アルゴリズムの効率化―特にグラフの列挙問題―に関する研究

１９７７年１月

築　山　修　治

内容梗概

本論文は，著者が大阪大学大学院工学研究科（電子工学専攻）在学中に尾崎研究室において行った研究のうち，アルゴリズムの効率化 ── 特にグラフの列挙問題 ── に関する研究をまとめたものである．

第1章緒論では，本研究の意義およびこの分野での研究の現状について述べ，本研究によって得られた諸成果を概説する．

第2章では，次章以後で述べるアルゴリズムの効率の良さ（複雑度）を評価する方式について述べる．まず，計算機モデルRAM（random access machine）の概略を述べ，アルゴリズムの効率の良さ（複雑度）をそのRAMプログラムを用いて定義する．次に，問題の規模を表わす変数nの増大に伴って，RAMプログラムが必要とする演算時間T(n)の増大する割合 ── オーダ(order) O(f(n)) ── を定義し，それをアルゴリズムの時間複雑度を示す関数とする妥当性について述べる．最後に，以下で述べるアルゴリズムの複雑度については，問題の規模を表わす変数として，入力の規模を示す変数と出力の規模を示す変数の2つを用いる必要のあることを示す．

第3章では，有向グラフにおけるサイクルの列挙問題について考察し，新しいアルゴリズムを提案する．まず，従来のアルゴリズムについて概観し，それらの長所・短所を考察する．それによって，より効率的なアルゴリズムを見い出すための1つの接近法として，サイクルを見い出す際に探索すべき辺の個数を削減すること，すなわち探索範囲の削減が挙げられることを示す．次に，探索範囲の削減を効果的に行うために，与えられた有向グラフに対してDFS（depth-first search）の技法で1つの森を生成し，その森を用いてサイクルの特徴付けを行う．それによって得られた結果と従来のアルゴリズムの一部を結合して，より効率的なアルゴリズムを導出する．従来のアルゴリズムのうち最良のものは$O((|V|+|E|)\cdot(|\mathcal{C}|+1))$の時間複雑度と$O(|V|+|E|)$のスペース複雑度を持っていたが，このアルゴリズムもこれらと同じ時間およびスペース複雑度を持っている．しかし，従来のものが$O((|V|+|E|)\cdot(|\mathcal{C}|+1))$の演算時間を要したいくつかの例に対して，このアルゴリズムは$O(\sum_{C\in\mathcal{C}}|C|+|V|+|E|)$の演算時間ですむ．ここで，$V$，$E$，および$\mathcal{C}$は，それぞれ有向グラフの頂点，辺，およびすべてのサイクルの集合であり，各サイクル$C\in\mathcal{C}$は頂点の系列で

出力されるものとしている．

第４章では，無向グラフにおける極大独立頂点集合の列挙問題について考察し，新しいアルゴリズムを提案する．まず，従来のアルゴリズムについて概観し，それらの長所・短所を考察する．それによって，極大独立頂点集合を列挙する場合には，極大でないものや重複したものを生成しないようにする効率的な手法が必要なことを示す．次に，このような手法を導くために，与えられた無向グラフGに対して，Gを張る部分グラフ$G(W_{i-1})$とそれから生成されるGを張る部分グラフ$G(W_i)$を考え，この$G(W_i)$のすべての極大独立頂点集合を$G(W_{i-1})$のそれらから見い出す問題を考察する．それによって得られた結果とbacktrackingの技法を組み合せ，効率の良いアルゴリズムを導出している．このアルゴリズムは，$O(|V|\cdot|E|\cdot|\mathcal{M}|+|V|+|E|)$の時間複雑度と$O(|V|+|E|)$のスペース複雑度を持っており，このオーダの時間およびスペース複雑度は従来のアルゴリズムでは得られなかったものである．ここで，V，E，および$\mathcal{M}$は，それぞれ無向グラフの頂点，辺，およびすべての極大独立頂点集合の集合である．

第５章結論では，本研究全般にわたり，その結果の意義と残された問題点についてまとめる．

関連発表論文

(1) 築山, 白川, 尾崎, "有向グラフのすべてのサイクルを求めるための一手法," 信学会, 回路とシステム理論研資, CST 73-66 (1973-12).

(2) 築山, 白川, 尾崎, "有向グラフのすべてのサイクルを求めるための一手法," 信学論(A), Vol. 58-A, No. 4, pp. 196-203 (1975-4).

(3) S.Tsukiyama, I.Shirakawa, H.Ozaki, "A survey: Generating all the cycles of a digraph," Proc. 2nd USA-JAPAN Computer Conf., pp. 92-96 (Aug. 1975).

(4) 井手, 築山, 有吉, 尾崎, "グラフのすべての極大独立頂点集合を求めるアルゴリズム," 信学会, 回路とシステム理論研資, CST 74-103 (1975-1).

(5) 築山, 井手, 有吉, 尾崎, "極大独立頂点集合を生成するための一手法," 信学論(A), Vol. 59-A, No. 6, pp. 433-440 (1976-6).

(6) S.Tsukiyama, M.Ide, H.Ariyoshi, I.Shirakawa, "A new algorithm for generating all the maximal independent sets," SIAM J. Comput., to appear.

アルゴリズムの効率化 ── 特にグラフの列挙問題 ── に関する研究

目　　次

第1章 緒論

人が自らの手で問題を解いていた時代には，小さな規模の問題しか解けなかったため，問題の解法（アルゴリズム）を解析する場合，そのアルゴリズムが正しく完了するかどうかという点にのみ注目した．しかし，電子計算機が発達し，種々の問題を電子計算機で解くようになると，アルゴリズムの効率の良さにも注目する必要が生じてきた．初期の電子計算機においてはメモリ容量が少なかったため，アルゴリズムの効率の良さの主な規準は必要なメモリスペースの量であったが，その後，メモリが大量にかつ安価に入手できるようになったため，現在では必要な演算時間がより重要な評価規準となっている．しかし，その評価方法も初めは理論的ではなく，実験結果を示し，「この結果からアルゴリズムの効率は良いように思われる」といった表現が用いられていた．ところが，1965年にEdmondsがグラフのマッチング（matching）に関する論文[1]において，「このアルゴリズムの困難さ（difficulty）は，グラフの規模の指数関数ではなく，中位いの次数の多項式に従って増大する」という表現を行い，効率の良さを問題の規模の関数で表現するという考えを示してからは，必要な演算時間を理論的に評価するようになった．それ以後現在まで，アルゴリズムの効率の良さ（それを時間複雑度と呼ぶ）は，問題の規模を示す変数nの増大に伴って，アルゴリズムに必要な演算時間の増大する割合 ── オーダ(order) $O(f(n))$ ── を用いて評価するようになっている[2,3]．例えば，時間複雑度が$O(n)$のアルゴリズムは，$O(n^3)$のアルゴリズムより効率が良いと評価される．

ところで，このような意味での効率の良いアルゴリズムは，近年ますますその必要性を増している．というのは，我々を取り巻くシステムが巨大化するにつれて解くべき問題の規模が増大し，効率の良いアルゴリズムが見い出されていないため，実用的に可能な時間内に解けない問題が数多く現われてきたためである．例えば，時間複雑度が$O(2^n)$，$O(n^3)$，および$O(n)$と表わされる3つのアルゴリズムが，$n=10$の問題を解くのにいずれも1分要したとすれば，$n=1000$の問題を解くために第1および第2のアルゴリズムはそれぞれ2^{990}分$\fallingdotseq 2^{984}$時間および10^6分$\fallingdotseq 16,700$時間を要するのに対して，第3のものは10^2分$\fallingdotseq 1.7$時間ですむ．さらに，このような意味での効率の良いアルゴリズムは，計算機の処理速度が増加した場合，その効率が一層良くなる．例えば，先ほどの例の3つのアルゴリズムを現在の計算機より10倍速い計算機で実行してみれば，第1および第2のものは1分以内にそれぞれ$n=13$および$n=21$の問題しか解けないのに対して，第3のものは$n=100$の問題を解くことができる．

従って，アルゴリズムの効率化（より効率の良いアルゴリズムを見い出すこと）は，現在，計算機科学の重要な課題の1つとなっており[4]，それは実用上有意義な問題でもある．本研究は，このアルゴリズムの効率化という立場からグラフ理論を考察したものである．

さて，与えられたシステムを解析あるいは構成する場合，通常はそのシステムのもつ本質的な挙動を抽出し，適当な数理モデルを用いて定式化する．このとき，システムの構造が離散的（あるいは組合せ論的）である場合には，その構造を表現する数理モデルとしてグラフを用いることが多い．このような例は，電子通信工学や化学など工学の分野だけでなく，社会学や心理学の分野においても見ることができる．これは，グラフがシステムの構造を表現する簡潔かつ適切な数理モデルであり，グラフを用いた表現によって効率の良いアルゴリズムを確立できることが多いためである[5,6]．このような特徴によって，グラフ理論の有用性は電子計算機の発達に伴ってますます増大し，各分野に応用されるようになっている．従って，グラフ理論における各種アルゴリズムを効率化することは，グラフ理論の応用面の広さから，極めて重要な問題である．

ところで，システムの解析や構成に付随したグラフの問題は，大きく3つに分類される．第1は，与えられたグラフあるいはサブグラフが指定された性質を持つか否かを判定する，いわゆる判定問題であり，第2は，与えられた制約のもとで，指定された目的関数を最適化する，いわゆる最適化問題である．第3は，指定されたある性質を持つサブグラフすべてを列挙する，いわゆる列挙問題である．

本論文では，これらのうち列挙問題に注目し，特に工学的に重要な，有向グラフにおけるサイクルの列挙問題ならびに無向グラフにおける極大独立頂点集合の列挙問題について考察する．

まず，第2章において，次章以後で述べるアルゴリズムの効率の良さ（複雑度）を評価する方式について述べる．初めに，複雑度評価に最も良く用いられている計算機モデルRAM（random access machine）[3,7]の概略を述べ，次に，オーダ（order）を用いて複雑度を評価する妥当性について論じる．最後に，次章以後で述べる列挙問題に対しては，問題の規模を示す変数として，入力の規模を示す変数と同時に，出力の規模を示す変数も用いる必要のあることを示す．

第3章において，有向グラフにおけるすべてのサイクルの列挙問題を考察し，新しいアルゴリズムを導く．従来のアルゴリズムのうち最良のものは，$O((|V|+|E|)\cdot(|C|+1))$の時間複雑度を持っていたが，このアルゴリズムもこれと同じ時間複雑度を持っている．しかし，従来のものが$O((|V|+|E|)\cdot(|C|+1))$の演算時間を要したいくつかの例に対して，このアルゴリズムは$O(\sum_{C\in\mathcal{C}}|C|$

$+|V|+|E|)$の演算時間ですむ．ここで，V，E，および$\mathcal{C}$は，それぞれ有向グラフの頂点，辺，およびサイクルの集合であり，各サイクル$C \in \mathcal{C}$は，頂点の系列で表現されているものとする．

第4章において，無向グラフにおけるすべての極大独立頂点集合の列挙問題を考察し，新しいアルゴリズムを導く．このアルゴリズムは，$O(|V|\cdot|E|\cdot|\mathcal{M}|+|V|+|E|)$の時間複雑度を持ち，このオーダの時間複雑度は従来のアルゴリズムでは得られなかったものである．ここで，V，E，および$\mathcal{M}$は，それぞれ無向グラフの頂点，辺，および極大独立頂点集合の集合である．

最後に第5章で，本研究全般にわたり，その結果の意義と残された問題点についてまとめる．

第2章 複雑度解析の方式

2.1 緒言

計算機科学（computer science）の中に，計算複雑度解析（computational complexity analysis）[8]と呼ばれる1つの分野があり，アルゴリズムの効率化やその評価などはこの分野に含まれている．この分野でよく用いられる計算機械のモデルに，チューリング機械（Turing machine）[3,9]とランダムアクセス機械（random access machine）RAM[3,7]の2つがあるが，これらは計算複雑度解析の上で密接な関係がある[7]．これらのうち，前者は問題を解くのに最小限必要とする計算手数の下限を解析したり，ある規準のもとでの問題間の等価性を解析[10]したりする際に用いられており，この方面での研究はここ2～3年で急速に進歩した[8]．後者は主にアルゴリズムの効率化やその評価などに用いられる計算機モデルであり，通常の電子計算機の理想的なモデルの1つである．この章では初めに，このRAMの概略について述べ，これを用いてアルゴリズムの効率の良さを計る尺度――複雑度（complexity）――を定義する．その後，次章以後に述べる列挙問題の複雑度を評価する方式について述べる．

2.2 計算機モデルの概要

ランダムアクセス機械（random access machine）RAMとは，1つのアキュミュレータ（accumulator）を持つ計算機であり，各命令はプログラム自身を修正することができない．その概略は図2-1に示すように，読み取り専用入力テープ，書き込み専用出力テープ，プログラム，および，メモリで構成されている．

