アルゴリズム 11－4
（1989. 10. 24）

動的Blocked Warshall法による推移的閉包アルゴリズム

舘下 直純　　大柳 俊夫　　大内 東
北海道大学 工学部 情報工学科

関係データベースの分野における再帰質問処理は、最終的に関係の推移的閉包を求めるものが多い。その推移的閉包を求めるアルゴリズムの一つに動的Blocked Warshall法がある。しかしながら、従来の動的Blocked Warshall法では使用者自身が前もって分割を与えなければならない。

本報告では、分割を自動的に決め二次記憶へのアクセスをできるだけ少なくする”改訂動的Blocked Washall法”を提案する。また、提案する方法と従来の方法を実験により比較し、その有効性を確認する。

The Algorithm for The Transitive Closure by The Dynamic Blocked Warshall Algorithm

Naosumi Tateshita, Tosio Ohyanagi and Azuma Ohuchi
Department of Information Engineering, Faculty of Engineering, Hokkaido University

N-13, W-8, Kita-Ku, Sapporo 060, Hokkaido, Japan

In this paper, we present a new algorithm for computing the transitive closure of database relations. The algorithm is a revised version of Dynamic Blocked Warshall Algorithm. The proposed algorithm minimizes I/O traffic, and there is no need to create partitions ahead of time.

We also present simulation results that show that this proposed algorithm perform uniformly better than the Dynamic Blocked Warshall Algorithm.

1．はじめに

関係データベースの分野において再帰的質問処理というテーマがある。これは親子関係から先祖関係を求めるようなもので、再帰的に自分を呼び出して処理するものである。それらは最終的に関係の推移的閉包を求めるものが多いことが知られている。その場合これは二項関係において、定義域(domain)の値を頂点、組(tuple) を辺と考えた場合の有向グラフの推移的閉包を求める問題と置き換えて考えることができる。

二次記憶環境下で大規模関係データベースの推移的閉包を求めるアルゴリズムの一つに動的Blocked Warshall法[1] がある。この方法はBlocked Warshall法における行列の分割を動的に行い、主記憶の容量の制約に柔軟に対応できるように改良したものである。しかしながら、文献[1] の方法は二次記憶へのアクセスが頻繁に起こり、推移的閉包計算の効率の悪化をまねく場合が多くなる。また実験は、静的な分割に対してのみ行われており、動的分割に対する実験データは示されていない。さらに、使用者自身が分割を与えなければならないが、はじめに適切な分割を与えることは難しいと考える。

本報告では、処理の順序を変えることにより、二次記憶へのアクセスをできるだけ少なくし、最適な分割をつくる新たな動的分割の方法を提案する。また、提案する方法と従来の方法を実験により比較検討する。

2．データ構造　および　戦略

ここでは、頂点ｉから到達可能である頂点の集合を頂点ｉの後続者リストとして持っている。

従来のWarshall法及びWarren法の基本戦略として、「ある頂点番号ｋ以下の頂点からなる後続者リストが計算済みである。」ということが挙げられる。これは次の二つの先行条件で表すことができる。

1.(i,k) precedes (i,j) for all k<j,
for all i.
2.(j,k) precedes (i,j) for all k<j,
for all i.

ここで用いた(i,j) は、頂点ｉから頂点ｊへ辺があるかどうかを調べ、それがあるときは書き換えをすること（ｊの後続者リストをｉの後続者リストに重複しないように同じものは省いて加えること）を意味する。そのような検査と書き換えを合わせて要素(i,j) の処理と呼ぶ。

