行置換によるスパース行列の効率的縮小アルゴリズム†

青江順一†† 山本米雄†† 島田良作††

Tarjan らは，スパース行列の縮小法として行置換による方法（ffd 法と呼ぶ）を採用し，次を満足するための条件（HD 条件と呼ぶ）とその理論的評価を与えた．(1)行列のすべての非零要素数を n，行の大きさを m とするとき，記憶量を $n+2m$ 語とする．(2)最悪の場合の探索時間を $O(1)$ とする．本論文では，種々のスパース行列に対する実験結果に基づいて ffd 法と HD 条件を評価し，ffd 法の改善法と HD 条件に代る経験的な条件を提案する．まず，ffd 法の行置換法と行ソート法が改善され，改善された縮小法を ffds 法と呼ぶ．次に，この ffds 法が上記の(1)と(2)を満足するための条件（AV 条件と呼ぶ）を提案する．この AV 条件は経験的なものであるが，HD 条件より判定が能率的に行え，しかも適用できるスパース行列の範囲が HD 条件より大幅に広くなる．最後に，AV 条件を満足しない行列に対する縮小法の拡張を考える．Tarjan らの拡張法では，HD 条件以外に ED 条件と呼ばれるもう一つの条件を必要とする．しかし，本論文による拡張法では AV 条件をそのまま使用できるので，行列縮小化の条件をつねに一つに統一できる特徴がある．

1. まえがき

電子計算機で処理される情報がスパース行列 (sparse matrix) で表現される場合は非常に多いが，行列が大きくなると膨大な記憶領域を必要とするので，その記憶倹約が問題となる[1)~9)]．

スパース行列の利用は，その要素の値が変化する動的 (dynamic) なスパース行列とその値が変化しない静的 (static) なスパース行列とに区別できる．そして，動的な場合の縮小法では，情報の追加と削除が能率的なデータ構造を要求するので，行列の縮小率および得られたデータ構造からの探索時間の低下は免れない．これに対して，静的な場合の縮小法では動的な場合におけるデータ構造の制限は不要となるので，縮小化の条件は有利になる[1),6),7),8)]．

ハッシュ法は，このようなスパースな表の探索問題を解決する最も実用的な方法である．また，Aho ら[3)]による行置換を使った縮小法（ffd 法と呼ぶ）はパーサの解析表[3),9),10)]，ストリングパターンマッチング[6)~8)]，ガウス消去法[12)]における静的スパース行列に対して有効なものである．Tarjan ら[1)]は，最悪の場合の探索時間に関して ffd 法がハッシュ法より優れているとして，ffd 法の効率化の条件（HD 条件と呼ぶ）を提案し，その理論的評価を与えた．しかし，彼らの目的は ffd 法の実用性を主張することよりむしろ，ffd 法の効率化に関して理論的基礎が存在し，それが実用的アルゴリズムの発見に役立つことを示すことにあった．したがって，Tarjan らの理論的結論を実験により実際的に評価すること，また行置換による縮小法に対する効率化の条件を経験的に決定することは興味深いものと思われる．

以上により，本論文の目的は次の3点にある．

（1） ffd 法の行置換と行ソートを改善し，新しい縮小法（ffds 法と呼ぶ）を提案する．

（2） 計算機の一様乱数により種々のスパース行列を構成し，ffd 法と ffds 法も含めた五つの縮小法に対する実験結果に基づき各縮小法を比較する．

（3） (2)の結果に基づき，ffds 法に対する効率化の条件を経験的に決定する．

2. 行置換によるスパース行列の縮小法

行と列の大きさがともに m (≥ 2) であるスパース行列を M で表し，$1 \leq i, j \leq m$ なる第 i 番目の行と第 j 番目の列（以後それぞれをたんに i 行，j 列と呼ぶ）に対応する行列 M の位置を (i, j)，その要素を $M(i, j)$ で表す．このとき，Tarjan ら[1)]は次のような行列 M を構成した．

N ($=m^2$) 個の値が0から $N-1$ の整数値として行列 M に定義でき，いま n ($\geq m$) 個の値が定義されているものとする．このとき，値 k に対する行列 M の位置 (i, j) の i と j は次のように計算される．

$$i = k \operatorname{div} m + 1,$$

$$j = k \bmod m + 1.$$

ここで，$k \operatorname{div} m$ と $k \bmod m$ はそれぞれ k を m で割った商と余りを与える．値 k が $M(i, j)$ に定義され

† An Efficient Algorithm of Reducing Sparse Matrices by Row Displacements by JUNICHI AOE, YONEO YAMAMOTO and RYOSAKU SHIMADA (Department of Information Science and Systems Engineering, Faculty of Engineering, The University of Tokushima).