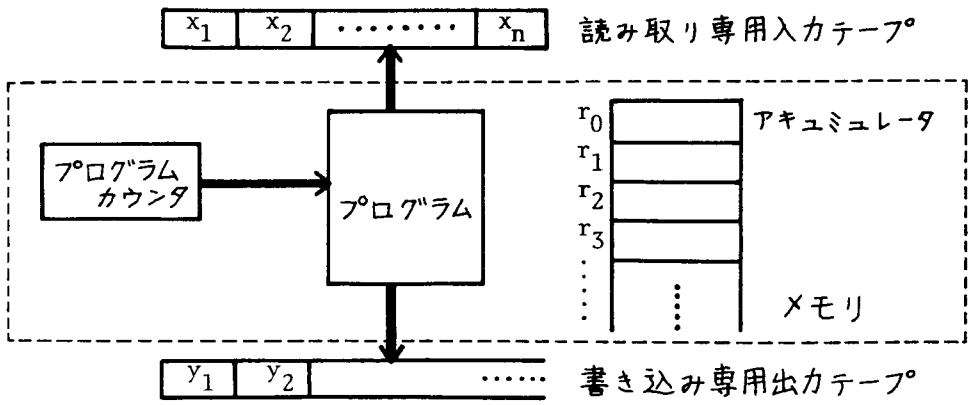


図2-1 ランダムアクセス機械RAM

メモリはレジスタ（register）の系列$r_0, r_1, r_2, \cdots$から成りr_0はアキュミュレータとして用いられる．各レジスタは十分大きな整数を表現することができ，使用できるレジスタの個数に制限はないものとする．このような仮定は，問題に現われる整数はアルゴリズムが実行される計算機の1語長に比べて十分小さく，問題の規模は計算機のメモリ容量に比べて十分小さいということに対応している．このようなレジスタ1個をメモリスペースの1単位とする．

プログラムはラベルが付けられた命令の系列から成り，それらはメモリ以外のところに貯えられており，そのため，プログラム自身を修正することができない．プログラムカウンタは次に実行されるべき命令のラベルを示しており，これはJUMP命令によって変更することができる．RAMの命令としては，通常の電子計算機に見られる機械語命令と同種のものを想定し，LOAD-STORE命令，READ-WRITE命令，および各算術命令などはアキュミュレータr_0を介して実行されるものとする．さらに，これらの命令はどれも1単位時間で実行され，レジスタの内容には依存しないものとする．

このような，レジスタ1個はメモリスペースの1単位を要し，各命令はそれを実行するのに1単位演算時間を要するという仮定（uniform cost criterion[3]）は，以下で取り扱うグラフの問題においては十分妥当なものであり，この仮定によってアルゴリズムの評価が容易になる．この他に，あるレジスタに整数iが入っているとすれば，そのレジスタは$[\log_2|i|+1]$単位のメモリスペースを必要とし，そのレジスタを用いる命令は$[\log_2|i|+1]$単位の演算時間を要すると仮定する場合もある（logarithmic cost criterion[3]）．ここで，$[x]$はxを越えない整数を表わす．このような仮定は，整数iがそれを取り扱う電子計算機の1語長よりもはるかに大きい場合などに適切であるが，アルゴリズムの評価が困難になる．そこで以下では，uniform costの仮定を用いることにする．

以下でアルゴリズムの効率の良さ（複雑度）を評価する場合には，そのRAMプログラムが必要とする演算時間およびメモリスペースの量を用い，それらが問題の規模の増大に伴って増大する割合をその評価規準とするため，ここではRAM命令の詳細については触れない．従って，RAMによって実行することができ，そのような割合を変化させない命令ならば，任意の命令を用意することができる．

2．3　複雑度解析の手法

問題の規模を示す整数の変数をnとし，その問題を解くアルゴリズムAを考える．ここで，問題の規模とは一般にRAMの入力テープの長さを言うが，それは多くの場合，自然な方法で定めることができる．

［定義２．１］　アルゴリズムＡの時間複雑度（time complexity）およびスペース複雑度（space complexity）とは，ＡをＲＡＭで実行した際，そのＲＡＭプログラムが必要とする演算時間t_Aおよびメモリスペースs_Aを，問題の規模nの関数として表現したもの$t_A(n)$および$s_A(n)$である．

今，アルゴリズムＡの時間複雑度を$t_A(n) = k_1 \cdot n^2 + k_2 \cdot n + k_3$と表わすことができたとしよう．ただし，$k_1$，$k_2$，$k_3$は定数である．ところで，このような定数は，単位時間の取り方やＲＡＭの命令の種類などに依存する上に，実際にアルゴリズムを実行する電子計算機がいく種類もあるため，このような定数を含んだアルゴリズムの評価方法は適当ではない．さらに，nが大きくなれば，これら定数の意味はうすれ，第１章緒論においても述べたように，実際にnはますます増大する傾向にある．それゆえ，このような定数を含んだ関数を用いて時間複雑度を表現することは適切ではなく，nが十分大きくなった際の関数$t_A(n)$のふるまいによって表現するのが適当である．スペース複雑度に関しても同種の議論が成り立つ．

［定義２．２］　非負の整数変数nの関数$g(n)$，$f(n)$に対して，どのようなnにおいても$g(n) \leq k_1 \cdot f(n) + k_2$を満たすような定数$k_1$，$k_2$が存在するならば，$g(n)$のオーダ（あるいは位数）は$f(n)$であると言い，$O(f(n))$と書く．

以下では，この位数$O(\cdot)$を用いて複雑度を表わすことにする．位数で表わされた時間およびスペース複雑度は，問題の規模nが増大した際，ＲＡＭプログラムが必要とする演算時間およびメモリスペースの増大する割合を示しており，このような表現において，より小さい時間複雑度を持つアルゴリズムの必要性は緒論において述べたとうりである．

ところで，規模nの問題といっても一般に多くの種類がある．例えば，$n \times n$の０−１行列といっても，その０要素および１要素の配列にはいく種類もある．従って，時間およびスペース複雑度も次の２種類を考えることができる．

［定義２．３］　規模nのある問題において，そのすべての種類の入力に関する時間複雑度の中で，最大のものを最悪時間複雑度（worst-case time complexity）と呼び，そのすべての種類の入力に関する時間複雑度の期待値を平均時間複雑度（expected time complexity）と呼ぶ．スペース複雑度に対しても，同様に最悪スペース複雑度（worst-case space complexity）および平均スペース複雑度（expected space complexity）を定義する．

一般に，最悪複雑度の良いアルゴリズムが平均複雑度も良いとは限らない．しかし，最悪複雑度の方が平均複雑度より解析しやすい上に，平均複雑度を解析するためには，規模nの入力すべての集合の上にある確立関数を定義する必要があり，第３章や第４章で述べるグラフの問題に対してそのような合理的な

確立関数を導入することは困難である．そこで以下では，最悪複雑度を用いてアルゴリズムを評価することにし，単に時間複雑度あるいはスペース複雑度と言えば，それは最悪時間複雑度あるいは最悪スペース複雑度を指すものとする．

さて，このように位数を用いて複雑度の評価を行うならば，アルゴリズムをRAMの命令で記述する必要はなく，高水準な言語で記述することができる．次章以後では，アルゴリズムをALGOL-likeな言語で記述するが，その文法はALGOL 60[11]に準じている．このALGOL-likeな言語で書かれたアルゴリズムをRAMプログラムに変換することは容易であり，それはALGOLコンパイラ（compiler）によってなされる．そのコンパイラの操作は煩雑ではあるが単純なものであるため，ALGOL-likeな記述のままでアルゴリズムの複雑度を評価することは容易である．ただし，この言語では手続き（procedure）の再帰呼び出し（recursive call）が許されており，以後述べるアルゴリズムでもそれを用いている．従って，再帰呼び出しのある手続きをRAMプログラムに変換した場合に必要となるスタックなどは，アルゴリズムの記述に陽には現われていない．このような点にのみ注意し，そのようなスタックもメモリスペースの評価の中に考慮されていることに気が付けば，以下で述べるアルゴリズムはRAMで実行できること，およびその複雑度の評価関数$O(f(n))$はRAMプログラムにおいても成立していることが容易に確められる．

2.4 列挙問題における複雑度解析

前節において複雑度の定義を行ったが，問題の規模を示す変数nについてはあいまいな表現であった．そこで，この節において，次章以後に取り扱うグラフの列挙問題に注目し，その問題の規模の定義を明確にしよう．

与えられた集合Vに対して，Vの任意の2元v, wの非順序対（unordered pair）を$\langle v, w\rangle$，順序対（ordered pair）を(v, w)で表わし，Vにおけるすべての非順序対の集合およびすべての順序対の集合を，それぞれ$V \triangle V$および$V \times V$と表わす．

［定義2.4］[12] 無向グラフGとは，空でない集合Vと，Vと素な集合E，および写像$\varphi : E \to V \triangle V$で定義される合成概念$G=[V, E, \varphi]$であり，$V$の各元を頂点，$E$の各元を無向辺と呼ぶ．

［定義2.5］[12] 有向グラフGとは，空でない集合Vと，Vと素な集合E，および写像$\psi : E \to V \times V$で定義される合成概念$G=[V, E, \psi]$であり，Vの各元を頂点，Eの各元を有向辺と呼ぶ．

［定義2.6］[12] 無向（あるいは有向）グラフ$G=[V, E, \varphi]$に対して，無向（あるいは有向）グラフ$G_0=[V_0, E_0, \varphi_0]$が，

(i) $V_0 \subset V$, $E_0 \subset E$,

(ii) 写像 ψ_0 は写像 ψ の E_0 へ縮小した縮小写像である,

の2条件を満たすとき, G_0 を G の部分グラフという.

グラフの列挙問題とは, 与えられた無向(あるいは有向)グラフにおいて, 指定された性質を持つ部分グラフをすべて見い出せという問題である. 従って, その入力は, 無向(あるいは有向)グラフ G であり, 出力は, 求める部分グラフすべての集合 S である. 普通, 問題の規模を示す変数として入力の規模を示す変数を用いるから, この場合は無向(あるいは有向)グラフの頂点の個数 n と無向(あるいは有向)辺の個数 m を問題の規模を示す変数として挙げることができる. というのは, 無向(あるいは有向)グラフをRAMの入力とするには, 例えば各頂点および各無向(あるいは有向)辺にそれぞれ1つの整数を割り当て, 各無向(あるいは有向)辺から $V \triangle V$ (あるいは $V \times V$) の元への対応をリストしておけばよいから, $O(n+m)$ の規模でグラフを入力することができるためである.

ところで, 第3章で述べる有向グラフにおけるサイクルの列挙問題では, 頂点の個数が n の有向グラフの中に, 出力集合 S の元の個数が $\sum_{i=2}^{n} {}_nC_i \cdot (i-1)! \sim n!$ であるようなグラフが存在し, また, 第4章で述べる無向グラフにおける極大独立頂点集合の列挙問題では, 頂点の個数 $n=3k$ の無向グラフの中に, 出力集合 S の元の個数が 3^k であるようなグラフが存在する(そのような例は各章において示す). 従って, 時間複雑度として最悪の場合を取ること, および出力のために必ず S の元の個数に比例した演算時間が必要なことに注意すれば, これらの列挙問題の規模を示す変数として入力の規模のみを用いている限り, それぞれ $O(n!)$ および $O(3^k)$ の時間複雑度より良い時間複雑度を得ることができない.

しかし, 頂点の個数および有向辺の個数が同じ有向グラフ G_1, G_2 でも, その写像 ψ_1, ψ_2 が異なればそのサイクルの個数も異なり, また, 頂点の個数および無向辺の個数が同じ無向グラフ G_1, G_2 でも, その写像 ψ_1, ψ_2 が異なればその極大独立頂点集合の個数も異なる. 従って, このような列挙問題に対するアルゴリズムの時間複雑度を評価する場合には, 入力の規模を示す変数だけでなく, 出力の規模を示す変数も用いる必要があることがわかる.

2.5 結言

この章では, 次章以後で述べるアルゴリズムの効率の良さの評価方法, すなわち複雑度解析の方式について述べた. それをまとめると次のようになる.

