転置ファイルおよび接尾辞配列の効率的圧縮法

定兼邦彦† 今井浩†

単語ブロックソート圧縮法を提案する．これは文書と全文検索のための索引を圧縮する方法であり，圧縮データから転置ファイルを高速に生成できる．文書は圧縮時に単語に区切られるため，復号時には形態素解析などの時間のかかる処理は必要ない．これにより，全文検索のための索引を保存する際のディスク容量やネットワークを介して転送する際の負荷を減らすことができる．htmlに対する実験から，圧縮率はgzipよりも良く，圧縮データから転置ファイルを生成する時間は転置ファイルを0から作るよりも短く，形態素解析にかかる時間を含めると5倍以上速いことを確認した．また，単語ブロックソート圧縮法よりも圧縮率の良い通常のブロックソート圧縮法で圧縮された文書から単語を切り出し転置ファイルを生成するアルゴリズムも提案する．

Efficient Compression of Inverted Files and Suffix Arrays

KUNIHIKO SADAKANE† and HIROSHI IMAI†

We propose word-based block sorting, which is used for compressing both texts and their full-text indexes, inverted files. Since texts are separated into words, morphological analysis, which is time consuming, is not necessary in the decoder. By using the proposed compression scheme, we can reduce space for storing full-text indexes and a load for transferring them via network. We confirmed by experiments that our compression scheme has better compression ratio than gzip and creating the inverted file from compressed data is faster than creating it from scratch. Furthermore, this is more than five times faster if time for morphological analysis is included. We also propose an algorithm for creating an inverted file from a compressed file by the ordinary block sorting which has better compression ratio than the word-based block sorting.

1. はじめに

Webページなどの大量のテキストデータからの検索のためにインデックスを作成する方法は良く用いられている．これらのテキストやインデックスのサイズは大きいため，保存する際やネットワークで転送する際には圧縮しておく方が良い．テキストの圧縮にはgzipが良く用いられるが，それよりも圧縮率の良いものはいくつか提案されている．

検索のためにデータを転送するものではWeb検索エンジンが有名であるが，そのデータ量が問題になっている．山名ら[16]はWebページを収集するロボットを複数配置し，各ロボットは自分の近くにあるサイトのページを収集し，それを圧縮して検索サーバに転送する方法を提案している．検索サーバでは圧縮されたhtmlファイル群を伸長し，それらに対して索引を作成する．圧縮には一般的なgzipが使われているが，圧縮にブロックソート圧縮法[4]を用いることにより圧縮率を上げ，ネットワークの負荷を減らすことが考えられる．ブロックソート圧縮法を用いることのもう1つの利点は，圧縮された文書を復号する際に，文書自身とその文書に対する接尾辞配列が得られるため，全文検索のための索引を高速に作成できる[11]という点である．これにより，各ロボットでのファイルの圧縮にかかる時間は増大するが，検索サーバで索引を作成するときの負荷を大幅に減らすことができる．

本論文ではこの方法を拡張し，復号時に文書の転置ファイルが得られるような文書圧縮法を提案する．これは単語を単位としたブロックソート圧縮法であり，単語ブロックソート圧縮法と名付ける．圧縮後のサイズは転置ファイルの半分以下であり，また文書のみをgzipで圧縮するよりも小さい．ブロックソート法の復号にかかる時間は短

† 東京大学理学系研究科情報科学専攻
Department of Information Science, University of Tokyo
{sada,imai}@is.s.u-tokyo.ac.jp

いため，文書から転置ファイルを作成する時間よりも高速であり，線形時間で行なえる．また，転置ファイルを作成するには形態素解析を行ない文書を単語に区切らなければないが，単語ブロックソート法では文書は圧縮時に単語に区切られているため，復号時には形態素解析は必要ない．この圧縮法により，全文検索のためのインデックスを効率良く圧縮保存することができる．

2. 転置ファイルと接尾辞配列

2.1 転置ファイル

転置ファイルとは，文書中に現れる各単語に対してその出現位置のリストを格納したものである．出現位置は単語ごとに連続した領域に置かれ，その領域の先頭へのポインタを単語ごとに保存しておく．これを単語表と呼ぶことにする．単語の出現位置はソートしてあるため，位置を隣の値との差で符号化することで索引のサイズを小さくすることができる．なお，転置ファイルでは単語の出現位置を単に文書番号のみにすることで索引のサイズをさらに小さくできるが，本論文では単語の近接検索 (proximity search)[12] を行なえるようにするため文書中の位置まで保存することにする．

転置ファイルの長所としては，検索の際のディスクの読み込み回数が少いこと，単語の先頭位置のみに索引を作るためサイズが小さくなること，単語のポインタを隣との差分で符号化できるため索引をさらに小さくできること，単語の出現位置がソートされているため近接検索が行ないやすいという点がある．

