D-019

圧縮アルゴリズムによる大容量データの効率的検索に関する研究
Study on High Performance Retrieval from the Mass Data by using Compression Algorithms

比嘉 克也† Katsuya HIGA
長山 格‡ Itaru NAGAYAMA

1. はじめに

多くのデータベースアプリケーションにとって重要なデータ処理の一つとして，大容量データベースから任意のパターンを検索することが挙げられる．このような大容量データベースは，プロセッサや記憶装置等の計算機資源を節約する為に圧縮されて格納されることが多い．しかし，圧縮されたデータを一時的にメモリ等に展開してマッチングを行うと，多くの計算機資源が必要になる．また，圧縮されない通常のデータには冗長性が含まれているため，余計なマッチング処理を行わなければならない．そこで，圧縮されたデータに対し直接的なマッチングを行うことにより，データの展開処理やデータの冗長性を無くした高速な検索が行えるようにしたい．そのためには，圧縮されたデータに対する直接的なマッチングを行う技術が望まれる．

一般に，与えられたテキスト中に含まれる目的のパターンを検索するという文字列パターン照合問題は，文字列処理に関する最も基本的かつ頻繁に行われる処理の一つである．また，文字列のパターン照合は多くのデータベースにおける情報検索において最も重要な処理の一つである．現在は，遺伝子配列データ等の極めて長く大量なテキストデータを扱う機会も多くなり，目的のパターンを高速に検索する事の重要性が高まっている．

本研究では，高速な圧縮検索を実現するために，まず現存するいくつかの圧縮アルゴリズムにより圧縮されたデータに対する圧縮検索の可能性について比較検討する．

2. 圧縮データに対するパターンマッチング

通常，圧縮されたデータに対する情報の検索方法として以下の方法が挙げられる [1].

(A) 展開してから検索

(B) 展開しながら検索

(C) 展開せずに検索

(A) の「展開してから検索」とは，圧縮されたデータを一時的にメモリや記憶装置に展開し，それに対して検索を行う方法である．これは，最も単純かつ一般的な方法であるが，「展開に要する時間」＋「検索に要する時間」がかかるために非常に低速である．(B) の「展開しながら検索」は，展開プログラムに対し検索するルーチンを組込み，データを展開する側から検索を行う手法である．(C) の「展開せずに検索」は，今回の研究の目的となっており，圧縮されたデータに対し直接的に検索を行う方法である．

†琉球大学大学院理工学研究科, The Graduate School of Engineering and Science, University of the Ryukyus
‡琉球大学工学部, Department of Engineering, University of the Ryukyus

表 1: 圧縮パターン検索

圧縮法	圧縮パターン検索アルゴリズム
Run-length	Amir and Benson[2]
LZ78/LZW	Amir, et al.[3]
LZ77	Farach and Thorup[4]

これまでに上記の「展開せずに検索」的手法である圧縮されたテキストに対するマッチングに関する解法として，Amir と Benson らによる Run-length 符号化 [2], Amir, Benson, Farach による LZW 法 [3], Farach と Thorup による LZ77 法 [4] に対する圧縮パターン検索アルゴリズムが提案されて来た (表 1)．しかしながら，これらの研究では検索速度に関する問題が考慮されていなかった．

そこで，本研究では以下に述べる Run-length 符号化により圧縮されたデータに対する直接的な検索を行い，検索速度について考察すると共に高速な検索方法を提案する．

3. Run-length 符号化によるパターン検索

3.1 Run-length 符号化

本研究では，圧縮アルゴリズムを用いてデータの冗長性を無くし高速な検索を実現させることが目的である．そのために，今回は最も基本的な圧縮アルゴリズムである Run-length 符号化を用いてデータを圧縮し，それに対し直接的なマッチングを行う．

Run-length 符号化は，図 1 に示すようにデータ内で同じ値が並んでいる際に，その並びの個数を記録して行く方法である．この符合化法は，隣り合うデータが同一である個数が多い場合に有効な手法である．

3.2 検索手法

まず，ここでは Run-length 符号化により圧縮されたデータを値，その個数をデータ数と呼ぶことにする．また，検索される本体文字列を Text，検索する部分文字列を Key と呼ぶ．

