ブール行列の乗算アルゴリズムの高速化について†

高岡忠雄†† 福地陽一†††

本論文では，ブール行列の乗算のための新しく能率のよいアルゴリズムを開発し，計算機実験により，従来のアルゴリズムと性能の比較をしている．

ブール行列 A, B が与えられたとき，その積 $C=A\cdot B$ は

$$c_{ij}=\sum_{k=1}^{n} a_{ik}\cdot b_{kj}$$

で与えられる．上の計算を定義通りやると，もちろん $O(n^3)$ の手間がかかる．この計算において，$a_{ik}\cdot b_{kj}=1$ となるところで Σ の計算を打切るようにすれば，1の発生確率を p として，$O(n^3/p^2)$ の時間で計算できる．この方法をループ脱出法と呼ぶことにする．

従来知られているブール行列乗算の高速化の代表的なものは Strassen 法と4人のロシア人の方法である．Strassen 法では，ブール代数を環の中に埋め込むことによって，$O(n^{2.81})$ で計算する．4人のロシア人ではビットパタン 1101 を数値 13 のようにデータ圧縮することによって，$O(n^3/\log n)$ の時間で計算する．

本論文の方法は，4人のロシア人におけるデータ圧縮の考え方とループ脱出法の考え方を共存させることにより，平均 $O(n^3/(p^2\log n))+O(n^2)$，最悪 $O(n^3/\log n)$ の時間で走る．計算機実験でも，このアルゴリズムが速く走ることを確かめた．したがって，本論文の方法は実用的評価に耐えるものと考えられる．

1. まえがき

グラフ理論において推移的閉包 (transitive closure) は重要な役割を果す．推移的閉包を求めることは与えられたグラフのすべての節点の対に対し，その間の到達可能性を決定することである．

推移的閉包は，Karp[1]によれば次のように実用的応用性を持っている．いま大きなプログラムをいくつかのセグメントに分けて一部を二次記憶媒体上に置き，必要に応じてオーバーレイするような状況を考える．与えられたプログラムにおける制御の流れをグラフとしてモデル化する．このグラフの推移的閉包を求め，それよりこのプログラムをどのようにセグメントに分ければオーバーレイの回数を少なくできるかを決めることができるであろう．

推移的閉包を高速に求めるアルゴリズムは，まず Warshall[2]によって実行時間 $O(n^3)$* のものが求められた．また Purdom[3]によって実用上より速いアルゴリズムが提出された．一方，Warshall によっても指適されたように，推移的閉包を計算することとブール行列の積を計算することとは密接な関係がある．Fischer and Meyer[4] と Munro[5] を合せ読めば推移的閉包の計算の複雑さ (computational complexity) とブール行列の積のそれとが同じオーダであることがわかる．したがって，ブール行列の積を高速に求めることは，理論上及び実用面から重要な課題となる．本論文では，n 次元のブール行列の乗算アルゴリズムとして，中位の n に対し従来知られているものより速いアルゴリズムを提出する．

一般にブール行列の乗算は定義的に計算しようとすると次のようになる．algol 風に書くと，

〔初等的アルゴリズム〕

```
1 for I :=1 step 1 until N do
2 for J :=1 step 1 until N do
3 C[I,J] := Σ (K=1..N) A[I,K]・B[K,J];
```

ここで A, B は入力される $n\times n$ のブール行列，C は積を表す行列である．そして，“・”，“+”はブール演算における乗算，加算を表わす．(以後，本論では特にことわりのない場合は，“・”，“+”はブール積，ブール加算を表す．) このアルゴリズムの実行時間は，$O(n^3)$ である．そこでこの実行時間を何とか減らせないかといろいろなアルゴリズムが提案された．Strassen[6] の方法を応用したアルゴリズム Furman[7] においては，$O(n^{2.81})$，Four Russians[8] においては，$O(n^3/\log n)$ である．本論においては，4人のロシア人のアルゴリズム (Four Russians' algorithm) を改良したよ

---

† On the Speed-up of Algorithms for Boolean Matrix Multiplication by TADAO TAKAOKA and YOICHI FUKUCHI (Faculty of Engineering, Ibaraki University).

