マイクロコンピュータと
ワークステーション 52－3
（1988. 9. 27）

自然言語辞書に対する記憶管理の一方式

安留　誠吾　　青江　順一

徳島大学工学部情報工学科

本論文ではブロック化された辞書に対する記憶管理法を提案し，その有効性を説明する．ここで，提案する管理法は，時間的効率を低下させないために局所的な解決法を導入する．また，辞書が可変長にブロック化されて補助記憶に格納されているときに，これらの可変長のブロックを主記憶上の一つの作業領域を使って管理し，辞書記憶の使用効率を向上させることを目的としている．

A Method of Managing Divided Dictionaries on a Natural Language

Seigo YASUTOME　Junichi AOE

Department of Information Science and System Engineering,
Faculty of Engineering, Tokushima University
2-1 Minami-Josanjima-Cho, Tokushima, 770, Japan

In this paper, we propose a new memory management scheme of the divided dictionaries on a natural language and the efficiency is supported by experimental results. The method to be introduced here solves a problem on storage of the divided dictionaties without degrading the time efficiency. Specially, it is suitable for the case that each size of the divided dictionaries is mixed and that they are placed in an auxiliary memory.

1. まえがき

言語処理システムに於ける辞書の利用形態としては，まず基本辞書が提供され，次にユーザが必要な単語を適時追加する形態が一般的である．また，膨大な数の単語に対して，効率的な検索時間と記憶領域を実現するために，辞書順に基づきデータベースをブロック化して記憶管理する方法がとられる場合が非常に多い．しかし，ブロック化された辞書の最大の欠点は追加単語数の制限にあり，この点を克服する記憶管理法が，導入されているシステムは非常に少ない．[1]-[3]

そこで，本論文ではブロック化された辞書に対する記憶管理法を提案し，その有効性を説明する．ここで，提案する管理法は，時間的効率を低下させないために局所的な解決法を導入する．

自然言語の辞書において，高速な検索の実現も大切であるが，基本辞書に加えて必要となる各種専門辞書や個人用辞書等の膨大化と多様化に対処するためにも，そのコンパクト性にも十分配慮する必要がある．また，辞書に於けるキーの削除は，キーの挿入に比べ少ないと見なせるので，これを増進的性質と呼ぶ．更に，辞書管理の立場からキーの昇順検索は必要不可欠である．これらの性質を満足し、しかもポインタをもたない辞書検索法の最も単純でしかもコンパクトな手法は，次の方法によるものである．

1) 各キーをソートし，ブロック転送可能な大きさにブロック化する．

2) キーに対応するブロックは，主記憶上のディレクトリ表の２分探索で決定する．

3) ブロック内のキー探索は線形検索(Linear Search)で行なう．

この方法では，ブロックの転送回数を１回にでき，またブロック数をn，ブロック内の単語数をmとすれば，探索時間は

$$O(\log_2 n)+O(m)$$

となるので，mを大きくしなければ，十分な検索速度が得られる．しかし，この方法の欠点はキーの挿入あるいはレコードの更新によりブロック分割の再構成が必要になる点である．これは，各ブロックに空領域を取ることで一時的に待避できるが，根本的な解決にはならない．また，このブロックを固定長にすると記憶管理は簡単になるが，内部断片化による空領域の増加は免れない．

本論文の目的は，辞書が可変長にブロック化されて補助記憶に格納されているときに，可変長のブロックを記憶管理して，記憶の使用効率を向上させる点にある．

記憶領域の管理は，従来主記憶上の動的管理を意味し，外部あるいは内部断片化による記憶領域の再構成（詰め直し）を避けるために，次の共通した基本的な構成要素を含んでいる．

1) 使用可能な領域のリスト

2) 記憶領域を見つけだす方法（最適割当て法，先着順割当て法）

3) 未使用領域を返還する方法

4) ごみ集めの方法

本論文の記憶管理でも，これらの要素はそのまま考慮する必要があるが，図１のように，対象となる各ブロックが補助記憶に存在するので，その転送回数が重要な評価基準となる．

