アルゴリズム 11－1
（1989. 10. 24）

ループ領域解析アルゴリズムとその応用

牧野 正士

日本アイ・ビー・エム㈱ 東京基礎研究所

アセンブラや FORTRAN、C 等で書かれたプログラムの構造の理解や、最適化、再構造化、テスト等の支援にとってそのフローグラフのループ構造の解析は重要である。本稿では、任意のフローグラフ G=(N,A,s) (N:ノードの集合、A:枝の集合、s:開始ノード)とその深さ優先探索図 G´ に関して、G のループを特徴付ける、「ループ領域」の概念を導入し、G の(G´ に関する)全てのループ領域を O(k|A|) 時間で求めるアルゴリズムを与える。ここで、 k は G´のループ領域の数を表す。本結果から reducible なフローグラフに対する新しい特徴付けが得られる。更に本結果の理論的な問題への応用についても考察する。

A Loop Region Analysis Algorithm and Its Applications

Seishi Makino

IBM Research, Tokyo Research Laboratory
5-19, Sanbancho, Chiyoda-ku, Tokyo 102, Japan
(bitnet: makino@jpntscvm.bitnet)

Loop structure analysis of a flow graph is useful for the understanding, optimization, restructuring, and testing of programs written in such languages as Assembler, FORTRAN, and C. In this paper, we define a 'loop region' of any flow graph G, with respect to its depth-first presentation G´, and give an O(k|A|) time algorithm for finding all the loop regions of G (with respect to G´). Here, |A| and k denote the number of arcs and loop regions of G´, respectively. Our result contains a new characterization of a reducible flow graph. We also discuss an application of our algorithm to some theoretical problems.

1. はじめに

プログラムのコントロール・フローグラフ(以下フローグラフと略す)のループ構造を解析することは、アセンブラや FORTRAN、C 等で書かれたプログラムの構造の理解や、最適化、再構造化、テスト等の支援にとって重要な要因の一つである。フローグラフのループ構造を解析するためには、まず、「ループとは何か」を定義しなければならない。一方、reducible なフローグラフに対する「ループ」の定義としては、「backward arc (back edge) による(自然な)ループ」が一般に広く知られている([He77]、[ASU87])。ここで、reducible なフローグラフとは、図1のような、入口点を2つ以上持つ単純有向サイクルを部分グラフとして含まないようなフローグラフのことである[HU74]。しかし、実際のループ解析への応用において、「backward arc (back edge) による(自然な)ループ」の定義では、やや不都合な場合があり、特に、フローグラフが reducible でない場合のループ構造は、従来の定義や手法によっては調べることができない。そこで本稿では、一般のフローグラフに対して、その枝重み付き深さ優先探索図に関する「ループ領域」を定義し、ループ領域を効率よく求めるアルゴリズムを与える。 本結果から、reducible なフローグラフに対する新しい特徴付けが得られる。更に本結果の、理論的な問題への応用についても考察する。

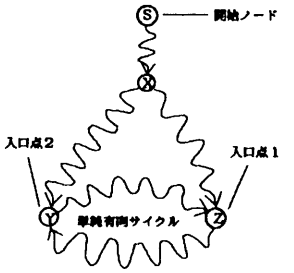

図1 破線矢印はノードを共有しない有向パスを表す。

2. ループ領域

[定義1] フローグラフとは、有向グラフ G=(N,A,s) (N:ノードの集合 A: 枝の集合 ⊂ N × N s: 開始ノード) であって、次の条件 1°、2° を満たすものを言う。
1° s から N の各ノードに至る有向パスが存在する。
2° s は G のどの有向サイクルにも含まれない。

[定義2] フローグラフ G=(N,A,s) の有向サイクル C に対して、C の入口点とは、s から C 内のノードに至る有向パスが最初に出会う C のノードを言う。

我々は、Hecht-Ullman による characterization ([HU74]) に基づいて、reducible なフローグラフを次のように定義する。