†† 茨城大学工学部情報工学科
††† 三菱電機(株)電子技術部

* ここでは n はグラフの節点の数を表す．

り速いアルゴリズムを提出する．そしてこのアルゴリズムと他のアルゴリズムを比較検討する．なお，本論の中でのアルゴリズムは，algol 風で書かれている．

また，本論文では実行時間の測定は，ALGOL や FORTRAN のような高級言語を基礎にして考える．すなわち基本的な操作としては，四則演算と ALGOL の制御構造及び記憶装置での random accessability を仮定する．

2. 4人のロシア人のアルゴリズム

Strassen のアルゴリズムによる行列の乗算方法は，漸近的に速いアルゴリズムである．（ブール行列の乗算についても同じことがいえる．Aho et al.[9]）そこで本章の4人のロシア人のアルゴリズムであるが，中位の n に対して，より実用的なアルゴリズムといえる．これは容易にビットベクトルの計算に帰着できる．まずそのアルゴリズムを提出しておこう．（以下のアルゴリズムは，Aho et al. より転載された．）

〔4人のロシア人のアルゴリズム〕

```
begin
1  for i := 1 step 1 until ⌈n/m⌉ do
     begin
       comment We compute the sums of the rows
               b_1^(i), ..., b_m^(i) of B_i;
       ROWSUM [0] := [0, 0, ..., 0];
                     (n)
3      for j := 1 step 1 until 2^m − 1 do
         begin
4          let k be such that 2^k ≤ j < 2^(k+1);
5          ROWSUM [j] := ROWSUM [j − 2^k] + b^(i)_(k+1);
         end;
6      let C_i be the matrix whose jth row, 1 ≤ j ≤ n,
       is ROWSUM [NUM (a_j)], where a_j is the jth
       row of A_i;
     end;
7  let C be Σ_{i=1}^{⌈n/m⌉} C_i;
end;
```

ここでアルゴリズムの説明に移るわけであるが，$m := \lfloor \log n \rfloor$ とし，A の行列を $A_1, \cdots, A_{\lceil n/m \rceil}$ に分割する．A_i は A の第 $m(i-1)+1$ 列から第 mi 列までから成る．$A\lceil n/m \rceil$ は残りの列を拡大して m 列にするため必要ならば "0" から成る特別の列を付け加えて構成される．B の行列は $B_1, \cdots, B_{\lceil n/m \rceil}$ に分割される．B_i は B の第 $m(i-1)+1$ 行から第 mi 行までから成る．$B\lceil n/m \rceil$ は残りの行を拡大して m 行にするため必要ならば "0" から成る特別の行を付け加えて構成される．それから NUM (v) なる関数はあるビットパターン v を数値化するものでたとえば，NUM ([0, 1, 1])=6 である．

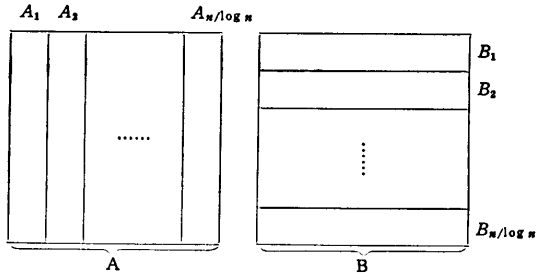

図 1 行列 A, B の分割
Fig. 1 Partition of boolean matrices A and B.

4人のロシア人のアルゴリズムが何故に，$O(n^3/\log n)$ で実行されるかは次の説明で理解される．今便宜上 n は，$\log n$ で割りきれると仮定すると，**図1**のように，A を $n \times (\log n)$，B を $(\log n) \times n$ の小行列達に分割することができる．

