アルゴリズム 66－9
（1999．1．27）

疎な有向グラフの強連結成分を求める効率良い並列アルゴリズム

多田昭雄
（熊本工業大学）

中村良三
（熊本大学）

有向グラフ（節点数n,辺数m）の強連結成分を求める問題は、最も基本的なグラフ問題の一つであり、現段階での最良の結果はＣＲＥＷ－ＰＲＡＭ並列計算機モデルで、$O(n^3)$台のプロセッサを用いて、$O(\log^2 n)$時間を実現したアルゴリズムである。これらのアルゴリズムは、行列の計算に依存しているので、$O(n^3)$台のプロセッサ数を下げることは困難である。また、辺の密度が低い疎グラフにおいては、行列計算や行列領域に無駄が発生する。本稿では、疎な有向グラフに対して、分割統治法と超節点法で効率良く強連結成分を求める並列アルゴリズムを提案する。このアルゴリズムはＣＲＥＷ－ＰＲＡＭ並列計算機モデルの上で、プロセッサ数が$O(n)$、計算時間が$O(\log^2 n)$である。

An Effective Parallel Algorithm for Finding Strongly Connected Components of a Sparse Directed Graph

Akio Tada (Kumamoto Institute of Technology)
Ryozo Nakamura (Kumamoto University)

A problem for finding strongly connected components of a directed graph is one of the most basic problems in graph theory. The best effective parallel algorithm is the one that takes $O(\log^2 n)$ time using $O(n^3)$ processors on a CREW-PRAM model so far. Because the complexity of the algorithm depends upon the matrix multiplication, it is difficult to reduce the number of processors futhermore. Specially in a sparse graph, a lot of wasteful processor time and memory space occur to matrix computations.

This paper presents an efficient parallel algorithm for finding strongly connected components of a sparse directed graph, using devide and conquer technique and supernode method. This algorithm requires $O(n)$ processors and $O(\log^2 n)$ time on a CREW-PRAM model.

1　はじめに

有向グラフ（節点数をｎ、有向辺数をｍとする）の強連結成分を求める問題は、最も基本的なグラフ問題の一つであり、他の多くのグラフ問題の部分問題になっている。すなわち、そのアルゴリズムの改良は、他の多くのグラフ問題の改良に結びつくという点で非常に重要である。

この問題に対する並列アルゴリズムの研究も長期間に渡ってすすめられており、現段階での最良の結果は、ＣＲＥＷ－ＰＲＡＭ並列計算機モデルにおいて、$O(n^3)$台のプロセッサを用いて、$O(\log^2 n)$時間を実現したアルゴリズム[1]である。しかし、このアルゴリズムはプロセッサの数が非常に多くて効率が悪い。強連結成分問題に関する並列アルゴリズムは、そのほとんどが、行列の計算に依存しており、この手法に依存している限り、$O(n^3)$台のプロセッサ数を下げることは困難である。また特に辺の密度が低い疎グラフ（sparse graph）に対しては、行列計算や行列領域の無駄が多く、最適な並列アルゴリズムの設計は難しい。

本稿では、有向グラフが疎グラフ（$m=O(n)$）であるとき、分割統治法と超節点法[2][3]を用いて非常に効率よく強連結成分を求める並列アルゴリズムを提案する。具体的には、有向辺を出節点・入節点の組で入力させた有向グラフを入出次数が高々１の部分グラフ（線形リスト）に分割し、強連結成分を超節点法で管理しながらグラフ全体の強連結成分を求める並列アルゴリズムである。このアルゴリズムはＣＲＥＷ－ＰＲＡＭ並列計算機モデルの上で、プロセッサ数が$O(n)$、計算時間が$O(\log^2 n)$で強連結成分を求める並列アルゴリズムである。

2　強連結成分を求める並列アルゴリズムの概要

対象とする有向グラフは、自己閉路や多重辺を含まない有向単純グラフＧ（Ｖ,Ｅ），|Ｖ|＝ｎ，|Ｅ|＝ｍで、辺の密度が低い疎グラフ（$m=O(n)$）とする。グラフの各節点は１から通し番号が付けられる。図１は有向グラフの例を示し、その入力は図２のように配列の形で与えられ、各有向辺はOVが出節点、IVは入節点を示し、上段から下段への向きで表される。すなわちOV[1..m]とIV[1..m]の対で表され、出節点は節点番号の順でソートされている。

