順次インデックスファイルに対する差分圧縮法の具体的提案

程 亜非[†] 桧垣泰彦[††] 池田宏明[†]

差分圧縮法の考え方は古くからあるが，その具体的な実現法および性能は検討されていない．本論文は2次記憶装置上のソート済み順次インデックスファイルに差分圧縮を適用することにより，従来の汎用圧縮法に比べて高速アクセスが可能なことを示している．この目的には圧縮率に着目するだけでは不十分で，伸張時間も重要な指標であることに着目して，まず，圧縮対象の性質に基づく冗長性を最小にした圧縮情報と伸張手数の小さい圧縮ファイルフォーマットを提案し，それを用いた差分圧縮法を実現した．次に，提案した方法の有効性と限界を示すために，圧縮ファイル量，伸張処理量，およびプロセッサ速度とディスク速度の相互関係のモデルに基づいて代表的な汎用圧縮法との性能評価を行った．この結果，提案した差分圧縮法が有効であることを示した．具体例に応用した場合，提案した差分圧縮法の平均読み込み伸張時間が圧縮しない場合より32.5%短くなった．

Proposal of a Method of Differential Compression for Sequential Index Files

YAFEI CHENG,[†] YASUHIKO HIGAKI [††] and HIROAKI IKEDA [†]

For a short access time of sorted sequential index files in a secondary memory device, an implementation method of differential compression is proposed. In this method, an optimized description code of fixed length and a new compression file format were introduced. An evaluation model of decoding time was also discussed based on the relation among algorithm complexity, compression ratio, speeds of a disk and a processor. The proposed method was compared with the conventional compression methods with respect to access speed to give more feasible result than the others. It can, in average, save 32. 5% of the time required for accessing the index files incorporated in the experiments.

1. はじめに

データベースシステムではキー検索のために木構造，ハッシュ関数が用いられ，簡便のために，あらかじめソートされた順次インデックスファイル[1]も使われている．例えば，検索キーとインデックスの部分一致照合を行う場合などである．2次記憶装置上のソートされた順次インデックスファイルを用いて検索する場合は，検索時間には読み込みと文字列の照合時間が含まれる．検索時間を短縮するために文字列照合アルゴリズムの改善[2]のほか，2次記憶装置のアクセス時間の短縮も重要である．そのためには，ファイル圧縮が有効な手段と考えられる．圧縮したインデックスファイルを用いて検索する方式には2種類ある．一つは圧縮したインデックスファイルの伸張を行わず，そのまま用いる方法である．これについては，ある順序保存符号を用いて圧縮した順次インデックスを使用し，ハフマン符号[3]を用いた場合より検索速度を改善した報告がある[4]．この方法の問題点は部分一致検索ができないことである．

もう一つの方法は検索キーとインデックスの照合をする前に，圧縮されたインデックスの伸張を行う方法である．

本論文では部分一致検索機能を実現するために後者を対象にして，2次記憶装置上のインデックスファイルの読み込み伸張時間の短縮を検討する．この場合，圧縮率のみではなく，伸張時間も重要な評価指標となる．

汎用圧縮法としては，ハフマン符号法，算術符号法[5),6)]および *Lempel-Ziv* 符号法（以下LZ符号法と略記する）[7]などがある．ハフマン符号法と算術符号法は符号（例えばASCIIコード）の出現確率に基づいて，

† 千葉大学大学院自然科学研究科
Graduate School of Science and Technology, Chiba University
†† 千葉大学工学部
Faculty of Engineering, Chiba University

最適可変長符号で再符号化することによりファイルを圧縮する．伸張はビット単位処理となるので，伸張時の処理量は大きい．LZ符号法には，いくつかの変種がある．LZ符号法に基づいて実現した圧縮ツールの種類は多く，性能の良い圧縮法として評価されている[8]．

本論文で対象とするソートされた順次インデックスファイルに対しては，従来の汎用圧縮法のほかに差分圧縮も適用できる．しかし，具体的な実現方法や圧縮率および伸張速度などについて汎用圧縮法との性能比較は論じられていない[9)~13)]．そこで，本論文では，従来の方法に比べて有効な差分圧縮の具体的実現のために，圧縮対象のデータの性質を利用して，冗長性を最小にする最適複写長情報を用いることを提案する．また，複数個の複写長情報をブロック化することによって，圧縮ファイル中でバイト境界に整合しないデータ部分が複写長情報ブロックに限られるようにし，伸張処理をこの部分の処理のみに限定するという工夫によって，インデックスファイルの読み込み伸張時間を短縮する．

