの部分から順次行う必要がある。また、主記憶の様な比較的速い記憶デバイスに対する圧縮イメージの展開時間は、まだ満足なレベルにあるとは言い難いであろう。今後の圧縮技術の発展が期待される。

4. まとめと今後の展望

テキスト情報圧縮アルゴリズムの最近の動向を、文法式圧縮法を中心に紹介した。これは展開を速くできる点で有望である。また、記憶システムの上で実用となっている圧縮手法の現状について紹介し、問題点を指摘した。今後は、文法式圧縮法の理論的裏づけ、処理時間、とくに復元時間の高速化、それに関連して専用回路による実行系の提案などを考えている。

参考文献

[1] 武田，篠原，「圧縮されたテキスト上のパターン照合 －データ圧縮とパターン照合の新展開－」，情報処理 43 巻 7 号，763-769 頁，2002 年 7 月

[2] 山本，「ユニバーサルデータ圧縮アルゴリズムの変遷 －基礎から最新手法までー」，2001 年情報論的学習理論ワークショップ(IBIS2001)予稿集，339-348 頁，2001 年 8 月

[3] 喜田，「文法変換に基づく圧縮」，情報源符号化ワークショップ予稿集, 3-10 頁, 2002 年 11 月

[4] Gage, P., “A New Algorithm for Data Compression”, C/C++ Users Journal, Vol.12, No.2, 1994

[5] Gilchrist, J., “Archive Compression Test”, http://compression.ca/

[6] Tremaine, R.B., et al., “IBM Memory Expansion Technology (MXT)”, IBM Research and Development, Vol.45, No.2, pp.271-285, 2001

[7] Smith, T.B., et al., “Memory Expansion Technology (MXT): Competitive impact”, IBM Research and Development, Vol.45, No.2, pp.303-309, 2001