数理解析研究所講究録
第716巻 1990年 19-36

有向グラフに対する極大パスカバー問題の計算量

Complexity of Maximal Path Cover Problems for Directed Acyclic Graphs

山神憲司（東京電機大学理工学部），植村憲治（都留文科大学），
夜久竹夫（東京電機大学理工学部）
Kenji Yamagami[1] , Kenji Uemura[2] and Takeo Yaku[3]

[1] Department of Mathematical Science , Tokyo Denki University , Hatoyama , Saitama 35003 , JAPAN
[2] Tsuru University , Tsuru , Yamanashi 402 , JAPAN
[3] Department of Information Sciences , Tokyo Denki University , Hatoyama , Saitama 35003 , JAPAN

Abstract. We consider the covering problem of directed acyclic graphs(DAG) by node disjoint paths . (The graph which consists of one node is regarded as the path of length zero.) The set of such paths is called path cover, chain matching or simply covering. If the number of paths in a path cover of a DAG is minimum, we call that path cover maximal. We give algorithms for finding maximal path cover of given DAG.
First we show that this problem for DAG is reducible to the one for alternating subgraphs, and give a (node) linear time algorithm for DAG of outdegree or indegree at most two. Also we analyze the average number of visited nodes by this algorithm bounded by $n + \log n$. Moreover an efficient parallel algorithm is given. Finally we show the eqivalence between the maximal path cover problem and maximam matching problem for some bipartite graphs.

1 まえがき

有向グラフからパスカバーを求める問題は有向グラフ状のデータ構造を”線形化”[3]して，連続状の記憶構造により表現する問題に関係する．例えばプログラムフローチャートをcodingにより線型化することは，そのフローチャートのパスカバーを求めることである．その際に，最小のブロック数を伴うパスカバーは分岐（goto文）最小のcodingに対応する[6]．

一般の（サイクルを持つ）有向グラフに対して，極大パスカバー問題はNP完全であることが知られている[5]．サイクルのない有向グラフ($G=(V,E)$，ここに$|V|=n$とする．)，DAGと略す，に対して，極大なパスカバーを求めるアルゴリズムついては$O(n^{5/2})$時間のものが知られている[5]．入次数、出次数とも高々2のDAGに関しては$O(n)$時間のものが知られている[6]．われわれは出次数，又は入次数のみを高々2と制限して，4節では$O(n)$時間のアルゴリズムを示し[4]，更にこのアルゴリズムがvisitする頂点の平均個数の上限の数値解は$n+\log n$であることを示す[9]．5節ではこの問題が並列計算量のクラスNCに属すことを示す．

この研究の一部は東京電機大学総合研究所の研究助成及びイチマルコーポレーション（株）の依託研究により行なわれた．

またこれらのアルゴリズムは，制限された二部グラフのマッチング問題に応用することができる．一般のマッチング問題は多項式時間で解けることが知られているが，その並列計算量は未解決で，我々の知る最良の結果は二部グラフにおけるマッチング問題で $O(n^{3/4}polylog(n)) \notin NC$ 時間である (see [12])．従って我々は制限された2組グラフについて NC アルゴリズムを与えたことになる．

2 準備

有向グラフを $G=(V,E)$ で表す．ここで V は頂点の集合，E は辺の集合である．有向グラフ G における頂点 v の入次数と出次数，$indegree(v,G)$ と $outdegree(v,G)$ とそれぞれ示す，を次のように定める：

$indegree(v,G) = \mid \{u \in G; (u,v) \in E\} \mid$

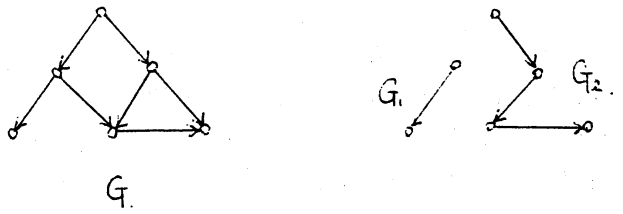
$outdegree(v,G) = \mid \{u \in G; (v,u) \in E\} \mid$.

次にパスカバーを定める．$G_1=(V_1,E_1), G_2=(V_2,E_2), \cdots, G_n=(V_n,E_n)$ を有向グラフ $G=(V,E)$ の部分グラフとする．もし $V_1 \cup V_2 \cup \cdots \cup V_n = V$ かつ $V_i \cap V_j = \phi (i \neq j)$ ならば P=$\{G_1, G_2, \cdots, G_n\}$ を G の分割という．各部分グラフ $G_i=(V_i,N_i)$ を P の成分という．分割 P は各 G_i が次の条件 (1) 又は (2) を満たすときパスカバーといわれる：

(1) G_i は一つのパスからなる．

(2) G_i は一点 ($\{$v$\}$,ϕ) である．

また，Pによるグラフ，$G(P)=(V,E(P))$ と書く，とは $G(P)=G_1 \cup G_2 \cup \cdots \cup G_n$ のことである．



G

有向グラフ $G=(V,E)$ のパスカバー $M=\{G_1, G_2, \cdots, G_n\}$ が極大であるとは，G の任意のパスカバー N に対して次がなりたつことである：

$\mid (E-E(M)) \mid \leq \mid (E-E(N)) \mid$ 即ち，

$\mid E(M) \mid \geq \mid E(N) \mid$.