†† 徳島大学工学部情報工学科

```
Algorithm A.  First-Fit Decreasing Method.
Input:  Nonzero position (i,j).
Output: The row displacement function r and the array ENTRY.
Method:
    begin
Step 1. for i:=1 to m do
            begin
               count(i):=0; list(i):=φ
            end;
        for each nonzero position (i,j) do
            begin
               count(i):=count(i)+1; list(i):=list(i)∪{j}
            end;
Step 2. for p:=1 to m do
            bucket(p):=φ;
        for i:=1 to m do
            bucket(count(i)):=bucket(count(i))∪{i};
Step 3. for k:=1 to n+m do
            ENTRY(k):=false;
        for p:=m downto 1 do
           for each i in bucket(p) do
              begin
                 r(i):=0;
  overlap:       for each j in list(i) do
                    if ENTRY(r(i)+j)=true then
                       begin
                          r(i):=r(i)+1; goto overlap
                       end;
                 for each j in list(i) do
                    ENTRY(r(i)+j):=true
              end
    end.
```

図 1 ffd 法
Fig. 1 ffd method.

ているとき，その要素を非零要素と呼び，そうでなければその要素を零要素と呼ぶ．

Tarjan らの採用した行列 M の縮小法は，行列 M の非零要素 $M(i,j)$ を1次元配列 ENTRY の

ENTRY$(r(i)+j)$

なる要素に写像することである．ここで，r は行番号に対する行置換関数 (row displacement function) を表す．この方法では，行列 M の二つ以上の非零要素が配列 ENTRY の同じ要素に写像されてはならないし，しかも行列 M のすべての非零要素が効率よく配列 ENTRY の要素に写像されなくてはならない．そこで，Tarjan らは非零要素数の多い行より順に行置換関数の値を決定し，しかとその値がつねに最小となるような方法を採用した．以後，この方法を“first-fit decreasing”の略として ffd 法と呼ぶ．この ffd 法をアルゴリズムAとして**図1**に示す．

Step 1 の list(i) は i 行に存在する非零要素に対する列番号のリストであり，count(i) はこれら非零要素の数を表す．Step 2 は非零要素数の多い順による行ソートである．Step 3 は配列 ENTRY を初期化し，行置換関数を決定する．ただし，簡単のために配列 ENTRY の要素は論理値とし，行列 M の非零要素が写像された要素を true として表す．

$l \geqq 0$ なる l に対して $l+1$ 以上の非零要素数をもつ行に存在する総非零要素数を $n(l)$ で表すとき，Tarjan らは ffd 法が効率的に働くための条件を提案し，次の定理を導いた．

[定理 1] 行列 M が次の条件 (“harmonic decay”を略して HD 条件と呼ぶ)

$n(l) \leqq n/(l+1)$

を満足するとき，$1 \leqq i \leqq m$ なるすべての $r(i)$ は ffd 法によりつねに次の式を満足する．

$0 \leqq r(i) \leqq n.$

(証明) 文献1)の定理1参照．

(証明終)

行列 M が HD 条件を満たすとき，配列 ENTRY は $n+m$ 以下の大きさとなり，行置換関数も含めて行列 M は $n+2m$ 語に圧縮できる．そして，得られたデータ構造からの探索時間は $O(1)$ となる[1]．また，文献1)の定理3より ffd 法の実行時間は $O(n^2+m)$ 時間となることがわかる．以後，行ソートを行わない行置換による縮小法を ff 法 (“first-fit”法の略) として ffd 法との比較のために使用する．

(例1) **図2**に $m=5$ なる行列 M とその縮小結果を示す．ただし，行列 M の＊印が非零要素を表し，行ソートの順は2，1，3，5，4である．

この行列 M において，$n=10, n(1)=9$ となるので，

$n(1) \leqq n/2$

は成立しない．したがって，この行列 M は HD 条件を満足しない．

3. 行列縮小法の改善

本論文での実験は，次の値を使用して評価を行う．

行列 M の全要素数 m^2 $(=N)$ に対する全非零要素数 n の割合をスパース率 (sparse rate) sr と呼び，次のように定義する．

$$\mathrm{sr}=\frac{n}{m^2}\times 100(\%)$$

また，行列 M の行に存在する平均非零要素数 av を

$$\mathrm{av}=n/m$$

と定義する．各縮小数において構成された配列 ENTRY 中のすべての false 要素数を α とするとき，縮小率 (reduction rate) rr を次のように定義する．

$$\mathrm{rr}=\frac{\alpha}{n}\times 100(\%)$$

3.1 行置換関数の決定手順の改善

ffd 法と ff 法による行列 M の縮小結果を**表 1** に示す．ただし，表中の縮小率の α は，$r(i)$ の最大値に m を加えた値を配列 ENTRY の大きさとして計算したものである．