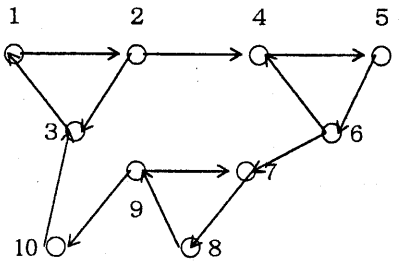

図１：有向グラフの例

	1	2	3	4	5	6	7	8	9	10	11	12	13
OV	1	2	2	3	4	5	6	6	7	8	9	9	10
IV	2	3	4	1	5	6	4	7	8	9	7	10	3

図２：入力配列

強連結成分を求める並列アルゴリズムは大きく分けて、次の３つのステージから構成される。

１．有向グラフを入出力次数が高々１の部分グラフに分割するアルゴリズム

２．分割された部分グラフ（線形リスト）から直接に強連結成分を求めるアルゴリズム

３．部分グラフを併合して強連結成分を求めるアルゴリズム

以下に各ステージのアルゴリズムの詳細を述べる。

3　アルゴリズムの詳細

ｽﾃｰｼﾞ１．有向グラフを入出力次数が高々１の部分グラフに分割するアルゴリズム

与えられた入力配列に基づいて、各節点の入出力次数が高々１となるように節点を分割し、新節点番号を用いてグラフを分割する。このとき、分割された元の節点を分割節点（旧節点）と称し、一つの節点が分割され新たに生成された節点を兄弟節点（新節点）と呼ぶ。

ｽﾃｯﾌﾟ［１.１］　各節点の出次数、入次数を求め、大きい方の値をその節点の最大次数とする。そして、この最大次数を各節点の兄弟節点数とする。兄弟節点の総数は高々(n+m)個である。

例えば、各節点の出次数は出節点 OV[1..m] から配列 O[1..n]に次のように求めることができる。

```
for  1 ≦ e ≦ m-1  in parallel do
     O[e] ← 0
     if  OV[e] ≠  OV[e+1]
        then  O[OV[e]] ← e
O[OV[m]] ← m
for  2 ≦ i ≦ n  in parallel do
     if  O[i] ≠ 0
        then  O[i] ← O[i] - O[i-1]
```

上記の並列アルゴリズムの計算時間は$O(1)$である。

入次数は始めに入節点 IV[1..m]をソートして、その後は出次数と同様に求めることができる。

このステージはプロセッサ数は$O(m)$で、計算時間はソートに要する時間$O(\log m)$　となる。

ｽﾃｯﾌﾟ［１.２］　各節点の兄弟節点数に基づき、兄弟節点に連続した通し番号を付ける。さらに入力された有向グラフの節点番号の大きさの順序を保つように連続した新しい節点番号を付け、有向グラフを新節点番号に基づき分割する。

具体的には、節点番号１から各節点における兄弟節点の累計和を求め、それに基づき各節点は兄弟節点に通し番号を付け、新しい節点番号に変更して有向グラフを分割する。

例えば、各節点の兄弟節点数とその累計和を与える配列を、それぞれ B[1..n]、S[1..n]とすると、新節点番号と旧節点番号の対応表：配列 N[1..S[n]]は次のような計算で求めることができる。はじめに、各節点において兄弟節点の先頭に対する新節点番号と旧節点番号の対応を次のように作成する。

```
N[1] ← 1
for  2 ≦ i ≦ n  in parallel do
        j ← S[i-1] + 1
        N[j] ← i
```

次に、各分割節点において先頭の兄弟節点が保持している旧節点番号を兄弟節点にﾌﾞﾛｰﾄﾞｷｬｽﾄする。以上によって、新節点番号と旧節点番号の対応表ができあがる。図１の例では図３のような対応表となる。