さらに，実現した差分圧縮法の伸張時間を検討して，伸張処理量，圧縮ファイル量，およびプロセッサ速度とディスク速度（処理環境）の相互関係を示し，定量的に検討する．また，従来の汎用圧縮法との対比実験も行った．これらの結果から，提案した差分圧縮法の限界も明らかにする．

2. 差分圧縮の原理

2.1 圧縮対象とその構造

図1 (a) はソートされ，かつ各インデックスがユニークである順次インデックスファイルの構造の一例である．<D>は各インデックスの区切りを表す情報であり，<P>は各インデックスに対応するポインタである．この構造ではポインタ部分のデータも読まなければならない．本論文では無駄となるポインタ部分を読み込む必要のない，図1 (b) のような構造を想定する．すなわち，インデックス語，ポインタ部分を別々のファイルに保存する．インデックス語とポインタには1対1の関係を持たせ，検索時に，ヒットしたインデックス語の順序番号に対応するポインタを直接ポインタファイルから得る．以下，図1 (b) のインデックス語をレコードと呼ぶ．インデックス語ファイルをファイルと呼ぶ．このファイルを圧縮対象とする．

2.2 典型的な差分圧縮法

差分圧縮法は，図1 (b) のようなファイルでは隣接したレコードの先頭から数文字が一致することが多いという特徴を利用する圧縮法である．

隣接した第 $k-1$ レコードと第 k レコードを次のように表現する．

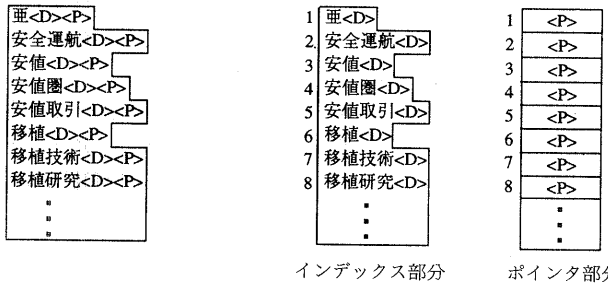
$k-1$：$x_1x_2...x_j...x_p$〈D〉

k：　$y_1y_2...y_j...y_q$〈D〉

ここで，x_j と y_j はバイト単位のデータであるとし，<D>はレコードの区切りを表す情報で，通常1バイトのコードである．第 $k-1$ レコードの先頭の部分列 $X_j=x_1...x_j$ が第 k レコードの先頭の部分列 $Y_j=y_1...y_j$ と同一ならば，Y_j をより短い情報（これを複写長情報〈C_L〉と呼ぶ）で置き換えることによってファイルを圧縮できる[9),10)]．こうした圧縮ファイルの複写長情報は，第1レコードを除いて常にレコードの先頭に置かれ，直前のレコードの区切り情報と隣接するから，伸張時には，複写長情報〈C_L〉は区切り情報〈D〉によって他の情報と識別される．両者をまとめて差分圧縮情報〈C_I〉（以下，圧縮情報と略記する）を考えれば，圧縮情報によって圧縮されたファイルのフォーマットは次のようになる．

$U_1C_{I_2}U_2C_{I_3}U_3...C_{I_{k-1}}U_{k-1}C_{I_k}U_k...C_{I_n}U_n$〈EOF〉

ここで，U は各レコードの圧縮できない部分のデータである．記号<EOF>は圧縮したファイルの終わり（end of file）情報である．



(a) 通常のインデックスファイルの例
(a) Indices and pointers are combined into one file.

(b) インデックスとポインタが分けられた場合の例
(b) Indices and pointers are divided into different files.

図1 ソート済みインデックスファイルの例
Fig. 1 Example of a sorted index file.

3. 差分圧縮の実現

3.1 提案する差分圧縮法の設計

キーとの部分一致の高速化には圧縮されたファイルの伸張時間が重要である．一般に良い圧縮率を実現するために複雑な圧縮法を採用すると，その代償として伸張時間がかかる．本論文で扱う対象においては，伸張時間の増大することは避けなければならない．ここでは，差分圧縮法を具体的に設計するにあたり新たに考慮した点について述べる．

3.1.1 固定ビット長複写長情報

図1のようなファイルを全体から見れば，各隣接レコードの共通部分のバイト数は0，1，…，lの範囲にある．0，1，…，lのそれぞれの出現確率によって，ハフマン符号[3]や算術符号[5),6)]などで符号化すれば，各レコードの複写長情報の冗長性を最小にすることができる．この場合，複写長情報のビット数は個々のレコードに依存し，一般にバイト境界に整合しないので，2.2節に述べた典型的な差分圧縮法をそのまま適用すると**図2**に例示したように圧縮できないデータU（図2(b)のC，D）もバイト境界に整合しなくなる．したがって，ビット単位で位置を調整する必要があり，伸張に時間がかかると考えられる．