表 1 より，ffd 法は ff 法を十分改善していないことがわかる．これは，配列 ENTRY の中心部のみで要素 true の存在率が高くなり，両端では false 要素が多く残される関数 r の決定法に原因がある．したがって，本節では，関数 r の決定手順における

$r(i):=0$

を次のように改める．

$r(i):=1-\min(i)$

ここで，$\min(i)$ は $\mathrm{list}(i)$ の要素中の最小の列番号を表す．また，配列 ENTRY 中の true 要素の最大のインデックスを max とするとき，位置 (i,j) の探索法を次のように改める．

if $1\leqq r(i)+j\leqq$ max **then**
　return ENTRY $(r(i)+j)$
else return false.

本節の改善により，配列 ENTRY の大きさはつねに max となり，しかも要素 true も一様に高い率で存在するようになる．以後，縮小法および縮小率はつねにこの改善に従ったものとする．

3.2 行ソートに対するもう一つの提案

本節では，次の NZ 長 (nonzeros length の略) を利用した行ソート法を提案する．

```
     1   2   3   4   5
1    0   *   0   0   *
2    0   *   *   0   *
3    *   0   *   0   0
4    0   *   0   0   0
5    *   0   0   0   *
```

(a) Matrix M.

```
0   *   *   0   *                                   r(2)=0
        0   *   0   0   *                           r(1)=2
                *   0   *   0   0                   r(3)=5
                            *   0   0   0   *       r(5)=8
                            0   *   0   0   0       r(4)=8

1   2   3   4   5   6   7   8   9   10  11  12  13
f   t   t   t   t   t   t   t   t   t   f   f   t   ENTRY

t=true, f=false
```

(b) Function r and array ENTRY.

図 2 ffd 法の結果
Fig. 2 Results of ffd method.

表 1 ff 法と ffd 法の結果
Table 1 Results of ff and ffd methods.

m	縮小率 (%) av=1		av=2		av=3		av=4	
	ff 法	ffd 法	ff 法	ffd 法	ff 法	ffd 法	ff 法	ffd 法
200	20.3	35.0	30.4	44.1	23.6	24.9	17.7	18.0
150	85.4	44.6	33.7	33.7	22.0	19.6	21.8	20.5
100	71.2	84.1	31.3	42.2	21.2	23.1	25.2	18.4
80	51.3	82.5	28.5	39.9	25.2	22.7	26.9	22.1
50	43.4	60.4	34.1	20.0	30.0	21.6	33.3	20.7
30	40.0	28.0	37.7	44.3	20.8	20.8	28.1	25.6

[定義 1] 行列 M において，$1\leqq i\leqq m$ なる i 行の非零要素のなかで最大の列番号と最小の列番号をそれぞれ $j_{\max}$, $j_{\min}$ $(1\leqq j_{\min}, j_{\max}\leqq m)$ とするとき，i 行の NZ 長 $\mathrm{length}(i)$ を次のように定義する．

$\mathrm{length}(i)=j_{\max}-j_{\min}$.

行置換による行列 M の縮小法では，行の適用回数が多くなるにつれて，配列 ENTRY 中の true 要素の存在率は高くなる．したがって，ffd 法のように非零要素数の少ない行を後で適用することは関数 r の決定が能率的となり，その結果として配列 ENTRY の大きさを小さくできる．また，NZ 長の小さい行を後で適用することは配列 ENTRY の大きさの増加をできる限り小さくするので，同様に配列 ENTRY の縮小

表 2 各種法の縮小結果
Table 2 Results of reduction by each method.