(i) RAMの1つのレジスタをメモリスペースの1単位，各命令を実行するのに要する時間を演算時間の1単位とする．

(ii) アルゴリズムをRAMプログラムとして実現したとき，そのRAMプログラムに要する演算時間およびメモリスペースが，問題の規模nの増大に伴って増大する割合の上界をアルゴリズムの時間複雑度およびスペース複雑度と呼び，位数$O(f(n))$を用いて表わす．

(iii) アルゴリズムはALGOL-likeな言語で記述するが，それをRAMプログラムに変換する操作は容易であり，従って，ALGOL-likeな言語のままで時間およびスペース複雑度を評価することができる．

(iv) 以下で着目するグラフの列挙問題に対しては，問題の規模を示す変数として，入力の規模を示す変数だけでなく，出力の規模を示す変数も用いる必要がある．

第3章 有向グラフにおける サイクルの列挙問題

3.1 緒言

有向グラフのすべてのサイクルを見い出す問題は，グラフ理論における基本的な列挙問題の1つであると同時に，フィードバックシステムのシグナルフロウグラフ的解析や，プログラムの解析や評価など，広範囲にわたる応用を持っている[12,13]．そのため，これまで多数の著者がこの問題を研究して来た[14,15-25]，それらの手法は大きく2つに類別できる．1つは，隣接行列などの行列を用いた代数的な手法[14,15,25]であり，他の1つは，backtrackingの技法を用いた探索手法（search method）[16-25]である．これらのうち，前者の行列を用いた手法は，データ構造が煩雑なうえ時間複雑度も実用的なものではなく[25]，その改良もあまりなされていない．それに対して，後者のbacktrackingの技法を用いたアルゴリズムでは，求めるサイクルを1つずつ出力することができるため，前者のように求めるすべてのサイクルをメモリ内に貯えておく必要がなく，さらに，ここ4～5年の間にその時間複雑度は飛躍的に改良され[24]，$O((|V|+|E|)\cdot(|\mathcal{C}|+1))$のオーダの時間複雑度を持つアルゴリズムが見い出されるに至った[21-23]．ここで，V，E，および$\mathcal{C}$は，それぞれ有向グラフの頂点，有向辺，およびサイクルの集合である．

このようなbacktrackingの技法を用いたアルゴリズムにおいて，これまで考察されてきたことは，いかにして無駄な探索（結果として所望のサイクルが発見されなかった探索）の繰り返しを防ぐかという点であり，探索範囲を削減する（すなわち，1つのサイクルを見い出す際に探索すべき有向辺の個数を少くなくする）という点に関しては，ほとんど考察されていない．

ところで，サイクルを頂点の系列によって出力することにすれば，すべてのサイクルを出力するために少くなくとも$O(\sum_{C\in\mathcal{C}}|C|)$の演算時間を必要とするから，入力に要する演算時間を考慮して，最良のアルゴリズムは$O(\sum_{C\in\mathcal{C}}\cdot|C|+|V|+|E|)$の時間複雑度を持つものと考えられる．このような時間複雑度を持つ最良のアルゴリズムを得るためには，サイクル1つ当りを見い出すために$O(|V|+|E|)$の演算時間を要するわけにはいかず，どうしてもサイクルが存在する範囲だけを探索する必要があると考えられる．従って，従来のアルゴリズムを改良し，$O((|V|+|E|)\cdot(|\mathcal{C}|+1))$の時間複雑度より良いオーダの時間複雑度を持つアルゴリズムを見い出すためには，探索範囲を削減するという点

についても考察する必要があるであろう．

そこで，この章では，この観点から[20]で示したアルゴリズムに[22]で用いられた無駄な探索の繰り返しを防ぐ技法（blocking-unblocking技法）を導入して，新しいアルゴリズムを導く．このアルゴリズムの時間複雑度は従来のアルゴリズムと同じ$O((|V|+|E|)\cdot(|C|+1))$であるが，探索範囲削減に対する1つの接近法が付け加えられているため，従来のアルゴリズムより実際に必要とする演算時間が短かく，さらに，従来のアルゴリズムが$O((|V|+|E|)\cdot(|C|+1))$の演算時間を要した代表的ないくつかの最悪例に対して，$O(\sum_{C\in\mathcal{C}}|C|+|V|+|E|)$の演算時間で解を見い出すことができる．

初めに，従来のアルゴリズムのうちbacktrackingの技法を用いているものについて概観し，それらの長所・短所について考察する．次に，与えられた有向グラフに対してDFS（depth-first search）[26]の技法を用いて1つの森を生成し，その森を用いてサイクルの特徴付けを行う．それによって得られた結果にblocking-unblockingの技法を導入し新しいアルゴリズムを導く．最後に，そのアルゴリズムを評価し，実験結果を示す．

3.2 諸定義

有向グラフ$G=[V,E,\Psi]$において，相異なる有向辺e_1, e_2が$\Psi(e_1)=\Psi(e_2)=(v,w)\in V\times V$を満たすとき，これらの$e_1$, e_2は互いに<u>同方向に並列</u>であると言い，$\Psi(e)=(v,v)$なる有向辺eを<u>自己サイクル</u>と言う．この章で対象とする有向グラフは，V, E共に有限集合であり，自己サイクルおよび同方向に並列な有向辺のいずれをも含まないものとする．そこで，この章においては，単に<u>グラフ</u>と言えばこのような有向グラフをさすものとし，さらに，グラフGを，Gの頂点集合$V(G)$とGの辺集合$E(G)$を用いて単に$G=[V(G),E(G)]$と書く．また，この章では有向辺を単に<u>辺</u>と呼び，$\Psi(e)=(v,w)$を単に$e=(v,w)$と表現する．辺eに対して，$e=(v,w)$であるとき，辺eは頂点vを<u>始点</u>とし頂点wを<u>終点</u>とする，あるいはeはvから<u>出て</u>wに<u>入る</u>と言う．また，vおよびwをeの<u>両端点</u>と言い，vおよびwは互いに<u>隣接する</u>と言う．さらに，以下では，eの始点vおよび終点wをそれぞれ$s(e)$および$t(e)$で表わす．

［定義3.1］ グラフGの辺の順序列$S=[e_1,e_2,\cdots,e_k]$が，

$$e_i=(v_{i-1},v_i) \quad (1\leq i\leq k) \qquad (3\text{-}1)$$

$$v_i\neq v_j \quad (0\leq i<j\leq k) \qquad (3\text{-}2)$$

を満たすとき，SのなすGの部分グラフ$R=[\{v_0,v_1,\cdots,v_k\},\{e_1,e_2,\cdots,e_k$

}] を v_0 から v_k に至る長さ k の順路と言い，v_0 および v_k をそれぞれ順路Rの始点および終点と言う．特に，(3-1)を満たすSが，

$$v_0 = v_k, \quad v_i \neq v_j \quad (1 \leq i < j \leq k) \qquad (3-3)$$

を満たすとき，Sのなす Gの部分グラフを長さ k のサイクルと言う．

［定義３．２］ グラフ $G=[V(G), E(G)]$ の任意の２頂点 x，y に対して，$[v_1=x, v_2, \cdots, v_k=y]$ なる頂点の系列が存在し，どの v_i および v_{i+1} ($1 \leq i \leq k-1$) も互いに隣接しているとき，このGは弱連結であると言う．また，Gの任意の２頂点 x，y に対して，x から y に至る順路および y から x に至る順路の両方が存在するとき，このGは強連結であるという．

［定義３．３］ グラフGの極大な強連結部分グラフのおのおのを，Gの強連結成分と言う．

［定義３．４］ グラフGの頂点の集合 $W \subset V(G)$ に対して，頂点集合 $V(H) \triangleq W$ および辺集合 $E(H) \triangleq E(G) \cap (W \times W)$ で定義されるGの部分グラフ $H \triangleq [V(H), E(H)]$ を，Gのセクション部分グラフと言い，$\langle W \rangle$ で表わす．

［定義３．５］ Gの強連結な非可分成分 $K=[V(K), E(K)]$ とは，どの頂点 $v \in V(K) \subset V(G)$ に対しても，Kのセクション部分グラフ $\langle V(K)-\{v\} \rangle$ が弱連結であるような，極大な強連結部分グラフである．

なお，以下では，順路 $R=[\{v_1, v_2, \cdots, v_k\}, \{(v_1, v_2), (v_2, v_3), \cdots, (v_{k-1}, v_k)\}]$ およびサイクル $C=[\{v_1, v_2, \cdots, v_k\}, \{(v_1, v_2), (v_2, v_3), \cdots, (v_{k-1}, v_k), (v_k, v_1)\}]$ をしばしば頂点の系列 $R=[v_1, v_2, \cdots, v_k]$ および $C=[v_1, v_2, \cdots, v_k, v_1]$ で表わす．また，頂点 v に対して，

$$Adj(v) \triangleq \{ w \mid (v, w) \in E(G) \} \qquad (3-4)$$

を，v に対する隣接リストと言い，グラフGは各頂点に対する隣接リストの集合で表現されているものとする．

さて，ここで，n 個の頂点を持つ完全対象グラフGに着目する．完全対象グラフとは，どの２頂点 v，w に対しても，v から出て w に入る辺および w から出て v に入る辺の両方を丁度１個ずつ持つ有向グラフであるから，$|E(G)| = n(n-1)$ であり，サイクルの総数は $\sum_{i=2}^{n} {}_nC_i \cdot (i-1)!$，すなわち $O(n!)$ である．

従って，第２章において述べたように，この列挙問題に対するアルゴリズムを解析する場合には，入力の規模ばかりではなく，出力の規模を示す変数をも用いる必要がある．

３．３ 従来のアルゴリズム

３．３．１ backtracking の技法を用いたアルゴリズム

backtrackingの技法を1言で表現すれば，「各段階で選択できるものを1つ選択し，次の段階へ進む．それを選択できるものがある限り繰り返し，選択できるものがなくなれば，1つ前の段階へ戻り他の選択を行う」という技法である．このように，backtrackingの技法それ自体は極めて単純なものであり，あまり効率的であるとは言えないが，それに適切な条件を付加し，無駄な選択をなくした「制限付きbacktracking技法」にすれば，極めて効率の良いアルゴリズムが見い出されることがある．サイクルの列挙問題に対するアルゴリズムは，その代表的な1つの例である．

この技法を初めてサイクルの列挙問題に適用したのはTiernan[16]であるが，彼のアルゴリズムはこのような制限付きの技法ではなく，制限のないback-tracking技法であった．

彼のアルゴリズムでは，各頂点を1からn($=|V(G)|$)までの正整数で表現し($V(G)=\{1,2,\cdots,n\}$)，初めに$s\leftarrow 1$として以下の操作を行う．

Ⅰ．$v_1>s$($2\leq i\leq k$)なる順路$P=[v_1=s, v_2, \cdots, v_k]$をbacktrackingの技法ですべて生成し，$s$を通り$s$より大きい番号の頂点から成るサイクルをすべて見い出す．

Ⅱ．$s<n$であれば，$s\leftarrow s+1$として操作Ⅰを繰り返し，$s\geq n$であれば，操作終了．

この操作Ⅰのbacktracking技法においては，ある段階で生成されている順路$P_k=[v_1, v_2, \cdots, v_k]$に対して，$Adj(v_k)$に含まれている頂点が順次探索され，その際，頂点$v_{k+1}\in Adj(v_k)$が，$v_{k+1}\notin V(P_k)$および$v_{k+1}>s$なる条件を満たせば，次の段階として順路$P_{k+1}=[v_1, v_2, \cdots, v_k, v_{k+1}]$が生成される．ただし，$v_{k+1}=s$である場合にはサイクルが見い出されたことになる．

しかし，backtrackingにおいてこのような条件だけでは，無駄な順路(結果として，その順路を含むサイクルが見い出せなかったもの)が数多く生成されるため，このアルゴリズムの時間複雑度は$O(2^{\alpha m}\cdot C+n+m)$となり[24]，制限付きのbacktracking技法であるとは言えない．ここで，αは$0<\alpha<1$なる定数，Cはすべてのサイクルの個数，mは辺の個数である．

このTiernan[16]のアルゴリズムと同じ時間複雑度を持つアルゴリズムとして，Weinblatt[17]のアルゴリズムがある．

Tarjan[18]は，Tiernanのアルゴリズムにmarking-unmarkingの技法を導入し，時間複雑度が$O(n\cdot m\cdot c+n+m)$のアルゴリズムを導いた．このmarking-unmarkingの技法とは，各頂点にマーク(mark)を付けたり取り除いたりするものであり，そのマークは次のことを示している．すなわち，「$P_k=[v_1, v_2, \cdots, v_k]$が生成されており，頂点$v_{k+1}$にマークが付けられているならば，$P_{k+1}=[v_1, v_2, \cdots, v_k, v_{k+1}]$を含むサイクルは存在しない」．従って，

Tarjan [18] のアルゴリズムでは，次の段階の頂点が満たすべき条件として，マークが付いていないという条件が新らたに付加されており，制限付きのback-tracking技法であるということができる．

この Tarjan [18] のアルゴリズムと同じ時間複雑度を持つアルゴリズムとしては，[19] あるいは [20] で提案されたアルゴリズムがある．

Johnson [22] は，Tiernan [16] のアルゴリズムに，Tarjan [18] の marking-unmarking 技法をさらに改良したblocking-unblocking の技法を導入し，$O((n+m)\cdot(c+1))$ の時間複雑度を持つアルゴリズムを導いた．彼のアルゴリズムと同じ時間複雑度を持つアルゴリズムとして，[21] および [23] で提案されたものがある．

Szwarcfiter-Lauer [21] のアルゴリズムは，ある段階で生成されている順路 $P_k=[v_1, v_2, \cdots, v_j, \cdots, v_k]$ に対して，ある頂点 v_j ($1<j<k$) が $v_j \in Adj(v_k)$ であったとき，Johnson [22] や Read-Tarjan [23] のアルゴリズムでは無視していたサイクル $[v_j, v_{j+1}, \cdots, v_k, v_j]$ をサイクルとして出力する点に特徴がある．

しかしながら，彼らの手法には，marking-unmarking の技法に誤りがあり，そのため，図3-1のようなグラフなどにおいてはすべてのサイクルが見い出されない．すなわち，このグラフにおいて，頂点1から探索が始まり，辺(1, 2)が最初に，次に辺(1, 3)が，最後に辺(1, 5)が探索されたとすれば，辺(1, 5)の探索の際，頂点5にはマークが付けられたままになっており，サイクル[1, 5, 3, 1]が見い出されない．

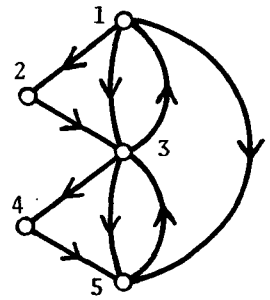


図3-1 [21] のアルゴリズムの誤りを示す例

しかし，この誤りも Johnson [22] の用いた blocking-unblocking の技法を適用して解消することが可能である．

一方，Read-Tarjan [23] のアルゴリズムの特徴は，「予見的な技法」を導入している点にある．「予見的な技法」というのは，Johnson の blocking-unblocking の技法が，それまでの探索によって得られた情報を頂点のマークという形で貯えているのに対し，順路の長さを1つ延長する際，その順路を