注1． 有向グラフ G の極大パスカバー $M=\{G_1, G_2, \cdots, G_n\}$ は最小の成分数を持つ．即ち，G の任意のパスカバー $N=\{H_1, H_2, \cdots, H_m\}$ に対して $n \leq m$ である．

3 交互グラフ

この節では

問題1．極大パスカバーを求める．
が

問題2．分岐最小の生成出グラフを求める[4]．
に帰着できることを示す．有効グラフの辺の間には交互同値という関係が知られている[4]．更に問題2は次の問題3に帰着できる．[4]

問題3．上の同値関係により分割された各部分グラフについて，分岐最小の生成出グラフを求める．

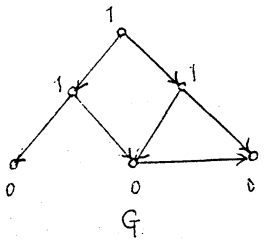
定義. 有向グラフ $G=(V,E)$ における頂点 v の分岐数は

$$branch(v,G)=\begin{cases} outdegree(v,G)-1 & outdegree(v,G)>0 \\ 0 & outdegree(v,G)=0 \end{cases}$$

と定められる．G の分岐数は，

$$branch(G)=\sum_{v\in N} branch(v,G)$$

である．



G

branch(G) = 3

定義. 有向グラフ $G=(V,E)$ の部分グラフ $H=(V,F)$ は各頂点 v に対してただ一つの辺 $(u,v)\in F$ $(u\in V)$ が存在するとき，G の生成出グラフという，ただし $indegree(v,G)\geq 1$．

また，頂点の列 $v_1,v_2,...,v_{n+1}$ が存在して $e_i=(v_i,v_{i+1})$ であるか又は $e_i=(v_{i+1},v_i)$ であるとき，辺の列 $e_1e_2...e_n$ をセミパスという．また，セミパスがサイクルをなすとき，それをセミサイクルという．

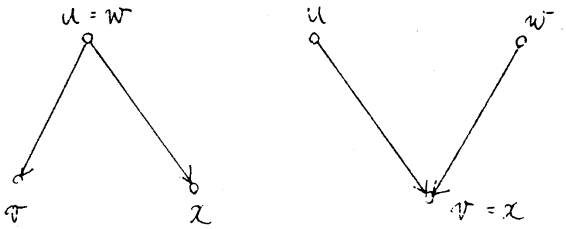
有向グラフの辺の間に同値関係を定め，その同値関係によって有向グラフを分割する．このとき分岐数最小の生成出グラフは，その分割により得られる各部分グラフ（極大交互グラフという）に対する分岐数最小の生成出グラフを求めることにより得られる．

定義. [4] 有向グラフ $G=(V,E)$ の辺 (u,v) と (w,x) が交互隣接である，$(u,v)\sim(w,x)$ と書く，とは

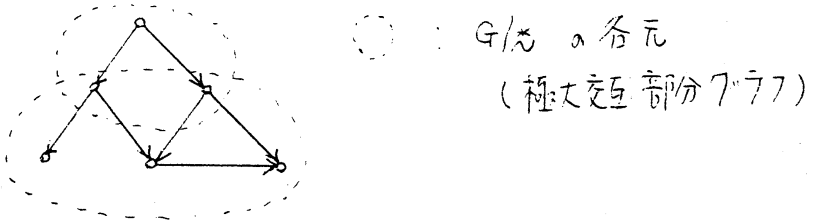
(1) $(u,v)\neq(w,x)$ であり，

(2) $u = w$又は$v = x$である.

を満足するときをいう.



記号$\overset{*}{\sim}$で$\sim$の反射推移閉包を表わす. $G/\overset{*}{\sim}$の各元をGの極大交互部分グラフという. $G/\overset{*}{\sim} = G$のときGを交互グラフという.



補題1. [4] $G_1, G_2, ..., G_n$をDAG Gの極大交互部分グラフのすべてとする. $i \leq n$に対してH_iをG_iの分岐数最小の生成出グラフとすると, $\bigcup_{1 \leq i \leq n} H_i$は$G$の分岐数最小の生成出グラフである.

4 逐次アルゴリズムとその平均時間

4.1 アルゴリズム A1

Gから極大交互部分グラフを得る$O(n)$時間アルゴリズムは, 深さ優先探索[1]によって容易に構成できる. よって以下では, 交互グラフの分岐数最小の生成出グラフを求めるアルゴリズム A1 を示す.

有向グラフはリスト表現(リンク表現)により表現されていると仮定する. 更に, $inlist(v)$は頂点vに入ってくる頂点の並び, $outlist(v)$はvから出てゆく頂点の並びをそれぞれ表わす.

PROCEDURE ALTSEARCH(v,G,F)

G (input,output) is an alternating graph; verteces in G are marked "UNCOVERED", "COVERED", "UNVISITED" and/or "VISITED".

v (input) is vertex in G; searching starts from v.

F (output) is the set of the edges; a spanning out forest for G of minimal branches.

begin
mark v "$VISITED$";

```
    while vertex u remaining in "outlist(v)" mareked "UNCOVERED" do
      begin
        add (v,u) to F; mark u "COVERED";
        while vertex x remaining in "inlist(u)" marked "UNVISITED" do
          ALTSEARCH(x,G,F)
      end
  end
```

PROCEDURE ALTMATCH(G,F)

G (input) is an altDAG with outdegree atmost two.