2.2 接尾辞配列

文字列の接尾辞 (suffix) とは，文字列 $T[1..n]$ から先頭の何文字かを除いた残りの文字列 $T[i..n]$ のことである．文字列の全ての接尾辞を表すデータ構造があれば，文字列の任意の部分文字列の検索を高速に行なうことができる．接尾辞配列 (Suffix array)[8] は，部分文字列検索のためのデータ構造である．これは Suffix tree[9),15] よりも省メモリであるため，大規模なデータに対して使われている．文字列 T の接尾辞配列とは，T の全ての接尾辞 $T_i = T[i..n]$ を辞書順にソートしたときの接尾辞の添字 i を並べた配列である．T の任意の部分文字列へのポインタは接尾辞配列上で連続した領域に格納されているため，それらは二分探索で求めることができる．

接尾辞配列の特徴は，どんな部分文字列でも簡単に検索できるという点である．転置ファイルでは文字列を単語に区切ってから索引を作るため，任意の部分文字列の検索が難しくなる．検索したい文字列を含む全ての単語を列挙し，それらの出現位置を合わせたものを求める必要がある．よって，高速な検索のためには文書が正しく単語に区切られている必要がある．

3. ブロックソート圧縮法

3.1 オリジナルの方法

ブロックソート圧縮法[4] は文字列の圧縮法で，現在最も圧縮率の良いと言われている PPM 法[5] にせまる圧縮率を達成し，速度はこれらの圧縮法よりも高速である．現在では bzip2[14] というプログラムが公開され，普及しつつある．ブロックソート圧縮法は compress や gzip で使われている Ziv-Lempel 系の圧縮法とは異なり，圧縮する文字を並び替えることにより文字列を圧縮しやすくしてから単純な方法で圧縮するものである．ブロックソート法は Burrows-Wheeler 変換 (BW 変換，BWT)，move-to-front 変換 (MTF)，符号化の 3 段階に分けられる．

3.1.1 BW 変換

図 1は文字列 BANaNa の BW 変換とその逆変換を表している．まず，圧縮したい文字列を 1 文字ずつ左に巡回させた文字列を作り，それらを辞書順にソートする．次に，辞書順にならんでいる文字列の最後の文字を取ると BaaANN となり，これが BW 変換後の文字列 W となる．この文字列は元の文字列の並び替えになっている．また，元の文字列がソートした文字列中で何番目かという数字も復号に必要になる．図 1では元の文字列が右の行列中の上から 2 番目にきている．

復号時にはこの逆変換を行なうが，それは次のようになる．まず，BWT 後の文字列 W 中の文字をアルファベット順に並べ，文字列 Y を作る．これは各文字の頻度を数えることで線形時間で行なえる．次に，アルファベット中の各文字に対し，W の中で上から i 番目のものと Y の中で上から i 番目のものをリンクする．最後に，元の文字列が辞書順で何番目かがわかっているので，そこからリンクをたどれば元の文字列が復元される．これで復元できる理由は以下の通りである．図で同じ行にある W と Y の文字は右向きの矢印で結ばれているが，これは元の文字列中でそれら

の文字が並んでいることを表している．Y 中の2つの a は W 中の a と結ばれているが，a の次の文字は B と N であるため，どちらの a と結ぶかを決めなければならない．しかし，Y 中の文字はソートの定義よりそのあとに続く文字列の辞書順で並んでおり，この文字列の辞書順は先頭の a を取っても変わらないため，Y 中の上の a の右が B，下の a の右が N であることがわかる．よって，Y と W の同じ文字は上から順に結んでいけばよい．

BW 変換には文字列のソートが必要なため時間がかかるが，その逆変換は線形時間で行なえるため，ブロックソート圧縮法は圧縮よりも伸長が高速という特徴がある．

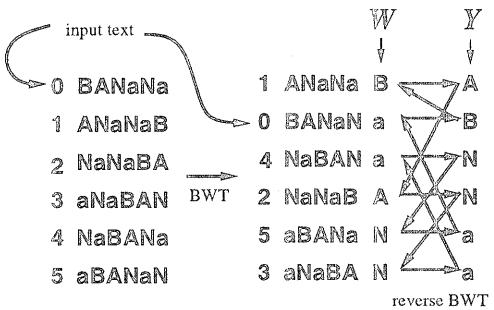

図1 BWT と逆BWT

3.1.2 MTF 変換

move-to-front 変換とは，文字列中の文字をランクと呼ばれる数字に変換することである．ランクには recency rank と interval rank がある[3]．ランクは1以上の整数である．文字 x_i と等しい文字で i より左にあるもののうち最も右のものを x_j とすると，x_i の interval rank は $i-j$ になり，recency rank は x_j から x_{i-1} の間の異なる文字の数になる．つまり文字の recency rank は interval rank よりも大きくなることはない．図 2は前節の BW 変換の結果の文字列 W = BaaANN を interval rank と recency rank に変換した結果を表している．なお変換時には W の左にアルファベット中の全ての文字を決まった順番で含む文字列があると仮定している．

BW 変換後の文字列に対して MTF 変換を行なうと，ランクの値は小さいものに偏るが，その理由は以下の通りである．BW 変換前の文字列中によく現れる部分文字列は BW 変換により連続した領域に集められるため，BW 変換後の文字