3．動的分割および静的分割について

ここで用いられるデータは、その推移的閉包が主記憶の容量に対して大きすぎて全てのデータを主記憶にいれて計算することが不可能なものであるとする。そこで主記憶の大きさに合うようにいくつかに分割して計算を行わなければならない。分割はいくつかの頂点の後続者リスト（その頂点から到達可能である頂点のリスト）からなる。さらにこれらのリストは推移的閉包を計算するにつれて大きくなり、最初の分割では主記憶に合わなくなってしまう。もし分割が静的であれば、リストが大きくなりすぎて主記憶の容量をオーバーする場合には、その分割をより小さいものにして再計算しなければならなくなる。あるいはそれを恐れて分割をあまりにも小さいものとして計算すると一度に処理できるものをわざわざ分けて計算するのだから当然効率の悪いものとなる。そこで処理中に主記憶が溢れた場合に柔軟に対応して処理するための動的分割を考える。

以下で用いる”対角ブロック”とは、その行番号と列番号がともに現在主記憶上にあるような分割の範囲の中にあるような処理の対象をいう。それに対して対角ブロックにないものを”非対角”という。

4. Blocked Warshall法

a)静的分割アルゴリズム

静的分割アルゴリズムを図１に示し、５行５列対角ブロックの処理順序を図２に示す。

b)動的分割アルゴリズム

Blocked Warshall法では最初に列分割に当たる列の後続者リストを主記憶に読み込んで、対角ブロックの処理を行い、その後で非対角行の後続者リストを読み込んで処理をする。非対角行の処理においては、対角ブロックにある後続者リストに加えられるものはなく、すべてその非対角行の後続者リストに加えられ、次の非対角行の処理のための後続者リストを読み込む前に二次記憶に書き戻されるので、主記憶上のリストが増えることはない。よって対角ブロックの処理を除いて主記憶が溢れることはないと考えて良いとしている。しかし、非対角行の処理において後続者リストを読み込むための容量がない場合には非対角行の後続者リストを読み込んだ時点で主記憶が溢れる。その場合には列分割の最後の列を次の分割に充てれば良い。（対角ブロックの最後の後続者リストを放棄する。）

対角ブロックの処理では、あらかじめ分割を決め、その分割に対応する後続者リストを全て主記憶上に読み込み（当然読み込み中に主記憶が溢れたらそこまでしか読み込めないが）、列方向に処理を行っていく。その処理中に主記憶が溢れたら列分割の最後の列に対応する後続者リストを放棄、あるいは二次記憶に書き戻し、縮小した分割で対角ブロックの処理を続けて行う。さらに主記憶が溢れた場合には、いつでも同様の処理を施す。

それぞれの列分割に対して

（列分割はJb～Je）

```
/* 対角ブロックの処理 */
For j= Jb to Je
   For i= Jb to Je
      If  辺(i,j) がある  then
          jの後続者リストを
              iの後続者リストに加える。

/* 非対角行の処理 */
For i=1 to n
```

但し、対角ブロックで処理を終えた行は除く。

```
   For j=Jb to Je
     If  辺(i,j) がある  then
         jの後続者リストを
             iの後続者リストに加える。
```

図１　静的分割Blocked Warshall法

1	6	11	16	21
2	7	12	17	22
3	8	13	18	23
4	9	14	19	24
5	10	15	20	25

図２　Blocked Warshall法における
５行５列対角ブロックの計算順序

5．改訂動的Blocked Warshall法

今回我々が提案する方法を”改訂動的Blocked Warshall法”と呼ぶ事にする。

a)アルゴリズムの概略

アルゴリズムの概略を図３に示し、５行５列対角ブロックの処理順序を図４に示す。

b)アルゴリズムの説明

列分割に属する列にあたる頂点の後続者リストは、最初の一つはその列分割の処理が始まる前に主記憶に読み込んでくる。次の頂点の後続者リストからは、その頂点の処理が始まる前に主記憶に読み込んでくる。そしてその列分割の非対角行の処理がすべて終わってから、その列分割に属する列にあたる頂点の後続者リストをすべて二次記憶に書き戻す。