F (output) is the set of edges of a spanning out forest for G of minimal branches.

```
  begin
    F:=φ
    mark all vertices v with outdegree(v) ≥ 1 "UNVISITED";
    mark all vertices v with indegree(v) ≥ 1 "UNCOVERED";
    if there is a vertex v with outdegree(v)=1
      then ALTSEARCH(v,G,F)
      else
      begin
        if there is an alternate semicycle [4] C in G
          then
            begin
              choose any vertex v in C with outdegree(v)=2;
              ALTSEARCH(v,G,F)
            end
          else
          begin
            choose any vertex v with outdegree(v) ≥ 1 ALTSEARCH(v,G,F)
          end
      end
  end
```

与えられた出次数高々2のDAG $G=(V,E)$ の極大交互部分グラフの各々について，$ALTMATCH$ は depth first に動き，$O(m)$ 時間で分岐最小の生成出森を与える．得られた生成出森から極大パスカバーが得られる．

前に述べたように G から極大交互部分グラフの全てを得る $O(m)$ 時間アルゴリズムが構成できる．よってこれらのアルゴリズムと上の $ALTMATCH$ を組み合せて全体として $O(m)$-時間で動くような深さ優先のアルゴリズムを構成できる．

仮定より $m \leq 2n$ であるから，得られるアルゴリズムは $O(n)$ 時間で動く．よって以下の定理が得られる．

定理１． G を出次数高々２の非サイクル的有向グラフとする．G の極大パスカバーを与える頂点-線型時間アルゴリズムが存在する．□

4.2 平均時間の解析

この節ではアルゴリズム **A1** の平均時間を解析する．ここでは交互グラフはつぎの２条件を満たすものとする．

(C_1) 入次数は高々３，出次数は高々２．

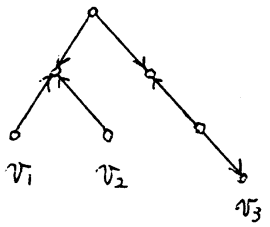
(C_2) 辺の向きを無視しても非サイクル的．

また各交互グラフには根とよばれる出次数が０でない頂点が存在するものとし，根を持つ交互グラフを根付交互グラフという．各根付交互グラフは同じ確率であらわれるものとする．

定義. 根付交互グラフ G の平均的深さを次のように定義する．

$$\sum_{a}(1/2)^k * q(a)$$

ここで，a は G のターミナル頂点(次数１の頂点)，k は根と a を結ぶセミパスにおける枝分かれの個数，$q(a)$ はセミパスの長さとする．



Gの平均的深さ
$= (\frac{1}{2})^2 \times 2 + (\frac{1}{2})^3 \times 2 + (\frac{1}{2}) \times 3$
$= \frac{5}{2}$

定義. n 頂点交互グラフの平均的深さを次のように定義する．

$$SMD(n)/NAL(n)$$

ここで $NAL(n)$ は n 頂点根付交互グラフの総数，$SMD(n)$ は各 n 頂点根付交互グラフの平均的深さの総和とする．

補題２． $NALT(n)$ を根がターミナル頂点である n 頂点交互グラフの総数とし，$SMDT(n)$ をそれらのグラフに対する平均的深さの総和とする．このとき次の **(1)-(4)** 式が成立する．

(1) $NALT(2n) = \sum_{i=0}^{n-2} NALT(2n-2-i) * NALT(i)$
$+NALT(n-1) * \{NALT(n-1)+1\}/2$

$(n \geq 2)$

$NALT(2n+1) = \sum_{i=0}^{n-1} NALT(2n-1-i) * NALT(i)$

$(n \geq 1)$

$NALT(0) = NALT(1) = 1, NALT(2) = 0$

(2) $NAL(2n) = \sum_{i=1}^{n} NALT(2n+1-i) * NALT(i)$
$NAL(2n+1) = \sum_{i=1}^{n} NALT(2n+2-i) * NALT(i)$
$+NALT(n+1) * \{NALT(n+1)+1\}/2$

(3) $SMDT(2n) = SMDT(2n-2) + 2 * NALT(2n)$
$+\{\sum_{i=1}^{2n-3} SMDT(2n-2-i) * NALT(i) + SMDT(n-1)\}/2$

$(n \geq 1)$

$SMDT(2n+1) = SMDT(2n-1) + 2 * NALT(2n+1)$
$+\sum_{i=1}^{2n-2} SMDT(2n-1-i) * NALT(i)/2$

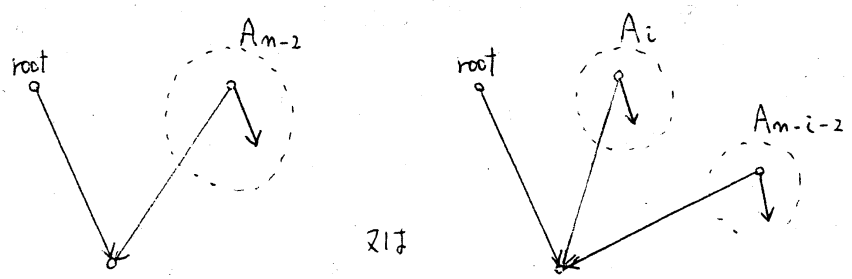
$(n \geq 1)$

$SMDT(0) = SMDT(1) = 0$

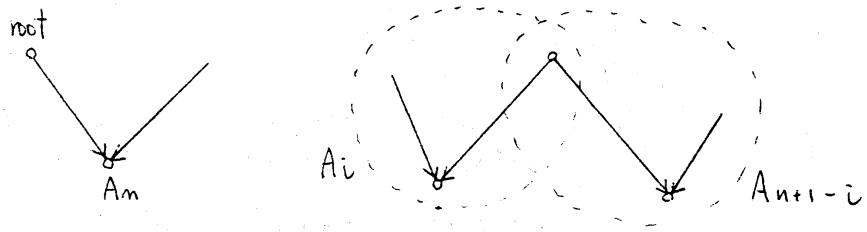
(4) $SMD(2n) = SMDT(2n)$
$+\{\sum_{i=2}^{2n-1} SMDT(2n+1-i) * NALT(i)\}/2$
$SMD(2n+1) = SMDT(2n+1)$
$+\{\sum_{i=2}^{2n} SMDT(2n+2-i) * NALT(i) + SMDT(n+1)\}/2$

証明の概略.