av	sr	m	n	縮小率 (%)				
				ff 法	ffds 法	ffsd 法	ffd 法	ffs 法
2	1.01	200	404	18.8(5)	0.0(1)	0.0(1)	13.1(4)	0.0(1)
	1.35	150	303	21.8(5)	0.0(1)	0.0(1)	9.6(4)	0.0(1)
	2.11	100	211	24.6(5)	0.0(1)	0.0(1)	21.8(4)	0.0(1)
	2.47	80	158	26.6(5)	0.0(1)	0.0(1)	17.1(4)	0.0(1)
	3.92	50	98	33.7(5)	0.0(1)	0.0(1)	8.2(4)	0.0(1)
	6.78	30	61	8.1(5)	0.0(1)	0.0(1)	0.0(1)	0.0(1)
3	1.51	200	606	16.2(5)	0.0(1)	0.0(1)	13.2(4)	0.0(1)
	2.06	150	464	14.0(5)	0.0(1)	0.0(1)	4.5(4)	0.0(1)
	3.12	100	312	20.2(5)	0.0(1)	0.0(1)	17.6(4)	0.0(1)
	3.63	80	232	18.1(5)	1.7(3)	0.0(1)	6.9(4)	1.3(2)
	5.76	50	144	34.7(5)	0.0(1)	2.7(3)	11.8(4)	1.0(2)
	10.11	30	91	12.1(5)	5.5(2)	4.4(1)	9.9(4)	4.4(1)
4	2.02	200	807	10.5(4)	5.9(3)	4.1(1)	12.1(5)	4.6(2)
	2.60	150	584	17.1(5)	1.4(1)	3.4(3)	7.7(4)	2.1(2)
	4.13	100	413	20.1(5)	5.1(1)	10.1(3)	9.6(2)	14.5(4)
	5.17	80	331	24.2(5)	8.5(3)	6.0(1)	13.9(4)	7.9(2)
	8.00	50	200	28.0(5)	8.5(3)	7.5(2)	16.5(4)	6.5(1)
	13.44	30	121	19.8(4)	14.9(1)	14.9(1)	21.5(5)	19.0(3)
5	2.56	200	1,024	17.7(5)	9.3(1)	11.9(3)	15.3(4)	11.4(2)
	3.18	150	715	21.3(5)	4.3(1)	8.8(2)	9.8(3)	11.0(4)
	5.16	100	516	28.7(5)	11.6(1)	16.2(3)	14.5(2)	19.5(4)
	6.27	80	401	27.9(5)	14.0(1)	16.7(3)	15.2(2)	17.5(4)
	9.76	50	244	34.8(5)	16.4(3)	10.2(1)	17.2(4)	15.6(2)
	17.11	30	154	28.6(4)	20.8(1)	30.0(5)	22.1(2)	25.3(3)
6	3.09	200	1,236	26.6(5)	14.7(1)	21.0(4)	19.0(2)	20.1(3)
	4.19	150	942	33.8(5)	16.6(2)	20.4(4)	16.3(1)	18.1(3)
	6.30	100	630	29.5(5)	19.2(1)	27.1(3)	28.6(4)	25.7(2)
	7.30	80	467	35.8(5)	21.6(2)	21.8(4)	21.6(2)	21.1(1)
	11.76	50	294	39.1(5)	17.1(1)	25.5(4)	22.8(2)	23.2(3)
	19.00	30	171	36.8(5)	27.1(1)	27.5(2)	28.7(3)	33.3(4)

化に効果がある．しかし，行列 M はスパースであるので，各行に存在する非零要素数の分布は小さい値に偏る．すなわち，ffds 法の行ソートはスパースな行列に対してその効果が悪くなることがわかる．これに対して，NZ 長による行ソースでは NZ 長が 1 の場合を除いて 2 から m の範囲に広く分布するので，行ソートの効果は行列 M がスパースな場合でも十分期待できる．

以上の考察に基づいて，本節では非零要素数と NZ 長を利用した次の五つの行ソートによる縮小法の実験を行った．

ff 法：“first-fit” 法の略．

ffds 法：非零要素数を NZ 長より優先した行ソートを行う方法であって，“first-fit decreasing and shortening” 法の略．

ffsd 法：NZ 長を非零要素数より優先した行ソートを行う方法であって，“first-fit shortening and decreasing” 法の略．

ffd 法：“first-fit decreasing” 法の略．

ffs 法：NZ 長の長い順に行ソートを行う方法であって，“first-fit shortening” の略．

実験結果は，スパース率 sr に従って集計するより行の平均非零要素数 av に従ったほうがよい評価が行えるので，後者に従った結果を**表 2** に与える．ただし，( )内の値は各種法の縮小率の順位を表す．