$A_i \cdot B_i$ を計算するのに，$O(n^2)$ の実行時間がかかる．なぜなら，$A_i \cdot B_i$ を計算するために，A_i の各行 a_j について，$a_j \cdot B_i$ を求めるために，a_j が "1" をもつ列番号 k ごとに B_i の第 k 行を見つける．こうして B_i の中でみつけた行を n 次ビットベクトルとみなし全部加える．ところで A_i のすべての行の中には，高高 $2^{\log n} = n$ 個の異なった行しか存在しないので，B_i の行の加え方は n 通りしかない．すべての加え方について，B_i の行のすべての n 通りの部分和を前もって計算して ROWSUM に入れておく．（アルゴリズムの 3, 4, 5 行によって計算時間 $O(n^2)$）そして，$a_j \cdot B_i$ を計算するかわりに，単に答を探すために，a_j によって表を引けば良い．(ROWSUN [NUM (a_j)] によって表を引けば良い．）これより $A_i \cdot B_i$ を計算するのに，$O(n^2)$ の実行時間を要することになる．

$$C[i, j] := \sum_{k=1}^{n/\log n} C_k[i, j] \quad \text{なので}$$

A と B の乗算には全体で，$O(n^3/\log n)$ の実行時間がかかることになる．

3. テーブル参照法によるスピードアップ

このアルゴリズムは前処理の段階（A, B を小行列に分割すること）では同じであるが，その後の処理のしかたが，よりスピーディにかつ簡素化されている．それは，新しく TABLE なる配列を導入し，表を見ることによりブール行列の乗算を求める方式をとっているからである．ではまず，アルゴリズムを提出してみよう．

〔テーブル参照法 (Modified algorithm)〕

```
begin
1  for i:=1 step 1 until ⌈n/m⌉ do
2      for j:=1 step 1 until n do
         begin
3          DA[j,i]:=NUMA(A_i,j,m);
4          DB[i,j]:=NUMB(B_i,j,m);
         end;
5  for i:=1 step 1 until n do
6    for j:=1 step 1 until n do
       C[i,j]:=Σ_{k=1}^{⌈n/m⌉} TABLE[DA[i,k], DB[k,j]];
end;
```

4人のロシア人のアルゴリズムにおいては，A_i の行ベクトルに応じて B_i の行ベクトルを選び出して部分和をつくりそれより C_i を作る．それに対してこのアルゴリズムは A_i の行ベクトルのビットパターンをNUMAなる関数により数値化 (NUMA (A_i, j, m) は小行列 A_i の第 j 行の m ビットを左を下位のビットとして（数値化する），小行列 A_i を DA_i という列ベクトルに変換するものである．そして B_i の列ベクトルに対しては，そのビットパターンを NUMB なる関数により数値化 (NUMB (B_i, j, m) は小行列 B_i の第 j 列の m ビットを上を下位のビットとし数値化する）して，小行列 B_i を DB_i という行ベクトルに変換するものである．これによってできあがる，第 i 列が DA_i なる行列 DA と第 i 行が DB_i なる行列 DB より，求めるべき $C=A\cdot B$ を導出する．DA と DB で乗算するさいに，このアルゴリズムの最大のポイントである TABLE なる配列を利用してスピーディに行う．例えば，TABLE(5,6) を参照すると1が発見されるが，これは初等的アルゴリズムで 101 と 011 の AND と OR の操作，すなわち $1\cdot0+1\cdot0+1\cdot1=1$ を一挙に行うわけである．

このアルゴリズムを解析してみよう．まず 1〜4 行で DA, DB を計算するが，この時間は明らかに $O(n^2)$ かかる．なぜなら NUMA, NUMB を計算するのに $O(m)$ かかるので，1〜4 行全体で $O(n^2)$ である．5〜7 行においては，TABLE なる配列を使って $C=A\cdot B$ を計算するのに，DA と DB で計算することができる．

$$C[i,j]:=\sum_{k=1}^{\lceil n/m\rceil}\mathrm{TABLE}\,(DA(i,k),\ DB(k,j))$$

として計算する．これによれば $O(n^3/\log n)$ の実行時間がかかる．前章のアルゴリズムとオーダ的には変わりないが，4人のロシア人のアルゴリズムは ROWSUM の配列を何度も作るので，それに $O(n^3/\log n)$ かかる．この ROWSUM を作るかわりに DA, DB の行列を作って一括して計算しているので実際の実行時間は速くなる．メモリにおいても，C_i の配列を作らない分テーブル参照法がまさっている．