そこで筆者らは圧縮率の改善に伴う伸張手数の増加を最小限に抑えるために，圧縮対象ファイルに対応した固定の最適複写長情報を使うことを考えた．複写長情報が$b(b\geq 2)$ビットと固定された場合は，圧縮できる最大バイト数が2^b-2となる*．第kレコードの実際の圧縮可能なバイト数がc'_kの場合，2^b-2を越える$c'_k-(2^b-2)$バイトが圧縮できない．つまり，bビットでは不足である．一方，圧縮可能なバイト数が2^b-2未満の場合は，bビットでは冗長である．いずれの場合も圧縮率を劣化させるので，bに最適値が存在することになる．

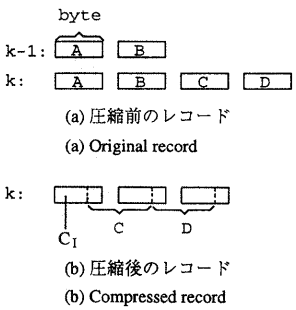


図2 バイト境界に合わない圧縮情報C_Iを使う場合の圧縮されたレコードの状況の例
Fig. 2 An example of compressd record where C_I does not match with the byte boundary.

* 数値2^b-1はファイルの終わりを識別するために予約しておくものとする．

複写長情報がbビットの場合，1レコードあたりの差分圧縮可能な平均ビット数Nはbの関数として次のように書ける．

$$N(b)=\frac{1}{n}\sum_{k=1}^{n}\{8c_k(b)-b\} \qquad (1)$$

ここで，nはレコード数であり，$c_k(b)$は複写長情報bによって実際に圧縮できるバイト数で，

$$c_k(b)=\begin{cases} c'_k, & c'_k\leq 2^b-2 \\ 2^b-2, & c'_k>2^b-2 \end{cases} \qquad (2)$$

である．式(1)の$N(b)$を最大とする$b=b_o$が最適複写長情報となる．

3.1.2 複写長情報のブロック化

具体的な圧縮対象に対して，式(1)と(2)で求められるb_oは必ずしも8ビットではなく，それより小さい．圧縮されたファイル中でバイト境界に整合しないデータ量を最小に抑えるためには，あるb_oに対して，正の整数iの集合

$$A=\{i \mid b_o i\equiv 0 \pmod 8\} \qquad (3)$$

を満たすi個のレコードの複写長情報を組み合わせてブロックとすればよい．ブロック化された複写長情報は

$$w=\frac{1}{8}b_o i \text{（バイト）} \qquad (4)$$

である．このブロックをBと書けば，圧縮したファイルフォーマットは以下のようになる．

$$HB_1U_1\cdots U_iB_2U_{i+1}\cdots U_{2i}$$
$$\cdots B_mU_{(m-1)i+1}\cdots U_{mi}\cdots$$

ここで，Hはb_o，iの情報を記述したヘッダであり，Uは各レコードの圧縮できない部分のデータである．<EOF>は最後のBに置かれる．Bはバイト境界に整合しているので，すべてのUもバイト境界に整合する．**図3**にはこれを$b_o=6$として例示した．

このフォーマットでは，圧縮ファイル中のバイト境界に整合しない部分はブロックBに押し込められているから，不整合が他に波及しない．これによって，良い圧縮率を得ると同時に，伸張アルゴリズムが複雑になることも抑えることができると考えた．

3.1.3 区切り情報

典型的な差分圧縮法における区切り情報は二つの機能を持っていた．すなわち，伸張時の複写長情報位置

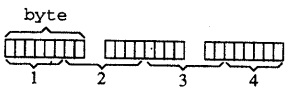


図3 4個の複写長情報を3バイトに合成した例
Fig. 3 An example of 4 C_L's combined into 3 bytes ($b_0=6$).