表 2 より，av が 2 と 3 の場合は行ソートに NZ 長を考慮した ffds, ffsd, ffs 法が ffd 法より圧倒的に優れていることがわかる．また，av が 4 から 6 へと増加するにつれてこれらの方法と ffd 法の差はなくな

る．これは，ffd 法の行ソートの効果が出てきたためと考えられる．しかし，ffds 法はつねによい縮小率を与えている．これは，非零要素数による行ソートの効果が十分でないときに NZ 長による行ソートがこれをうまく補足しているためと考えられる．ffsd 法では，つねに NZ 長による行ソートが非零要素数による行ソートに優先するので，ffds 法におけるこのような補足は行えない．以上より，本論では ffds 法を ffd 法の改善法として採用する．ffds 法のアルゴリズム B を付録 1 に示す．

後になったが，3.1 節の改善法の有効性は表 1 と 2 における ff 法と ffd 法の結果より容易にわかる．

3.3 HD 条件の問題点

HD 条件を満たす行列 M は，次の二つの制限を受ける．

（1） 行列 M のすべての非零要素の少なくとも半数は一つの非零要素だけをもつ行に存在する．

（2） $\log_2 n$ 個以上の非零要素数をもつ行は存在しない．

これらの制限からもわかるように，HD 条件はかなり厳しい条件であるといえる．

ここで，m を 30, 50, 80, 100, 150, 200 と変化させた行列 M において，HD 条件を満足する範囲の境界を行列 M の全非零要素 n，スパース率 sr，行の平均非零要素数 av の値で**図 3** に示す．図 3 において，各グラフの下方が HD 条件を満足する行列 M の存在範囲である．

図 3 より，HD 条件を満足する行列 M の sr と av の値は m が増加すると急激に減少するので，n は行列

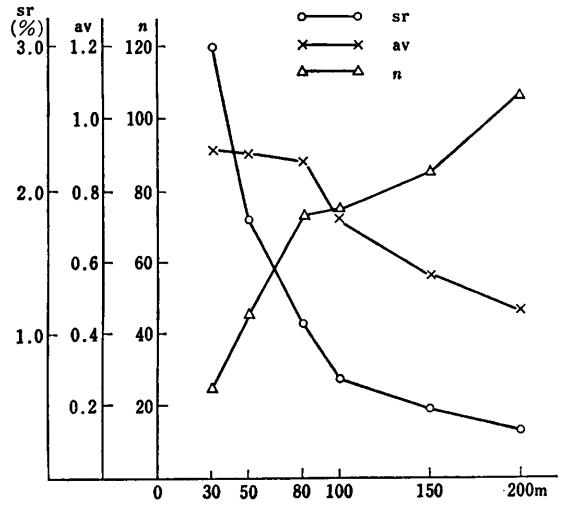

図 3 HD 条件の範囲
Fig. 3 Range of HD condition.

M の大きさ m を増加してもその率だけ増加しないことがわかる．とくに，図 3 において av が 1 以上にならないので，n と m の関係はつねに

$$n < m$$

となり，2 章の Tarjan らによる条件

$$n \geqq m$$

と矛盾する．

以上より，HD 条件は n を極端に制限するものであり，実用的条件でないといえる．この点は，HD 条件を満足する行列 M の縮小率が前節のどの方法においても 0 %となり*，比較実験にならないことよりもわかる．

3.4 HD 条件に代る条件

Tarjan らの採用した評価に従って，行列 M が $n+m$ 以下の大きさの配列 ENTRY に圧縮できるとき，その縮小率は“良い”とする．ここでは，行列の av が与えられたとき，ffds 法による縮小率がどの程度であれば行列 M を $n+m$ 以下の大きさの配列 ENTRY に圧縮できるかを考える．このときの縮小率の最大値 rr_{max} は