4. テーブル作成のアルゴリズム

テーブル参照法においては，TABLE という配列を使用するが，この入力はカード読み込みにより入力してやってもよいがそれが面倒な場合次のような再帰的手続 Get Table (N, TABLE) で作ることができる．そのアルゴリズムを提出しておこう．このアルゴリズムは，Table 1 の 4×4 行列において左上，右上及び左下の小行列は同じものであり右下にはすべて "1" の小行列があるという事実に着目している．

```
procedure Get Table (N, TABLE);
  value N;
  integer N;
  integer array TABLE;
  begin
1   if N=1 then TABLE[0,0]:=0 else
      begin
2       Get Table (N/2, TABLE);
3       for I:=N/2 step 1 until N-1 do
4         for J:=0 step 1 until N/2-1 do
            begin
5             TABLE[I,J]:=TABLE[I-N/2,J];
6             TABLE[J,I]:=TABLE[J,I-N/2];
7             TABLE[I,J+N/2]:=1;
            end;
      end;
end;
```

ここで TABLE なる配列の説明に移る．この配列を見ることにより，2つの数のビットパターンのブール演算を行うものである．（アセンブラにおいては，AND という関数によりこの操作は容易にできるが，本論においては高級言語でのアルゴリズムの研究であり，アセンブラ言語のようにビットごとの演算はできないと考える．）

たとえば，入力ブール行列の大きさが $N=4$ の場合を考えて見よう．

TABLE[1,2]=0 となるのは，右を下位の桁として

$1_{10}=01_2$

表 1 修正されたアルゴリズムのためのテーブル
Table 1 TABLE used for modified algorithm.

$N=4$

	0	1	2	3
0	0	0	0	0
1	0	1	0	1
2	0	0	1	1
3	0	1	1	1

$2_{10}=10_2$

左辺が TABLE の要素であり，右辺はそれを2進化したビットパターンである．右辺のそれぞれの数を桁ごとに見てゆく．

TABLE [1, 2]=0×1+1×0=0+0=0

同様に，TABLE (2, 3)=1 となるのは

TABLE [2, 3]=1×1+0×1=0+1=1

このように，ある2つのビットパターンを見た時，おのおの同じ桁に "1" がひとつでもあれば，その TABLE の評価は "1" となる．

したがってこの TABLE をあらかじめ計算しておき，その表を使ってブール行列の乗算を計算することができる．なお，この TABLE 作成の手続の実行時間が $O(n^2)$ であることは明らかである．従って，入力文や代入文で TABLE を作らず，プログラム的に作ってもかなり高速に作成される．

5. 例

例を使ってアルゴリズムの流れを追ってみよう．まず入力ブール行列 A, B を次のようにする．($n=4$)

$$A=\begin{bmatrix}1&0&0&0\\1&1&0&1\\0&1&1&0\\0&1&1&0\end{bmatrix}\quad B=\begin{bmatrix}0&1&0&1\\0&1&1&0\\0&0&1&1\\1&0&0&0\end{bmatrix}$$

まず，$m=\lfloor \log n \rfloor=2$，であり，次のように分割する．

$$A_1=\begin{bmatrix}1&0\\1&1\\0&1\\0&1\end{bmatrix}\quad A_2=\begin{bmatrix}0&0\\0&1\\1&0\\1&0\end{bmatrix}\quad B_1=\begin{bmatrix}0&1&0&1\\0&1&1&0\end{bmatrix}\quad B_2=\begin{bmatrix}0&0&1&1\\1&0&0&0\end{bmatrix}$$

分割されたそれぞれの小行列は，テーブル参照法によって新しい行列 DA, DB を作る．

$$DA\begin{bmatrix}1&0\\3&2\\2&1\\2&1\end{bmatrix}\quad DB=\begin{bmatrix}0&3&2&1\\2&0&1&1\end{bmatrix}$$