[定義3] フローグラフ G=(N,A,s) に対して、G が reducible であるとは、次の1°、2°のいずれかの条件を満たすときをいう。
1° G は有向サイクルを含まない。
2° G の任意の単純有向サイクル (両端点以外を共有しない有向サイクル) C は、唯一の入口点 h ∈ C を持ち、s から C 内のノードに至る任意のパスは、h を通る(すなわち、C の任意のノードは、h によって dominate される)。

図2、4の G_1, G_2 はいずれ も reducible なフローグラフ の例である。フローグラフの枝重み付き優先探索図 (以後、' 探索図'と略記する) とは、与 えられたフローグラフ G の 各分岐ノードに対して、そこから出る各々の枝に予め優先順位(重み)を与えておき、s を開始点とする深さ優先探索([AHU83])を、各分岐ノードで、優先順位の高い(重い)枝が優先的に選択されるように行ったときに 探索順序に応じて G の各枝を通常の tree arc, back arc, forward ark, cross arc に分類して表した図のことである。

図3は、図2のフローグラフの分岐ノードの各枝の重みを、分岐先のノードのアルファベットの若いものが重くなるように付けた時の探索図である。探索図において、各 back arc の始点を tail、終点を head と呼ぶ。また、探索図上の head h に対して、h を終点とする back arc の始点(tail)の集合を T(h) で表す。

[定義４] フローグラフ G とその探索図 G´ が与えられている時、G´ の任意の head h に対して、次の 1°、2° によって定められる G の部分グラフ R=(V,E,h) を、h を head とする、G の (G´ に関する) ループ領域という。

1° V = {h} ∪ T(h) ∪ { v ∈ N | h → v かつ v → t (ただし、t ∈ T(h)) }

2° E = { (v,w) | v,w ∈ V かつ (v,w) ∈ A }

ここで、v, w ∈ N に対して、v → w は、G´ 上で v から w に至る back arc を経由しない有向パスが存在することを表す。h を R の head 、t (∈ T(h))を R の tail という。

定義４から明らかに、探索図上の各ヘッドと、ループ領域が一対一に対応する。そこで、R=(V,E,h)を R(h) (または Rh) と略記する。ループ領域を用いると、reducible なフローグラフの ループをうまく特徴づけることができる。 例えば、図２ のフローグラフ G_1 のループ領域は、図３の探索図で、二点鎖線で囲った、ノード a, b, c, d, eから成る領域 R=(V,E,a)である。また、図４の G_2 の場合、ループ領域は、破線の領域 R_1と、一点鎖線の領域 R_2の２つである。 フローグラフが、reducible なときは、探索図の back arc の集合は、探索図の形によらず常に一定であることが知られている([HU74])。従って、ループ領域について次の性質が成り立つ。

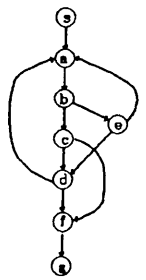

図2 G_1 =(N,A,s)

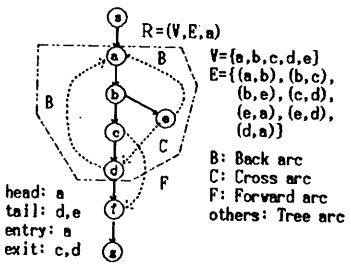

図3 G_1の深さ優先探索図と ループ領域

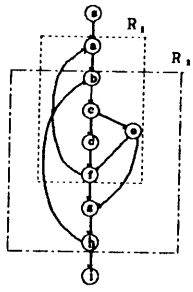

図4 G_2 =(N,A,a)

[定理１] reducible なフローグラフのループ領域は、枝の重み付けとは無関係に、常に一定である。

フローグラフが reducible でない場合には、定理１は必ずしも成り立たない。例えば、図５のフローグラフ G_3 の探索図は、枝の重み付けによって変わる(図６,７)。しかし、探索図の選び方に無関係に次の性質が成り立つ。