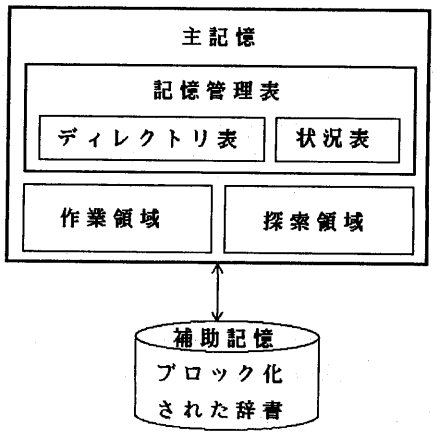


図１　記憶管理の構成

2. 記憶管理アルゴリズム

2. 1 準備

任意のブロックは主記憶上の探索領域(Search Area)に転送され，検索或は更新操作

が実行される．従って，探索領域は最大のブロックを保持できる大きさを持つものとし，ブロックのオーバーフローはこの探索領域上で検出できるものとする．従来の記憶管理では，未使用ブロックと使用ブロックが別に管理されるが，本手法では，これらを同じブロックとして管理する点に特徴がある．従って，ブロックは次のように定義される．

［定義１］　ブロックは実質的にデータが格納されている領域（占有領域と呼ぶ）とそれ以外の未使用領域（空領域と呼ぶ）とに分割され，この二つ領域の合計をそのブロックの大きさと呼ぶ．そして，占有領域がブロックの大きさを越えたブロックを，オーバーブロックと呼ぶ．但し，ブロック内の構成は便宜上占有領域の先に（番地の小さい方に）空領域が格納されているものとする．

［定義２］　ブロックqの大きさをn，占有領域の大きさを■とするとき，

$$RATE(q) = ■ / n$$

をブロックqの占有率と呼ぶ．特に，全ブロックの占有率の平均値をTOTALで表す．

［定義３］　占有率の基準β（占有基準値と呼ぶ）を定義し，$\beta \leqq RATE(q)$なるブロックqを基準ブロックと呼び，$\beta > RATE(q)$なるブロックqを非基準ブロックと呼ぶ．

本章の議論では，オーバーブロックは基準ブロックのみで起こり，非基準ブロックでは起こらないと仮定する．また，図１のように探索領域以外に作業領域(Work Area)が一つ主記憶上に存在するものと仮定する．また，各ブロックの格納番地に対する昇順のリンク（番地リンクと呼ぶ）と非基準ブロックに対するブロックの小さい順のリンク（非基準と呼ぶ）を定義し，これらの情報は次の状況表として主記憶に在中させる．

［定義３］　状況表の配列要素となるレコードP_CELLの各フィールドを定義する．

1)　ADDR: ブロックが格納されている先頭番地．

2)　SIZE: ブロックの大きさ．

3)　R_SIZE: 占有領域の大きさ．

4)　A_POS: 番地リンクを表すポインタ．

5)　G_POS: 非基準リンクを表すポインタ．

状況表を配列BLOCKで表し，ブロックqに対するP_CELLの要素は，BLOCK[q].フィールド名として参照するものとする．

２．２　提案アルゴリズム

ブロックpへの情報の追加（このときにBLOCK[p].R_SIZEも変更される）は，主記憶上の探索領域で行なわれ，

BLOCK[p].R_SIZE>BLOCK[p].SIZE

ならば，ブロックpはオーバーブロックとなる．また，この判定の外，ブロックpの非基準から基準へ或はその逆への変更（非基準リンクの変更も含む）もこの探索領域中の更新操作の後で実行される．

提案される手法は，辞書ファイル全体の占有率TOTALを減少させないでブロックpの領域不足を局所的に解決するものであり，次の優先順位に従って解決される．

1)　オーバーブロックpが次に連続するブロックqの空領域を借用することで解決するならば，図２の様にブロックpの一部をブロックqの空領域に書き込む．但し，このときブロックp，qの占有率は同一にする．この操作は，手続きMIX(p,q)で実行される．

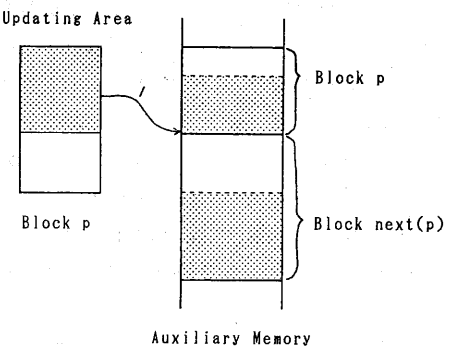


図２　手続きMIX(p,NEXT(p))の説明図

2)　非基準ブロックの空領域を利用して，オーバーブロックの解決を実行する．但し，非基準ブロックの各候補に対しては，オーバーブロックpとの交換と吸収のみを考え，非基準ブロック数を減少させるものを優先する．