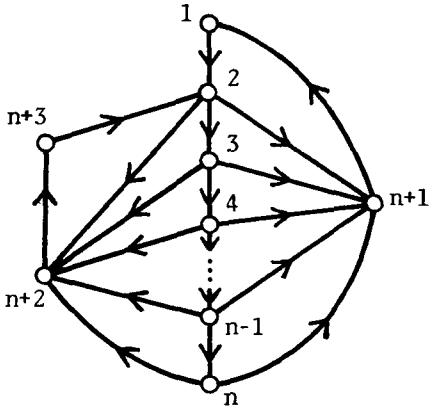


図3-2 [23] のアルゴリズムが効率的でない例

含むサイクルが2つ以上存在するか否かを予かじめ調べ，それによって無駄な順路の生成を減少させようという技法である．

しかし，隣接リストの探索回数の大小という観点から見れば，このアルゴリズムよりJohnson[22]のアルゴリズムの方がその回数は少く，効率が良い．そのような例の1つとして，図3-2のグラフを挙げることができる．このグラフを表わす各隣接リストの頂点は番号の小さいもの順に並んでいるものとすると，隣接リストの探索回数（辺の探索回数）は，Read-Tarjan[23]のアルゴリズムの方がJohnson[22]のアルゴリズムより $n(n+1)/2$ 回程度多い．

なお，これら[21]，[23]のアルゴリズムも，以下で述べる探索範囲の削減という点に関しては，ほとんど考察されておらず，改良する必要がある．

3．3．2 Johnson のアルゴリズム

ここでは，従来のアルゴリズムの中では最良のJohnson[22]のアルゴリズムについて概観し，blocking-unblocking 技法の紹介を行う．その後，このアルゴリズムを改良し，より効率の良いアルゴリズムを導くために，探索範囲の削減という点に注目する．

Johnson[22]のアルゴリズムも，各頂点を1からnまでの正整数で表わし，初めに $s \leftarrow 1$ として以下の操作を行う．

Ⅰ．与えられたグラフのセクション部分グラフ $<\{s, s+1, \cdots, n\}>$ において，$|V(K)| \geq 2$，かつ最小の番号の頂点を持つ強連結成分Kを求める．

Ⅱ．このようなKが存在しなければ，操作完了．そうでなければ，Kの中の最小の番号の頂点をsとし，sを通りKに含まれるすべてのサイクルを求める．この操作が終れば，$s \leftarrow s+1$ として操作Ⅰ，Ⅱを繰り返す．

この操作Ⅱにおいて，sを通るすべてのサイクルを見い出すために，blocking-unblocking 技法の導入された「制限付きbacktracking 技法」が用いられている．blocking-unblocking技法とは，marking-unmarking 技法と同様各頂点にマークを付けたり取り除いたりして無駄な探索の繰り返しを防ぐ技法であるが，それは次のような考察に基ずいている．

(i) 順路 $P_k=[v_1, v_2, \cdots, v_{k-1}, v_k]$ を含むサイクル $[v_1, v_2, \cdots, v_{k-1}, v_k, \cdots, v_1]$ が存在しないならば，順路 $[v_1, v_2, \cdots, v_{k-1}, \cdots, v_k]$ を含むサイクルも存在しない．

(ii) 順路 $P_k=[v_1, v_2, \cdots, v_{k-1}, v_k]$ を含むサイクルが存在するとき，順路 $[v_1, v_2, \cdots, v_{k-1}, v_k, \cdots, x]$ を含むサイクルは存在しないが，順路 $[v_1, v_2, \cdots, v_{k-1}, \cdots, x]$ を含むサイクルが存在する可能性がある頂点xは，順路 $P_{k-1}=[v_1, v_2, \cdots, v_{k-1}]$ と交差することなく，xから v_k に到達することのでき

る頂点である．例えば，図3-3に示す黒い丸印の頂点である．

blocking-unblocking 技法が Tarjan [18] の marking-unmarking 技法と異なる点は，この (ii) の考察であり，これによって，時間複雑度の改良が可能となった．

これらの考察に基ずき，blocking-unblocking技法では次のような操作で各頂点のマークを付けたり取り除いたりする．

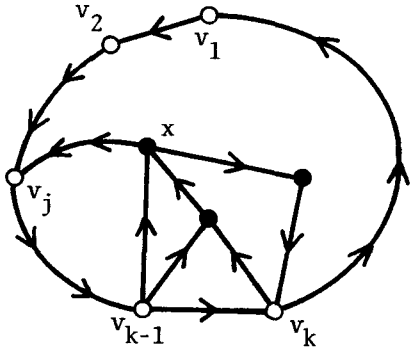


図3-3　blocking-unblocking 技法の説明図

0°. 最初はどの頂点にもマークを付けない．

1°. 各段階で生成される順路 $P_k=[v_1, v_2, \cdots, v_k]$ 上のすべての頂点にマークを付ける．（順路の長さが1つ延長されるたびに，延長された頂点にマークを付けてゆけばよい）．

2°. 順路 $P_k=[v_1, v_2, \cdots, v_{k-1}, v_k]$ が生成されている段階から1段階戻り，順路 $P_{k-1}=[v_1, v_2, \cdots, v_{k-1}]$ に関して探索を繰り返す際，P_k を含むサイクルが見い出されていなければ，v_k のマークはそのままにしておく．（頂点に付けられたマークは，次の3°によってのみ取り除かれる）．

3°. P_k の段階から P_{k-1} の段階に戻る際，P_k を含むサイクルが見い出されているならば，P_{k-1} と交差することなく v_k に到達することのできる頂点のマークを取り除く．

このような操作を backtracking の技法に付加することにより，無駄な探索の繰り返しが防がれ，サイクル1つ当りに要する演算時間が $O(n+m)$ となった．

そこで，次に，このアルゴリズムをさらに改良し，サイクル1つ当りに要する演算時間を，そのサイクルの長さのオーダにするための1つの接近法について考察しよう．

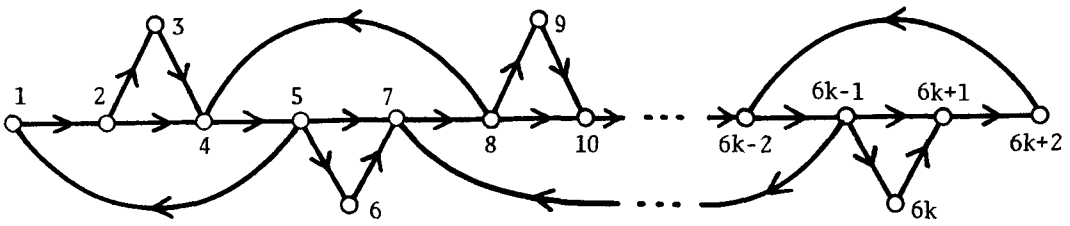


図3-4　[22] のアルゴリズムが効率的でない例

まず，図3-4に示されるようなグラフGを考えよう．このグラフにおいては，

$n=6k+2$, $m=10k+1$, $C=4k$であり，従来の$O((n+m)\cdot(C+1))$の時間複雑度を持つアルゴリズムが$O(k^2)$の演算時間を要した例の1つである．しかし，このグラフの各サイクルの長さの総和γは$18k$であるから，$O(k)$の演算時間ですべてのサイクルを見い出すアルゴリズムがあれば，その方が望ましい．ところで，このようなグラフにおいては，頂点$3i-2$（$i=1,2,\cdots,2k$）を通り$3i-2$より大きな番号の頂点から成るサイクルはすべてセクション部分グラフ$\langle\{3i-2, 3i-1, 3i, 3i+1, 3i+2\}\rangle$に含まれているから，すなわち，サイクルの存在している範囲が限られているから，何らかの方法で探索の範囲を制限してやれば，$O(k)$のアルゴリズムを導くことも可能であろう．従って，探索範囲の削減は，従来のアルゴリズムの時間複雑度を改良する1つの接近法であり，その操作が容易に行えるならば，必要な演算時間の短縮に寄与するであろう．

ところで，サイクルは必ず1つの強連結な非可分成分に含まれることに注意すれば，前述のJohnson[22]のアルゴリズムの操作Iにおいて，強連結成分Kを強連結な非可分成分とすることができる．このように変更することも，探索範囲を削減する1つの方法であり，サイクルを見い出す際の演算時間を短縮できるが，操作Iは最悪の場合$O(n)$回繰り返されるから，$O(n+m)$で強連結な非可分成分が見い出せたとしても[26]，この処理を行うためにはかなりの演算時間を必要とするであろう．さらに，このような変更だけでは，図3-4のグラフにおいて必要な探索の制限を行うことはできない．

そこで以下では，探索範囲の削限という観点から，次の性質を持つアルゴリズムを導こう．

(i) 繰り返し強連結成分（あるいは強連結な非可分成分）に分割するという操作を行なわなくとも，サイクルを見い出すために行なわれる探索は，1つの強連結な非可分成分内にある頂点の隣接リストに限られる．

(ii) 図3-4のようなグラフにおいて，頂点1を通るサイクルを見い出す際，頂点$v\in\{6,7,\cdots,6k+2\}$のような頂点（すなわち，1からvに至る順路とvから1に至る順路とは，必ず1，v以外の頂点を共有することが容易に解るような頂点）には探索が及ばない．

3．4　前処理操作

3．4．1　前処理アルゴリズム

与えられたグラフGのすべてのサイクルを列挙する際，探索すべき範囲を制限することができるように，次のような前処理アルゴリズムを実行する．

前処理アルゴリズムの第1の目的は，与えられたグラフGに対して，1つの外向森Tを生成することである．この外向森T=[V(T), E(T)]はDFS技法（depth-first search技法[26]）を用いて生成されたものであり，

(i) V(T) = V(G)，

(ii) Tはサイクルを持たない，

(iii) どの頂点v∈V(T)も，それに入る辺e∈E(T)を高々1つしか持たない，

という3つの性質を満たしている．

以下では，与えられたグラフGに対して，外向森Tが決定されているとき，このGを特にG_Tと書き，辺b∈E(T)および$l \in E(G_T) - E(T)$をそれぞれTの木枝およびリンクと呼ぶ．また，各頂点vに対して，次の集合を定義する．

$$\Gamma_T(v) \triangleq \{ w \mid (v, w) \in E(T) \} \qquad (3\text{-}5)$$

$$\Gamma_T^{-1}(v) \triangleq \{ w \mid v \in \Gamma_T(w) \} \qquad (3\text{-}6)$$

[定義3.6] 2頂点v, w∈V(G_T)に対して，v=wであるかあるいはvからwに至るT上の順路が存在するとき，vはwの上位にある，あるいはwはvの下位にあると言い，vの下位にある頂点および上位にある頂点の集合を，それぞれ$\hat{\Gamma}_T(v)$および$\hat{\Gamma}_T^{-1}(v)$によって表わす．

この関係を用いれば，Tのリンクを次の3つに類別することができる．

[定義3.7] G_Tのリンクlに対して，

(i) $s(l) \in \hat{\Gamma}_T(t(l))$であれば，lをBリンク（backward link）と言い，

(ii) $t(l) \in \hat{\Gamma}_T(s(l))$であれば，lをFリンク（forward link）と言い，

(iii) それ以外のとき，lをCリンク（cross link）と言う．

リンクはこのように3種に分類できるが，以下で述べるサイクル生成アルゴリズムにおいては，FリンクとCリンクを同等なものとして取り扱うことが可能となる．それゆえ，与えられたグラフGに対して，DFS技法を用いて外向森Tを生成する際には，辺を木枝・Bリンク・CまたはFリンクの3種類に分類しておくものとする．この分類が，前処理アルゴリズムの第2の目的である．

第3の目的は，各頂点v∈V(G)に対して，1からn≜|V(G)|までの番号N(v)を割り当てることである．この割り当て方は，隣接リストの頂点がすべて探索し終った最初の頂点に1，その次の頂点に2という順に，すなわち1段階戻るたびに番号を割り当てて行く．

以上の目的を達成する前処理アルゴリズム（手続きDIFOREST）を，図3-5にALGOL-likeな言語で記す．ここで，STATE(v)は，隣接リスト$Ad_j(v)$に含まれるどの頂点もまだ探索されていないとき0，そのどれかの頂点が調べられているとき1，すべての頂点が調べられたとき2の値を取る．また，$F_T(v)$は，$\Gamma_T^{-1}(v) \neq \phi$のとき，$\Gamma_T^{-1}(v)$の唯1つの元を示し，$\Gamma_T^{-1}(v) = \phi$のとき，

0の値を取るものとする。なお，スタックSORTには，各頂点がこの手続きで探索された順に入れられるが，これは後のサイクル生成アルゴリズムにおいて，Bリンクの選択順序決定に用いられる。

```
procedure  DIFOREST ;
     comment  This routine is a prepositive procedure for generating all
        the cycles of a digraph G represented by adjacency lists Adj(·).
        Integer n is a global variable denoting the number of vertices ;
 begin    integer array  STATE(n), N(n) ;    integer  i ;
          vertex array  F_T(n) ;    vertex stack  SORT(n) ;
        procedure  DFS(  most recently reached vertex  v  ) ;
          begin
              put v on stack SORT ;
              STATE(v) := 1 ;
              for   w ∈ Adj(v)   do
                 if   STATE(w) = 0   then
                   begin    comment  edge (v,w) is classified as a branch ;
                      F_T(w) := v ;
                      DFS( w )
                   end
                 else   if   STATE(w) = 1   then
                              edge (v,w) is classified as a B-link
                        else  edge (v,w) is classified as a C- or F-link ;
              STATE(v) := 2 ;
              i := i+1 ;
              N(v) := i
          end  DFS ;
     i := 0 ;
     empty stack SORT ;
     for   each vertex r   do
        if   STATE(r) = 0   then
          begin
             F_T(r) := 0 ;
             DFS( r )
          end
 end  DIFOREST ;
```