列分割に属さない列に当たる頂点の後続者リストは、その頂点の非対角行の処理が始まる前に主記憶に読み込んで、処理が終わったらすぐに二次記憶に書き戻す。

このアルゴリズムにおいて分割をあらかじめ設定しない。列分割の先頭Jbは前の列分割の最後の列Jeに１を加えたものである。（最初は１）、いまの列分割の最終列Jeは決めずに基本的に主記憶の容量が溢れるまで（当然［Je≦すべての頂点数］である。）分割の列数を増やしていく方法を考える。そのときに主記憶の容量が溢れるのは行または列方向の処理中である。また非対角行の処理中においても主記憶が溢れることがありえる。その場合の対応をそれぞれ b1)～ b3)に示す。

b1) ｉ行の行方向の処理中に
主記憶の容量が溢れた場合

いま確定している列分割の最後の列に対応するi-1 の後続者リストを書き戻し、列分割はi-2 までと考え対角ブロックを形成する。それからｉ行

それぞれの列分割に対して（列はJb～）

```
/* 対角ブロックの処理 */
k=Jb-1
repeat
   k=k+1
   For j=Jb to k-1
     i=k
     If  辺(i,j) がある  then
         ｊの後続者リストを
              ｉの後続者リストに加える。
     If  主記憶が溢れた  then
         b1) の処理を行い、
                  非対角行の処理に続く。
   For i=Jb to k
     j=k
     If  辺(i,j) がある  then
         ｊの後続者リストを
              ｉの後続者リストに加える。
     If  主記憶が溢れた  then
         b2) の処理を行い、
                  非対角行の処理に続く。
 until（主記憶が溢れる）

 /* 非対角行の処理 */
 For i=1 to n
```

但し、対角ブロックで処理を終えた行は除く。

```
    For j=Jb to Je
                （確定した列分割はJb～Je）
      If  辺(i,j) がある  then
          ｊの後続者リストを
               ｉの後続者リストに加える。
      If  主記憶が溢れた  then
          b3) の処理を行う。
```

図３　改訂動的Blocked Warshall法

の残った部分の行方向の処理を行い、ｉの後続者リストを二次記憶に書き戻す。それによって、非対角行の処理の際には i-1行とｉ行の処理を省略することができる。

b2) ｊ列の列方向の処理中に
主記憶の容量が溢れた場合

現在処理中の列の前の j-1列までを列分割と考え、対角ブロックを形成する。処理を終えたｊ行については非対角での処理が終わったものと考え後続者リストを二次記憶に書き戻す。それによって、非対角行の処理の際にはｊ行の処理を省略することができる。

b3) ｉ行の非対角の処理中に
主記憶の容量が溢れた場合

列分割の最後の列にあたるJeの後続者リストを二次記憶に書き戻し、列分割をJe-1までとし、残りの非対角行の処理をする。そうしてJeを次の列分割に割り当てる。そして新たに列分割の最後の列をJe列とする。

また、非対角行ｉの処理でｉの後続者リストに加えられるものがなかった場合（対角ブロックをJb～Jeとすると、(i,Jb),(i,Jb+1),...,(i,Je)が存在しないか、あるいは存在してもJb～Jeの後続者リストにあってｉの後続者リストにはないものが一つもない場合）には、ｉの後続者リストはその処理が終わった後、二次記憶に書き戻す必要はないので（二次記憶にはそのリストそのものが残っているので）、そのまま放棄すればよい。

c)先行条件の充足性

もし、(i,j) が対角ブロックにあれば、すべての(i,k) はいつもその行の(i,j) よりも左側に現れる。それはすでに処理済みの列分割か、現在処理中の対角ブロックにある。すでに処理済みの列分割中にあるものについては問題はない。現在処理中の対角ブロックにおいて行方向処理中の時はその行を左側から（頂点番号の小さい順から）処理しているのですでに処理済みであり、問題はない。次に列方向の処理中はその列の左側の列の対角ブロック内は処理済みなので問題はない。よって１番目の先行条件を満たす。２番目の先行条件については、すべての(j,k) はすでに確定した対角ブロックの中またはそれらの行の左側にあるので、これも同様に処理済みである。よって、対角ブロックの中に(i,j) がある時は先行条件を両方とも満たしている。