(1) 根がターミナル頂点である n 頂点根付交互グラフはつぎのように書き表すことができる.
(A_i は i 頂点交互グラフ.)



(2) n 頂点根付交互グラフはつぎのように書き表すことができる.



(3) 根がターミナル頂点である n 頂点根付交互グラフにおいて，i 頂点グラフは，$i = n/2 - 1$ の場合をのぞいて，$NALT(n-2-i)$ 回あらわれる．よってこの場合の平均の深さは，$SMDT(i) * NALT(n-2-i)$ である．

$i = n/2 - 1$ のとき i 頂点グラフは $NALT(n/2-1)+1$ 回あらわれる．よってこの場合の平均の深さは，$SMDT(n/2-1) * \{NALT(n/2-1)+1\}$ である．

$i \neq n-2$ のとき，この n 頂点根付交互グラフには i 頂点グラフの他にもうひとつ根から出る枝がある（図1(b)）．よってこの場合の平均の深さは，$2 * NALT(n)$ である．

(4) **(3)** と同様 □

補題3． 任意の $t > 1$, $A > 1$ に対し，ある N が存在して任意の $n > N$ に対してつぎの **(a)**-**(c)** 式が成立する．

(a) $\log_A n < n$

(b) $(6n+1)^t > (6n-1)^t + 1$

(c) $\{(3n+i)^t + (3n-i-1)^t\} - \{(3n+i-1)^t + (3n-i)^t\}$ $\begin{cases} > 0 & i > 0 \\ > 1 & 3n > i > n \end{cases}$

証明. 省略 □

定理2． **(1)**-**(4)** における等式の左辺の増加は，任意の $t > 1$, $A > 1$ に対して，A^n より大きく A^{n^t} より小さい．

証明. フィボナッチ数列との比較により A^n より大きいことは明らか．

まず $NALT(n) > O(a^n)$ を示す．

$NALT(n) = O(A^n)$ を満たす $A > 0$ が存在したとすると，ある $c > 0$ と任意の $\epsilon > 0$ に対し，ある整数 N が存在して，任意の $n > N$ について，

$$c - \epsilon < NALT(n)/A^n < c + \epsilon$$

を満たす．

NALT(4n+1)

$$\begin{aligned} &= NALT(4n-1) + NALT(4n-2) + \cdots + NALT(3n-1) * NALT(n) \\ &\quad + \cdots + NALT(2n) * NALT(2n-1) \\ &> NALT(3n-1) * NALT(n) + \cdots + NALT(2n) * NALT(2n-1) \\ &> (n-1)A^{4n-1} * (c-\epsilon)^2. \end{aligned}$$

したがって $NALT(4n+1)/A^{4n+1} = (n-1)(c-\epsilon)^2/A^2 > c + \epsilon$ となってこれは矛盾である．ゆえに $NALT(n) > O(a^n)$ が示される．$NALT(4n+2)$, $NALT(4n+3)$, $NALT(4n)$ についても同様にして証明できる．

つぎに $NALT(n) < O(A^{n^t})$ を示す．

$NALT(n) = O(A^{n^t})$ を満たす $t > 0$ が存在したとすると，ある $c > 0$ と任意の $\epsilon > 0$ に対し，ある整数 N が存在して，任意の $n > N$ について，

$$NALT(n)/A^{n^t} < c + \epsilon$$

を満たし，またある $n > N$ について，

$$c - \epsilon < NALT(n)/A^{n^t}$$

を満たす．
$M = max\{NALT(i)/A^{i^t},\ 0, 1, \cdots, N\}$ とすると，$NALT(i) < M * A^{i^t}$ である．このとき，

NALT(6n+1)

$$\begin{aligned}
&= NALT(6n-1) + NALT(6n-2) + \cdots + NALT(5n-1) * NALT(n) \\
&\quad + \cdots + NALT(3n) * NALT(3n-1) \\
&< (c+\epsilon)M(A^{(6n-1)^t+0^t} + \cdots + A^{(5n-1)^t+n^t}) \\
&\quad +(c+\epsilon)^2(A^{(5n-2)^t+(n+1)^t} + \ldots + A^{(4n-1)^t+(2n)^t} \\
&\quad +A^{(4n-2)^t+(2n+1)^t} + \cdots A^{(3n)^t+(3n-1)^t})
\end{aligned}$$

したがって任意の $n > N$ に対して $NALT(6n+1) < A^{6n-1)^t+0^t+1}$ である．
ゆえに $NALT(n) < O(A^{n^t})$ が示される．

また $NAL(2n) = \sum_{i=1}^{n} NALT(2n+1-i)NALT(i)$
$< \sum_{i=0}^{n} NALT(2n+1-i)NALT(i)$
$= NALT(2n+3)$