図3-5　前処理アルゴリズム

この手続きによって定められる外向森Tは，配列$F_T(\cdot)$によって一意的に定められ，かつすべての辺が3種類に正しく分類されていることは，容易に解るであろう[26]．さらに，この前処理アルゴリズムの時間およびスペース複雑度が共に$O(n+m)$であることも明らかであろう[26]．ただし，$m \triangleq |E(G)|$である．

3．4．2　外向森の性質

前処理アルゴリズムによって定められた頂点番号$N(\cdot)$およびBリンクに対して以下の補題が成立する．

［補題３．１］　グラフ G_T において，次の関係が成立する．

(i)　$v \in \hat{\Gamma}_T(w)$ ならば，$N(v) \leq N(w)$ である．

(ii)　$v \in \hat{\Gamma}_T(w)$ なる v，w に対して，$N(v) \leq N(u) \leq N(w)$ であるような頂点 u が存在すれば，$u \in \hat{\Gamma}_T(w)$ である．

（証明略）

［補題３．２］　グラフ G_T において，辺 $e=(v,w)$ がBリンクであるための必要十分条件は，$N(v) < N(w)$ である．

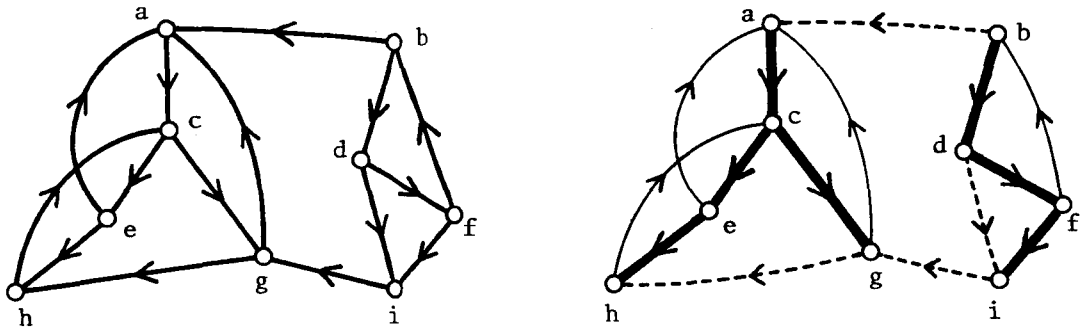
（証明略）

［補題３．３］[26]　グラフ G_T において，$N(v) < N(w)$ なる２頂点 v，w に対して，v から w に至る順路が存在するならば，その順路上には $t(l) \in \hat{\Gamma}_T^{-1}(v) \cap \hat{\Gamma}_T^{-1}(w)$ なるBリンク l が少くなくとも１つ存在する．

（証明略）

これらの補題の証明は，前処理アルゴリズムの記述より容易に行うことができる．

〔例３．１〕　図3-6 (a)のグラフ G に対して，前処理アルゴリズムを適用すれば，同図(b)のグラフ G_T が得られ，各頂点 v の $F_T(v)$ および $N(v)$ は表3-1のように与えられる．この G_T においても，上記３つの補題が成立していることは容易に確められるであろう．



(a)　グラフ G　　(b)　グラフ G_T

木枝　　Bリンク

Cリンクまたは Fリンク

図３－６　前処理アルゴリズムの適用例

表3-1　v および $F_T(v)$, $N(v)$

頂点 v	a	b	c	d	e	f	g	h	i
$F_T(v)$	o	o	a	b	c	d	c	e	f
$N(v)$	5	9	4	8	2	7	3	1	6

3.5 基本定理

ここでは，以下で提案するアルゴリズムの基本となる方針について述べる．初めに，次の補題を示す．

［補題3.4］　グラフ G_T の任意のサイクル C は，C 上のどの頂点 $v \in V(C)$ も $t(l)$ の下位にあるようなBリンク l を含む．

（証明）　C 上の辺 (v, w) が $N(v) < N(w)$ を満たせば，これは補題3.2よりBリンクであるから，C はBリンクを持つ．また，$N(v) > N(w)$ であれば，補題3.3より v から w に至る C 上の順路には必ずBリンクが含まれる．そこで，C 上のBリンクの中で，最大の終点番号 $N(t(l_1))$ を持つBリンクを l_1 とする．この l_1 が補題の条件を満たすことを示すために，ある頂点 $v \in V(C)$ が $t(l_1)$ の下位になかったと仮定する．このとき，明らかに $t(l_1)$ から v に至る C 上の順路には，次のようなリンク l_k が存在する．

$$o(l_k) \in \hat{\Gamma}_T(t(l_1)) \qquad (3\text{-}7)$$

$$t(l_k) \notin \hat{\Gamma}_T(t(l_1)) \qquad (3\text{-}8)$$

(i)　$N(o(l_k)) < N(t(l_k))$ の場合には，l_k はBリンクであり $o(l_k) \in \hat{\Gamma}_T(t(l_k))$ であるから，(3-8)より $t(l_k)$ は $t(l_1)$ の上位になければならない．すなわち，$t(l_k) \in \hat{\Gamma}_T^{-1}(t(l_1)) - \{t(l_1)\}$ であるから，$N(t(l_k)) > N(t(l_1))$ となり，l_1 の選び方に矛盾する．

(ii)　$N(o(l_k)) > N(t(l_k))$ の場合には，$N(t(l_1)) > N(o(l_k))$ より，$N(t(l_1)) > N(t(l_k))$ である．従って，補題3.3より $t(l_k)$ から $t(l_1)$ に至る C 上の順路には，$\hat{\Gamma}_T^{-1}(t(l_k)) \cap \hat{\Gamma}_T^{-1}(t(l_1))$ に終点を持つBリンク l_h が存在する．それゆえ，$t(l_k) \notin \hat{\Gamma}_T(t(l_1))$ より $t(l_h) \in \hat{\Gamma}_T^{-1}(t(l_1)) - \{t(l_1)\}$ となり，$N(t(l_h)) > N(t(l_1))$ となるから，この場合にも l_1 の選び方に矛盾する．　（証明終）

この補題により，G_T の1つのBリンク l を通りセクション部分グラフ $\langle \hat{\Gamma}_T(t(l)) \rangle$ に含まれるサイクルをすべて見い出すという操作を，すべてのBリンクについて繰り返し行えば，G_T のすべてのサイクルを見い出すことができ，

これらのサイクルの間に重複はない．ところで，あるBリンクlを通り$<\hat{\Gamma}_T^A(t(l))>$に含まれるすべてのサイクルを見い出すためには，$t(l)$から$o(l)$に至る$<\hat{\Gamma}_T(t(l))>$内のすべての順路を見い出せばよいが，これを行うために，本アルゴリズムは$o(l)$から始めて，辺の方向（矢印）を逆にたどるback-trackingの技法を用いる．すなわち，ある段階で生成されている順路$R_k=[x_k, x_{k-1}, \cdots, x_2=o(l), x_1=t(l)]$に対して，辺$(x_{k+1}, x_k)$が探索される．以下では，この順路$R_k$を拡張して順路$R_{k+1}=[x_{k+1}, x_k, x_{k-1}, \cdots, x_2=o(l), x_1=t(l)]$を生成し，$x_{k+1}$に入る辺を探索する必要があるのは，$x_{k+1}$がどのような条件を満足する場合かについて考察する．

辺(x_{k+1}, x_k)がCまたはFリンクである場合には，次の補題は補題3.1より明らかであろう．

［補題3．5］ ある段階での順路$R_k=[x_k, x_{k-1}, \cdots, x_2=o(l), x_1=t(l)]$に対して，辺$(x_{k+1}, x_k)$がCまたはFリンクであり，かつ$N(x_{k+1})>N(x_1)$であれば，$R_k$と辺$(x_{k+1}, x_k)$とから成る順路$R_{k+1}$は，$l$を通り$<\hat{\Gamma}_T(t(l))>$に含まれるどのサイクルにも含まれない．

（証明略）

次に，辺(x_{k+1}, x_k)がBリンクの場合について考察する．このとき，$x_1=t(l)$からx_{k+1}に至る$<\hat{\Gamma}_T(x_1)>$内の順路が必ずx_kを通るならば，R_kと(x_{k+1}, x_k)とから成る順路R_{k+1}が，lを通り$<\hat{\Gamma}_T(x_1)>$に含まれるサイクルに含まれることはない．従って，この事実をうまく利用すれば，探索範囲の削減に役立たせることができるであろう．

そこで，各Bリンクlに対して，次のような値を割り当てる．

$$LOWN(l) \triangleq \begin{cases} \min_{R \in \tilde{\mathcal{R}}(l)} [\, N(s) \mid s \text{ は } R \text{ の始点} \,] & : \tilde{\mathcal{R}}(l) \neq \phi \text{ のとき} \\ \infty & : \text{それ以外のとき} \end{cases} \quad (3-9)$$

ただし，$\tilde{\mathcal{R}}(l)$は始点sからBリンクlに至る順路$R=[x_k=s, x_{k-1}, \cdots, x_2=o(l), x_1=t(l)]$のうちで，$s=x_k$を$\hat{\Gamma}_T(t(l))$かつ$x_i \in \hat{\Gamma}_T(t(l))$（$1 \leq i \leq k-1$）を満たすものすべての集合である．外向森Tの作り方から明らかなように，$R \in \tilde{\mathcal{R}}(l)$上のsから出て行く辺$(x_k=s, x_{k-1})$はCまたはFリンクである．

この$LOWN(\cdot)$の値を用いれば，次の補題が成立する．

［補題3．6］ ある段階での順路$R_k=[x_k, x_{k-1}, \cdots, x_2=o(l), x_1=t(l)]$に対して，辺$(x_{k+1}, x_k)=l_b$がBリンクであり，かつ$N(x_1)<LOWN(l_b)$であれば，$R_k$とBリンク$l_b$とから成る順路$R_{k+1}$は，$l$を通り$<\hat{\Gamma}_T(t(l))>$内にあるどのサイクルにも含まれない．

（証明） l を通り $<\hat{\Gamma}_T(t(l))>$ に含まれる任意のサイクル C に注目したとき，C に含まれる l 以外のどの B リンク l_b に対しても，$LOWN(l_b) \leq N(t(l))$ が成立することを示せば十分である．そこで，$t(l)$ から $o(l_b)$ に至る C 上の順路を $t(l)$ から順にたどる過程で，$o(l_C)$ を $\hat{\Gamma}_T(t(l_b))$ および $t(l_C) \in \hat{\Gamma}_T(t(l_b))$ を満たす最後のリンクを l_C とすれば，$o(l_C)$ から $t(l_b)$ に至る C との順路は明らかに $\tilde{\mathcal{R}}(l_b)$ の元となる．それゆえ，$o(l_C) \in \hat{\Gamma}_T(t(l))$（補題3.4より），および補題3.1より $N(o(l_C)) \leq N(t(l))$ となり，$LOWN(l_b) \leq N(t(l))$ が成立する．（証明終）

以上の補題に基ずいて，B リンク l を通り $<\hat{\Gamma}_T(t(l))>$ 内に含まれるサイクルを見い出すための探索手順として，次のようなものが考えられる．

0°. 現段階において生成されている順路を $R_k = [x_k, x_{k-1}, \cdots, x_2 = o(l), x_1 = t(l)]$ とする．

1°. 辺 (x_{k+1}, x_k) が C または F リンクであれば，$N(x_{k+1}) \leq N(x_1)$ のときのみ順路 $R_{k+1} = [x_{k+1}, x_k, \cdots, x_1]$ を生成する．

2°. 辺 (x_{k+1}, x_k) が B リンクであれば，$LOWN((x_{k+1}, x_k)) \leq N(x_1)$ のときのみ順路 R_{k+1} を生成する．

3°. 辺 (x_{k+1}, x_k) が木枝であれば，順路 R_{k+1} を生成する．

この手順を backtracking 技法によって繰り返せば，l を通り $<\hat{\Gamma}_T(t(l))>$ に含まれるサイクルすべてを生成することができる．さらに，この手順に，Johnson[22] の blocking-unblocking 技法を導入し，x_{k+1} にマークが付いていないときのみ順路 R_{k+1} を生成するというように変更する．それによって，この手順に従ったアルゴリズムの時間複雑度を $O((n+m)\cdot(c+1))$ にすることができる．

3．6 B リンクに対する考察

補題3.5に示された C または F リンクに対する条件によって，探索の範囲は $<\hat{\Gamma}_T(t(l))>$ 内に制限されたわけであるが，補題3.6で示された B リンクに対する条件によって，探索の範囲はどのように制限されたかを考察しよう．

B リンク l を通り $<\hat{\Gamma}_T(t(l))>$ に含まれるサイクルを見い出している際，$\hat{\Gamma}_T(t(l)) - \{t(l)\}$ に終点を持つ B リンク l_b に対して，$LOWN(l_b) > N(t(l))$ が成立していたとすれば，$t(l)$ から $t(l_b)$ の下位にある頂点 v に至る $<\hat{\Gamma}_T(t(l))>$ 内の順路は必ず $t(l_b)$ を通る．