この対応表に基づき、入力配列を新しい節点番号で作り直す。

このステップは$O(m+n)$すなわち$O(n)$プロセッサ数で、計算時間はﾌﾞﾛｰﾄﾞｷｬｽﾄに要する時間$O(\log n)$となる。

新節点番号	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
旧節点番号N	1	2	2	3	3	4	4	5	6	6	7	7	8	9	9	10

図３：新節点番号／旧節点番号対応表

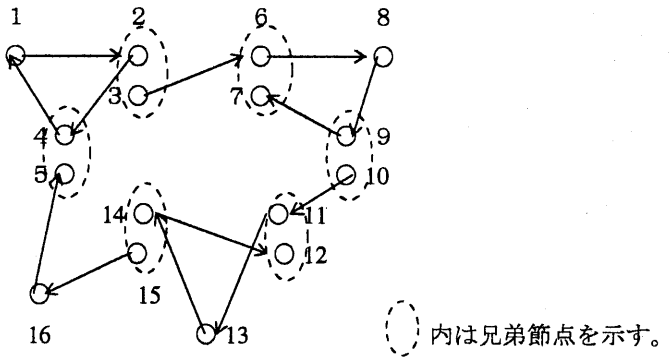

図４：分割されたグラフ

1	2	3	4	5	6	7	8	9	10	11	12	13
1	2	3	4	6	8	9	10	11	13	14	15	16
2	4	6	1	8	9	7	11	13	14	12	16	5

図５：新しく構成された入力配列

図１の有向グラフの例は図４のように分割され、４つの部分グラフ（線形リスト）に分かれる。また、図２の入力配列は図５のように構成される。

ｽﾃｰｼﾞ２．分割された線形リストから直接に強連結成分を求めるアルゴリズム

ｽﾃｯﾌﾟ［２.１］　各線形リストをダブリング技法を用いてなぞり、リスト上の各要素のランク値とリスト上の最小節点番号を求め、最小節点番号をそのリストのリスト番号とする。

ｽﾃｯﾌﾟ［２.２］ log n 時間のなぞりで終点に達しないリストはループを形成するので強連結成分となる。従って、そのリストのすべての要素はランクの値を１とし、強連結成分を表すマーク（＊）付けをする。

ｽﾃｯﾌﾟ［２.３］　上記のｽﾃｯﾌﾟ[2.2]を満たさない線形リストに対して、同一リスト上に兄弟節点があるかどうか調べる。具体的には、各分割節点において、その兄弟節点をリスト番号順、ランク順（昇順）にソートする。そして分割節点内の兄弟節点に同一リスト番号が存在すれば、ランクの小さい値の兄弟節点からランクの大きい値の兄弟節点間はループを形成する。すなわち強連結成分となるので、それらの強連結成分にマークを付け、それらの節点のランクをそれらの一番大きいランク値で統一する。

以上のステージはプロセッサ数が$O(n+m)$すなわち$O(n)$で、$O(\log\ n)$ 時間を要する。

図４の例はこのｽﾃｰｼﾞ２の結果、図６のようになる。（図６では分割節点内の兄弟節点のソートは表示していない。以下同様。）

図６において、リスト番号１はｽﾃｯﾌﾟ[2.2]でマーク付けされ、節点１,２と４が強連結成分となる（図６強連結成分(1)）。

新節点番号	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
分割節点	1	2	2	3	3	4	4	5	6	6	7	7	8	9	9	10
行き先	2	4	6	1	5	8	7	9	7	11	13	12	14	12	16	5
リスト番号	1	1	3	1	5	3	3	3	3	10	10	10	10	10	5	5
ランク	1	1	1	1	3	2	5	3	4	1	2	5	3	4	1	2
強連結成分(1)	*	*		*												
強連結成分(2)						*	*	*	*			*	*	*	*	
ｽﾃｰｼﾞ２後のﾗﾝｸ	1	1	1	1	3	5	5	5	5	1	5	5	5	5	1	2