検出と，レコードの区切りの機能である．新しく提案した方法では，i 個のレコードを伸張すれば，次のデータが圧縮情報ブロック B であることは自明であるから B と U の区切り情報は必要ない．一方，U の列は可変長バイト列であるので，各 U には区切り情報が必要である．ここでは，2個の7ビットコードである JIS 漢字コードと ASCII コードで表されるテキストデータを仮定し，各 U の最後のバイトの MSB を1に設定することによって区切りを識別することにした．

3.2 圧縮と伸張アルゴリズム

次に本論文で新しく提案する圧縮と伸張アルゴリズムの概要を示す．この方法は2パス方式である．パス1は圧縮対象ファイルのデータの性質を調べて最適ビット長 b_o を求める処理を実行し，圧縮アルゴリズムの Step 1 に対応する．パス2では下のアルゴリズムの Step 2～Step 6 に記述したように，具体的な圧縮を行う．B_{max} は圧縮ファイルの複写長情報ブロック B を格納するバッファ buf 0 の大きさである．隣接のレコードを格納するバッファは buf 1 と buf 2 である．

圧縮アルゴリズム

Step1：圧縮対象ファイルについて，式(1)と式(2)によって，b を1～8の間に変化させて，N を最大にする最適複写長情報 b_o を求める．

Step2：求めた b_o と式(3)によって，複写長情報ブロック B のサイズ w が B_{max} を越えない最大の $i \in A$ を決める．b_o，i の値を圧縮ファイルのヘッダ H に書き込む．buf 1 を初期化する．

Step3：圧縮ファイルの現在の位置から大きさ w の空間を用意する．

Step4：i 個のレコードの差分圧縮を行う．

1. 圧縮対象ファイルから一つのレコードを buf 2 に読み込む．もしファイルの終わりなら，Step 6 へ行く．そうでなければ，buf 1 のレコードと比較して，$2^{b_o}-2$ を越えない圧縮可能なバイト数を b_o ビットの複写長情報で表し，これを複写長情報バッファ buf 0 に格納する．
2. 圧縮できない部分 U の最後のバイトの MSB を1にセットして，U を圧縮ファイルの現在の位置以後に書き込む．buf 2 のレコードで buf 1 の内容を更新する．
3. もし i 個レコードの圧縮が終わりなら，Step 5 に行く．そうでなければ，Step 4 の 1. へ．

Step5：現在の圧縮ファイルの書き込む位置を記憶し，Step 4 で buf 0 に格納した i 個のレコードの複写長情報ブロック B を Step 3 で用意した圧縮ファイル中の空間に書き込む．記憶した書き込む位置に戻ってから，Step 3 へ．

Step6：圧縮ファイルの終わり情報 $2^{b_o}-1$ を複写長情報バッファ buf 0 に格納する．さらに，Step 3 で用意した圧縮ファイルの空間に書き込む．終了．

次に，この圧縮アルゴリズムに対応した伸張アルゴリズムの概要を述べる．ここで，buf 0 は伸張されたレコードを格納するバッファであり，また，buf 1 は読み込まれた複写長情報ブロック B を格納するバッファである．さらに，buf 2 は buf 1 の b_o ビット長の各レコードの複写長情報をバイト単位のデータに伸張して格納するバッファで，p_0，p_2 はそれぞれ，buf 0，buf 2 のポインタである．

伸張アルゴリズム

Step1：圧縮ファイルのヘッダ H を読み込んで，パラメータ w，b_o および i に代入する．また，$p_2 \Leftarrow i$．

Step2：もし $p_2 \neq i$ なら，Step 4 へ．

Step3：圧縮ファイルから複写長情報ブロック B を buf 1 に読み込む．buf 1 の各 b_o ビット長のデータをバイト単位のデータに伸張して buf 2 に格納．また，$p_2 \Leftarrow 0$．

Step4：buf 2 の p_2 番目の複写長情報を p_0 に代入．

Step5：一つのレコードを伸張．

1. buf 0 の p_0 が指す位置に1バイトのデータを読み込む．
2. それが区切り情報を含んでいれば Step 6 へ，そうではない場合は $p_0 \Leftarrow p_0+1$ として Step 5 の 1. へ．

Step6：出力と次の複写長情報の判断．

1. 区切り情報を消す．
2. buf 0 を出力する．
3. $p_2 \Leftarrow p_2+1$．
4. もし buf 2 の p_2 番目の情報が圧縮ファイルの終わり情報 $2^{b_o}-1$ なら終了．そうではない場合は Step 2 へ．

4. 圧縮率および読み込み伸張時間の総合評価

4.1 前　　提

新しく提案した差分圧縮法評価のための圧縮と伸張実験は，大規模な全文書（新聞記事，書誌情報，用語解説）から機械的に求めた[14]性質の異なる複数のインデックスファイル（News 1～3，Book 1～2，Term 1～3），UNIX-BSD 版の英語単語ファイル web 2* および UNIX のツール spell の辞書ファイル words を対象とした．提案した差分圧縮法との比較のために，ハフマン符号法，算術符号法** と LZ 符号