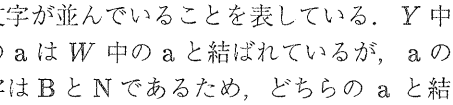

図2 interval rank と recency rank

列中にはその部分文字列中の同じ文字が並ぶようになる．つまり BW 変換後の文字列中には同じ文字が並んでいる場所が多くなる．このような文字列に MTF 変換を行なうと同じ文字が並んでいる部分は文字の種類に関係なくランクが1のものが並ぶようになり，全体的にランクが1のものが多くなる．

3.1.3 符 号 化

オリジナルのブロックソート法ではランクは recency rank が使われ，ハフマン符号や算術符号で圧縮されるが，これらの符号を決めるにはランクの頻度が分かっていなければならない．これとは別に，δ 符号[6] などの初めから決まっている符号を用いる方法も考えられる．δ 符号を用いると，ランク r は $1+\lfloor\log_2 r\rfloor+2\lfloor\log_2(1+\log_2 r)\rfloor$ ビットで表される．

bzip2 ではランクは連長圧縮をした後にハフマン符号で圧縮されている．

3.2 次数制限ブロックソート法

Schindler[13] は BW 変換での文字列のソートを各文字列の先頭の s 文字の比較だけで行ない，圧縮を高速にする方法を提案している．先頭の s 文字が等しい場合は，文字列は出現位置順に並べられる．s が4程度でも十分な圧縮率を達成している．次数を制限した逆 BW 変換はオリジナルの逆 BW 変換よりは遅くなる．ただし次数1の場合は逆 BW 変換はオリジナルの場合とあまり差はない．

3.3 修正 BW 変換

Sadakane[10] は接尾辞配列を圧縮するための修正 BW 変換 (MBWT) を提案している．図 3はその例である．まず変換前の文字列 T の各文字の同一視 (unify) を行ない，文字列 U を作る．文字の同一視とは，アルファベット $\mathcal{A}$ から別のアルファベット $\mathcal{B}$ への変換である．ここでは $\mathcal{B}$ は8ビットの文字，数字，英小文字，カタカナの 256+10+26+86 = 378 個の文字からなる．変換規則は

- 2バイト英数 → 1バイト英数
- 英大文字 → 英小文字
- ひらがな → カタカナ

である．$\mathcal{A}$ の文字の比較は，$\mathcal{B}$ に変換してから行なう．

次に，U の各接尾辞を辞書順にソートする．BW 変換と異なり，U の最後には終端記号 \$ があるとし，\$ はどの文字よりも順位が小さいとする．次に，ソートされた各接尾辞に対して，それの1つ前の位置の文字に対応する T の文字を並べ，W を作る．これが修正 BW 変換の結果である．つまり，T の文字を，U の接尾辞配列に従って並び替えている．

逆変換は，W の文字を unify し，その文字列に対して通常の逆 BW 変換を行ない，生成されたリンクに従って W を並び替えればよい．この逆変換により，元の文字列 T と正変換の時に使われた接尾辞配列が復元される．この接尾辞配列では接尾辞の大文字小文字の区別などはしていないため，検索の際に都合が良い．

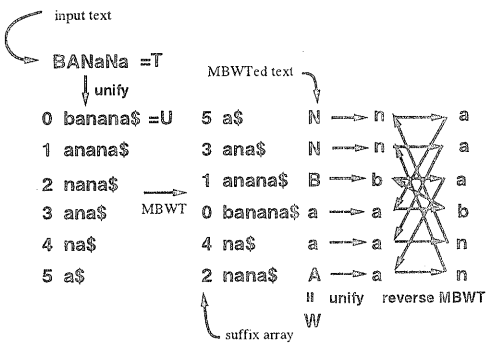

図3 修正 BW 変換と逆変換

4. 単語ブロックソート圧縮法

本節では文書及びその転置ファイルを圧縮するための単語ブロックソート圧縮法を提案する．これは，文章を単語に区切り，各単語をアルファベットの要素だとみなしてブロックソート圧縮法を適用するものである．これにより，圧縮された文書から単語単位の接尾辞配列を得ることができる．これは簡単に転置ファイルに変換することができる．このときの BW 変換を単語 BW 変換と呼ぶ．ここで修正 BW 変換を使えば，同一視を行ない同じとみなされる単語が単語単位の接尾辞配列上で連続した領域に格納される．同一視を行う単語としては，本論文の実験では文字の同一視を行うと同じ文字列になる場合だけを考えているが，これを拡張して読みや意味が同じになる文字列を同一視することも可能である．

4.1 圧縮方法

文書が与えられた時，単語ブロックソート法の圧縮の手順は次のようになる．

(1) 文書を単語に区切り，1から通し番号をつける．同じ単語は同じ番号になるようにする．
(2) 単語と番号の変換表を出力する．
(3) 数字に変換された単語の列を (修正)BW 変換で並び替える．
(4) MTF 変換を行なう．
(5) ランクを符号化する．

なお，(3) 以降は通常のブロックソート法と同じである．単語 BW 変換においても修正 BW 変換や次数1の BW 変換を用いることができる．MTF 変換ではランクとして recency rank を使うか interval rank を使うかの2通りが考えられる．ランクの符号化には，δ 符号を用いる．