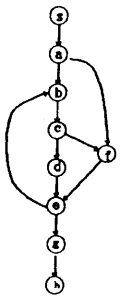

図5 G_3

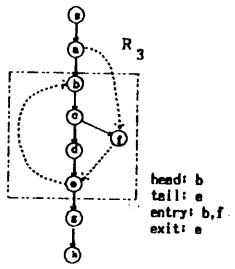

図6 アルファベット順を優先した時の G_3 の探索図

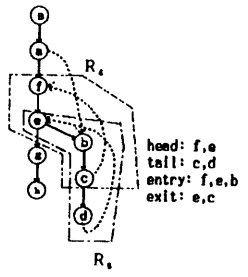

図7 アルファベットの逆順を優先した時の G_3の探索図

[定理２] フローグラフ G=(N,A,s) とその探索図 G´ に対して、G が reducible であるための必要十分条件は、G´ 上の任意のループ領域 R に対して、R 外のノードを始点とし、R 内のノードを終点とする、cross arc または forward arc (R への '飛びこみ枝' と呼ぶことにする)が存在しないことである。

フローグラフ G のループ領域 R=(V,E,h) において、h と,R への飛び込み枝の終点を、R の entry ノードと定義する。また、G の探索図上で、R 内のノードを始点とし、R 外のノードを終点とする任意の枝を R からの '飛び出し枝' と呼び、その始点を R の exit ノードと定義する。

［定理２の証明］

必要性（背理法による）G が reducible であるとき、G のあるループ領域 R(h) に対して、R(h) への飛び込み枝 (u,v) が存在したとすると、定義４から、h から v への back arc を経由しない有向パスと、v から t (∈ T(h))への back arc を経由しない有向パス及び back arc (t,h) からなる単純有向サイクル C が存在する。このとき、定義４より、h から u (∉ R(h)) への back arc を経由しないパスは存在し得ない。従って、G´ 上で s から tree arc のみを経由して u に至る有向パス P に対して、P+(u,v) は s から h を経由せずに v (∈ C) に至るパスなので、定義３に矛盾する。

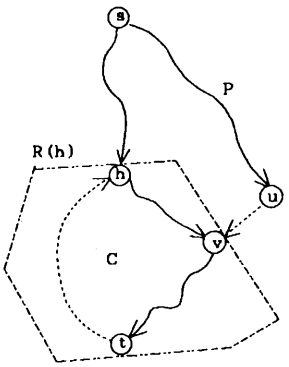

図8

十分性　対偶、すなわち、次を証明する。

「G が reducible でない、すなわち、G のある単純有向サイクル C に対して、C が２つ以上の入口点 $e_0, e_1, .., e_n$ を持つなら、G´ 上のあるループ領域 R に対して、R への飛び込み枝が存在する」

C は G´ 上でも単純有向サイクルであり、これを C´ とすると、C´ は少なくとも一つの back arc を含む。それらを $b_1, b_2, .., b_p$ とし、$b_1, .., b_p$ の終点をそれぞれ、$h_1, h_2, .., h_p$ とすると、探索図の性質により、s から各 $h_1, .., h_p$ には、back arc を含まない有向パス、$p_1, p_2, .., p_p$ が存在するので、$h_1, .., h_p$ はいずれも、C´ の入口点である。そこで、back arc の一つ，例えば b_p に着目して、h_p を head とするループ領域を R_p とすると、b_p の始点 t_p は R_pのノードである。t_p から C´ の枝を逆方向にたどる時、最初に出会う入口点を e_0 ($\neq h_p$)とすると、s から e_0 に至る、back arc を含まず C´ と交差しない有向パス p_0 が存在する。s から p_0 と C´ 上の枝をたどって t_p に至る単純有向パス (ノードを共有しない有向パス) $P = p_0 + e_0, .. t_p$ は h_p を含まず、また、back arc を含まない。また定義１から明らかに $s \notin R_p$ なので、ある枝 $(x,y) \in P$ が存在して、$x \notin R_p$ かつ、$y \in R_p$ 。一方、(x,y) は tree arc で有り得ない。なぜなら、探索図及びループ領域の性質から h_p から y に至る、R_p 内のノードと tree arc のみを通る有向パス Q が存在し、$(x,y) \notin Q$ なので、(x,y) が tree arc であるとすると、y を終点として共有する２本の tree arc が存在することになり、探索図の性質に反するからである。従って、(x,y) は cross arc または forward arc でなければならなず、このとき (x,y) は R_p への飛び込み枝であり、y は h_p 以外の R_p の entry である。よって、十分性が示された。証明終。