$$rr_{max} = \frac{m}{n} \times 100(\%) \qquad (1)$$

で与えられ，$av = n/m$ より式(1)は

$$rr_{max} \frac{1}{av} \times 100(\%) \qquad (2)$$

となる．すなわち，av＝4 の場合であれば

$$rr_{max} = 25\%$$

となり，av＝5 の場合であれば

$$rr_{max} = 20\%$$

となる．この値を表 2 の結果と比較すると av が 4 以下であれば，行列 M が ffds 法により十分効率的に圧縮可能であるといえる．

以上より，本節では ffds 法が効率的に働く条件を

$$av \leqq 4$$

で与え，これを AV 条件と呼ぶ．

Tarjan らによる HD 条件は判定するために，ffd 法の Step 1 と 2 を必要とするので，$O(n)$ 時間（詳しくは $O(n)+O(m)$ から $n \geqq m$ を利用して $O(n)$ となる）を要する．これに対して，AV 条件の判定は $O(1)$ 時間でよい．また，次の定理（証明は付録 2 参照）より ffds 法の実行時間は $O(n^2+2mn)$ 時間となり，ffd 法の $O(n^2+m)$ 時間より若干多くなる．しかし，静的スパース行列の縮小法である点と縮小率の改善の特長を考慮すれば，これが大きな問題となること

* ただし，ff 法は 0 %に近い値となった．

```
0   *   *   0   *                                          r(2)=-1
            *   0   0   0   *                              r(5)=2
                0   *   0   0   *                          r(1)=3
                                        *   0   *   0   0  r(3)=8
                    0   *   0   0   0                      r(4)=4

    1   2   3   4   5   6   7   8   9   10  11
    t   t   t   t   t   t   t   t   t   f   t    ENTRY

t=true, f=false
```

図 4 ffds 法の結果
Fig. 4 Results of ffds method.

はない.

[定理 2] 行列 M が ffds 法により $n+m$ 以内の大きさの配列 ENTRY に圧縮されるとき，ffds 法の実行時間は $O(n^2+2mn)$ 時間となる.

(例2) 例1の行列 M は HD 条件を満足しないが，次式により AV 条件は満足する.

$$av=n/m=10/5=2.0$$

そして，ffds 法の行ソート 2，5，1，3，4 の順に**図4**の縮小結果が得られる.

例1での ffd 法の縮小率は $\alpha=3$ より 30% となるが，本章の改善による ffds 法の縮小率は $\alpha=1$ より 10%となる.

4. 行列縮小法の拡張

Tarjan らは行列 M が HD 条件を満足しない場合，行置換の前に列の置換も行うように縮小法を拡張した．すなわち，列の置換関数を c とし，列置換された行列 M に対応する行列を B とするとき，行列 M の非零要素 $M(i,j)$ は

$$B(i+c(j),j)$$

なる行列 B の要素に写像される．そして，この行列 B に対して ffd 法を適用して行列 M の圧縮を実現する．Tarjan らは列置換関数 c を決定する基準として次の ED 条件 (exponential decay condition の略) を提案した.

行列 M において最初に置換された j 個の列より成る行列を B_j とし，n_j を行列 B_j の全非零要素数とする．また，行列 B_j において $i+1$ $(i\geqq 0)$ 以上の非零要素を含むすべての行に存在する全非零要素数を $n_j(i)$ とするとき，ED 条件は次式で表される.

$$n_j(i)\leqq n_j/2^{i(2-n_j/n)}$$

しかし，ED 条件は HD 条件を満足する行列 B を行列 M から得るためのものであるので，結局は前章の結論に従い ED 条件も有効性が低いといえる.

本章でも，AV 条件を満足しない行列 M の縮小法への拡張を考えるが，次の性質を利用して AV 条件をそのままここでも使用する.

大きさが $e\times h$ でスパース率が q である行列 A と大きさが $e'\times h'$ でスパース率が q' である行列 A' において，

$$e\cdot h=e'\cdot h'$$
$$q=q'$$

が成立するならば，行列 A と A' は等価であるという．ただし，e,h,e',h' は2以上の整数である.

(性質1) 等価な二つの行列 A と A' のそれぞれの行の大きさ e と e' が

$$e>e'$$

ならば，ffds 法による縮小率は行列 A のほうが行列 A' よりよくなる.

性質1も経験的なものであるが，行列 A の av と平均 NZ 長が行列 A' よりともに小さくなる理由より容易に説明できる．この性質を実証する実験結果を**図5**に示す.