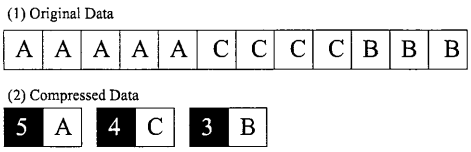

図 1: Run-length 符号化

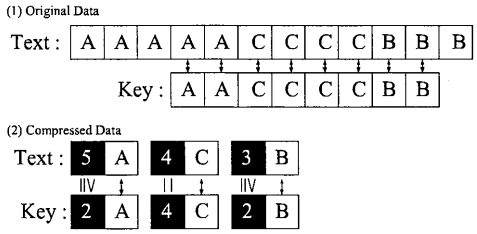

図 2: マッチング法

マッチングに用いるアルゴリズムは，最も基本的なブルートフォースマッチングとする．マッチングの際には，Key において以下の条件を用いて行う．

- **Key の先頭データと後尾データ**

$$\begin{array}{c}(\text{Text の値} = \text{Key の値}) \\ \text{and} \\ (\text{Text のデータ数} \geq \text{Key のデータ数})\end{array}$$

- **Key のその他部分**

$$\begin{array}{c}(\text{Text の値} = \text{Key の値}) \\ \text{and} \\ (\text{Text のデータ数} = \text{Key のデータ数})\end{array}$$

上記条件の概略を図 2 に示す．

4. 実験

4.1 実験条件

前節の手法を検証するために，圧縮されたテキストデータに対し直接的に検索する実験を行った．今回の実験は，全て Vine Linux 2.6 (kernel 2.4.22)，512MB，Intel Pentium III 1.2GHz 上で行った．また，実験に用いたデータはアルファベット"A","B"を一様乱数によって発生させた容量 1G byte のテキスト P と，Run-length 符号化によって圧縮率約 10%になる 1G byte テキスト Q を用いた．

4.2 実験結果

図 3 に一様乱数によって発生させたテキストデータ P に対する非圧縮，圧縮時の検索時間を示す．また，図 4 に Run-length 符号化によって圧縮率約 10%になるテキストデータ Q に対する非圧縮，圧縮時の検索時間を示す．

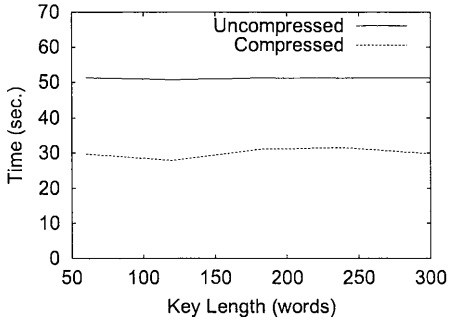

図 3: 一様乱数によるテキスト P に対する検索結果

表 2: 一様乱数によるテキスト P に対する検索結果 (秒)

Key 長	60	120	180	240	300
非圧縮時	51.380	50.810	51.299	51.335	51.299
圧縮時	29.608	27.779	30.921	31.413	29.697

5. まとめ

今回の研究では，Run-length 符号化により圧縮されたテキストデータに対する直接的な文字列検索を行った．明らかに Run-length 符号化によって圧縮率が高くなるテキストデータ Q は，非圧縮データに対し検索を行うより，圧縮データに対して行う方がより高速である．また，非圧縮データでは Key 長に比例し検索時間が増加して行くのに対し，圧縮データではほぼ一定の検索速度が得られた．

参考文献

[1] 竹田 正幸，篠原 歩:圧縮されたテキスト上のパターン照合，IPSJ Magazine Vol.43 No.7 pp.763-769(2002)

[2] Amir, A. and Benson, G.: Efficient two-dimensional compressed matching,Proc. Data Compression Conference, p.279(1992)

[3] Amir, A., Benson, G. and Farach, M.: Let sleeping files lie: Pattern matching in Z-Compressed files, J. Computer and System Sciences, Vol.52, pp.299-307(1996)

[4] Farach, M. and Thorup, M.: String-matching in Lempel-Ziv compressed strings, Algorithmica, Vol.20 No.4, pp.388-404(1998)

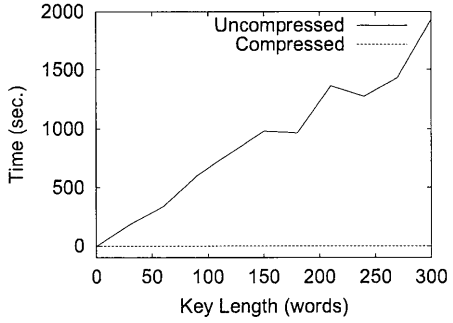

図 4: 圧縮率 10%のテキスト Q に対する検索結果

表 3: 圧縮率 10%のテキスト Q に対する検索結果 (秒)

Key 長	60	120	180	240	300
非圧縮時	341	794	966	1277	1936
圧縮時	0.447	0.448	0.448	0.475	0.569