そこで，今，前節で述べた手順に従って，l を通り $<\hat{\Gamma}_T(t(l))>$ に含まれるサイクルを見い出す際，$<\hat{\Gamma}_T(t(l))>$ 内の l を含む強連結な非可分成分 K 以外の頂点 $v \in \hat{\Gamma}_T(t(l))$ に探索が至ったとする．そうすれば，そのときの順路 R_k

$=[x_k=v, x_{k-1}, \cdots, x_2=o(l), x_1=t(l)]$ の中には，v を含む強連結な非可分成分と K との両方に含まれる頂点 x_j が必ず存在する．そこで，x_j に入る R_k 上の辺を (x_{j+1}, x_j) とすれば，$t(l)$ から x_{j+1} に至る $<\hat{\Gamma}_T(t(l))>$ 内の順路はすべて x_j を通ることから，この (x_{j+1}, x_j) はBリンクとなる．それゆえ，x_{j+1} と $t(l)$ とが同一の強連結な非可分成分に含まれないことから，$LOWN((x_{j+1}, x_j)) > N(t(l))$ となり，前節の手順に従ったという仮定に矛盾する．従って，$LOWN(\cdot)$ の値を用いることにより，探索の範囲は1つの強連結な非可分成分内に制限される．

さらに，$v \in \hat{\Gamma}_T(t(l))$ から $t(l)$ に至る $<\hat{\Gamma}_T(t(l))>$ 内の順路は必ずBリンク l_b を通り，かつ $t(l)$ から v に至る $<\hat{\Gamma}_T(t(l))>$ 内の順路は必ず $t(l_b)$ を通るような場合には，Bリンク l_b を探索すべき順路の中に加える必要はないが，これも $LOWN(l_b)$ の値を用いることによって，容易に探索から取り除くことができる．従って，図3-4に示したようなグラフに対しても，探索範囲を適切に制限することが可能となった．

ところで，探索範囲の削減にこの $LOWN(\cdot)$ の値を用いるためには，あるBリンク l を通りかつ $<\hat{\Gamma}_T(t(l))>$ に含まれるサイクルを見い出す操作を行う前に，少なくとも $\hat{\Gamma}_T(t(l)) - \{t(l)\}$ 内に終点を持つBリンク l_b の $LOWN(l_b)$ の値を求めておく必要がある．

そこで，以下に示すような規準ですべてのBリンクをソート（sort）し，その結果に従ってBリンク l を選び，その l を通り $<\hat{\Gamma}_T(t(l))>$ に含まれるすべてのサイクルを見い出す過程で，同時にこの l の $LOWN(l)$ の値を決定して行けば，$LOWN(l)$ の値が必要となるまでにその値を知ることができる．

≪Bリンクの sorting の規準≫

2つのBリンク l_i，l_j に対して，$t(l_j) \in \hat{\Gamma}_T(t(l_i)) - \{t(l_i)\}$ であるときに限り，l_j は l_i より優先する．

この規準によれば，同じ終点を持つBリンクはどちらが先でもよい．それゆえ，この規準を満たす1つの順序付けは，手続きDIFOREST において作られたスタック SORT を用いて行うことができる．すなわち，スタック SORT か

表3-2　Bリンクの選択順序とスタック SORT

選択順序	1	2	3	4
Bリンク	(f, b)	(h, c)	(e, a)	(g, a)

スタック SORT

a	c	e	h	g	b	d	f	i

スタックの先頭 ↑

る順に頂点を取り出し，取り出した順にその頂点を終点とするBリンクを順序付ける．ただし，その取り出された頂点を終点とするBリンクがいくつかある場合には，それらの間の順序は任意に与える．このようにして得られたBリンクの順序付けは前記の規準を満している．

〔例3．2〕 例3.1のグラフ G_T に対しては，表3-2のような順序が与えられる．

3．7 アルゴリズム

図3-7に，前節までの考察に基ずいた，G_T のすべてのサイクルを生成するアルゴリズムを，ALGOL-like な言語を用いて記す．ここで，与えられたグラフ G_T は，各頂点ごとに入ってくるCまたはFリンクのリスト $C^-(v)$，Bリンクのリスト $B^-(v)$，および $\Gamma_T^{-1}(v)$ の元 $F_T(v)$ を与えることによって表現されているものとする．これらのリストは，前処理アルゴリズムの後に，与えられたグラフ G_T の各隣接リスト $Adj(\cdot)$ から，$O(n+m)$ の演算時間で（各隣接リストを1度探索するだけで）作ることができる．また，各頂点ごとに割り当てられた論理変数 marked(v) は，vがマークされているときtrue，それ以外のときfalseの値を取るものであり，さらに，各vに割り当てられたリスト $M^+(v)$ には，vから出て行く辺の終点が入れられ，頂点のマークを取り除く際に用いられる．なお，スタック SORT は手続き DIFOREST において作成されたものであり，スタック RS には各段階で生成されている順路上の頂点系列が入れられる．

```
procedure  CYCLE ;
     comment  This routine generates all the cycles of a digraph G_T represented
         by array F_T(·) and list arrays C⁻(·) and B⁻(·).  Integers n and m are
         global variables denoting the numbers of vertices and edges, respec-
         tively.  Both vertex stack SORT and integer array N(·) are the outputs
         of the prepositive procedure DIFOREST ;
 begin    list array  M⁺(n) ;    integer array  LOWN(m) ;    B-link  ℓ_B ;
          logical  flag ;    logical array  marked(n) ;    start vertex  s ;
          vertex stack  RS(n) ;
        procedure  BACKTRACK(  most recently reached vertex  v,
                               logical result  f  ) ;
          begin    logical  g ;
                 procedure  UNMARK(  marked vertex  u  ) ;
                   begin
                      marked(u) := false ;
                      for   w ∈ M⁺(u)   do   begin
                          delete w from M⁺(u) ;
                          if   marked(w) = true   then   UNMARK( w )
                      end
                   end  UNMARK ;
```

```
            f := false ;
            put v on stack RS ;
            marked(v) := true ;
            for   (w,v) ∈ C⁻(v)   do   comment  investigation for a C- or F-link ;
                 if   w = s   then   begin
                      output cycle designated by stack RS ;
                      f := true
                 end
                 else   if   N(w) > N(s)   then
                      LOWN(ℓB) := MIN[ LOWN(ℓB), N(w) ]
                 else   if   marked(w) = false   then   begin
                      BACKTRACK( w, g ) ;
                      if   g = true   then   f := true
                 end ;
            for   (w,v) ∈ B⁻(v)   do   comment  investigation for a B-link ;
                 if   LOWN((w,v)) > N(s)   then
                      LOWN(ℓB) := MIN[ LOWN(ℓB), LOWN((w,v)) ]
                 else   if   marked(w) = false   then   begin
                      BACKTRACK( w, g ) ;
                      if   g = true   then   f := true
                 end ;
            w := FT(v) ;              comment  investigation for a branch ;
                 if   w = s   then   begin
                      output cycle designated by stack RS ;
                      f := true
                 end
                 else   if   marked(w) = false   then   begin
                      BACKTRACK( w, g ) ;
                      if   g = true   then   f := true
                 end ;
            if   f = true   then   UNMARK( v )
            else   begin
               for   (w,v) ∈ C⁻(v)   do
                    if   N(w) ≤ N(s) and v ∉ M⁺(w)   then   put v on M⁺(w) ;
               for   (w,v) ∈ B⁻(v)   do
                    if   LOWN((w,v)) ≤ N(s) and v ∉ M⁺(w)   then   put v on M⁺(w) ;
               w := FT(v) ;
                    if   v ∉ M⁺(w)   then   put v on M⁺(w)
            end ;
            delete v from stack RS
         end  BACKTRACK ;
   for   each vertex v   do   begin
         marked(v) := false ;
         M⁺(v) := φ
   end
   while   stack SORT is not empty   do   begin
         delete s from stack SORT ;
         if   B⁻(s) ≠ φ   then
              for   ℓB = (v,s) ∈ B⁻(s)   do   begin
                   LOWN(ℓB) := ∞ ;
                   put s on stack RS ;
                   BACKTRACK( w, flag ) ;
                   delete s from stack RS
              end
   end
end  CYCLE ;
```

図3-7　サイクル生成アルゴリズム

このサイクル生成アルゴリズムの正しさを証明する以下の補題が成立する．

[補題３．７][22]　このアルゴリズムのある実行段階において，スタックRSに順路$R_k=[x_k, x_{k-1}, \cdots, x_2=o(l), x_1=t(l)]$が入っているとする．このとき，与えられたグラフ$G_T$に$l$を通り$<\hat{\Gamma}_T(t(l))>$に含まれるサイクル$C=[x_1, \cdots, x_{k+1}, x_k, x_{k-1}, \cdots, x_2, x_1]$が存在するならば，頂点$x_{k+1}$にはマークは付けられていない．すなわち，$marked(x_{k+1})=false$である．さらに，スタックRSから$x_2=o(l)$が取り除かれ，$l$を通り$<\hat{\Gamma}_T(t(l))>$に含まれるサイクルの探索が終了した時点では，すべての頂点にマークは付いていない．

（証明略）　[22]と同様に，kに関する帰納法により容易に証明できる．なお，補題の後半は，lを通り$<\hat{\Gamma}_T(t(l))>$内にあるサイクルは少なくとも１つ存在することに気が付けば，容易に証明できる．

[補題３．８]　このアルゴリズムによって，各Bリンクの$LOWN$の値は正しく決定される．

（証明）　Bリンクの選択順序iに関する帰納法で証明する．$i=1$の場合は，$i=1$のBリンクl_1の終点$t(l_1)$の下位（$\hat{\Gamma}_T(t(l_1))-\{t(l_1)\}$）に終点を持つBリンクがないことから，次の$i=k$の場合と同様な証明が可能である．そこで，$i=k-1$までのBリンクの$LOWN$の値はすべて正しく決定されているものとし，$i=k$のBリンク$l_k$に注目する．すなわち，$l_k$を通り$<\hat{\Gamma}_T(t(l_k))>$に含まれるサイクルを見い出す際，$\hat{\Gamma}_T(t(l_k))-\{t(l_k)\}$に終点を持つBリンクの$LOWN$の値はすべて正しく決定されているものとする．今，$LOWN(l_k)$の値を与える順路$R_k\in\tilde{\mathcal{R}}(l_k)$を$R_k=[x_k, x_{k-1}, \cdots, x_2=o(l_k), x_1=t(l_k)]$とし，この$R_k$上の頂点で，かつ$l_k$を通るサイクルの探索過程で少なくとも１度はスタックRSに挿入されたことのある頂点のうち，R_k上で最もx_kに近い頂点をx_j（$k>j\geq 2$）とする．このx_jに入るR_k上の辺(x_{j+1}, x_j)は，l_kを通り$<\hat{\Gamma}_T(t(l_k))>$内に含まれるサイクルの探索過程で必ず調べられるが，このときx_{j+1}にマークが付いているとすれば，補題3.7よりl_kを通るサイクルの探索過程でx_{j+1}が１度はスタックRSに挿入されたことになり，x_jの選び方に矛盾する．従って，辺(x_{j+1}, x_j)が調べられる際には，x_{j+1}にマークは付いていない．ところで，この(x_{j+1}, x_j)が木枝であれば，x_{j+1}にマークが付いていないことから，x_{j+1}は必ずスタックRSに挿入されるから，(x_{j+1}, x_j)はCまたはFリンクであるか，あるいはBリンクである．そこで，CまたはFリンクであるとすれば，x_{j+1}がスタックRSに挿入されなかったことより，$N(x_{j+1})>N(x_1)$となり，$x_{j+1}=x_k$となるから，アルゴリズムによって$LOWN(l_k)$の値は正しく決定されることが解る．一方，Bリンクである場合には，順路$[x_k, x_{k-1}, \cdots, x_{j+1}, x_j]$

は，$r>j$ なる頂点 $x_r \in \bar{V}(R_k)$ がスタック RS に挿入されなかったことより，$LOWN((x_{j+1}, x_j))$ の値を定める順路でなければならず，この場合もアルゴリズムの操作により，$LOWN(l_h)$ の値は正しく決定されることが解る．

（証明終）

以上の補題3.4－3.8により，次の定理が成立することは明らかであろう．

［定理３．１］ このアルゴリズムは，与えられたグラフ G_T のすべてのサイクルを重複なく見い出す．

（証明略） アルゴリズムで生成される順路の長さの帰納法で証明できる．

３．８ 複雑度解析および計算結果

この章で提案したアルゴリズムは３つの部分から成っている．第１は手続き DIFOREST であり，第２は各頂点 v に入ってくる C または F リンクのリスト $C^-(v)$ および B リンクのリスト $B^-(v)$ を作る操作であり，第３は手続き CYCLE である．第１および第２の部分では，与えられたグラフの隣接リスト $Adj(\cdot)$ は丁度１回ずつ探索されるだけであるから，それに必要な演算時間は第３の部分に比べて短かく効率の良いものである．さらに，このアルゴリズム全体に対して，次の定理が成り立つ．

［定理４．２］ このアルゴリズムの時間およびスペース複雑度は，それぞれ $O((n+m)\cdot(c+1))$ および $O(n+m)$ である．ここで，n，m，および c は，それぞれグラフ G の頂点，辺，およびサイクルの個数である．