4.2 実　　装

4.2.1 文書から単語番号への変換

文書を単語に区切る際は，英語の場合は問題はなく，アルファベットを A-Z と a-z，0-9，空白文字，それ以外に分け，それらが連続している間は1つの単語とみなす．日本語の場合は単語に区切ることは難しいため，形態素解析プログラムである chasen 1.51☆ を使用した．`chasen -F "%m\n"` により，文書を形態素に区切り，これにより単語の長さを決定した．

単語を単語番号に変換するには，単語をハッシュ表に登録しておき同じ単語がない場合は新しい番号をつけ，すでに同じ単語がある場合はその単語と同じ番号をつける．単語の長さは分かっており，前方一致を考慮する必要ないため，ハッシュ関数の計算は容易である．

4.2.2 修正 BW 変換

修正 BW 変換には，Larsson，Sadakane の接尾辞ソートアルゴリズム[7] を用いる．このアルゴリズムは単語の出現数を m とすると最悪でも $O(m \log m)$ 時間である．このアルゴリズムは Manber, Myers[8] のものと最悪の計算量は等しいが，常に高速であり，文書中に長い反復がある場合でも速度が落ちない．また，次数1の修正 BW 変換は，単語が k 種類あるとするとサイズ k の配列を利用する bucket ソートで線形時間で行な

☆ http://cactus.aist-nara.ac.jp/lab/nlt/chasen.html

える．

4.2.3 MTF変換

ランクとしてinterval rankを用いる場合，単語ごとに最後に出現した位置を覚えておけば1つの文字の変換は定数時間で行なえる．recency rankを用いる場合，単語の種類を k 個とすると $O(\log k)$ 時間で1つの文字の変換は行なえる[3]．しかし実装は複雑であるため，$O(\sqrt{k})$ 時間のアルゴリズムを提案する．これは単語をサイズが $\sqrt{k}$ のグループに分け，各グループの中では循環バッファを利用することでランクの値を書き換える回数を1回の変換につき $O(\sqrt{k})$ 回にするものである．

単語をそのランクに従い 1 から $M = \lceil\sqrt{k}\rceil$ のグループに分け，単語からグループ番号への変換表 GRP を用意する．また，ランクから単語番号への変換表 R も用意する．R はサイズ M のグループごとに循環バッファになっており，各グループの先頭位置を示すサイズ M の配列 HEAD を使う．単語 c のランクを求めるには GRP[c] により c の属するグループ g を求め，そのグループの中の各ランクの単語を c と比較する．ランクを求めた後は，c のランクを1にし，c よりランクが小さかった単語のランクを全て1増やす必要がある．グループ g の文字に関しては1つずつ位置をずらすが，それより小さいグループに関してはそのグループ内の一番最後の単語を1つ右のグループに移し，最初の単語は1つ左のグループの一番最後の単語にする．これは HEAD の値を1減らせばよく，GRP の値は変化する単語に対してのみ変更する．1つランクを求める際の表の更新の回数は HEAD と GRP が $\sqrt{k}$ 回，グループ g での R, GRP の更新が $\sqrt{k}$ 回であるため，$O(\sqrt{k})$ 時間となる．

なお，recency rankから単語番号を復元する際には，配列 GRP は必要ない．

4.2.4 ランクの符号化

ランクの符号化は，単純には数字を δ 符号で符号化すればよいが，実際のランクの出現確率に従った符号長にはならないため圧縮率が落ちる．よってランクの実際の出現確率に従い算術符号で符号化する．ただしランクの値の種類は単語の数になるため，全てのランクの実際の確率を考慮することは難しい．また，ランクは 1 になる確率が非常に高いため，その場合だけを特別に扱うだけでも効果がある．Balkenholら[2]はまずランクが1, 2, それ以上かを算術符号化し，ランクが2より大きい場合にはその値を符号化する方法を提案している．ランクの値の確率の推定は，過去の3つのランクの値から行なっている．この方法を単語ブロックソート法に適用し，まずランクが1かそうでないかを算術符号化する方法を使用した．

ランクの算術符号化は次のようになる．まず，3つの連続したランクがそれぞれ1かそれ以外かにより8つの状態を考え，それぞれの状態でランクが1である確率を 0.5 に初期化する．まず，ランクを1つも符号化していないときの状態 s は 0 とする．各ランクの符号化は次のようになる．

(1) ランクが1である時は1，そうでない時は0を算術符号化する．この時の1の確率は，状態 s から決まる．

(2) ランクが1ではない場合は，さらに (ランク -1) を δ 符号で符号化する．これは δ 符号の各ビットごとに，0と1の確率を 0.5 とした算術符号を用いる．

(3) 状態 s でのランク1とそれ以外の頻度を更新する．

(4) 状態 s を更新する．ランクが1の時は $s := (s \cdot 2 + 1) \bmod 8$，1以外の時は $s := (s \cdot 2) \bmod 8$ とする．