図６：ｽﾃｰｼﾞ２の強連結成分の検出

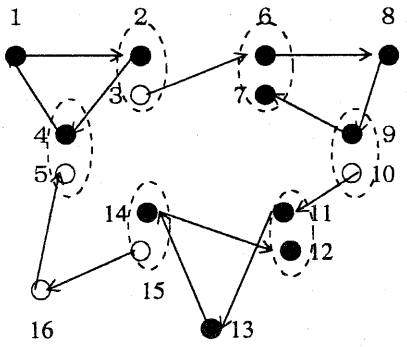

図７：ｽﾃｰｼﾞ２の強連結成分

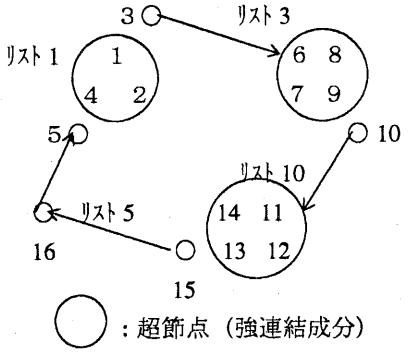

図８：超節点による構成

また、リスト番号３および10は、それぞれ分割節点４および７で同一リスト上に兄弟節点があるので、ｽﾃｯﾌﾟ[2.3]より、節点６,７,８と９および節点11,12,13と14が強連結成分となる（図６強連結成分(2)）。図７において、黒丸の部分が強連結成分を表す。黒丸を超節点で置き換えると、図７は図８のように構成される。

ｽﾃｰｼﾞ３．部分グラフを併合して強連結成分を求めるアルゴリズム

概観的には、各分割節点内の兄弟節点を介して、重複しない部分グラフの組を併合しながら強連結成分を求める。

初期段階では部分グラフは線形リストであり、強連結成分が存在するためには、各リスト上に少なくとも２個以上の分割節点が存在する必要があるので、分割節点が１個の線形リストは取り除く。

repeat log（線形リストの総数） do

{

ｽﾃｯﾌﾟ［３.１］　兄弟節点を介して重複しない部分グラフ（リスト）の組を作る。

具体的には各分割節点内でソートされた兄弟節点に対して、すべての可能な部分グラフの組を列挙し、重複しないすべての部分グラフの組を作る。この処理は疎グラフという条件から、兄弟節点の数が定数に制限されるので、組のソートと平衡２分木法を用いて、$O(\log n)$時間で処理できる。

ｽﾃｯﾌﾟ［３.２］　各部分グラフ（リスト）の組を併合点を介して併合する。

ここで、併合点とは、部分グラフの組で部分グラフ間にまたがる兄弟節点のある組をもつ分割節点と定義する。

基本的には図９のように、部分グラフ（リスト）の組の併合点間に互いに方向の異なる辺が存在すればループがあることになる。このループの検出は各節点のランクにより(1)式のように判定する。なお、(2)式は図９の矢印がすべて逆方向の場合である。

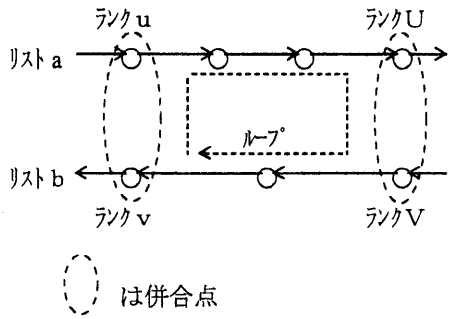

図９：部分グラフの併合によるループの検出

rank(u)＜ rank(U) かつ rank(v)＞ rank(V)
・・・(1)

あるいは

rank(u)＞ rank(U) かつ rank(v)＜ rank(V)
・・・(2)

次にｽﾃｯﾌﾟ[3.2]のアルゴリズムを下段に示す。

併合点間にループが形成されるのは、併合点に存在する節点のランクが(1)式あるいは(2)式を満足し、かつ併合点が指し示している相手の併合点の分割節点番号と併合点に到達した分割節点番号の中に同じ番号がある場合である。