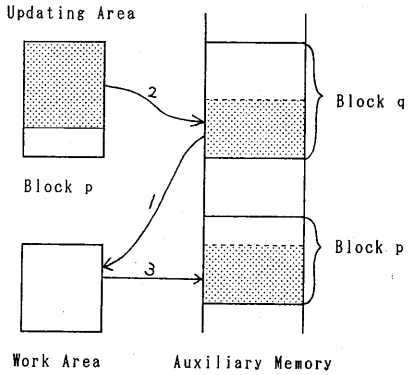


図３　ブロックpとqの交換

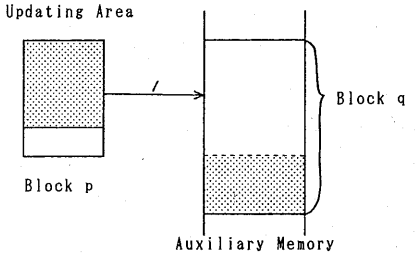


図４　ブロックpの吸収

3)　ファイルの最後尾にブロックpを移動する.

2)で述べた交換は，図３に示すように，ブロックqを作業領域に待避させ，オーバーブロックpをブロックqの場所へ，ブロックqをブロックpの場所へ書き込む．また，吸収は従来の記憶管理に使われているのと同様で，図４に示すように単にオーバーブロックpをブロックpの空領域に書き込む．この二つブロックp,qに関する交換と吸収は非基準ブロック数の増減により，次の関数で操作コードとして分類される．但し，次の関数を使用する．

G_SUM(q): ブロックqの空領域の大きさを表す．即ち，

G_SUM(q)=BLOCK[q].SIZE-BLOCK[q].R_SIZE.

NEXT(p): ブロックpの次のブロック番号を表す．即ち，

NEXT(p)=BLOCK[p].A_LINK.

[関数 A_CASE(q,p)]

(c-1) 次式(1)を満足すれば(c-2)へ進み，満足しなければ(c-3)へ進む.

$$BLOCK[q].SIZE \geqq BLOCK[p].R\_SIZE$$
$$BLOCK[p].SIZE \geqq BLOCK[q].R\_SIZE \quad (1)$$

(c-2) 次式(2,3)を満足すればコード１を；式(2)だけを満足すればコード３を；式(3)だけを満足すればコード４を返す．また，何れも満足しない場合，コード６を返す.

$$\beta \leqq BLOCK[p].R\_SIZE / BLOCK[q].SIZE \quad (2)$$
$$\beta \leqq BLOCK[q].R\_SIZE / BLOCK[p].SIZE \quad (3)$$

(c-3) $G\_SUM(q) > BLOCK[p].R\_SIZE$ (4)

が成立しない場合，コード７を返す．また，式(4)が成立し，

$$\beta \leqq (BLOCK[p].R\_SIZE + BLOCK[q].R\_SIZE) / BLOCK[q].SIZE \quad (5)$$

が成立すればコード２を返し，式(5)が成立しなければコード５を返す.

手順(c-1)は，オーバーブロックpと非基準ブロックqの領域が交換可能か否かを式(1)の条件で調べ，交換可能ならば手順(c-2)に於て交換操作による非基準ブロック数の増減を式(2,3)の条件に従って分類する．即ち，式(2,3)は交換した後，ブロックpとq基準であるか否かをそれぞれ判定し，コード１，３，４，６は両ブロックが基準ブロックになる；ブロックpは基準でブロックqが非基準になる；ブロックqの非基準が解消し，代わりにブロックpが非基準になる；両ブロックとも非基準となる場合をそれぞれ意味する．手順(c-3)はオーバーブロックpが非基準ブロックqの空領域に吸収されるか否かを式(4)で調べ，手順(c-2)と同様に非基準ブロック数の増減の分類を式(5)で行う．従って，コード２は両ブロックが基準ブロックとなるが，ブロックNEXT(p)($\neq$NIL)はブロックpが抜けた空領域を自分のブロックに取り込むので，非基準ブロックになる確率が極めて高い．この理由は，基準ブロックの空領域を使用領域より非常に小さい値に設定しているからである．もちろん，ここで基準か非基準かのチェックを導入するのは容易であるが，記述を簡単にするために基準ブロックが抜けた次のブロックは非基準ブロックとなるものと仮定して議論する.