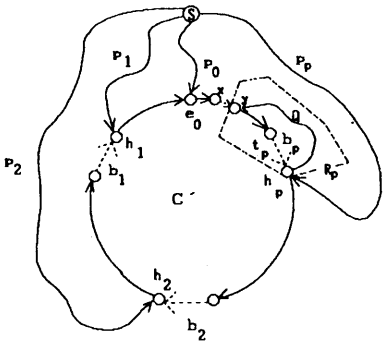

図9

３．アルゴリズム

フローグラフ G=(N,A,s) の探索図上で、G の全てのループ領域および、その entry ノード tail ノード、exit ノードを求める再帰的アルゴリズムを図１０に示す。R(i), E(i), T(i), X(i) はそれぞれ、i を head とするループ領域のノード、entry, tail, exitの集合を表し、初期状態では、各集合は空である。また、集合演算の効率化のために、各集合は k × |N| ビット (k は ループ領域の数)の配列で表現されており、例えば、v ∈ R(i) は、R(i)を表す k × |N| ビット の配列の、(i,v)-成分が 1 であることと同値であるとする。このことにより、図１０における v ∈ R(i) のチェックや、R(i) ← R(i) ∪ {v} (追加)、R(i) ← R(i) / {w} (削除)等の集合演算は、ランダム・アクセス計算機上で O(1) 時間で行える。起動には、ループ領域(s) を実行する。

```
    procedure ループ領域(v)
    begin
1.  ┌── for each tree arc (v,w) do
    │   begin
2. (イ)  for each i ∈ H such that v ∈ R(i) do R(i) ← R(i) ∪ {w}; { ループ属性の伝搬。}
3.  │    ループ領域(w); { 再帰的呼び出し }
    └── end;
4.  ┌── for each i ∈ H such that v ∈ R(i) do
    │   begin
5. (ロ)   if v has no successor w which satisfies (v,w) ≠ back arc, and w ∈ R(i)
    │    then R(i) ← R(i) / {v}; { v を R(i) から削除する。}
    └── end;
6.  ┌── for each back arc (v,w) do  { この時、w ∈ H であることに注意。}
    │   begin
7.  │     R(w) ← R(w) ∪ {w};
8.  │     E(w) ← E(w) ∪ {w}; { w は entry }
9.  │     v から w まで、tree arc を逆向きに辿る時に出会う各ノード p に対して do
    │     begin
10. (ハ)    if p ∉ R(w) then R(w) ← R(w) ∪ {p} { p を R(w) に追加。}
11. │       else leave; { tree arc を辿るのを中止する。}
    │         { p ∈ R(w) の時、これ以降のノードは全て R(w) の要素となっていることに注意。}
    │     end;
12. │     R(w) ← R(w) ∪ {v};
13. │     T(w) ← T(w) ∪ {v}; { v は tail }
    └── end;
14. ┌── for each successor w of v do
    │   begin
15. │     for each i ∈ H such that v ∈ R(i) and w ∉ R(i) do X(i) ← X(i) ∪ {v};
   (ニ)   { この時、(v,w) は R(i) からの飛び出し枝 }
16. │     if (v,w) is a cross arc or a forward arc then
17. │     for each i ∈ H such that w ∈ R(i) and v ∉ R(i) do E(i) ← E(i) ∪ {w};
    │      { この時、(v,w) は R(i) への飛び込み枝 }
    └── end;
    end;
```

図１０　ループ領域解析アルゴリズム

4．アルゴリズムの正当性と計算量

フローグラフ G=(N,A,s) に対して与えられた探索図を G´=(N,A,s,T,B,F,C,H) （ ここで、T, B, F, C, H は、それぞれ G´ の tree arc, back arc, forward arc, cross arc, head の集合 ）とすると、次が成り立つ。

> [定理3] G´ に対して、ループ領域(s) は、ランダム・アクセス計算機上で、高々 O(k|N|+|A|log|A|) ビットのメモリと、O(k|A|) 時間(k=|H|) で動作し、停止時において、各 R(h), E(h), T(h), X(h) (h ∈ H)はそれぞれ、h を head とする G の(G´ に関する) ループ領域のノード、entry, tail, exit の集合に等しい。