ｽﾃｯﾌﾟ［３.４］　部分グラフの組を併合する。

部分グラフの組を併合するには、部分グラフのリスト番号を小さい方のリスト番号に統一する。またランクをすべての併合点において、大きい方のランクの値で統一し、併合点間の節点のランクは併合点のランクからの増分で表し、有向辺をなぞる順序を保持する。
}

以上のように、このｽﾃｰｼﾞ［３］は［log（初期線形リストの数)］回繰り返され、すべての線形リストについて強連結成分があるかどうか調べられる。

このステージはプロセッサ数は$O(m+n)$すなわち$O(n)$となり、計算時間は部分グラフの併合に要する時間$O(\log^2 n)$となる。

その後、検出した強連結成分の節点を元のグラフの節点（分割節点）に戻すと、それらが求める強連結成分である。

```
for 〔すべての部分グラフの組〕 in parallel do
    for 〔すべての併合点〕 in paralell do
      〔併合点で、分割節点番号と相互に部分グラフ番号（リスト番号)、ランクを受け渡す〕
       repeat log（部分グラフの組における最大の併合点の数） do
          〔次の併合点まで上記データをﾌﾞﾛｰﾄﾞｷｬｽﾄする〕
           if （併合点間にループが形成される）
             then 〔その併合点間のすべての節点を強連結成分としてマーク付けし、
                    ランクをそれらの節点の中で一番大きいランク値で統一する〕
             else if 〔両節点間のランクが(1),(2)式を満足しない〕
                then 〔この部分グラフ間には強連結成分がない.
                       従って、ﾌﾞﾛｰﾄﾞｷｬｽﾄは中止する〕
                else 〔併合点に到達した分割節点番号を記憶する〕
```

［ｽﾃｯﾌﾟ[3.2]のアルゴリズム］

ｽﾃｰｼﾞ３の実行例を図８をもとに示す。最初に部分グラフ(リスト)の組(1,3)と(5,10)が選ばれる。まず、部分グラフの組（1,3）に対しては、分割節点２の兄弟節点２に（2，3，1)、兄弟節点３に（2，1，1）のデータを記憶する。（ ）内の値は順に、分割節点番号、相手の部分グラフ番号（リスト番号)、相手のランクを表す。そして、次の併合点までデータをﾌﾞﾛｰﾄﾞｷｬｽﾄすると、兄弟節点２は一周して元に戻るので、節点３を強連結成分に加える。また、節点３は強連結成分でループし終了する。そして、リスト番号を１に統一して、部分グラフの組（1,3）を併合する。

一方、部分グラフの組（5,10）に対しては、分割節点９の兄弟節点14に（9，5，1）と兄弟節点15に（9，10，5）のデータを記憶する。そして、次の併合点までデータをﾌﾞﾛｰﾄﾞｷｬｽﾄすると、節点14は節点15に出会うので、節点15を強連結成分に加える。また、節点15は終端（節点５）に達して終了する。そして、リスト番号を５に統一し、部分グラフの組（5,10）を併合する。このとき、ランクは併合点９で５に統一し、以下リスト番号５の節点のランクは増分を割り付ける（図１２(*1)）。この結果、図８は図１０のようにリスト番号１とリスト番号５の２つの部分グラフになる。

次に併合後の部分グラフに対して、リスト番号１と５を組み合わせる。この部分グラフの組は併合点が分割節点３（兄弟節点４と５）と分割節点６（兄弟節点９と10）にあるので、図１１のように、兄弟節点４には(3，5，52)、節点５には（3，1，1)、節点９には（6，5，1)、また節点 10 には（6，1，5）を記憶する。そして、次の併合点までそれぞれデータをﾌﾞﾛｰﾄﾞｷｬｽﾄすると、節点４は分割節点６の併合点に出会うので、ランクを比較する。

リスト1:　1＜5　，リスト5:　52 ＞1

となり、(1)式を満足し、分割節点も指し示した分割節点６と到達した分割節点６が等しいので、この併合点間ではループを形成し、強連結成分となる。節点 10 も同様にループ（この場合は同一ループ）となる。従って、本稿の例では、すべての節点が一つの強連結成分となる。