* Webster's Second International

** ハフマン符号法と算術符号法の符号数は 256 とする．

法* を用いた．すべての方法はC言語でコーディングした．

4.2 圧縮率と圧縮時間

表1には各圧縮法の圧縮率および参考のために平均圧縮時間** を示す．ここで，S_u は圧縮前のファイル量，r は圧縮率で，$r=S_c/S_u$ (S_c は圧縮後のファイル量）である．

この結果は，提案した差分圧縮法（DC）の圧縮率はハフマン符号法（Huffman）と算術符号法（Arith）の圧縮率より良いが，LZ符号法（LZ）の圧縮率より多少劣ることを示している．しかし，後に述べるように本来の目的である読み込み伸張時間においては，提案した差分圧縮法が優れている．

圧縮時間は，本論文の目的では，本質的でないが，提案した差分圧縮法は他の方法に比べて短い．

4.3 読み込み伸張時間のモデル

本論文では圧縮手法を用いて2次記憶装置上の圧縮されたファイルのアクセス時間を短縮することを目的としているから，単純なアクセス時間と伸張時間を含む読み込み伸張時間を考慮しなければならない．読み込み伸張時間は，ディスク，CPU，計算機のアーキテクチャ，コンパイラおよびOSなどの環境に依存する．しかし，これらをすべて変数として評価するのは困難であるから，本論文では同じアーキテクチャの計算機とコンパイラおよびOSのもとに評価モデルを与え，評価を行う．

表1 圧縮率と平均圧縮時間の比較
Table 1 Compression ratio and average compression time of each method.

file	S_u (byte)	圧縮率 r			
		DC(b_0)	LZ	Huffman	Arith
News 1	2,988,200	0.448,(4)	0.417	0.779	0.717
News 2	1,635,113	0.407,(4)	0.404	0.691	0.632
News 3	224,031	0.538,(3)	0.542	0.696	0.669
Book 1	520,890	0.532,(4)	0.441	0.764	0.733
Book 2	305,308	0.438,(4)	0.370	0.844	0.531
Term 1	902,408	0.462,(4)	0.437	0.773	0.720
Term 2	266,014	0.434,(3)	0.431	0.692	0.637
Term 3	88,876	0.468,(4)	0.401	0.542	0.529
web 2	2,477,182	0.354,(4)	0.296	0.557	0.514
words	201,039	0.430,(4)	0.376	0.569	0.527
圧縮時間(s/Mbyte)		8.61	26.80	27.11	33.41

* LZ符号に基づいて実現した圧縮ツールはgzip, lha, compressなどがある．そのうちgzipの圧縮率が最も良くかつ伸張時間も最短であるので，差分圧縮とLZ符号の対照実験をgzipで行った．

** 平均時間 $t=\sum_{i=1}^{m}t_i/\sum_{i=1}^{m}S_{u_i}$，$t_i$ は各ファイルの圧縮時間である．

単純なディスクアクセス時間は圧縮されたファイル量に比例する．伸張時間は伸張アルゴリズムの複雑さに関係する．また，ディスク装置とプロセッサの速度などの環境パラメタも読み込み伸張時間に影響を与える．ここではファイルアクセス時間，すなわち，読み込み伸張時間を以下のようにモデル化した．

読み込み伸張時間 T をとすると，

$$T=T_c+T_d \tag{5}$$

ここで，T_d はディスクから圧縮ファイルを読み込む時間である．T_c は読み込まれたデータを伸張する時間である．T_d はディスクのシーク時間を含む平均読み込み時間 t_d に，T_c はプロセッサの単位処理実行時間 t_c に比例する．すなわち，

$$T_c=pt_c \tag{6}$$

$$T_d=S_ct_d \tag{7}$$

ここで，p は総伸張処理量，S_c は圧縮後のファイル量である．式(5)～(7)から，

$$T=pt_c+S_ct_d \tag{8}$$

となる．表1に例示したファイルに対する各圧縮法の T をファイル量 S_c に対して実測した例を**図4**に示す．いずれも T と S_c は比例関係にあることがわかる．したがって，式(8)は以下のように書ける．