[定理3の証明] 正当性　図10の ループ領域(v) において、G´ の各ノード v に対して、v に関する再帰的呼び出しが終了した順番に v に番号 p(v) を与えることにする、すなわち、G´ に関する v の postorder ([Ta83]) を p(v) とする。 ループ領域(v) において、(イ) の ループ領域(w) の再帰的呼び出し以後、v よりも以前に再帰的呼び出しが終了したノード、すなわち、G´ 上で v よりも小さな postorder を持つノードに関して再び、各 R(i) の内容が更新されることはない。また、G´ の各枝 (v,w) に対して、w が任意の E(i) の要素となるかどうか、v が 任意の X(i) の要素となるかどうかは、 ループ領域(v) の (ニ) でただ一度決定される。一方、v の子孫 (v から tree arc を辿って到達できるノード)の postorder は、常に vより小さい。従って、次の命題を v の postorder, p(v) に関する帰納法で証明すれば、十分である。

(命題) ループ領域(v) の実行後、v 及び v の全ての子孫 (v から tree arc を辿って到達できるノード) に関して、各 R(i) の内容が正しく計算されている。ここで、「ノード w に関して R(i) の内容が正しく計算されている」とは、「w ∈ R(i) ⇔ w は G´ において、i を head とするループ領域(以下 Ri と表記する) の要素」が成り立つことである。

(I) p(v) = 1 の時、

この時、v は successor を持たないか、または、v の任意の successor w に対して、(v,w) が back arc であるかのいずれかである。v が successor を持たないなら、明らかに v はどのループ領域にも含まれない。一方 このとき、ループ領域(v) において実行されるステートメントはなく、しかも、ループ領域(v) が呼び出された時点ではまだ各 R(i) は空であり、実行後も v は R(i) の要素とはならない。次に、v の successor w に対して、(v,w) が back arc であるとき、定義4より、 v は R_w の要素 (tail)である。一方、このとき、ループ領域(v) において実行されるのは (ハ) だけであり、実行後、v は R(w) の要素 (tail)であり、かつ v が R(w) 以外の要素となることはない。ゆえに、このとき (命題) は成り立つ。

(II) ある m ($2 \leq m \leq |N|$) が存在して、p(v) < m の時 (命題) が成り立つと仮定する。

(III) p(v) = m の時、

v の任意の successor w に対して、p(w) < p(v) = m なので、帰納法の仮定より、(イ) のループ領域(w) の実行後、w および w の子孫に関して、各 R(i) は、正しく計算されている。

v ∈ Ri ⇒ v ∈ R(i) の証明

v ∈ Ri なら、定義4より明らかに、次の1°, 2°の少なくとも一方が成り立つ。

1°「 v は Ri の tail である」

2°「 v のある successor w (ただし、(v,w) ≠ back arc) は Ri の要素である」

1° の場合

(v,i) は back arc だから、(ハ) の実行後、v は常に R(i) の要素である。

2° の場合

定義4より、v から Ri の tail に至る、back arc を経由しないパスがある。一方、forward arc は、tree arcのみからなるパスで置き換えることができるから、実際には次のいずれかが成り立つ。

(i) v から Ri のある tail, t に至る tree arc のみで構成されるパスがある。

(ii) v から Ri の任意の tail, t´ に至る全てのパスは、少なくとも一本の cross arc と、(0本以上の) tree arc で構成される。

(i) の場合　i は v の祖先であり、t は v の子孫の一つなので、明らかに、(イ) の再帰的呼び出しの実行中

に、ループ領域(t)が実行され、その中で、 back arc (t,i) に対して、(ハ) が実行され、このとき v は R(i) の要素となる。従って、(イ) が終了した時点で、v は既に R(i) の要素となっているので (ロ) の実行後も v は常に R(i) の要素である。