次に配列 TABLE を参照しながら C を導出する．

$$C=\begin{bmatrix}0+0&1+0&0+0&1+0\\0+1&1+0&1+0&1+0\\0+0&1+0&1+1&0+1\\0+0&1+0&1+1&0+1\end{bmatrix}=\begin{bmatrix}0&1&0&1\\1&1&1&1\\0&1&1&1\\0&1&1&1\end{bmatrix}$$

このような方法によってブール行列の乗算を行う．

6. 平均 $O(n^2)$ のアルゴリズム（ループ脱出法）

この章においては，初等的アルゴリズム，4人のロシア人のアルゴリズム，テーブル参照法に共通して使っているブール加算の回数を減らすことによって得られる平均 $O(n^2)$ のアルゴリズムを提出する．

ここでは，初等的アルゴリズムを通して考えてみる．入力ブール行列 A, B について，その行列要素が "1" となる確率を p，"0" となる確率を $(1-p)$ のように入力するとする．本来は，

$$C[i, j]=\sum_{k=1}^{n} A[i, k]\cdot B[k, j]$$

によって求めるのだが，この総和の計算をするとき，$A[i, k]\cdot B[k, j]=1$ ならば $C[i, j]=1$ とする．すなわち，ここで k のループを出るわけである．この方法をループ脱出法という．

k 番目で始めて，1となる確率は，$(k-1)$ 番目までは $A[i, k]\cdot B[k, j]=0$ でなくてはならない．その確率は，$(1-p^2)^{k-1}$．そして，k 番目で1となる確率が p^2 である．したがって，k 番目で始めて1となる確率は，$(1-p^2)^{k-1}\cdot p^2$ となる．このことより，$A[i, k]\cdot B[k, j]=1$ となる平均の k を求めると，

$$\sum_{k=1}^{n}(1-p^2)^{k-1}\cdot p^2\cdot k$$

$$=\frac{1-(1-p^2)^n-np^2(1-p^2)^n}{p^2}\leq\frac{1}{p^2}$$

（この場合の演算は普通の四則演算である）

となる．$n\to\infty$ にすると，$1/p^2$ となる．したがって，$C[i, j]$ を求める場合，n が漸近的に大きくなった場合，$1/p^2$ 番目において $A[i, k]\cdot B[k, j]=1$ となり，$C[i, j]$ が求まる．故に，$A\cdot B=C$ を導出するためには，$O(1/p^2\cdot n^2)$ つまり $O(n^2)$ の実行時間で計算できる．このようなループ脱出法は，4人のロシア人のアルゴリズム，テーブル参照法にも使用できる．ただし，4人のロシア人のアルゴリズムでは，ループ脱出法でスピードアップできるのはプログラムの7行目において C_i 行列を加えるときだけである．3, 4, 5 行目の ROWSUM の計算は依然として $O(n^3/\log n)$ の時間を要する．テーブル参照法では，プログラムの5, 6, 7 行目でこの方法が使える．従ってこの場合，全体の実行時間は $O(n^2)$ となるが，7行目の文のテーブル参照の回数は，n が十分大きいとき大体 $\lceil n^2/(p^2\cdot\log n)\rceil$ となる．したがって，初等的アルゴリズムもテーブル参照法も，ループ脱出法によると実行時間は $O(n^2)$ であるが（厳密にいえば後者の実行時間は $O(n^2)+O(n^2)/(P^2\log n)$），後者の場合，$n^2$ の前の係数がかなり小さくなるわけである．なお，strassen の方法ではブール代数を環の中に埋め込むため本章の方法は使うことはできないことに注意しておこう．

7. あとがき

ここで，ブール行列の乗算に関するアルゴリズムを実際，計算機 (HITAC 8350 の上でのプログラミング言語 EDOS ALGOL) を使用して実験した結果を提出しておく．(最悪の場合とは，入力行列の要素が all-zero のときであり，ループ脱出法を用いないのと同じである．)

表 2 EDOS ALGOL を用いて HITAC 8350 の上で計則した各ブール行列乗算アルゴリズムの実行時間

Table 2 Net execution time of each boolean matrix multiplication algorithm on HITAC 8350 using EDOS ALGOL.