$$T=(\alpha t_c+t_d)S_c \tag{9}$$

ここで，$\alpha=p/S_c$ はバイト当たりの伸張処理量であり，以下では伸張手数と呼ぶ．図4の直線の傾きを q とすると，

$$q=\alpha t_c+t_d \tag{10}$$

である．

圧縮法に依存する伸張手数 α は式(9)，(10)に最小二乗法を用いて求められる．具体的には式(10)の α と

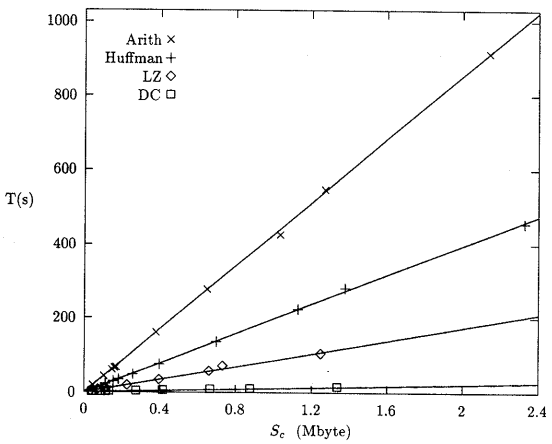


図4 各圧縮法の読み込み伸張時間の例
Fig. 4 Examples of decoding time by different compression methods.

表2 各圧縮法の伸張手数
Table 2 Decoding complexity of each method.

圧縮法	α
DC	0.0460
LZ	0.4096
Huffman	0.9469
Arith	2.1050

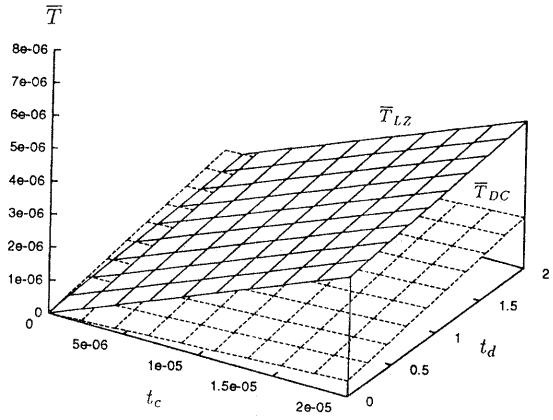


図5 t_c, t_d に応じた $\bar{T}_{DC}$(点線), $\bar{T}_{LZ}$(実線)の変化
Fig. 5 Variations of $\bar{T}_{DC}$, $\bar{T}_{LZ}$ in terms of t_c and t_d.

t_d を未知数として求めるために，条件の異なるプロセッサ(t_c)の環境での q を知る必要がある．t_c はプロセッサ速度評価に適したベンチマークプログラム[15]を用いて測定できる．通常，t_c は MIPS* で表されるが，ここではベンチマークプログラムの基準単位の逆数 s/dhry を使用する．したがって，S_c の単位をバイトとすれば，α の単位は dhry/byte である．t_d の単位は s/byte である．

本論文では五種類の PC のプロセッサ（メモリ 640 Kbyte，MS-DOS Ver. 3.30）および一種類のディスク（平均シーク時間 18 ms）の環境で，ディスク上に連続に記録した表1に例示した各ファイルに対して，各圧縮法に対応したプログラムの読み込み伸張時間 T を実測した．プログラムは Microsoft C Ver. 5. 10 でコンパイルしたものである．各プロセッサの t_c は C 言語版 Dhrystone Ver. 1. 1 を用いて測定した．求めた各圧縮法の伸張手数 α を**表2**に示す．使用したディスクの t_d は約 2 s/Mbyte であった．

4.4 読み込み伸張時間の評価

具体的な圧縮ファイル量と関係なくするために，読み込み伸張時間 T の評価モデル式である式(9)を元のファイル量 S_u で割ると，

$$\bar{T}=(\alpha t_c+t_d)r \tag{11}$$

$\bar{T}$ は S_u で正規化した1バイトの平均読み込み伸張時間，r は圧縮率である．式(11)を用いて各圧縮法の平均読み込み伸張時間 $\bar{T}$ を評価する．

4.4.1 提案した差分圧縮法と LZ 符号法の読み込み伸張時間の比較

表2によれば提案した差分圧縮法（DC）の伸張手数が小さい．一方，表1によればLZ符号法（LZ）の圧縮率がよい．性能比較上，最もきびしい条件を求めるために，両方法の圧縮率の差の一番大きな場合（Book 1）について，提案した差分圧縮法（DC）と LZ 符号法（LZ）の1バイトの平均読み込み伸張時間を $\bar{T}_{DC}$, $\bar{T}_{LZ}$ として，t_c, t_d に応じた $\bar{T}_{DC}$, $\bar{T}_{LZ}$ の変化を**図5**に示す．実線の平面は $\bar{T}_{LZ}$ で，点線の平面は $\bar{T}_{DC}$ である．両平面の交線は二つの圧縮法の読み込み伸張時間の転換条件を与える．式(11)によって，$\bar{T}_{DC}\leq\bar{T}_{LZ}$ のための条件は

$$t_c\geq\frac{\gamma_{DC}-\gamma_{LZ}}{\alpha_{LZ}\gamma_{LZ}-\alpha_{DC}\gamma_{DC}}t_d \tag{12}$$