次に(i,j) が非対角行にあるときについては、現在の列分割の前の列分割まではすでに処理済みであり、現在の列分割の非対角行を左側から処理しているので、１番目の先行条件については満足している。２番目の先行条件についても、(j,k) はすでに確定した対角ブロックかその左側の列分割にあるので、処理済みである。よって２番目の先行条件も満たす。

対角ブロック、非対角行すべてにおいて１、２の先行条件を満たしているので、この順序で計算（処理）することについて問題はない。

1	3	7	13	21
2	4	8	14	22
5	6	9	15	23
10	11	12	16	24
17	18	19	20	25

図４　改訂動的Blocked Warshall法における
５行５列対角ブロックの計算順序

6．実験結果

今回の実験は頂点数をすべて２００とし、主記憶の容量を2.4 kbytesとした。これは、主記憶上には最低でも２個以上の後続者リストが載らなければ計算ができないことから、２個の後続者リストがすべての頂点へ到達可能であるリストを持ったとき必要とする容量を主記憶の容量とした。

データの概略については表１、表２に示す。

実験結果については図５～図８に示す。それらは、Input/Outputの合計で縦軸の単位はk-byteである。そして▨は改訂動的Blocked Warshall法によって実験を行った結果で、□は右にある数値の列数をそれぞれ最初に分割として与えた場合の動的Blocked Warshall法での結果である。

この実験において演算時間を評価の基準とせずに、I/O の量をkbyte 単位でアルゴリズムの評価の基準としたのは、二次記憶を使用した場合にはほとんどの演算時間は主記憶と二次記憶間のI/Oによって決められるが、二次記憶としてディスクファイルを用いた場合には、演算時間は二次記憶のデータのアクセスの際のシーク時間や回転待ち時間に影響されるので、それらに影響されないでアルゴリズムを評価するパラメータとして、最適であると考えたからである。

7．考察

非対角行の処理中に主記憶が溢れた場合にはどちらのアルゴリズムにおいても処理の方法は同じである。よって効率の違いはないといえる。しかし対角ブロックの処理中に主記憶が溢れた場合には、読み込んでいたが処理をせずに放棄するような無駄な読み込みがない分だけ今回提案した改訂動的Blocked Warshall法の方が効率がよい。また、実際の活用においては、動的Blocked Warshall法で主記憶が溢れることのない効率的な分割を与えることは難しいと考える。しかしながら改訂動的Blocked Warshall法を用いることにより、分割をあらかじめ考えずにその状況に応じて自動的に分割をつくることが可能になる。

実験においても、ほとんどのデータに対して二次記憶と主記憶のI/O の合計は改訂動的Blocked Warshall法の方が従来の動的Blocked Warshall法よりも少なくなっている。動的Blocked Warshall法においては、与えられた分割が小さくなると一度に読み込もうとする後続者リストが少なくなるので、最終的に列分割を形成する後続者リストから漏れるものが少なくなってくる。それ故無駄な読み込みが少なくなる。しかしある程度以上分割を小さくしすぎると列分割数が多くなるので、そ

表１　データの概要

ﾀｲﾌﾟ	名前	頂点数	初期閉係辺の数	推移的閉包辺の数
上三角行列	DAT1	200	２９９	９８１
	DAT2	200	２９９	９４１
下三角行列	DAT1	200	３００	９９２
	DAT2	200	２９９	９７６