復号化も符号化と同様に状態を更新しつつ行なう．

4.3 圧縮ファイルからの転置ファイルの生成

単語ブロックソート法で圧縮された文書を復元し，その文書に対する転置ファイルを作成するには，逆BW変換で単語に対する接尾辞配列を生成し，それを転置ファイルに変換すればよい．単語に対する接尾辞配列では接尾辞は先頭の単語だけでなく，その後に続く単語列の辞書順に従って並んでいるが，転置ファイルでは接尾辞の先頭の1単語の単語番号順になっており，先頭の単語が同じ接尾辞の間はその出現位置の順になっている．つまり，接尾辞を (先頭の単語番号，出現位置) に従って基数ソートすればよい．図 4では，テキスト X 中に単語 that, the, this が存在し，それらに対する単語単位の接尾辞配列 I は左のようになる．ここでは同じ単語が出現位置順になっておらず，また異なる単語の間の境界も分からない．これを右のように単語の境界を求め，同じ単語は出現位置順に並び換えると転置ファイルになる．なお，圧縮時に次数1の単語BW変換をした場合には，逆BW変換を行なった時点で単語

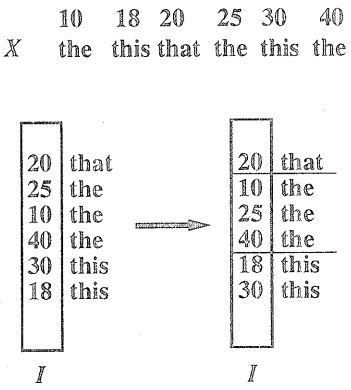

図4 単語の接尾辞配列から転置ファイルへの変換

は転置ファイルの順に並んでいるため，並び替えは必要ない．

単語列 $X[0,n-1]$ とその単語単位の接尾辞配列 $I[0,n-1]$ が与えられた時，転置ファイルを生成するアルゴリズムは以下のようになる．

(1) 一時的な配列 $J[0,n-1]$ を用意する．

(2) 接尾辞を辞書順に見て，先頭の文字 $X[I[i]]$ が等しい範囲には同じ数字 k を割り当て，$J[I[i]]=k$ とする．k は等しい k を持つ接尾辞のうちで最も辞書順で小さいものの辞書順の値とする．また，先頭の文字が辞書順で一つ後のものと異なる時には k を更新し印 $I[i]=-i$ をつける．この処理は基数ソートでの値ごとの累積頻度を求める操作に対応する．

(3) 配列 J を左から走査し，単語を出現位置順に並び替える．これは $j=0,\ldots,n-1$ に対し $i=J[j], k=I[i]++, J[j]=abs(k)$ とすればよい．これにより，J には文書中の各単語の転置ファイルでの位置が格納される．

(4) 単語の位置は単語単位の値であるため，これを文字単位の値に直す．これは J を左から走査し，それぞれの単語の長さを表から求め，その長さを合計したものを $I[J[j]]$ に入れればよい．

なお，次数1の単語BW変換を使用している場合は，$J[I[i]]=i$ とした後にステップ(4)を行なうだけでよい．

文書のサイズが大きい場合は，文書をメモリ内で処理できるサイズのブロックに区切り，ブロックごとに単語BW変換を行ない圧縮する．圧縮ファイルから転置ファイルを作成する場合は，ブロックごとに転置ファイルができるため，それらを併合する必要がある．しかし，各転置ファイル中の単語は出現位置順にならんでおり，それを別の転置ファイル中に挿入することはないため，併合は簡単に行なえる．

5. 転置ファイルと単語ブロックソート法の圧縮率

5.1 理論的な圧縮率の上限

Arimura, Yamamoto[1] により，ブロックソート圧縮法の理論的な性能は解明されているが，単語ブロックソート法の圧縮率に関しても同様の解析を行なう．文書の長さを n，単語の種類を k，単語の出現頻度の合計を m とする．転置ファイルでは単語ごとに出現位置の差を δ 符号で符号化するが，この時の総符号長は $\sum_{i=1}^{k} m_i(1+\log_2(\frac{n}{m_i})+2\log_2(1+\log_2(\frac{n}{m_i})))$ 以下となる (m_i は単語 i の頻度)．ここで，単語の出現位置を文字単位ではなく単語単位で表すと，符号長は $\sum_{i=1}^{k} m_i(1+\log_2(\frac{m}{m_i})+2\log_2(1+\log_2(\frac{m}{m_i})))$ 以下になる．$m<n$ であるため，この方が圧縮率が良い．また，この符号長は，H_0 は各単語が一定の確率で現れるとしたときのエントロピーとすると，$m(1+H_0+2\log_2(1+H_0))$ で押えられる．

単語ブロックソート法の場合，単語の並びを考慮して圧縮することになるのでさらに圧縮率が向上する．次数1の単語BW変換を行ない，recency rankを δ 符号で符号化する場合の総符号長は，単語 i の前にある単語 j の総数を m_{ij}，$m_i=\sum_j m_{ij}$ とすると，$\sum_{i=1}^{k}\sum_{j=1}^{k} m_{ij}(1+\log_2(\frac{m_i}{m_{ij}})+2\log_2(1+\log_2(\frac{m_i}{m_{ij}}))) \le m(1+H_1+2\log_2(1+H_1))$ で押えられる (H_1 は単語の出現確率がその直後の1単語によって決まるとしたときのエントロピー)．