記憶管理アルゴリズム（アルゴリズム１と呼ぶ）を図５に示す．但し，次の変数と手続きを利用する.

```
Algoritm 1
begin
(1-1)  for i:=2 to 7 do INFO[i] := -1;
(1-2)  if(NEXT(p)≠NIL
          and G_SUM(NEXT(p))≧OVER(p)then
           begin
             MIX(p,NEXT(p));
             return(TRUE)
           end;
       q := G_HEAD;
       while(q≠NIL)  do
         begin
           CODE := A_CASE(p,q);
           if(CODE = 1) then
              begin
                EXCHANGE(p,q,1);
                return(TRUE)
              end
(1-3)       else INFO[CODE] := q;
           q := BLOCK[q].G_LINK
         end;
       for i:=2 to 7 do
         if(INFO[i]≠-1) then
            begin
(1-4)         EXCHANGE(p,INFO[i],i);
              return(TRUE)
            end
end;
```

図５　アルゴリズム１

INFO: 操作コード（2～7）をインデックスとし，その操作コードのブロック番号を格納する配列である．

A_HEAD,G_HEAD: それぞれ番地リンクと非基準リンクの先頭のブロック番号を表す変数とする．

EXCHANGE(p,q,r): オーバーブロックpと操作対象ブロックqに対して，操作コードrの処理を実行する．

OVER(p): オーバーブロックの余剰領域の大きさ．即ち，

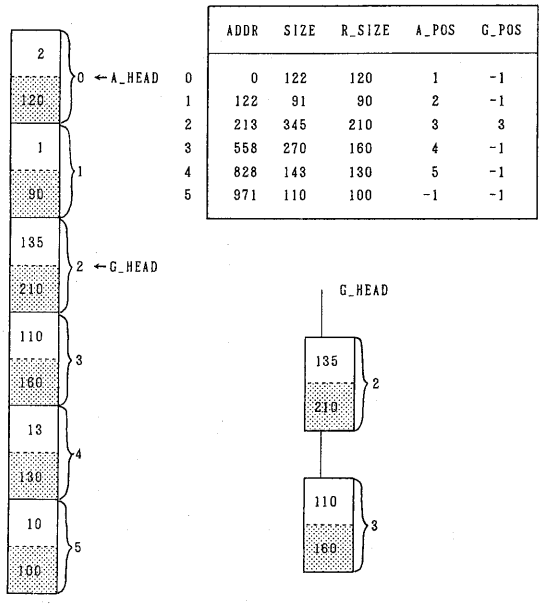
OVER(p)=BLOCK[p].R_SIZE-BLOCK[p].SIZE

アルゴリズム１では，まず行(1-1)で配列INFOを初期化し，行(1-2)でブロックpがファイルの最後尾ではなく（NEXT(p)≠NIL）かつオーバーした余剰領域OVER(p)が次のブロックNEXT(p)の空領域G_SUM(NEXT(p))で賄えれば，手続きMIX(p,NEXT(p))を呼び，操作を終える．これ以外の場合は，非基準リンクを状況表上で辿り各非基準ブロックに対する操作コードをブロック番号を配列INFOに格納する（行(1-3)）．但し，操作コード１は最優先され，EXCHANGE(p,q,1)を実行して終える．全ての非基準ブロックを調査後，行(1-4)で優先度の高い操作コードに対してEXCHANGEを実行し，処理を終える．

［例１］　図６に，ブロックの例を示す．但し，占有基準値βは0.9とする．この例において，ブロック2と3が非基準ブロックであり，それ以外は基準ブロックである．

図６のに於て，ブロック0の大きさが125となり，オーバーブロックとなったと仮定する．このとき，次に続くブロックNEXT(0)=1の未使用領域の大きさは1であるので，非基準ブロックの未使用領域の利用を考える．最初の非基準ブロック2について調べると，(c-1)の交換は不可能であるが，(c-3)で