ゆえに $NAL(2n) < NALT(2n+3)$
同様にして $NAL(2n+1) < NALT(2n+3)$ が示される．
したがって $NAL(n) = O(NALT(n)) < O(A^{n^t})$．　同様にして $SMDT(n) < O(A^{n^t})$, $SMD(n) < O(A^{n^t})$ が示される．□

上の定理は，任意の $t > 1$ に対して，$SMD(n)/NAL(n)$ が n^t でおさえられることを示唆している．

予想.　条件 (a), (b) を満たす任意の n 頂点交互グラフを入力としたとき，アルゴリズム **A1** が visit する頂点の平均の個数は，任意の $t > 1$ に対して，$n + n^t$ でおさえることができる．□

さらに我々は，n^t の値は，各 n の値に対する $SMD(n)/NAL(n)$ の数値例から，$\log n$ でおさえられることを推測した（表1）．

n	NAL(n)	SMD(n)	SMD(n)/NAL(n)	{ SMD(n)/NAL(n)} /log n
20	1746	10307.3	5.90337	1.97059
40	3.77919E+07	2.68012E+08	7.09179	1.92248
60	1.12707E+12	8.56329E+12	7.59785	1.85569
80	3.89366E+16	3.06692E+17	7.87670	1.79750
100	1.46386E+21	1.17888E+22	8.05325	1.74874

表1

5 NCアルゴリズム

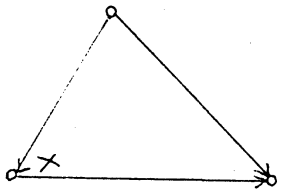
この節では極大パスカバー問題を，ある種の CRCW-PRAM(Concurrent Read Concurrent Write Parallel RAM) で $O(\log n)$ 時間, $O(n)$ プロセッサで解くことができることを示す.

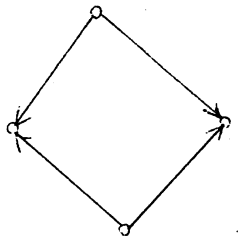
我々が用いる計算のモデルはarbitrary CRCW PRAMといわれる次のものである. (see eg. [7] , [12]) .
複数個のプロセッサ(RAM) , $P_1, P_2, ..., P_k$が共通メモリへ同時にアクセスし，計算を実行する．この際，共通メモリ ([2] のメモリと同様で，レジスタの並び) のある番地に対する，複数個のプロセッサの読み出し及び書き込みの同時性は許されているが，書き込みについては，どのプロセッサが書き込みに成功したかは，分からないものとする．P_iのローカルメモリの内容を L_{i_1}, L_{i_2},...の様に書くことにする.

5.1 完全交互グラフ

我々は議論を簡潔にするため，次の完全交互グラフの概念を導入する.

定義. $G = (V, E)$ を交互グラフとする．各 $e \in E$に対し，e に隣接する全ての辺 e'が $e \sim e'$を満足するとき G は完全であるという．□





また，完全交互グラフの方向を無視した生成木を交互生成木という.

さて，G を DAG とする．G の各頂点 vに接続する辺の向きによって，vを $v-IN, v-OUT$と 2 通りに見なし，G をいくつかの成分 $G_1, G_2, ..., G_k$に分解すると，各 G_i $(1 \leq i \leq k)$ は完全交互グラフになっている．このとき問題 3 は次の問題 4 に帰着できる（後にアルゴリズムの正当性の中で証明する）.

問題 4 . G を完全交互グラフに分解し，その各々の極大パスカバーを求める.

5.2 アルゴリズム A 1

この節では有向グラフから極大パスカバーを求めるアルゴリズム A 2を導入する.

このアルゴリズム A 2で，我々は頂点iと辺(i,j)に対し，それぞれ頂点$i-OUT$, $j-IN$及び辺 $(i-OUT, j-IN)$, $(j-IN, i-OUT)$ を考える.
我々は $D \in \{IN, OUT\}$ に対して，$D = IN$のとき$\bar{D}$でOUT, $D = OUT$のとき$\bar{D}$で

INを表わす.

プロセッサ P_kが頂点 $i-D$に対応する（$P_k = pr(i-D)$ とかく）．とは，P_kのローカルメモリについて $L_{k_1} = i-D$, $L_{k_2} = 0$ が成立するときをいう．プロセッサ P_kが辺$(i-D, j-\bar{D})$に対応する（$P_k = pr(i-D, j-\bar{D})$ とかく）とは，$L_{k_1} = i-D$, $L_{k_2} = j-\bar{D}$が成立するときをいう.

以下でA 1について述べる.

アルゴリズムA 2

入力　$G=(V,E)$, ここで$V=\{1,2,...,n\}$:有向グラフ
頂点番号iはメモリのi番地に格納される.
$E=\{(i_1,j_1),...,(i_m,j_m)\}$ に対して,
$i_1,j_1,i_2,j_2,...,i_m,j_m$が共通メモリの$n+1$番地から$n+2m$番地までにこの順番で格納される[7].

出力　F : Gの極大パスカバー

方法　次のステップ0からステップ5による.
ステップ0 (pre-computation)
ここでは$2n+2m$個のプロセッサと頂点及び辺を対応させる．(1)-(5) でその詳細を示す.

(1) k $(1 \leq k \leq n)$ に対し，$P_k = pr(i-OUT)$, $P_{n+m+k} = pr(i-IN)$ とする．ここでiは共通メモリのk番地の内容.