N=32 (matrix scale)

algorithm	ループ脱出法	最悪の場合
初等的	10.0(S)	33.2(S)
Strassen's		48.5
4人のロシア人	8.5	11.7
テーブル参照法	2.5	5.9

なお，入力行列は計算機の基本関数 RANDOM (0から1までの一様乱数) を 10 倍し，3未満ならば "1"，その他は "0" とした．したがって前章で述べた確率 p は，3/10となる．ここで初等的アルゴリズムにおいて，時間が約 1/3 となっているのは理論通りであり興味深い所である．

さて，テーブル参照法が何故4人のロシア人のアルゴリズムより速くなったかを考えてみよう．その原因は，4人のロシア人のアルゴリズムにおいては，ROWSUM が $(n/\log n)$ 回計算され，したがってその実行時間が合計 $O(n^3/\log n)$ になったのに対し，テーブル参照法では，TABLE を $O(n^2)$ 時間で一度だけ計算しておけばよかったことになる．また，TABLE が一度計算されるとそれを使って多くのブール行列の積が計算できることも長所の一つである．

使用メモリについても考察しておこう．入力と出力の $n\times n$ 配列 A, B, C はどうしても必要ということに対して，それ以外に何が必要かを考える．4人のロシア人のアルゴリズムでは，$n\times n$ 配列 ROWSUM が一つ．それからループ脱出法を使うときは，C_i が $(n/\log n)$ 個必要であり，使わないときは，結果の行列 C に ROWSUM からとってきた行を加算すれば良いから，$C_1, C_2, \cdots$ は実際にはいらない．テーブル参照法では，$n\times n$ の TABLE が一つだけでよい．なぜなら，DA, DB は A, B の場所を使うことができるからである．

最後に，本論でとり上げた4つのアルゴリズムの長所，短所を表にしておこう．

表 3 4つのアルゴリズムの比較

Table 3 Comparison of four algorithms.

algorithm	初等的	Strassen's	4人のロシア人	テーブル参照法
プログラム	簡単	複雑	やや複雑	簡単
ループ脱出法	可能	不可能	部分的可能	可能
A, B, C 以外のメモリ	なし	大	中	小
ループ脱出法による平均時間	$O(n^3/p^3)$		$O(n^3/\log n)$	$O(n^2)+O(n^3/(p^2\log n))$
最悪の場合の実行時間	$O(n^3)$	$O(n^{2.81})$	$O(n^3/\log n)$	$O(n^3/\log n)$

参考文献

1) R. M. Karp: A Note on the Application of Graph Theory to Digital Computer Programming, Inf. Control, Vol. 3, pp. 179-190 (1960).
2) S. Warshall: A Theorem on Boolean Matrices, J. ACM, Vol. 9, No. 1, pp. 11-12 (1962).
3) Paul Purdom Jr.: A Transitive Closure Algorithm, BIT, Vol. 10, pp. 76-94 (1970).
4) M. J. Fischer and A. R. Meyer: Boolean Matrix Multiplication and Transitive Closure, Conference Record, IEEE 12th Annual Symposium on Switching and Automata Theory, pp. 129-131 (1971).
5) I. Munro: Efficient Determination of the Transitive Closure of a Directed Graph, Information Processing Letters, 1, pp. 56-58 (1971).
6) V. Strassen: Gaussian Elimination is not Optimal, Numerische Mathematik, Vol. 13, pp. 354-356 (1969).
7) M. E. Furman: Application of a Method of Fast Multiplication of Matrices in the Problem of Finding the Transitive Closure of a Graph, Soviet Math. Dokl., Vol. 11, No. 5 (1970).
8) V. L. Arlazarov, E. A. Dinic, M. A. Kronrod and I. A. Faradzev: On Economical Construction of the Transitive Closure of a Directed Graph, Soviet Math. Dokl., Vol. 11, No. 5, pp. 1209-1210 (1970).
9) A. V. Aho, J. E. Hopcroft, and J. D. Ullman: The Design and Analysis of Computer Algorithms, Addition-Wesley (1974).

(昭和53年3月25日受付)
(昭和53年5月25日採録)