同様に，単語BW変換を行ないrecency rankを δ 符号で符号化する場合の符号長は，単語列のエントロピーを H とすると $m(1+H+2\log_2(1+H))$ でおさえられる．$H_0 \ge H_1 \ge H$ であるため，単語ブロックソート法の圧縮率は一般に転置ファイルよりも良く，またBW変換の次数が大きいほど良くなる．

5.2 圧縮率と速度の実験

単語ブロックソート圧縮法とその他のテキストの圧縮法の圧縮率と圧縮速度，および単語ブロックソート圧縮法により圧縮された文書から転置

ファイルを生成する時間と通常の転置ファイル構成アルゴリズムの時間を比較した．実験に使用したワークステーションは SUN Ultra60 (CPU UltraSPARC-II 360MHz, メモリ 2GB) である．使用した文書は 45325 個の html ファイルの集合 (サイズは 124M バイト) である．文書中の単語数は 386220，それらの頻度の合計は 46143595 である．時間は getrusage 関数で取得した user time である．

5.2.1 各サブルーチンの実行時間

まず，圧縮・伸長および転置ファイル作成時の各サブルーチンの実行時間を調べた (表 1, 2)．伸長の処理は表の下から上に行なわれる．単語ブロックソート法の圧縮では，単語 BW 変換が 2 種類 (次数制限なしと次数 1)，符号化が 3 種類 (interval rank (i) を δ 符号化 (δ)，recency rank (r) を δ 符号化，recency rank を算術符号化 (a)) が考えられる．伸長では，圧縮に対応して復号が 3 種類，逆単語 BW 変換が 2 種類ある．単語 BW 変換は，次数を 1 に制限するとソートが簡単になるため高速になるが，逆変換は反対に遅くなっている．ただし，次数 1 の場合は逆単語 BW 変換の結果の接尾辞配列は転置ファイルと同じ順番になっており，転置ファイルへの変換は単語単位の位置から文字単位の位置への変換のみでよいため高速になる．recency rank への変換はかなり時間がかかっているが，これは本論文で実装した $O(\sqrt{k})$ 時間のアルゴリズムではなく $O(\log k)$ 時間のもの[3] を使用すれば短縮されると思われる．

単語を単語番号に変換する作業は単語ブロックソート法の圧縮時よりも高速だが，これは転置ファイルでは同一視して同じになる単語には初めから同じ番号を与えているからである．単語ブロックソート法では同一視するまえの元の単語を復元するために，同一視すれば同じになる単語でも違う番号を与えている．

5.2.2 各圧縮方法の比較

表 3は圧縮後の文書のサイズと圧縮・伸長時間を表している．圧縮時間には形態素解析の時間 (884.35 秒) は含まれていない．表の圧縮方式の bw は単語 BW 変換，bw1 は次数 1 の単語 BW 変換，recency と interval はそれぞれランクに recency rank か interval rank を使用，arith と delta はランクを算術符号か δ 符号で符号化した単語ブロックソート法を表している．オリジナル，単語表は圧縮前のサイズ，単語表.gz は単語表を gzip で圧縮した時のサイズである．転置ファイルでは，各単語の位置は直前の同じ単語の位置との差を δ 符号で符号化している．bw, recency, arith が最も圧縮率が良く，bw1, interval, delta が最も圧縮が高速である．伸長に関しては bw1, interval, delta が最も高速であるが，bw, interval, delta とほとんど差がない．これは次数 1 の単語 BW 変換を用いると転置ファイルへの変換が高速だが，次数 1 の逆 BW 変換が次数を制限しないものよりも遅いからである．

表 1 単語ブロックソート法の圧縮・伸長時間

処理 (圧縮)	時間 (s)	処理 (伸長)	時間 (s)
形態素解析	884.35	インデックスの出力	43.85
単語番号に変換	244.87	転置ファイルに変換	44.34
単語の出力	16.56	テキスト出力	64.41
単語 BW 変換	244.74	逆単語 BW 変換	36.68
(次数 1)	67.17	(次数 1)	40.97
符号化 (i δ)	17.27	ランク復号 (i δ)	18.55
(r δ)	817.37	(r δ)	372.46
(r a)	954.48	(r a)	545.81

表 2 転置ファイルの作成時間

処理	時間 (s)
形態素解析	884.35
単語番号に変換	175.15
インデックスの出力	71.65

単語ブロックソート法では単語番号を圧縮したものの他に単語番号と単語の文字列の対応表 (単語表) が必要である．単語表は gzip などで簡単に圧縮できる．なお，単語表の圧縮は bzip2 よりも gzip の方が圧縮率が良かった．これは単語表の単語は辞書順にならんでいるため，単語の出現に局所性があるからと思われる．つまり，gzip は文字列の圧縮を，その長さと過去に現れた同じ文字列との距離で符号化するが，単語が辞書順に並んでいる文字列では同じ文字列との距離が常に小さくなり，少ないビット数で符号化されるからである．