(2) k $(n+1 \leq k \leq n+m)$ に対し，$P_k = pr(i-OUT, j-IN), P_{2m+n+k} = pr(j-OUT, i-IN)$ とする．ここでiは$n+2k-1$番地，jは$n+2k$番地の内容.
$(j-IN, i-OUT)$の全体を$\bar{E}$とかく.

(1), (2) でのプロセッサと頂点及び辺の対応付けは下の表のようになる.

メモリの番地	番地の内容	対応するプロセッサ
$1,2,...,n$	頂点	$P_1,...,P_n$ $P_{n+m+1},...,P_{2n+m}$
$n+1,...,n+2m$	辺	$P_{n+1},...,P_{n+m}$ $P_{2n+m+1},...,P_{2n+2m}$

次の（3）－（5）は入力データを[10]の入力表現に合わせるようにリスト構造に変更するために行なう.
この論文の残りの部分では，データは番地を意識せずに変数名で表すことにする．又，各頂点と各辺に対応するプロセッサのローカルメモリの内容は，グラフ

の表現がリスト構造に変更されるのに合わせて，適当に変更されるものとする．[10]

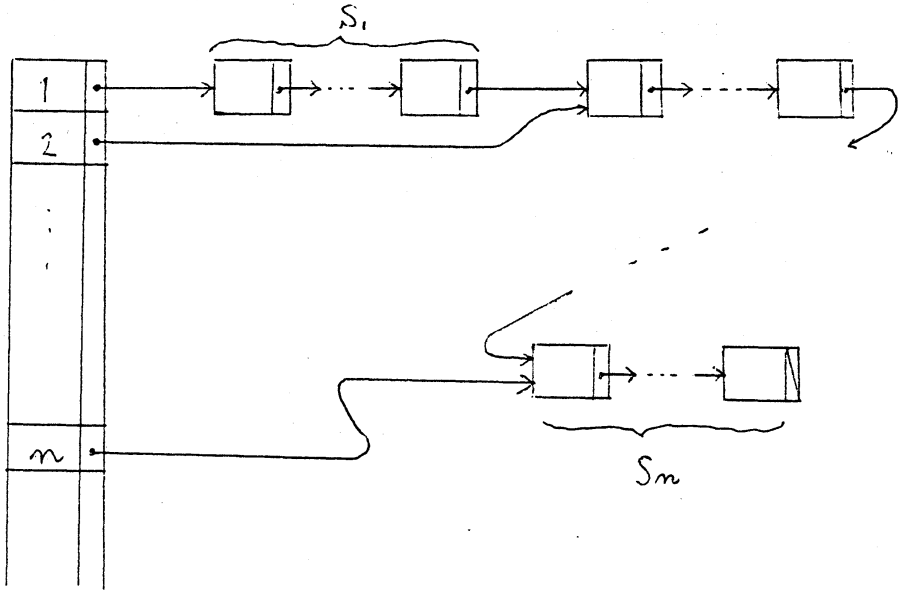
(3) Eの第1成分についてソートし，ソートリストSを作る[8],[11]．
ここに，$S=(S_1,S_2,...,S_n)$であり，
$(i,j_1),(i,j_2),...,(i,j_k)\in E$のとき
$S_i=((i-OUT,j_1-IN)(i-OUT,j_2-IN)...(i-OUT,j_k-IN))$とする．
ただし各S_iにおいて，$next(i-OUT,j_t-IN):=(i-OUT,j_{t+1}-IN)$へのポインタとする．

(4) 各i $(1\leq i\leq n)$に対して，
$adj(i-OUT):=S_i$の最初の辺$(i-OUT,j_1-IN)$へのポインタとする．
(doubling techniqueを用いる．)
S_iの最後の辺$(i-OUT,j_k-IN)$に対して$next(i-OUT,j_k-IN):=null$とする．

(5) $\bar{E}$について**(3),(4)**と同様のことを行なう．

ステップ0の終わり．

このステップ**0**でGは完全交互グラフの集まり$G_1,G_2,...,G_t$に分解され，各辺は2方向の辺として表現される．従ってGは無向グラフと同一視できる．これを$Alt(G)=\bigcup_{i=1}^{t}G_i$と書くことにする．また各頂点から出ていく辺のリストが，ポインタ$adj,next$の形で構成される．このステージの終了後，グラフはリストとして次のように表現される．



我々は，完全交互グラフG_iの極大パスカバーは，G_iの交互生成木を構成し，その極大パスカバーを求めれば良いことを示す．無向グラフの生成木(spanning tree)を構成する並列アルゴリズムは[10],[12]のなかで紹介されている．以下に**ステップ1-5**の概要を記す．

ステップ1 $Alt(G)$ の，根を持たず(Unrooted)方向を持たない(Undirected)生成木 $UUST(Alt(G))$ をつくる．$UUST(Alt(G))$ の辺に選ばれたものにはマークをする [10] , [7] ．(このステップで G の交互生成木が決る．)

ステップ2 $UUST(Alt(G))$ のオイラー閉路を構成し ($UUST(AltG)$ の各辺は両方向に表現されているため，必ずオイラー閉路が存在する．)，閉路のなかの Eに属するひとつの辺を切断することによって $UUST(Alt(G))$ のトラバースリストTを作る．その切断された辺に従って$UUST(Alt(G))$ の根が決定される．これを $RUST(Alt(G))$(Rooted Undirected Spanning Tree of Alt(G)) と書くことにする [10] .