表２　データの概要

ﾀｲﾌﾟ	名前	頂点数	初期閉係辺の数	推移的閉包辺の数
一様ランダム	DAT1	200	３００	１２７７２
	DAT2	200	２９８	１１１６８
	DAT3	200	２９９	１３１１６
	DAT4	200	２９８	１７６５９
	DAT5	200	２９７	１４７６４
	DAT6	200	３００	１３７４９
	DAT7	200	２９９	１３４３４
	DAT8	200	３００	１５３７０

の分読み込みと書き込みが急激に増えてくる。そのように動的Blocked Warshall法においては分割の選び方によって非常に効率が悪化する危険性がある。しかしながら、改訂動的Blocked Warshall法では、そのような危険性は全くないといえる。

もし全てのデータを入れて計算できるほど、主記憶の容量がある場合にはどちらのアルゴリズムにおいてもInput/Outputの量は同じで、主記憶に読み込むデータの大きさは初期関係に等しく、二次記憶に書き出されるデータはその推移的閉包と同じである。また、分割は静的なものと同じように振舞う。

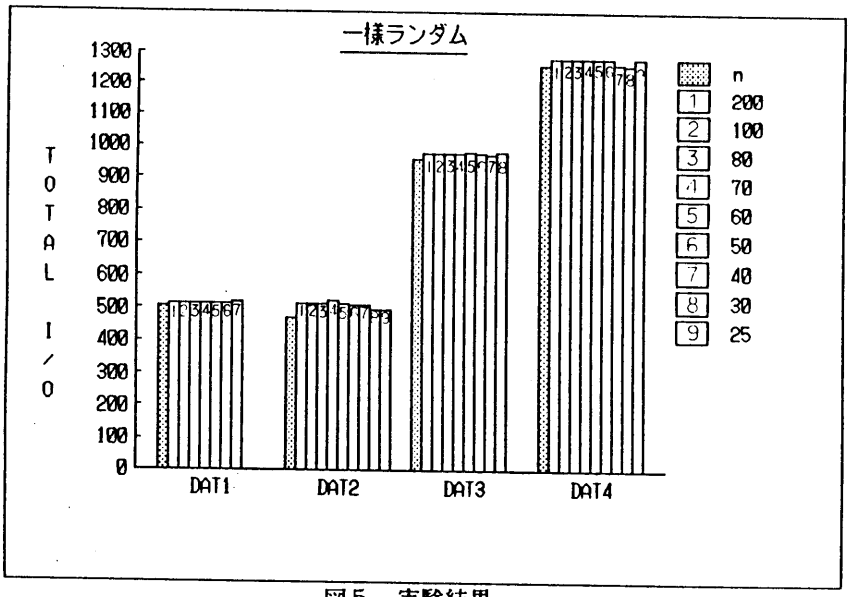