単語ブロックソート法の伸長時間は，文書自体を復元する時間と，転置ファイルを作成する時間を含んでいる．ランクが interval rank の場合の伸長時間は 211.39 秒であり，転置ファイルの作成時間 (247.52 秒) よりも高速である．転置ファイルの作成にはこの他に形態素解析の時間が必要である．すでに形態素解析がされており，各単語の長さが分かっていれば必要ないが，その場合でも各単語の長さを保存しておく必要がある．伸

長時間はgzipやbzip2の方が短いが，これらでは伸長後に形態素解析を行ない，単語を番号に変換する必要がある．検索サーバで圧縮文書から転置ファイルを作成する場合，従来の方法ではgzipで圧縮された文書の復元(9.45秒)，形態素解析(884.35秒)，転置ファイルの作成(247.52秒)の1141.32秒かかるが，単語ブロックソート法では211.39秒であり，5倍以上速くなっている．単語ブロックソート法の圧縮時には形態素解析と圧縮(538.96秒)が必要だが，これは検索サーバで行う必要がないため，サーバの負荷を減らすことができる．

圧縮後のサイズに関しては，単語ブロックソート法でinterval rankをδ符号化したものと単語表をgzipで圧縮したもののサイズの合計が31498620であり，gzipで圧縮したときのサイズ32917012よりもわずかだが小さい．recency rankを算術符号化した場合のサイズはbzip2と同じ程度である．bzip2ではテキストを一定の大きさのブロックに区切り，それぞれに対してBW変換，MTF変換，符号化を行なっている．bzip2ではBW変換は文字単位に行なっている．オリジナルのbzip2ではブロックのサイズが900Kバイトであるが，それと比較すると圧縮率は良くなっている．bzip2でのブロックのサイズを36Mバイトに変更すると，圧縮率は元よりも8.6%良くなっている．単語ブロックソート法ではブロックのサイズは文書全体になっているが，圧縮率はbzip2でブロックサイズを36Mバイトにしたものより4.9%悪いだけである．bzip2は文字単位のブロックソートであるが，その圧縮率は単語単位のものとあまり差はない．単語ブロックソート法でのrecency rankへの変換は現在の実装では遅くなっている．

なお，文書を10000000バイトずつ13個に区切り，それぞれを単語BW変換，interval rank，δ符号を用いて圧縮した場合は，圧縮後の転置ファイルのサイズの合計が30733476バイト，単語表を圧縮したものは2901647バイトになった．単語表は同じ単語がいくつもの表に現れているため，文書を分けるとサイズが大きくなるが，圧縮転置ファイルのサイズは分割しない場合の30133305と比べて約2%増えているだけである．よって，コンピュータのメモリが少ない場合や，圧縮速度を上げたい場合には文書を小さいブロックに分けて圧縮しても問題はない．ただし復号時には複数の転置ファイルをマージする必要がある．この時に必要なメモリはブロックのサイズにのみ依存する．

表3 圧縮・伸長時間とファイルサイズ

圧縮方式	サイズ	圧縮(s)	伸長(s)
オリジナル	124472592	—	—
単語表	5311472	—	—
単語表.gz	1365315	7.79	0.44
転置ファイル	74767587	247.52	—
bw recency arith	23723167	1460.97	736.94
bw recency delta	25349462	1338.13	563.53
bw interval delta	30133305	538.96	211.39
bw1 interval delta	37501980	362.90	207.74
gzip	32917012	81.00	9.45
bzip2	25983976	243.33	64.72
bzip2 (ブロック36M)	23910785	598.66	95.49

6. ブロックソート圧縮文書から転置ファイルの作成

単語ブロックソート法ではMTF変換(recency rank)が遅く，また，圧縮率は文字単位の方が良いため，通常のブロックソート法で圧縮された文書から転置ファイルを作成することを考える．これは次のようになる．

(1) 接尾辞配列Iから，その逆写像の配列Jを求める．
(2) Iで隣あっている接尾辞の間の一致長を配列Lcpに入れる．
(3) テキストを左から右に走査し，単語の先頭位置jを求め，それに対応する接尾辞配列の要素$I[J[j]]$に印をつける．その単語の長さも配列$L[J[j]]$に入れておく．
(4) Iで印のついているインデックスだけ残して，左につめる．LとLcpも対応するものだけ残し左につめる．
(5) Iを走査し，隣あう単語が異なる場所に印をつける．単語が異なるかどうかの判定は，単語の長さLが異なるか，隣あう接尾辞の一致長Lcpが単語の長さより小さいかである．
(6) 単語を同一視してソート．単語番号を同一視したものと等しくする．
(7) 単語を単語番号と出現位置でソートする．

なお，このように単語を切り出してから辞書順に並べるならば，修正BW変換ではなく通常のBW変換で構わない．

通常のブロックソート法では文字単位の接尾辞

配列を用いる．このとき単語の先頭位置を表す接尾辞は，接尾辞の先頭の単語が同じ場合でも接尾辞配列中で異なる領域に分断されてしまう場合がある．これは各単語の長さが異なるため生じている．一方，単語ブロックソート法では全ての単語の長さを1とみなすため，このような問題は生じず，圧縮データから転置ファイルを簡単に生成できる．