（証明） スペース複雑度が $O(n+m)$ で与えられることは，手続き DIFOREST および手続き CYCLE における再帰呼び出しの深さが高々 n であること，および各リスト配列に要するメモリスペースが $O(n+m)$ であることに気が付けば，容易に確められる．そこで，時間複雑度について考察する．手続き DIFOREST および リスト配列 $C^-(\cdot)$，$B^-(\cdot)$ を作成するのに要する演算時間，なびに手続き CYCLE において最初のサイクルが見い出されるまでに要する演算時間，さらに入力に要する演算時間は，それぞれ高々 $O(n+m)$ である．そこで，あるサイクルが見い出された後，次のサイクルが見い出されるまでに要する演算時間を考える．今，あるサイクルが見い出された後，そのサイクル上の１つの頂点 x がスタック RS から取り除かれる際，ある頂点 y のマークが取り除かれたとする．そうすると，この y に再びマークが付けられ，その後，再びそのマークが取り除かれるまでには，必ず新しいサイクルが見い出されていなければならない．そのため，今考えている時間内に同じ頂点が２度以上スタック RS に挿入されることはない．従って，あるサイクルが見い出された後，

次のサイクルが見い出されるまでに要する演算時間は高々$O(n+m)$である．なお，頂点w（$\langle w,v\rangle \in C^-(v)$，$\langle w,v\rangle \in B^-(v)$，あるいは$w=F_T(v)$）が$M^+(v)$に入っているか否かの判定は，各リスト$C^-(\cdot)$，$B^-(\cdot)$，および$F_T(\cdot)$に付随して，$w$が$M^+(v)$に入っているか否かを示す配列を用意しておけば，$O(1)$の演算時間で判定でき，この配列に必要なメモリスペースも$O(n+m)$でおさえられる．従って，このアルゴリズムの時間複雑度は$O((n+m)\cdot(c+1))$である．（証明終）

さらに，3.6節における説明から明らかなように，スタックRSに挿入される頂点は，現在調べているBリンクlに関して，$\langle \overset{A}{F}_T(t(l))\rangle$内の$l$を含む強連結な非可分成分の頂点に限られる．

また，サイクルの存在範囲がグラフ全体に比べると小さな部分グラフに限られており，その1つの部分グラフの頂点vからv以外の頂点wに至る順路と，wからvに至る順路とが必ずv，w以外の頂点（Bリンクの終点）と交差するようなグラフ（その代表的な1例が図3-4に示すグラフである）においては，本アルゴリズムは従来のアルゴリズムよりはるかに効率的なものとなる．その結果，図3-4に示すグラフにおいて，本アルゴリズムの必要とする演算時間は高々$O(\alpha+n+m)$となる．ここで，αは各サイクルの長さの総和である．しかしながら，この特徴のため，本アルゴリズムを無向グラフのすべてのサイクルを見い出す問題にそのまま適用することはできない．

さて，次に本アルゴリズムの実験結果を示す前に，手続きCYCLEに小さな変更を加える．この変更は，サイクルの探索過程でその範囲を制限するのではなく，探索に入る前に予じめ制限しておくものであり，その操作はDFS技法（手続きLISTDFS）で行なわれる．そのアルゴリズム（手続きCYCLE#2）を図3-8にALGOL-likeな言語で示す．ここで，各頂点vに与えられたリスト$Adj^-(v)$には，vに入ってくる辺の始点のうち，サイクルを形成するのに寄与するもののみが挿入され，$state(v)=true$であれば，そのようなリスト$Adj^-(v)$はすでに作成されたことを示す．また，$LNT(\cdot)$は$LOWN$の値を決定するために用いられる配列である．

これまでに手続きCYCLEに対して行った理論的評価は，手続きCYCLE#2に対しても成立すること，および手続きCYCLE#2が正しく完了することは，手続きLISTDFSの構造と，手続きLISTDFS(v)が呼び出された頂点に対しては必ず手続きBACKTRAK(v)が呼び出されていることに注意すれば，容易に確めることができるであろう．

この手続きCYCLE#2への変更によって，サイクルの個数が頂点の個数に比べて極めて多い完全対称グラフなどの場合には，サイクルの形成に寄与しない辺の探索回数を減少させることができ，実際に計算機で実行した際に要する

CPU時間を，10％程度短縮させることができる．

```
procedure  CYCLE #2 ;
     comment  This routine generates all the cycles of a digraph G_T represented
         by array F_T(·) and list arrays C⁻(·) and B⁻(·).  Integers n and m are
         global variables denoting the numbers of vertices and edges, respec-
         tively.  Both vertex stack SORT and integer array N(·) are the outputs
         of the prepositive procedure DIFOREST ;
 begin    list array  Adj⁻(n), M⁺(n) ;    integer array  LNV(n), LOWN(m) ;
          vertex stack  RS(n) ;    start vertex  s ;    B-link  ℓ_B ;
          logical  flag ;    logical array  marked(n), state(n) ;
       procedure  BACKTRACK(  most recently reached vertex  v,
                              logical result  f  ) ;
          begin    logical  g ;
                   procedure  UNMARK(  marked vertex  u  ) ;
                     begin
                         marked(u) := false ;
                         for   w ∈ M⁺(u)   do   begin
                             delete w from M⁺(u) ;
                             if   marked(w) = true   then   UNMARK( w )
                         end
                     end  UNMARK ;
             state(v) := false ;
             f := false ;
             put v on stack RS ;
             marked(v) := true  ;
             LOWN(ℓ_B) := MIN[ LOWN(ℓ_B), LNV(v) ] ;
             for   w ∈ Adj⁻(v)   do
                   if   w = s   then   begin
                            output cycle designated by stack RS ;
                            f := true
                   end
                   else   if   marked(w) = false   then   begin
                            BACKTRACK( w, g ) ;
                            if   g = true   then   f := true
                   end ;
             if   f = true   then   UNMARK( v )
             else   for   w ∈ Adj⁻(v)   do
                            if   v ∉ M⁺(w)   then   put v on M⁺(w) ;
             delete v from stack RS
          end  BACKTRACK ;
       procedure  LISTDFS( most recently reached vertex  v  ) ;
          begin
             Adj⁻(v) := φ ;
             state(v) := true ;
             LNV(v) := ∞ ;
             for   (w,v) ∈ C⁻(v)   do
                 if   N(w) ≤ N(s)   then   begin
                        put w on Adj⁻(v) ;
                        if   state(w) = false   then   LISTDFS( w )
                 end
                 else   LNV(v) := MIN[ LNV(v), N(w) ] ;
             for   (w,v) ∈ B⁻(v)   do
                 if   LOWN((w,v)) ≤ N(s)   then   begin
                        put w on Adj⁻(v) ;
                        if   state(w) = false   then   LISTDFS( w ) ;
                 end
                 else   LNV(v) := MIN[ LNV(v), LOWN((w,v)) ] ;
```

```
            w := F_T(v) ;
                if   N(w) ≤ N(s)   then   begin
                        put w on Adj⁻(v) ;
                        if   state(w) = false   then   LISTDFS( w )
                end
        end  LISTDFS ;
    for   each vertex  v   do   begin
        marked(v) := false ;
        state(v) := false ;
        M⁺(v) := φ
    end ;
    while   stack SORT is not empy   do   begin
        delete s from stack SORT ;
        if   B⁻(s) ≠ φ   then   begin
            for   ℓ_B = (v,s) ∈ B⁻(s)   do
                if   state(v) = false   then   LISTDFS( v ) ;
            for   ℓ_B = (v,s) ∈ B⁻(s)   do   begin
                LOWN(ℓ_B) := ∞ ;
                put s on stack RS ;
                BACKTRACK( v, flag ) ;
                delete s from stack RS
            end
        end
    end
end  CYCLE #2 ;
```

図3-8　手続きCYCLE＃2

この章で提案したアルゴリズムの第3の部分にこの手続きCYCLE＃2を用いたものを，FORTRANを用いてプログラムし，NEAC 2200/700 でいくつかのグラフについて実験を行った．

そのうち，完全対称グラフに対する結果を表3-3に示す．ここで，$n=5$ のグラフに対するサイクル1つ当りのCPU時間が $n=6$ のグラフに対するものよりも大きくなっているのは，手続きCYCLE＃2に要する演算時間に比べて，前処理に要する演算時間が比較的大きいことによる．しかし，頂点の個数が多くなりサイクルの個数が増加するに従って，前処理に要する演算時間の割合は極めて小さいものとなる．

表3-3　完全対称グラフに対する実験結果

n	5	6	7	8	9
number of cycles	84	409	2365	16064	125664
CPU time per cycle (μsec)	286.	281.	301.	331.	364.

また，ランダムに生成したグラフに対する実験結果を表3-4に示す．これらは，nおよびξ≜m/n(n-1)の各値に対して10個のグラフをランダムに生成し，それらの結果の平均を示したものである．これらの結果では，サイクルの個数cが300以上のグラフに対するCPU時間/γの値は，各サイクルの長さの総和γの大きさにかかわらず，表3-4に示す平均とほとんど同じであり，その差は高々±8μsecであった．

表3-4　ランダムに生成したグラフに対する実験結果

n	ξ≜m/n(n-1)	CPU time/c (μsec)	CPU time/γ (μsec)	CPU time/γ for graphs with c > 300. (μsec)
10	0.75	432.	52.	52.
10	0.50	510.	76.	70.
15	0.25	689.	77.	69.

3.9 結言

この章では，有向グラフのすべてのサイクルを列挙する新しいアルゴリズムを提案した．有向グラフのサイクルの列挙問題に対しては，backtrackingの技法が効果的な役割を果たし，これまでに時間複雑度が$O((n+m)\cdot(c+1))$のアルゴリズムが提案されていたが，この章では，それらのアルゴリズムの長所・短所を考察し，それらのアルゴリズムをさらに改良する1つの接近法として，探索範囲の削減という問題が考えられることを示した．そして，それを効率良く行う手順を示し，従来のアルゴリズムが$O((n+m)\cdot(c+1))$の演算時間を要したいくつかの例に対し$O(\gamma+n+m)$の演算時間ですむアルゴリズムを導いた．このアルゴリズムの特徴は，次の3点にまとめられる．

(i) 与えられたグラフに対して，DFS技法で1つの外向森を生成し，その外向森を用いてサイクルの特徴付けを行っていること．

(ii) それによって，他のアルゴリズムのように繰り返し強連結成分に分割する操作を行わなくとも，探索の範囲は1つの強連結な非可分成分内の頂点に制限されること．

(iii) さらに，探索の範囲が他のアルゴリズムより一層制限されていること．すなわち，頂点sを通るサイクルを見い出す際，sからある頂点vに至る順路とvからsに至る順路とが，vからsに至る順路上のBリンクの終点を共有する場合には，頂点vに探索が及ばないこと．（この性質のため，このアルゴリズムをそのまま無向グラフに適用することができなくなっている）．

第4章 無向グラフにおける極大独立頂点集合の列挙問題

4.1 緒言

よく知られているように，無向グラフGの1つの極大独立頂点集合は，Gの補グラフ$\overline{G}$において極大クリークとなるから[27]，すべての極大独立頂点集合を求める問題と，すべての極大クリークを求める問題とは互いに等価な問題である．これらの問題は，グラフ理論における1つの基本的な列挙問題として，計算複雑度解析の分野から興味深いだけでなく[23]，プリント基板上のICの配置問題や，情報システムにおける各種情報のclustering問題，さらには順序回路の状態数削減問題など，その応用範囲も広いため[13,28]，グラフ理論の応用という面からも重要な問題である．そのため，これまで多くの著者によって考察されてきたが，それらのうち代表的なものとして，[27]および[29]～[34]を挙げることができる．

これらのアルゴリズムは大きく3つに類別できる．第1は，ブール演算の手法を用いたもの[27,30]，第2は，部分グラフの系列を用いた手法[29,31]，第3は，backtrackingの技法を用いた手法[32,33]である．[34]は与えられた無向グラフにいくつかの無向辺が付け加えられたり取り除かれたりした場合を取り扱っており，それ自体は与えられた無向グラフのすべての極大クリークを見い出すアルゴリズムではないが，しいて上記3種に類別するとすれば，第2の種類に属している．

これらのアルゴリズムのうち，比較的大きな規模のグラフに対しても適用できるものとして，[31～33]のアルゴリズムを挙げることができるが，これらのいずれも時間複雑度の評価がなされていない．さらに，後に示すように，[31]に述べられているBierstoneのアルゴリズムでは，最悪の場合すべての極大クリークの個数の2乗に比例する演算時間を要するうえに，すべての極大クリークをメモリ内に貯えておく必要がある．また，backtrackingの技法を用いた[32]および[33]のアルゴリズムも，後に示すように，最悪の場合極大クリーク1つを見い出すために頂点の個数の指数関数に比例した演算時間を要する，いわゆる「制限のないbacktracking技法」である．そのため，いずれもあまり効率的なアルゴリズムではない．

そこでこの章では，$O(n \cdot m \cdot \mu + n + m)$の時間複雑度と$O(n+m)$のスペース複雑度を持つ，すべての極大独立頂点集合を見い出す新しいアルゴリズムを提案する．ここで，n，m，μは，それぞれ無向グラフの頂点，無向辺，および極大独立頂点集合の個数である．このアルゴリズムは，基本的には第2番目