注意 本論文ではステップ1及びステップ2については [7],[10] と同様なので詳細は省略する．ステップ1，ステップ2の実行後，トラバースリストTのなかでは $(i-D, j-\bar{D})$ の次の辺はポインタ $tournext((i-D, j-\bar{D}))$ で表される．

ステップ3 $RUST(Alt(G))$ の各辺をトラバースする順序に従って番号付けし，$(i-OUT, j-IN), (j-IN, i-OUT)$ のうち番号の若い辺にマークする．これによって $RUST(Alt(G))$ の各辺を親から子に向かう辺として表現する．このように表現された木を $RDST(Alt(G))$(Rooted Directed Spanning Tree of Alt(G)) と書くことにする．$RDST(Alt(G))$ を用いて各頂点の親を決定する．

ステップ4 $RDST(Alt(G))$ の葉$i-OUT$のうちから各連結成分ごとにひとつ選び，$i-OUT$の親を根までたどる．これをメインパスという．もし葉$i-OUT$がなければステップ5へ行く．

ステップ5 各頂点 $i-D, D \in \{OUT, IN\}$ に対し以下を実行する．
$i-OUT \in mainpath$ なら $F := F \cup (i-OUT, parent(i-OUT))$ とする．
$i-IN \notin mainpath$ なら $F := F \cup (i-IN, parent(i-IN))$ とする．
(ステップ4でメインパスが構成されなければこのステップが実行される.)
頂点 $i-D$が上の2つのステップで Fに属さないなら，$F := F \cup (i-D)$ とする．

Fの要素に選ばれた辺は，出力に指定された共通メモリのある領域にそれぞれ書き込まれる．

次にアルゴリズムの詳細を述べる．

ステップ3の詳細 トラバースリストTが構成されたら，そのリストにトラバースする順に順序付けし，各 $(i-OUT, j-IN)$ 及び $(j-IN, i-OUT)$ について番号の若いものを $RDST(Alt(G))$ の辺として採用する．それらは次のアルゴリズム NUMBERING, PARENT により実行される．

```
PROCEDURE NUMBERING((i-D, j-D̄), tournext(i-D, j-D̄))
  begin
    pardo for each (i-D, j-D̄) in T
      begin
```

```
        numtoend((i − D, j − D̄)):=1;
        temp(i − D, j − D̄):=tournext((i − D, j − D̄));
        repeat log n times do
          if temp(i − D, j − D̄) is not null then
            begin
              numtoend((i − D, j − D̄))
                := numtoend((i − D, j − D̄))
                    +numtoend(temp(i − D, j − D̄));
              temp(i − D, j − D̄) := temp(temp(i − D, j − D̄))
            end;
      end
    end pardo;
    number(i − D, j − D̄):=2(n-1)-numtoend((i − D, j − D̄))
  end
```

今，各$(i-D, j-\bar{D}), (j-\bar{D}, i-D)$に付いている番号の若い方にマークすることによって，木の各辺を親から子に向かう有向辺として表現する．すなわち$RDST(Alt(G))$を構成することができる．次に述べるのは，$RDST(Alt(G))$の各頂点の親を求めるアルゴリズムである．

```
PROCEDURE PARENT ((i − D, j − D̄), number(i − D, j − D̄)
begin
  pardo for each (i − D, j − D̄) in T
    if number(i − D, j − D̄) < number(j − D̄, i − D) then
      parent(j − D̄) := i − D
  end pardo
end
```

ステップ4の詳細　$RDST(Alt(G))$の葉で，$i-OUT$のものが，もしあれば一つ選び，$i-OUT$と根を結ぶパス（メインパス）を決定する．なければステップ5へとぶ．

ステップ4は次のアルゴリズム**MAINPATH**により実行される

```
PROCEDURE MAINPATH (i − D, parent(i − D))
begin
  anc(i − D) := parent(i − D);
  pardo for each i − OUT
    if i − OUT is a leaf in RDST(Alt(G))
      then mainpath := i − OUT
  end pardo
  repeat log n times do
    pardo for each i − D
      begin
        if i − D ∈ mainpath then
```

$mainpath := mainpath \cup anc(i-D)$;
$anc(i-D) := anc(anc(i-D))$;
end
end pardo
end

ステップ5の詳細 次のアルゴリズム CHAINMATCH により $RDST(Alt(G))$ の極大パスカバーを求める.

PROCEDURE CHAINMATCH $(i-D, parent(i-D))$
begin
$F := \phi$;
pardo for each $i-OUT \in mainpath$
$F := F \cup (i-OUT, parent(i-OUT))$
end pardo;
pardo for each $i-IN \notin mainpath$
$F := F \cup (i-IN, parent(i-IN))$
end pardo;
pardo for each $i-D \notin F$
$F := F \cup (i-D)$
end pardo
end

アルゴリズムの正当性はつぎの補題および定理から示される.

補題4. 完全交互グラフ G が木をなす時, Step5 のように選んだパスカバーは G において極大である.

証明. 木の葉と根を結ぶパスの偶奇性から明らかである. □

定理3. G を与えられた完全交互グラフとする. G の任意の交互生成木 $T(G)$ において極大なパスカバーは G においても極大となる.