図5　実験結果

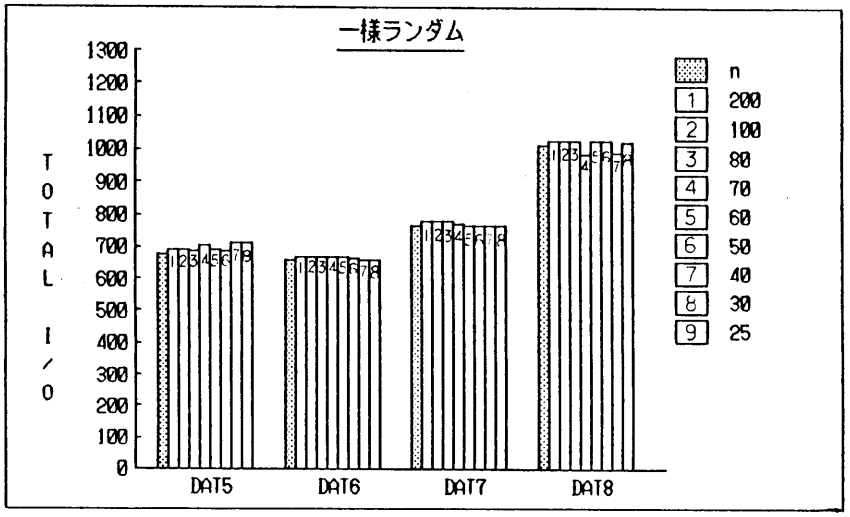

図6　実験結果

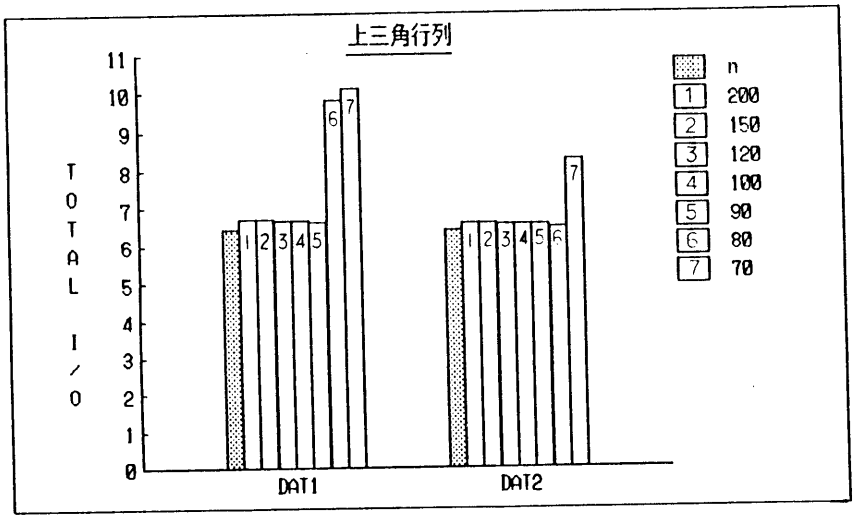

図7　実験結果

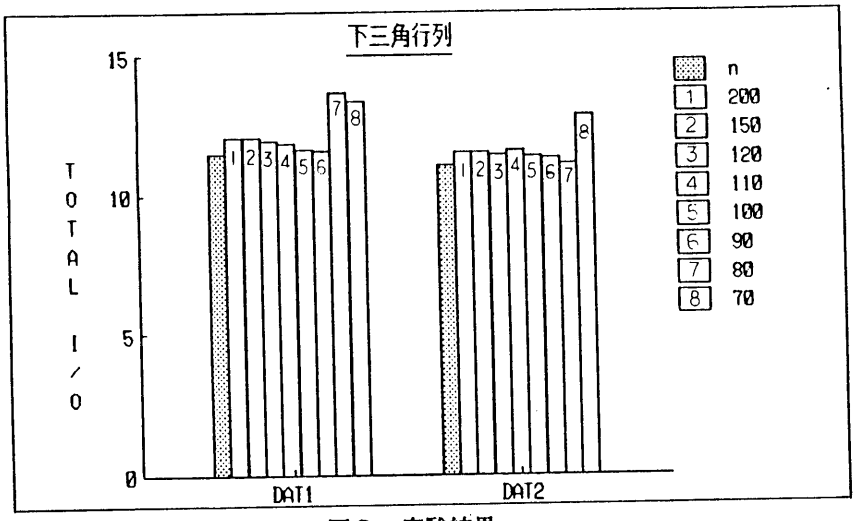

図8　実験結果

8．おわりに

この方法はただ単に有向グラフや関係データベースでの推移的閉包を求めるだけでなく、距離や経路等の情報を載せて最短経路問題等に応用することが出来ると考える。

今後の課題としては先行者リスト付きBlocked Warshall法、Blocked Warren法における動的分割の、より効率的な方法を考案することが挙げられる。

参考文献

[1] Rakesh Agrawal and H.V.Jagadish :
"DIRECT ALGORITHMS FOR COMPUTING THE TRANSITIVE CLOSURE OF DATABASE RELATIONS"
Proc.13th Conference VLDB Brighton,
pp.255-266 (1987).