G_SUM(2)=135＞BLOCK[0].R_SIZE=125



	ADDR	SIZE	R_SIZE	A_POS	G_POS
0	0	122	120	1	-1
1	122	91	90	2	-1
2	213	345	210	3	3
3	558	270	160	4	-1
4	828	143	130	5	-1
5	971	110	100	-1	-1

図６　ブロック化の例

となり，吸収は可能となる．そして，

β=0.9≦(BLOCK[2].R_SIZE
+BLOCK[0].R_SIZE)／BLOCK[2].SIZE(5)
=(210+125)／345=0.97

となるので，コード３が返される．即ち，INFO[3]=2が設定される．

次に小さい非基準ブロック3の大きさが270と125より大きいので，関数A_CASEの(c-1)でブロック3とオーバーブロック0の交換を試みるが，

BLOCK[3].SIZE=270≧BLOCK[0].R_SIZE=125

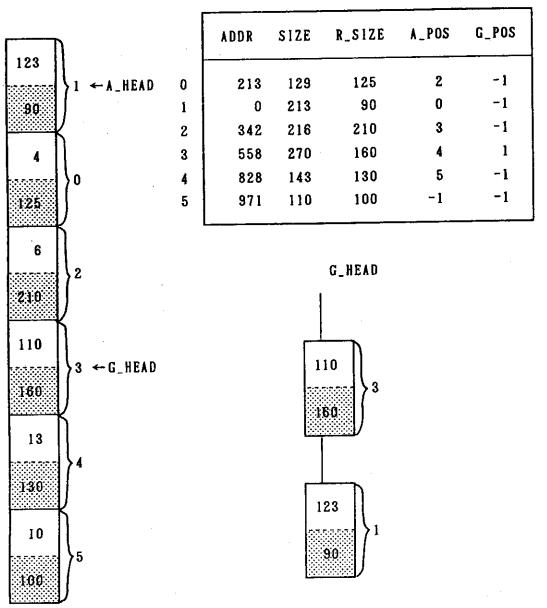
BLOCK[0].SIZE=122＜BLOCK[3].R_SIZE=160

により，交換は不可能となる．次に，(c-3)でブロック3の未使用領域への吸収を調べるが，

G_SUM(3)=110＜BLOCK[0].R_SIZE=125

により，吸収も不可能となる．次の非基準ブロックの候補はないので，INFOの情報により非基準ブロック2に対して，コード３の処理（EXCHANGE(0,2,3)）が実行される．

図７に最終結果を示す．



	ADDR	SIZE	R_SIZE	A_POS	G_POS
0	213	129	125	2	-1
1	0	213	90	0	-1
2	342	216	210	3	-1
3	558	270	160	4	1
4	828	143	130	5	-1
5	971	110	100	-1	-1

図７　オーバーブロックの処理

3.　実験による評価

占有基準値βを変化させて，使用ブロック数100，挿入数10,000回の場合の外部断片化による非基準ブロック数の変化を図８に，辞書全体の占有率TOTALの変化を図９に，オーバーブロックの起こった割合及びその操作コードの割合を表１に示す．

表１　占有基準値βによるオーバーブロック及び操作コードの割合

占有基準値β	0.75	0.80	0.85	0.88	0.90	0.93	0.95
オーバーMIX	6.75	8.08	9.19	10.01	10.57	11.18	12.38
	65.8	64.9	67.0	62.4	60.5	61.0	59.4
* CODE 1	2.5	3.8	3.4	2.3	2.7	2.6	1.9
* CODE 2	6.7	7.4	6.9	10.0	9.8	9.4	8.1
+ CODE 3	1.6	1.9	1.1	0.5	0.6	0.7	0.9
* CODE 4	5.6	6.8	6.0	8.1	9.5	8.6	10.7
* CODE 5	4.6	5.7	5.3	6.9	7.5	8.7	12.1
+ CODE 6	0.0	0.0	0.1	0.0	0.1	0.2	0.2
# CODE 7	13.2	9.5	10.2	9.8	9.3	8.8	6.7

