データベースシステム 119−31
（１９９９．７．２２）

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．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 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．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[15]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[10]という点である．

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で圧縮するよりも良い．ブロックソート法の復号にかかる時間は短いため，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．

† 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

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．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)[14] 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．

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)[7] 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[8),13] 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．

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．

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，速度はこれよりも高速である．現在では bzip2[12] 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．復号時にはこの逆変換を行なうが，これ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．

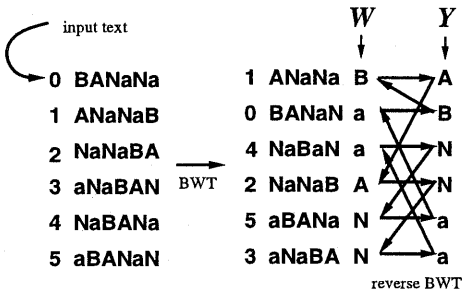

図 1 BWT と逆 BWT

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．

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．なぜなら同じ単語の中の文字はBW変換により連続した領域に集められるため，BW変換後の文字列中には同じ文字が並ぶようになり，ランクが1の確率が大きくからである．

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．

3.2 次数制限ブロックソート法

Schindler[11]はBW変換での文字列のソートを各文字列の先頭のk文字だけで行ない，圧縮を高速にする方法を提案している．先頭のk文字が等しい場合は，文字列は出現順に並べられる．k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ほぼ同じ時間になる．圧縮ファイルから転置ファイルを生成する場合，次数を制限していない場合は，接尾辞を単語ごとに出現位置順に並び替える必要があるが，次数1の場合には同じ単語は出現位置順にソートされているため，逆変換時に並び替える必要はない．

3.3 修正BW変換

Sadakane[9]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2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．これは，大文字から小文字への変換などである．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を並び替えればいい．この逆変換の際に，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，検索の際に都合がいい．

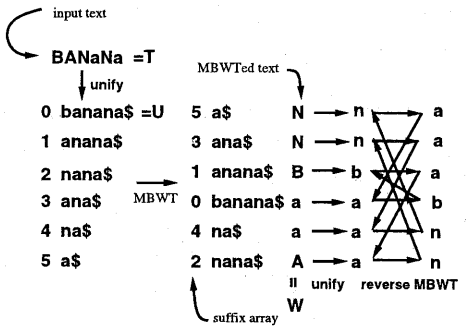

図2 修正BW変換と逆変換

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．

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．

修正BW変換は，次数1とすることもできる．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．単語の単語番号への変換にはハッシュ表を利用した．

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[6]を用いる．このアルゴリズムは単語数を m とすると最悪でも $O(m \log m)$ 時間である．また，次数1の修正BW変換は，単語の種類のサイズの配列を利用するbucketソートで線形時間で行なえる．

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用いる．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．

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．

5. 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いい．接尾辞配列では接尾辞はその中の単語の辞書順に並んでいるが，転置ファイルでは接尾辞の先頭の単語のアルファベット順になっており，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いい．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．

6. 転置ファイルと単語ブロックソート法の圧縮率

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，$\sum_{i=1}^{k}\sum_{j=1}^{k} m_{ij}(1 + \log_2(\frac{m_i}{m_{ij}}) + 2\log_2(1+\log_2(\frac{m_i}{m_{ij}}))) \leq m(1+H_1+2\log_2(1+H_1))$ 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．$H_0 \geq H_1 \geq H$ であるため，単語ブロックソート法の圧縮率は一般にBW

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

変換の次数が大きいほど良くなる．

7. 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．

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，recency rank (r) を δ 符号化，
recency rank を算術符号化) 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．recency rank への変換はかなり時間がかかっているが，これはアルゴリズムを改良すれば短縮されると思われる．

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)	481.90
(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

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．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．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 125.76 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．

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．recency rank への変換は現在の実装では遅くなっている．

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	1473.10	653.29
bw recency delta	25349462	1338.13	481.90
bw interval delta	30133305	538.96	125.76
bw1 interval delta	37501980	362.90	138.58
gzip	32917012	81.00	9.45
bzip2	25983976	243.33	64.72
bzip2 (ブロック 36M)	23910785	598.66	95.49

8. 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．

9. 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，最も圧縮率がいいものは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．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) 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/.

7) 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).

8) McCreight, E. M.: A space-economical suffix tree construction algorithm, *Journal of the ACM*, Vol. 23, No. 12, pp. 262–272 (1976).

9) 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.

10) 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 1552, pp. 434–445 (1999).

11) Schindler, M.: A fast block-sorting algorithm for lossless data compression, *IEEE Data Compression Conference*, p. 469 (1997).

12) Seward, J.: bzip2 (1996).
`http://www.muraroa.demon.co.uk/`.

13) Ukkonen, E.: On-line construction of suffix trees, *Algorithmica*, Vol. 14, No. 3, pp. 249–260 (1995).

14) 定兼邦彦今井浩: Suffix Arrayによる複数単語のProximity Search, 第10回データ工学ワークショップ (DEWS'99), 電子情報通信学会データ工学専門委員会 (1999).

15) 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).