以上の例の処理を配列で示すと、図１２のようになる。

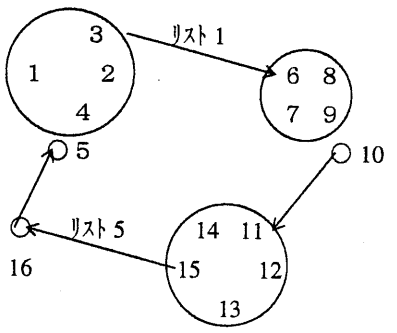

図１０：ｽﾃｰｼﾞ３による構成（１）

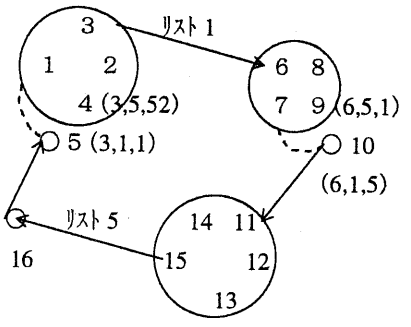

図１１：ｽﾃｰｼﾞ３による構成（２）

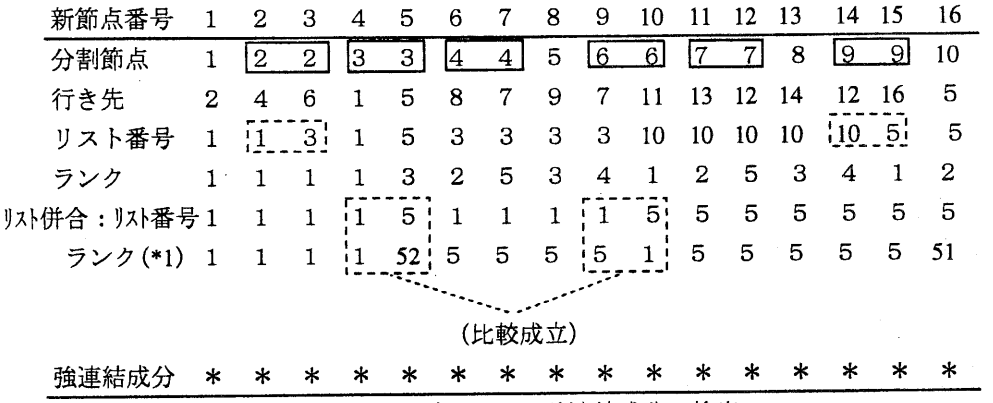

図１２：ｽﾃｰｼﾞ３による強連結成分の検出

4 まとめ

疎な有向グラフにおける強連結成分を求める問題について、分割統治法と超節点法を用いて求める効率的な並列アルゴリズムを提案した。このアルゴリズムはＣＲＥＷ－ＰＲＡＭ並列計算機モデル上で、プロセッサ数が$O(n)$、計算時間が$O(\log^2 n)$で達成できる。また、このアルゴリズムは、プロセッサ数を$O(n/\log^2 n)$にすれば、最適化並列アルゴリズムになる。

今後の課題として、並列計算機モデルをＥＲＥＷ－ＰＲＡＭモデルとした場合のアルゴリズムや辺密度が高い密な有向グラフ（dense graph）に対しても考察していきたい。さらに、いくつかの非連結部分グラフからなる有向グラフにおいて、最小の辺を加えることにより強連結となるグラフのアルゴリズム[1]にも、このアルゴリズムを適用していきたい。

参考文献

[1] Chaudhuri,P.: An Parallel Algoritms for Strong Connectivity Augmentation Probrem, Int. J. Comput. Math. vol.22 no3-4,187-197,1987.

[2] Gibbons,A. and Rytter,W.: Efficient Parallel Algorithms, Cambridge University Press, pp.6-18 (1988)

[3] Xavier,C. and Iyegar,S.S.: Introduction to Parallel Algoritms, Wiley-interscience, pp.188-201 (1998)

[4] 宮野 悟: 並列アルゴリズム － 理論と設計 －, 近代科学社(1993)

[5] 石畑 清: アルゴリズムとデータ構造, 岩波書店