(ii) の場合 このとき、i から tree arc のみを経由して v に至るパスがありしかも、t´ は v の子孫でなく、また、v は t´ の子孫でもないが、共通の祖先を持つ、すなわち、親戚関係にある。共通の祖先のうち、最も若い(postorder の小さい) ノードを a とすると、a は i 自身または、i の子孫である。なぜなら、v と t´ はともに、i の子孫だからである。一方、tree arc と forward arc の終点はいずれもその始点より postorder が小さいので、p(t´) < m 。従って、ループ領域(v)の実行時には、t´ に対する再帰的呼び出しは既に終了しており、また、ループ領域(t´) が実行された時に、 back arc (t´,i) に対して (ハ) が実行され、このとき a は R(i) の要素となっている。従って (イ) の伝搬作用により、ループ領域(v) が呼ばれた時には、v は R(i) の 要素となっているので、(ロ) の実行後も常に v は R(i) の要素である。

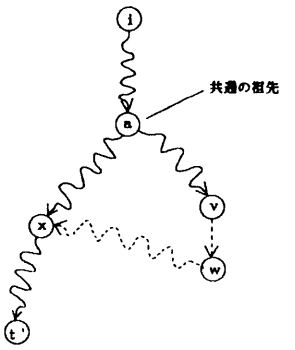

図11

v ∈ R(i) ⇒ v ∈ Ri の証明

対偶を証明する。v が Ri の要素でないなら、次の少なくとも一方が成り立つ。

(α) v から Ri の tail t に至る back arc を経由しない有向パスが存在しない。

(β) v は i の子孫でない。

(α) の場合 v の各 successor w から t に至る back arc を経由しない有向パスも存在し得ず、w も R の要素ではない。一方、p(w) < m なので、帰納法の仮定により、このとき w は R(i) の要素ではない。従って、(ロ)の実行後、v は R(i) の要素であり得ない。

(β) の場合 もし、(イ) の再帰的呼び出しの中で、v が初めて R(i) の要素となったとすると、それは、v の子孫のあるノード d に対して、(d,i) が back arc であり ループ領域(d) において (ハ) が実行された時に v ∈ R(i) となったことを意味する。しかし、このようなことが起こり得ない。なぜなら、もし起こったとすると、v は i の子孫でなければならないから、(β) の条件に反する。従って、v が R(i) の要素となるためには、(ロ) の条件から、v の successor の少なくとも一つは R(i) の要素で、かつ、ループ領域(v) が呼ばれる時既に、v は R(i) の要素でなければならない。そのためには v のある祖先 a´ に対して ループ領域(a´) の (イ) の再帰的呼び出しの中で、ループ領域(v) が呼ばれる前に a´ が R(i) の要素となっていなければならず、それは、a´ の子孫のあるノード d´ に対して、(d´,i) が back arc であり ループ領域(d´) において (ハ) が実行された時に a´ ∈ R(i) となったことを意味する。しかし、このようなことは起こり得ない。なぜなら、もし起こったとすると、a´ は i の子孫でなければならず、一方 v は a´ の子孫だから、結局 v は i の子孫であることになり、(β) の条件に反する。従って、p(v) = m のときにも、(命題) は成立する。

ゆえに帰納法によって、(命題) は、任意の 1 ≦ pv ≦ |N| について成立する。

計算量 明らかに、各 v に対して、ループ領域(v) は一回だけ呼ばれる。v の successor の数を deg(v) とすれば、ループ領域(v) において、(イ) の実行に要する時間は、再帰的呼び出しを除いて O(k×deg(v)) 時間、(ロ) 及び (ニ) に要する時間はいずれも O(k×deg(v)) 時間である。(ハ) については、(ハ) の脱出(leave)条件 から、既に R(w) の要素となっているノードは、再度 R(w) の要素として加えられることはなく、また、このとき加えられたノードは、全て Rv の要素であり、これ以後 R(w) から削除されることはない。また、w から v までの tree arc によって構成される有向パス上のノード p に対して、既に p ∈ R(w) であるときに p ∈ R(w) であるかどうかのチェックが行なわれることがあるのは、高々1回である。従って、ループ領域(s) の実行 (再帰的呼び出しを含めて) において (ハ) は高々、O(4×|A|)+O(k×|N|)+O(|N|) 時間を要する。ゆえに、ループ領域(s) 全体の実行 (再帰的呼び出しを含めて) に要する時間は、高々、$O(4\times|A|)+O(k\times|N|)+O(|N|)+\Sigma_{v\in N}\{O(3\times k\times deg(v))\}=O(k|A|)$ である。