である．具体的には，$t_c\geq 5.83\times10^{-7}t_d$ である．本実験で用いた t_d=2s/Mbyte のディスクを使う場合は，$t_c\geq 1.17\mu$s/dhry で，提案した差分圧縮法の優位性が発揮できる．この条件は測定に使用した最高速プロセッサ（$t_c=2.08\times10^{-4}$）の約 178 倍であり，十分に余裕を持った条件であると考えられる．したがって，提案した差分圧縮法の読み込み伸張時間が，最悪の場合でも短いことが明らかになった．

4.4.2 提案した差分圧縮法とハフマン符号法および算術符号法の読み込み伸張時間の比較

提案した差分圧縮法で得た圧縮ファイルの伸張手数はハフマン符号法と算術符号法のそれより小さい．しかも，提案した差分圧縮法の圧縮率はハフマン符号法と算術符号法より良いので，提案した差分圧縮法のディスクの読み込み時間 T_d と伸張時間 T_c の両方ともハフマン符号法と算術符号法の場合より優れていることがわかった．

4.5 総合評価実験

表3は UNIX ワークステーション* のシングルユーザモードの環境で，各圧縮法の読み込み伸張時間と圧縮しない場合のファイルの読み時間の対比実験の結果である．時間圧縮率は（読み込み伸張時間）/（圧縮しない場合の読み込み時間）で計算したものである．提案した差分圧縮法の時間圧縮率が最も良いことが判った．圧縮しない場合の各ファイルの読み込み時間を重みとする重み付き平均時間圧縮率で評価すれば，圧

* million instructions per second

* DECstation 5000/25, Ultrix-4.2

表3 時間圧縮の比較
Table 3 Decoding time of each method.

file	圧縮なし (s)	時間の圧縮率			
		DC	LZ	Huffman	Arith
News 1	4.03	0.672	1.667	6.11	8.24
News 2	2.17	0.691	1.677	5.92	7.37
News 3	0.31	0.935	2.065	5.90	7.63
Book 1	0.74	0.757	1.635	6.17	8.76
Book 2	0.43	0.744	1.628	6.03	5.15
Term 1	1.18	0.746	1.822	6.53	8.93
Term 2	0.38	0.789	1.763	5.18	6.67
Term 3	0.15	0.867	1.667	4.29	5.42
web 2	3.35	0.558	1.349	5.69	7.54
words	0.30	0.800	1.667	4.91	5.87

縮しない場合に比べて32.5%改善された．なお，LZ符号法，ハフマン符号法と算術符号法の読み込み伸張時間は圧縮しない場合の読み込み時間より長くなり，これらの方法による圧縮が逆効果であることも示している．

以上をまとめると，提案した差分圧縮法によって，良い圧縮率を得ると同時に伸張アルゴリズムが複雑になるという問題を解決した．この結果，部分一致検索にも対応できるインデックスファイルのアクセス時間短縮に関して，汎用圧縮法に比べて時間圧縮率効果が高い新しい差分圧縮法を得た．つまり，提案した差分圧縮法は所期の目的に有効と考えられる．

5. おわりに

本論文は部分一致検索を目的とした2次記憶装置上のソートされた順次インデックスファイルを対象として，その読み込み伸張時間を短縮することを目的とした差分圧縮の具体的な実現法を提案し，その性能を汎用圧縮法と比較した．その結果，新たに提案した圧縮法が有効であることを示した．本論文で提案した方法の特徴は，(1)圧縮対象のデータの性質によって，冗長性を最小にする固定ビット長の複写長情報を用いたこと，(2)伸張手数を最小に押さえるために，複数の複写長情報をブロック化した圧縮ファイルフォーマットを設計したことである．

提案した方法を各種のインデックスファイルに適用した結果，読み込み伸張時間を短縮できることを明らかにした．なお，従来の圧縮法を適用した場合には，逆に読み込み伸張時間が増大することも示した．

また，圧縮されたファイルの読み込み伸張処理時間モデルに基づいて伸張手数を検討した結果，本論文で具体的に提案した差分圧縮法の限界をCPU処理能力とディスクアクセス時間との関係として示すと同時に，通常の処理環境では提案した方法が有効であることを明らかにした．