証明.
G のある交互生成木で, そのパスの数が,G のパスの数と等しくなるようなものが存在することは明かである. なぜなら, IN とマークされたある頂点を根に指定し, そこから他の全ての頂点を含む, 方向を無視しても非サイクル的なグラフ（即ち木）をつくることができるからである.
実際には, あとひとつ辺を付け足せばサイクルとなってしまうまでパスをつくる. その最後の辺は, パスを構成していないから, それを削除することができる.
また補題2のパスの構成法より, G の交互生成木の葉で, OUT とマークされた頂点の数を最小とする木が, G の極大なパスカバーを与える交互生成木である. 従って我々は, G の交互生成木をどのように構成しても, G において OUT とマークされた葉の数は一定であることを示せばよい. また G において出次数1

の頂点は，交互生成木をどのように構成しても，必ず葉となり，よってその個数は一意的に定まる．ゆえに以下では G は出次数 1 の頂点は持たないものと仮定してもよい．

無向グラフ G' は，完全交互グラフ G に次のオペレーション $OP1$ を繰り返し作用した後 G' と同型になる時，G の縮小であるという．

オペレーション $OP1$ 全ての $(i,j),(i,k) \in E$ に対して，無向辺 $\{j, k\}$ をつくる．

さて任意の完全交互グラフ G に対し G の縮小となる無向グラフ G' を考える．いま G において，k を出次数が 2 である頂点の個数，$T(G')$ を G' の任意の交互生成木とし，$T(G)$ の縮小となる木を $T(G')$ とする． $|V(T(G'))|$=n-k より $|E(T(G'))|$=n-k-1 よって $|E(T(G))| = 2*|E(T(G'))|$=2(n-k-1) ゆえに $|\{E - E(T(G))\}|$=m-2(n-k-1) となる．従って $T(G)$ に含まれない G の頂点の個数は，{m-2(n-k-1)}/2, 更に m=2k であるから 2k-n+1 となる．このことと補題 2 から定理 5 が証明される．□

ステップ 0 − 5 が，O(log n) 時間，O(n) プロセッサで実行されるのは明らかである．従って次の定理が証明される．

定理 4． G を出次数又は入次数を高々 2 に制限した DAG とすると G の極大パスカバーを与える NC アルゴリズムが存在する．□

さらに完全交互グラフの極大パスカバーは，辺の方向を無視すれば，2 部グラフの最大マッチングそのものである．従って次の系を得る．

系 1． $G = (V, E)$, $V = V_1 \cup V_2$ を 2 部グラフとする．V_1 または V_2 における頂点の次数を高々 2 に制限すれば，G の最大マッチングを与える NC アルゴリズムが存在する．□

6 むすび

本論文では有向グラフの極大パスカバー問題の計算量を，グラフに交互同値という関係を導入し，(1) 線型時間アルゴリズムとその平均時間，(2) 並列アルゴリズムについて考察してきた．以上の結果をまとめると，

[6] では出次数，入次数とも 2 に制限した DAG について研究されているが，出次数だけ 2 に制限しても，アルゴリズムのオーダーは同じである．

(1) のアルゴリズムが visit する頂点の平均の個数の上限は $n + \log n$ であることを推測した．

(1) のアルゴリズムに対して”効率的”な並列アルゴリズムを構成した．

また，上の２つのアルゴリズムは，一方の集合における頂点の次数を２に制限した，二部グラフの最大マッチング問題に適用できる．特に最大マッチング問題の並列計算量は未解決問題になっていて，ここでの結果は，制限された二部グラフについて解かれたことになる．最後に，今後の研究課題は一般の DAG における極大パスカバー問題の並列計算量を求めることで，これは二部グラフにおける最大マッチング問題の並列計算量を求めることと同値である．

謝辞

平均時間の評価について貴重な助言をいただいた東海大学の大谷光春助教授に感謝致します．また，５節の並列アルゴリズムに対しご指導いただいた東京電機大学の西野哲朗助手，多くの論文を紹介してくださった日本電気（株）の土田賢省氏に感謝致します．

参考文献

[1] R. E. Tarjan, Depth-First Search and Linear Graph Algorithm, *SIAM J. Compute.* 1(2) (1972), 146-160.

[2] A. V. Aho, J. E. Hopcroft and J. D. Ullman, The Design and Analysis of Computer Algorithms, *Addison-Wesley, Reading MA*, (1974)

[3] T. B. Boffey, The Linearization of Flowcharts, *BIT* 15(1975),341-350.

[4] 夜久竹夫，フローグラフの分岐数について，**京大数解研講究録** 259 (1976), 158-164.

[5] F. T. Boesch and J. F. Gimpel, Covering the Points of a Digraph with Point-Disjoint Paths and its Application to Code Optimization, *JACM* 24 (1977),192-198.

[6] S. Iwata, Programs with Minimal GOTO Statements, *Inform. Control* 37 (1978), 105-114.

[7] Y. Sialoach, U. Vishkin, An O($\log n$) Parallel Connectivity Algorithm, *J. Algorithms* 3 (1982), 88-102

[8] M. Ajtai, K. Komlós, E. Szeméred, An O($n \log n$) Sorting Networks, *Proc. 15th ACM Symposium on Theory of Computing* (1983), 1-9

[9] 植村憲治，夜久竹夫，交代グラフの道被覆問題の平均計算量，**電子通信学会　オートマトンと言語研究会資料** AL-85(1985), 71-76

[10] R. E. Tarjan, U. Vishkin, An Efficient Parallel Algorithms, *SIAM J. Compute.* 35(2) (1985), 862-874

[11] R. Cole, Parallel Merge Sort, *Proc. 27th Annual IEEE Symp. on Foundations of Comp. Sci.* (1986), 511-516.

[12] R. M. Karp, V. Ramachandran, A Survey of Parallel Algorithms for Shared Memory Machines, *Tech. report UCB/CSD 88/408*, Computer Sience Division, University of California Berkeley, Calif. (1988)