また、計算に必要なメモリの量は、G´ を隣接リスト構造で格納するための O(|A|log|A|) ビット の領域と、各 R(i), E(i), T(i), X(i) を表現する k×|N| ビット の配列のための O(4×k×|N|) ビット の領域以外は、高々、O(|A|log|A|) ビット の作業領域だけであるから合計、O(k|N|+|A|log|A|) ビット である。証明終。

5．応用について

フローグラフ G=(N,A,s) の任意の探索図 G´ に対して本アルゴリズムを適用して得られた、'飛び込み枝'の集合を $\varepsilon(G')$ とすると、G から $\varepsilon(G')$ の枝を除いて得られる G の部分グラフ $\underline{G}=(N,A/\varepsilon(G'),s)$ は、常に reducible なフローグラフである。従って、本アルゴリズムは、reducible でないフローグラフをそれと関数的に等価な、reducible なフローグラフに変換するのに応用できる。一般に、reducible なフローグラフは、ループの最適化や、データフローの効率的解析に適している([ASU87]、[He77])ほか、一般には NP-困難([GJ79])な最小帰還枝(点)集合問題が多項式時間で解ける ([Ra88]、[Sh79]) などの性質があることが知られている。従って、例えば一般の reducible でないフローグラフ G の最小帰還枝集合 MFA(G) を求める場合などは、本アルゴリズムから得られる $\underline{G}$ に対する最小帰還枝集合 MFA($\underline{G}$) ([Ra88]によれば多項式時間で求まる)は一つの目安になると思われる。なぜなら、$\underline{G}$ は G の部分グラフであり明らかに、$|MFA(\underline{G})| \leq |MFA(G)|$ なので常に、$|MFA(\underline{G})| \leq |MFA(G)| \leq |MFA(\underline{G})| + |\varepsilon(G')|$ が成立するからである。

6．まとめ

ループ領域の概念を導入し、任意のフローグラフに対して適用可能なループ領域解析の算法を示した。定理2で述べた、reducible フローグラフの特徴付けは、一般によく知られているもの ([HU74]、[Ta74]、[He77] 等)と異なる。本アルゴリズムを、IBM 3081 ｼｽﾃﾑ 上で PASCAL/VS を用いてインプリメントした結果、実際の S/370 アセンブラ・プログラムから抽出された、ノード数1979、枝数 2638、ループ領域数 51 の reducible でないフローグラフのある探索図に対して、全ループ領域及び entry, exit を約 0.56 秒で計算した。

謝辞

本研究の発端となった、業務上のテーマ並びに援助を与えて下さった、日本アイ・ビー・エムの秋山義博氏、並びに、原稿を注意深く読んで記法上の誤り等を指摘して下さった、徳山豪氏に感謝する。

引用文献

[AHU83] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, Data structures and algorithms, Addison-Wesley 1983.

[ASU87] A. V. Aho, R. Sethi, and J. D. Ullman, Compilers, Addison-Wesley, 1987.

[GJ79] M. R. Garey and D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman, San Francisco, 1979.

[He77] M. S. Hecht, Flow analysis of computer programs, North-Holland, 1977.

[HU74] M. S. Hecht and J. D. Ullman, Characterizations of redicible flow graphs, J.ACM, vol21, no3, pp.367-375, 1974.

[Ra88] V. Ramachandran, Finding a mimimun feedback arc set in reducible flow graphs, J.Algorithms, 9, pp.299-313, 1988.

[Sh79] A. Shamir, A linear time algorithm for finding minimum cutsets in reducible graphs, SIAM J. Comput, 8, no4, pp.645-655, 1979.

[Ta74] R. E. Tarjan, Testing flow graph reducibility, J. Computer and System Sciences, 9, pp.355-365, 1974.

[Ta83] R. E. Tarjan, Data structures and network algorithms, SIAM, 1983.