図5の結果は，$e\cdot h=10,000$ なる行列 A を基準とする等価な行列 A' の縮小結果であり，$e=100$ の場合の av は小さい順に 8.46，10.51，15.56 であるので，どれも AV 条件は満足していない．しかし，e が

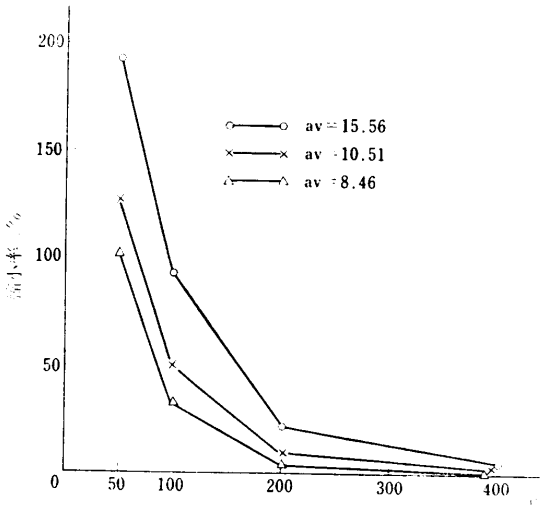

図 5 行の大きさと縮小率の関係
Fig. 5 Relation between row size and reduction rate.

200 から 400 へと大きくなると縮小率は激減し，また e が 50 へと小さくなると縮小率は激増することがわかる．厳密には，e が大きくなると行置換関数 r の記憶量も増加するので，縮小率のみで議論するのは十分でない．しかし，性質1による効果が大きいので，この関数 r の記憶増加を十分補償できる．

以上の結果より，本章では行列 M が

$\mathrm{av} \leqq 4$

なる AV 条件を満足しない場合，行列 M を次の大きさの行列 A に変換する．ただし，実数 a が整数 b に対して

$b-1 < a \leqq b$

を満たすとき，次の記法を使用する．

$[a]=b$

(手順1) 行列 M の行と列の大きさの変換

1) 行列 A の行の大きさ e を次のように決める．

$e=[n/4]$

2) 行列 A の列の大きさ h を次のように決める．

$h=[m^2/e]$

手順1の変換において

$m^2 \leqq e \cdot h$

が成立するので，行列 M に定義可能な値の総数 $N(=m^2)$ は行列 A においても保存される．そして，値 k に対応する行列 A の要素 $A(i,j)$ の i,j は

$i = k \ \mathbf{div}\ h+1,$

$j = k \ \mathbf{mod}\ h+1$

で表現される．そして，行列 A は AV 条件を満足するので，ffds 法により効率的に縮小可能となる．

本手法では，Tarjan らの方法に比べて次の特長を有する．

1) ED 条件を使って行列 M を行列 B に変換するには ff 法を利用するので，$O(n^2+m)$ 時間を要する[1]．しかし，AV 条件による行列 M から行列 A への変換は手順1により $O(1)$ 時間となる．

2) Tarjan らの列置換の導入は，列置換関数 c の記憶領域を必要とするが，本手法ではこれが不要となる．

5. むすび

以上，本論文では Tarjan らによって与えられた ffd 法の効率化の HD 条件を実験により具体的に評価し，ffd 法を ffds 法に，HD 条件を AV 条件に改善した．AV 条件は経験的なものであって HD 条件のような探索時間と記憶量に対する理論的保証はない．しかし，その適用範囲の広さと判定の容易さにより実際の行列縮小に対しては HD 条件より有効なものであると思われる．

本論文で提案した手法は文献3),5)～9)で述べられている記憶検索法に利用できる．そこで，具体例として XPL[10] の LALR(1) パーサ[3] の解析表に適用した．この表は大きさが 90×47，s が 8.1，av が 3.8 となり，HD 条件を満足しないが AV 条件は満足した．そして，ffds 法による縮小率は 0 %となった．