これにより，読み込み伸張時間は圧縮しない場合の読み込み時間に比べて，平均的に32.5%の短縮が図られた．

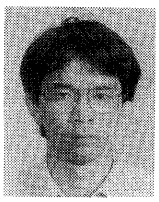
謝辞 日本コロムビア（株）小泉利雄氏には本研究の初期の段階で，便宜を提供していただいた．ここに感謝の意を表する．

参考文献

1) Knuth, D.: *The Art of Computer Programming,* Vol. 3, Addison Wesley, Mass. (1973).
2) Boyer, R. S. and Moore, J. S.: A Fast String Searching Algorithm, *Comm. ACM,* Vol. 20, No. 10, pp. 762-772 (1977).
3) Huffman, D. A.: A Method for Construction of Minimum-Redundance Codes, *Proc. IRE*, Vol. 40, No. 9, pp. 1098-1101 (1952).
4) 中津楢男：順序保存符号とその情報検索への応用，情報処理学会論文誌，Vol. 34, No. 2, pp. 312-319 (1993).
5) Rissanen, J. J.: Generalized Kraft Inequality and Arithmetic Coding, *IBM J. Res. Dev.*, Vol. 20, No. 5, pp. 198-203 (1976).
6) Witten, I. H., Neal, R. M. and Cleary, J. G.: Arithmetic Coding for Data Compression, *Comm. ACM*, Vol. 30, No. 6, pp. 520-540 (1987).
7) Ziv, J. and Lempel, A.: A Universal Algorithm for Sequential Data Compression, *IEEE Trans. Inf. Theory,* Vol. 23, No. 3, pp. 337-343 (1977).
8) Bell, T. C., Cleary J. G. and Witten, I. H.: *Text Compression,* Chapt. 8, Prentice-Hall (1990).
9) Date, C. J.: *An Introduction to Database Systems,* 3rd edition, Chapt. 2, Maruzen (1984).
10) Martin, J.: *Computer Database Organization,* 2nd edition, Chapt. 29, Prentice-Hall, Englewood Cliffs, N. J. (1977).
11) Mark, A. R. and Scott, J. V.: The Advantages of Database Compression, *Data Programming and Design,* Vol. 6, No. 7, pp. 54-60 (1993).
12) Bassiouni, M. A.: Data Compression in Scientific and Statistical Databases, *IEEE Trans. Softw. Eng.*, Vol. SE-11, No. 10, pp. 1047-1058 (1985).
13) 程 亜非，桧垣泰彦，池田宏明：部分一致検索用インデックスファイルの差分圧縮，信学技報，DE 90-19-23, pp. 1-8 (1990).
14) 桧垣泰彦，池田宏明：字種分類インデキシングによるフルテキスト検索システム（CLAX），著作権法施行令第24条の規定によるプログラム登録番号P第2148号-1 (Apr. 26, 1990).

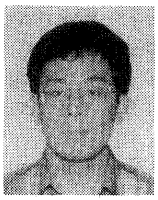
15) Weicker, R. P.: DHRYSTONE: A Synthetic System Programming Benchmark, *Comm. ACM*, Vol. 27, No. 10, pp. 1013-1030 (1984).

(平成5年10月27日受付)
(平成7年 6 月12日採録)



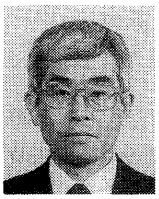
桧垣 泰彦（正会員）

昭和57年千葉大学工学部電子工学科卒業．59年同大学大学院工学研究科電子工学専攻修了．同年千葉大学工学部電子工学科（現電気電子工学科）助手となり現在に至る．インターネット上の情報システムに興味を持っている．電子情報通信学会会員．



程 亜非（学生会員）

昭和31年生．昭和57年中国北京工業大学第二分校計算機ソフトウェア工学科卒業．同年中国科学院計算技術研究所入所．平成3年千葉大学大学院修士課程修了．現在同大大学院博士課程在学中．データ工学，分散環境での情報検索および計算機ネットワークに興味をもつ．電子情報通信学会会員．



池田 宏明（正会員）

昭和17年生．昭和41年千葉大学工学部電気工学科卒業．昭和43年千葉大学大学院工学研究科電子工学専攻（修士課程）修了．同年同大学工学部助手となり，現在同大学教授．この間，AD変換方式の研究により東京工業大学より工学博士の学位を取得．その後，電子回路，電子計測，非数値解析，データ工学，色彩工学などの教育研究に従事．現在はマルチメディアシステムに興味を持っている．電気学会，電子情報通信学会，テレビジョン学会，IEEE各会員．