* 交換　　+ 吸収　　# 移動

実験結果より，TOTALは，最初オーバーブロックが起こり始めるまで上昇し，非基準ブロック数が増加すると共に下降し始める．しかし，TOTALが0.8近くまで降下すると非基準ブロック数も減少し始め，TOTALが0.9前後に安定し，非基準ブロック数も20前後に安定することが確認された．つまり，最初に辞書を構築する際にこのことをふまえてブロック化することにより，常に安定した辞書が構築できることが確認された．また表１より占有基準値βを変化させることによってオーバーブロックの起こる割合は，変化するもののオーバーブロック操作コードの割合は，変化しないことがわかる．このことからオーバーブロックの起こる割合が，即ち，時間的評価となることを意味している．よって占有基準値βを0.9前後に設定するのがよいものと思われる．

今後次の点を検討する必要がある．

a）非基準ブロックに追加するときに，非基準ブロック同士の交換処理で，基準ブロックになる場合があるかどうか？

b）最悪の転送回数と作業領域の個数を増加させると，局所的範囲が広がりより効率的となる．

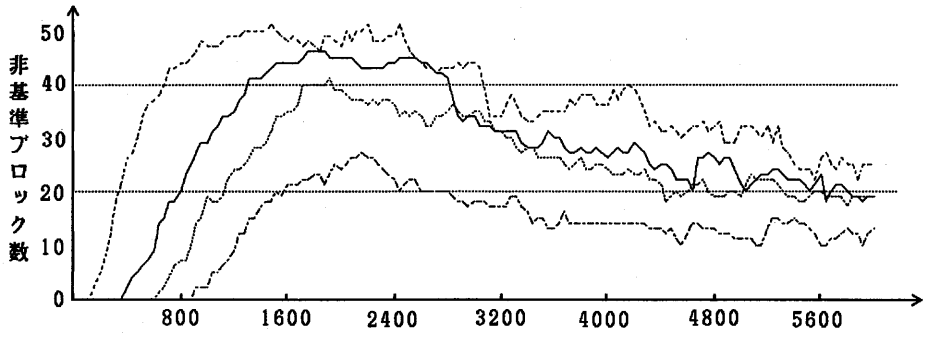


図８　非基準ブロック数の変化

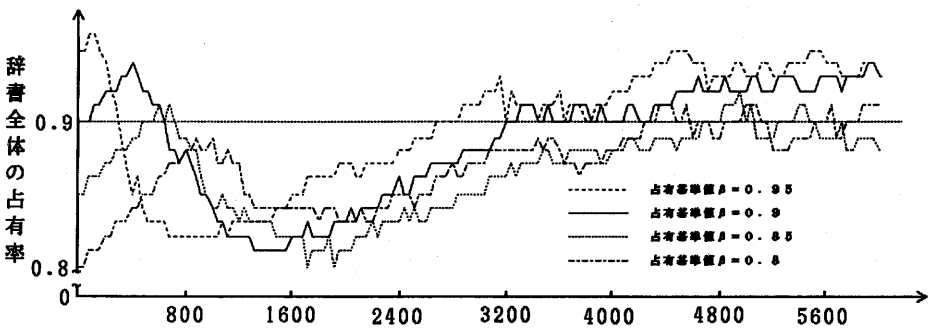


図９　辞書全体の占有率の変化

4.　むすび

以上，ブロック化された辞書ベースの記憶管理法を提案し，挿入のみに於て数値上で実験した．記憶管理の通常の処理では，オーバーブロックを転出させた領域を一つのゴミ領域として取り扱っているが，本手法では隣接するブロックの未使用領域として取り扱う．この処理により，通常の処理で取り扱われているゴミ領域への挿入だけでなく，ブロックの交換処理が効率的に実行されるようにした．また，占有基準値βに基づき，未使用領域の多い非基準ブロックのリンク情報を状況表に持つことにより，未使用領域の探索効率を向上させた．

提案された管理法の効率は，非基準ブロックの最適化を検討することで，更に向上すると思われる．[4] また，ブロックの転送回数と作業領域を増加させ，局所的範囲をより広く取ることも検討する必要がある．

参考文献

[1] A.V.エイホ,J.E.ホップクロフト,J.D.ウルマン著,大野義夫訳：データ構造とアルゴリズム，倍風館(1987)

[2] 宮地利雄著：データ構造とプログラミング，昭晃堂(1986)

[3] D.E.Knuth著,米田信夫他訳：基本算法，サイエンス社(1976)

[4] Rodney R.Oldehoeft,Stephen J.Allan: ADAPTIVE EXACT-FIT STORAGE MANAGEMENT, Communications of the ACM, May 1985 Vol.28