今後は，多くの応用例により本手法の有効性を確かめたい．

参考文献

1) Tarjan, R. E. and Yao, A. C.: Storing a Sparse Table, *Comm. ACM*, Vol. 22, No. 11, pp. 606-611 (1979).
2) Aho, A. V., Hopcroft, J. E. and Ullman, J. D.: *The Design and Analysis of Computer Algorithm*, pp. 44-74, Addison-Wesley, Reading, Mass. (1974).
3) Aho, A. V. and Ullman, J. D.: *Principles of Compiler Design*, pp. 73-124, 197-244, Addison-Wesley, Reading, Mass. (1977).
4) Knuth, D. E.: *The Art of Computer Programming*, Vol. 1, *Fundamental Algorithm*, pp. 295-304, Addison-Wesley, Reading, Mass., ibid., Vol. 3, *Sorting and Searching*, pp. 481-505 (1973).
5) 青江，福岡，山本，島田：不変要素をもつスパース行列の実際的縮小，信学論 (D)，Vol. 63-D, No. 12, pp. 1042-1049 (1980).
6) Tinsky: 表のたたみ込み，*bit*, Vol. 13, No. 2, プログラミングセミナー，pp. 52-57 (1981).
7) 青江，山本，島田：有限状態機械の効率的記憶検索法，信学論 (D)，Vol. 65-D, No. 10, pp. 1235-1245 (1982).
8) 青江，東條，山本，島田，稲田：パターンマッチングマシンの効率的記憶検索法，情報処理学会論文誌，Vol. 24, No. 4, pp. 414-420 (1983).
9) Aoe, J., Yamamoto, Y. and R. Shimada, R.: A Practical Method for Reducing Weak Precedence Parsers, *IEEE Trans. Softw. Eng.*, Vol. SE-9, No. 1, pp. 25-30 (1983).
10) Mckeeman, W. M., Horning, J. J. and Wortman, D. B.: *A Compiler Generator*, p. 126, Prentice-Hall, Englewood Cliffs (1970).
11) Moore, L.: *Fundamental of Programming with Pascal*, Ellis Horwood, New York (1980).
12) Tarjan, R. E.: Graph Theory and Gaussian Elimination, in Bunch, J. R. and Rose, D. J. (eds.): *Sparse Matrix Computations*, pp. 3-22, Academic Press, New York (1976).

付　　録

1. アルゴリズムB

図6に ffds 法であるアルゴリズムBを示す．Step 1 はアルゴリズムAの Step 1 に NZ 長の計算手順を追加したものである．Step 2 は行ソートであり，Pascal[11] により記述される次のレコード cell をリンクすることで実現される．

```
type cell=record num: integer;
                 next: ↑cell
          end.
```

したがって，root と P もレコード cell へのポインタである．**repeat** 文中の cond は次の条件式である．

(count(i)>count(p↑. num))
or((count(i)=count(p↑. num)) and
(length(i)≧length(p↑. num))
or(p=**nil**).

また，insert(p, i) は p=**nil** のときはリストの後に num=i, next=**nil** なるレコード cell を追加し，p≠**nil** のときは p の指す cell の前に num=i なるレコード cell を挿入する手続である．Step 3 は Step 2 の行ソートによる行置換関数の決定手順である．

2. 定理2の証明

Step 1 と Step 3 の配列 ENTRY の初期化は $O(n+m)$ 時間（ただし，$n+m>n\geqq m$ を利用してまとめてある）となり，Step 2 の行ソートは $O(m^2)$ 時間となる．Step 3 において i 行は count(i) 個の非零要素を含み，$r(i)$ は $1-m$ から $n+m-1$ の範囲の値をとるので，i 行に対する行置換関数 r の計算は

$O((n+2m-1)\cdot\text{count}(i))$

時間となり，Step 3 全体としては（配列 ENTRY の初期化は除く），$n\geqq m$ を利用して

$$O((n+2m-1)\sum_{i=1}^{m}\text{count}(i)+m)$$

```
Algorithm B.  First-Fit Decreasing and Shortening Method.
Input:  Nonzero position (i,j).
Output: The row displacement function r and the array ENTRY.
Method:
    begin
Step 1. for i:=1 to m do
            begin
               count(i):=0; list(i):=ϕ; length(i):=0; min(i):=0
            end;
        for each nonzero position (i,j) do
            begin
               if min(i)=0 then min(i):=j;
               length(i)=j-min(i)+1;
               count(i):=count(i)+1;
               list(i):=list(i)∪{j}
            end;
Step 2. root↑.num:=0; root↑.next:=nil;/* dummy cell */
        for i:=1 to m do
            begin
               p:=root;
               repeat p:=p↑.next until cond;
               insert(p,i)
            end;
Step 3. for k:=1 to n+m do ENTRY(k):=false;
        p:=root↑.next;
        repeat
               i:=p↑.num; r(i):=1-min(i);
  overlap:     for each j in list(i) do
                  if ENTRY(r(i)+j)=true then
                     begin
                        r(i):=r(i)+1; goto overlap
                     end;
               for each j in list(i) do ENTRY(r(i)+j)=true;
               p:=p↑.next
        until p=nil
    end.
```

図6 ffds 法
Fig. 6 ffds method.

$$=O(n^2+2mn-n+m)$$
$$=O(n^2+2mn)$$

時間となる．ここで，$n^2+2mn>m^2$ であるから ffds 法の実行時間は $O(n^2+2mn)$ 時間となる．

（昭和59年2月 6 日受付）
（昭和59年9月20日採録）