7. 結　論

全文検索のためのインデックスである転置ファイルを圧縮する単語ブロックソート法を提案した．圧縮ファイルからは文書自身とその転置ファイルを高速に復元できる．その時間は文書から転置ファイルを作成するよりも高速であり，圧縮率もgzipより良い．この圧縮法にはバリエーションがあり，最も圧縮率が良いものはbzip2よりも圧縮率が良いが圧縮・伸長速度は遅い．しかしこれはアルゴリズムの改良により克服できると思われる．これは今後の課題とする．

謝辞　htmlファイルを提供して下さった早稲田大学の黒田洋介氏に感謝致します．有益なコメントをくださった査読者の方に感謝いたします．本研究の一部は文部省科学研究費の援助を受けています．

参考文献

1) Arimura, M. and Yamamoto, H.: Asymptotic Optimality of the Block Sorting Data Compression Algorithm, *IEICE Trans. Fundamentals*, Vol. E81-A, No. 10, pp. 2117–2122 (1998).
2) Balkenhol, B., Kurtz, S. and Shtarkov, Y. M.: Modification of the Burrows and Wheeler Data Compression Algorithm, *IEEE Data Compression Conference*, pp. 188–197 (1999).
3) Bentley, J. L., Sleator, D. D., Tarjan, R. E. and Wei, V. K.: A Locally Adaptive Data Compression Scheme, *Communications of the ACM*, Vol. 29, No. 4, pp. 320–330 (1986).
4) Burrows, M. and Wheeler, D. J.: A Block-sorting Lossless Data Compression Algorithms, Technical Report 124, Digital SRC Research Report (1994).
5) Cleary, J. G. and Witten, I. H.: Data Compression Using Adaptive Coding and Partial String Matching, *IEEE Trans. on Commun.*, Vol. COM-32, No. 4, pp. 396–402 (1984).
6) Elias, P.: University codeword sets and representation of the integers, *IEEE Trans. Inform. Theory*, Vol. IT-21, No. 2, pp. 194–203 (1975).
7) Larsson, N. J. and Sadakane, K.: Faster Suffix Sorting, Technical Report LU-CS-TR:99-214, LUNDFD6/(NFCS-3140)/1–20/(1999), Department of Computer Science, Lund University, Sweden (1999). http://www.cs.lth.se/home/Jesper_Larsson/.
8) Manber, U. and Myers, G.: Suffix arrays: A New Method for On-Line String Searches, *SIAM Journal on Computing*, Vol. 22, No. 5, pp. 935–948 (1993).
9) McCreight, E. M.: A space-economical suffix tree construction algorithm, *Journal of the ACM*, Vol. 23, No. 12, pp. 262–272 (1976).
10) Sadakane, K.: A Modified Burrows-Wheeler Transformation for Case-insensitive Search with Application to Suffix Array Compression, *Proceedings of Data Compression Conference (DCC'99)*, p. 548 (1999). poster session.
11) Sadakane, K. and Imai, H.: A Cooperative Distributed Text Database Management Method Unifying Search and Compression Based on the Burrows-Wheeler Transformation, *Advances in Database Technologies*, LNCS, No. 1552, pp. 434–445 (1999).
12) Sadakane, K. and Imai, H.: On Proximity Searches for Many Keywords by using Suffix Arrays, *Proceedings of DEWS'99*, IEICE (1999). (in Japanese).
13) Schindler, M.: A fast block-sorting algorithm for lossless data compression, *IEEE Data Compression Conference*, p. 469 (1997).
14) Seward, J.: bzip2 (1996). `http://www.muraroa.demon.co.uk/`.
15) Ukkonen, E.: On-line construction of suffix trees, *Algorithmica*, Vol. 14, No. 3, pp. 249–260 (1995).
16) 山名早人 田村健人 河野浩之 亀井聡 原田昌紀 西村英樹 浅井勇夫楠本博之 篠田陽一村岡洋一: 分散型ロボットによるWWW情報収集, 第9回データ工学ワークショップ (DEWS'98), 電子情報通信学会データ工学専門委員会 (1998).

(平成11年6月20日受付)
(平成11年9月27日採録)

(担当編集委員 小川泰嗣)

定兼 邦彦

1971年生．1995年東京大学理学部情報科学科卒業，1997年東京大学大学院理学系研究科情報科学専攻修士課程終了．現在，同専攻博士課程在学中．テキスト圧縮，検索の研究に従事．IEEE DCC'98 Capocelli賞受賞．

今井 浩（正会員）

1958年生．1981年東京大学工学部計数工学科卒業，1983, 1986年東京大学大学院工学系研究科情報工学専門課程修士，博士課程修了，工学博士取得．1986～1990年，九州大学工学部情報工学科助教授，1990年より現在，東京大学大学院理学系研究科情報科学専攻助教授．1987年冬カナダMcGill大学訪問副教授，1988年秋IBM T. J. Watson研究所訪問研究員．アルゴリズム論，計算幾何学，最適化，学習理論の研究に従事．情報処理学会，電子情報通信学会，日本OR学会，ソフトウェア科学会，人工知能学会，数学会，ACM，IEEE各会員．