の種類に属する，部分グラフの系列を用いた手法であるが，[32, 33]で用いられたbacktracking技法とは異なるbacktracking技法が導入されているため，制限付きのbacktracking技法であるとも言える．

まず，従来のアルゴリズムについて概観し，それらの長所・短所を考察する．次に，与えられた無向グラフGに対して，Gを張る部分グラフ$G(W_{i-1})$とそれから生成される部分グラフ$G(W_i)$を考え，この$G(W_i)$のすべての極大独立頂点集合を$G(W_{i-1})$のそれらから見い出す問題を考察する．それによって得られた結果に基ずいて新しいアルゴリズムを導き，最後にそのアルゴリズムの評価を行う．

4.2 諸定義

無向グラフ$G=[V, E, \Phi]$において，相異なる無向辺e_1, e_2が$\Phi(e_1)=\Phi(e_2)=\langle v, w\rangle \in V \triangle V$を満たすとき，$e_1$, e_2は互いに並列であると言い，$\Phi(e)=\langle v, v\rangle$なる無向辺eを自己ループと言う．この章で対象とする無向グラフは，頂点集合V，無向辺集合Eのいずれも有限集合であり，かつ自己ループおよび並列辺のどちらをも含まないものとする．そこで，この章において単にグラフと言えば，このような無向グラフをさすものとし，グラフGを，写像Φを省略して$G=[V, E]$で表わす．また，以下で単に辺と言えば無向辺をさすものとし，$\Phi(e)=\langle v, w\rangle$を$e=\langle v, w\rangle$と表現する．

[定義4.1] グラフ$G=[V, E]$において，相異なる2頂点$v, w \in V$に対し，辺$e=\langle v, w\rangle \in E$が存在するとき，vおよびwは互いに隣接していると言い，vおよびwをeの端点と言う．また，$v, w \in V$が互いに隣接していないとき，vおよびwは互いに独立であるとも言う．

[定義4.2] グラフ$G=[V, E]$において，vに隣接している頂点の集合を，

$$\Gamma_G(v) \triangleq \{ w \mid \langle v, w\rangle \in E \} \qquad (4\text{-}1)$$

で表わし，その元の個数$|\Gamma_G(v)|$をvの縁度という．

[定義4.3] グラフGの頂点の集合$S \subset V$において，Sのどの2頂点も互いに独立であるとき，Sは独立頂点集合(independent set)であると言い，簡単のためISと略記する．

[定義4.4] グラフGにおいて，$S \subset V$が1つのISであり，Sを真に含む他のISが存在しないとき，Sは極大独立頂点集合(以下ではMISと略記する)であると言う．

[定義4.5] グラフ$G=[V, E]$に対して，辺集合$\overline{E} \triangleq V \triangle V - \{\langle v, v\rangle \mid v \in V\} - E$で定められるグラフ$\overline{G}=[V, \overline{E}]$を，Gの補グラフと言う．

［定義４．６］ グラフGの頂点の集合S⊂Vにおいて，Sのどの２頂点も互いに隣接しているとき，Sを<u>クリーク</u>（clique）と呼び，そのようなSを真に含む他のクリークがないとき，Sを<u>極大クリーク</u>と呼ぶ．

さてここで，図4-1に示すようなk個の3-クリーク（3個の頂点から成るクリーク）で構成されるグラフGを考えよう．このようなグラフにおいては，頂点の個数nおよび辺の個数mは共に3kであり，すべてのＭＩＳの個数は3^kである．従って，第２章で述べたように，与えられたグラフのすべてのＭＩＳを見い出す問題においては，入力の規模の他に出力の規模をも用いて，アルゴリズムの複雑度を評価する必要がある．

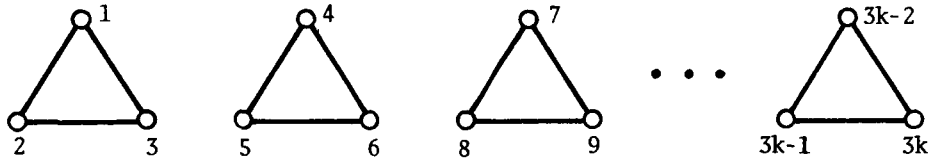


図4-1　　k個の3-クリークから成るグラフ

なお，3k個の頂点を持つグラフの中で，ＭＩＳの個数が最大であるようなグラフは，図4-1に示されるようなグラフであり，その最大値は3^kであることが知られている[35]．

４．３　従来のアルゴリズム

この節では，従来のアルゴリズムのうち，部分グラフの系列を用いたBierstoneのアルゴリズム[31]およびbacktrackingの技法を用いたアルゴリズム[32,33]に注目し，それらの特徴を概観する．

４．３．１　Bierstoneのアルゴリズム

Bierstoneのアルゴリズム[31]は，部分グラフの系列を用いてすべての極大クリークを求める手法であり，それをすべてのＭＩＳを求めるように変更すれば，その概略は次のように書ける．

0°. 与えられたグラフ$G=[V,E]$の頂点集合Vを$V \triangleq \{v_1, v_2, \cdots, v_n\}$とし，$V_1 \triangleq \{v_1\}$，$E_1 \triangleq \phi$として$G$の部分グラフ$G_1=[V_1,E_1]$を定義する．さらに，$G_1$のＭＩＳすべての集合を$\mathcal{M}_1 \triangleq \{V_1\}$とし，$i \leftarrow 2$として以下の操作を行う．

1°. $i<n=|V|$であれば，Gの部分グラフ$G_i=[V_i,E_i]$を，$V_i \triangleq V_{i-1} \cup$

$\{v_i\}$, $E_i \triangleq E \cap (V_i \triangle \overline{V}_i)$ と定め, $\mathcal{M}_i \leftarrow \phi$ として 2° を行う. $i=n$ であれば, $\mathcal{M}_n$ が求めるすべてのMISの集合である.

2° 各 $M_j \in \mathcal{M}_{i-1}$ に対して以下の操作を行い, それが終了すれば $i \leftarrow i+1$ として 1° へ戻る: M_j からある規則で生成される G_i の IS M_j' が, $\mathcal{M}_i$ のどの元にも含まれていない場合には M_j' を $\mathcal{M}_i$ の中に付け加え, その際, $\mathcal{M}_i$ の中に M_j' に含まれるものが存在すればそれを $\mathcal{M}_i$ から取り除く.

これからも解るように, 2° において, 新しいMISが見い出されるたびに, それまでに見い出されている $\mathcal{M}_i$ の元との比較が行われるため, GのMISの個数を μ とすれば, $i=n$ において, $1+2+\cdots+(\mu-1)=\mu(\mu-1)/2$ に(すなわち μ の2乗に)比例した演算時間が必要である. そのため, Bierstone のアルゴリズムの時間複雑度は, 少なくとも $O(\mu^2)$ より大きい.

このような, 時間複雑度がMISの個数 μ の2乗に比例するアルゴリズムは, μ が最悪の場合頂点の個数の指数関数に比例することを考えれば, あまり効率的であるとは言えない.

4.3.2 backtrackingの技法を用いたアルゴリズム

[32]および[33]のアルゴリズムは, 与えられたグラフのすべての極大クリークの集合を, 幾つかの互いに素な部分集合に分割し, その各々の部分集合をbacktrackingの技法を用いて見い出すというものである. これらのアルゴリズムを, すべてのMISを見い出すアルゴリズムに変更するためには, backtracking の技法において, いま調べているクリークのどの頂点にも隣接しているという条件で次の段階の頂点を選び出す代りに, いま調べているISのどの頂点にも隣接していないという条件で次の段階の頂点を選び出していけばよい.

ところで, [32]と[33]のアルゴリズムの差異は本質的なものではなく, その主な相異点は, 次の段階の頂点として選ぶことのできるものがいくつかある場合, そのような頂点 v の線度 $|\Gamma_G(v)|$ の大小によって先に選ぶか後にするかの優先度を付けるもの[33]と, 付けないもの[32]という点である. そこでここでは, [33]のアルゴリズムによってすべてのMISを求める際, その演算時間の上限はどのように表わされるかを考察しよう.

まず, [33]と同様に, 記号 $\mathcal{L}(\cdot)$ および $\mathcal{E}(\cdot)$ を定義する. グラフ $G=[V, E]$ のすべてのMISの集合を $\mathcal{M}$ とし, 頂点の集合 $S \subset V$ に対して, MISの集合 $\mathcal{L}(S)$, $\mathcal{E}(S)$ を次のように定める.

$$\mathcal{L}(S) \triangleq \{ M \in \mathcal{M} \mid M \cap S \neq \phi \} \qquad (4\text{-}2)$$

$$\mathcal{E}(S) \triangleq \{ M \in \mathcal{M} \mid S \subseteq M \} \qquad (4\text{-}3)$$

このとき, 任意の $v \in V$ に対して, $\mathcal{M}$ が次のように分割できることは明らか

である。

$$\mathcal{M} = \mathcal{L}(V) = \mathcal{E}(\{v\}) + (\mathcal{L}(V-\{v\}) \cap \mathcal{L}(\Gamma_G(v))) \qquad (4\text{-}4)$$

ここで，記号＋は互いに素な集合の和を示す．すなわち，集合X，YがX∩Y=φであるとき，X∪YをX＋Yと表わす．

今，図4-2に示されるようなグラフに[33]のアルゴリズムを適用して，すべてのＭＩＳを見い出すことを考える．まず，頂点aが選ばれ，aを含むすべてのＭＩＳが見い出され，次にbを含むもの（すなわち$\mathcal{E}(\{b\})$の元）がすべて見い出され，その後，頂点1を含むもの（すなわち$\mathcal{E}(\{1\}) \cap \mathcal{L}(\Gamma_G(b))$の元），頂点2を含むものという順に頂点kまで，このような$\mathcal{M}$の分割とその部分集合の元を見い出すという操作が行われてゆき，いま，頂点cを含むものを見い出す段階に至ったとする．すなわち，次のような集合の元を求める段階に至ったとする．

$$\mathcal{E}(\{c\}) \cap \mathcal{L}(\Gamma_G(b)) \cap \left[\bigcap_{i=1}^{k} \mathcal{L}(\Gamma_G(i)) \right]$$

$$= \mathcal{E}(\{c\}) \cap \mathcal{L}(\{3k+1, 3k+2, \cdots, 5k\}) \cap \left[\bigcap_{i=1}^{k} \mathcal{L}(\{k+2i-1, k+2i\}) \right]$$

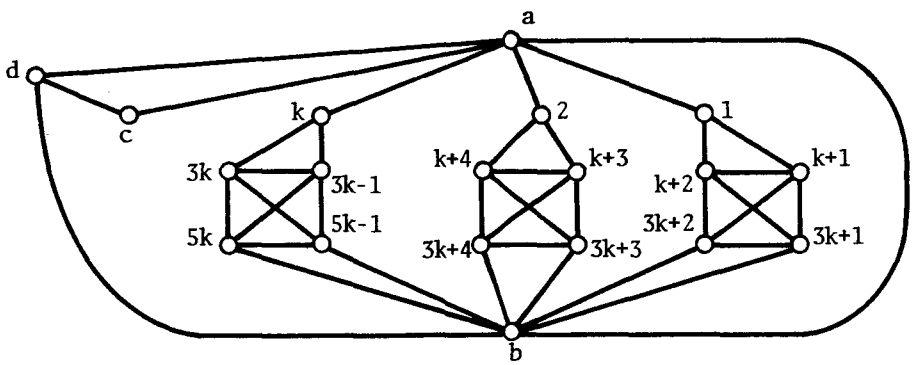


図4-2　[33]のアルゴリズムが効率的でないグラフの例

この集合は，グラフを見れば解るように空集合となり，新しいＭＩＳは見い出されない．しかしながら，そのことが判別されるためには，各集合$\{k+2i-1, k+2i\}$の中から1つずつ元を選び出す場合の数2^kに比例した演算時間を要する．すなわち，[33]のアルゴリズムは最悪の場合，見い出すべきＭＩＳ1つ当りに要する演算時間が，頂点の個数の指数関数に比例する．

このような状況は，次の段階で選び出す頂点を，線度が最大のものあるいは最小のものとするだけでは取り除くことができない．そのような例を作ることは比較的容易である．

以上の考察から解るように，従来のアルゴリズム[31-33]は，出力したＭＩＳ

に重複が起らないように，あるいは極大でないものを生成しないようにするために多くの演算時間を費やしていた．そこで，以下ではこのような点を改良し，ＭＩＳ１つ当りに要する演算時間は高々$O(n \cdot m)$でよく，かつＭＩＳの個数に比例する，すなわち$O(n \cdot m \cdot \mu)$の時間複雑度を持つアルゴリズムを提案する．ここで，$n \triangleq |V|$，$m \triangleq |E|$，$\mu \triangleq |\mathcal{M}|$である．

4.4 基本定理

グラフ$G=[V, E]$の頂点の集合$W \subset V$に対して，

$$E(W) \triangleq \{ \langle v, w \rangle \in E \mid v, w \in W \} \qquad (4\text{-}5)$$

とし，$E(W)$によって定められるGを張る部分グラフ$G(W)$を，

$$G(W) \triangleq [V, E(W)] \qquad (4\text{-}6)$$

と定義する．頂点の集合$W_{i-1} \subset V$，頂点$x \notin W_{i-1}$，および頂点の集合$W_i \triangleq W_{i-1} + \{x\}$を考え，$G$の部分グラフ$G(W_{i-1})$および$G(W_i)$のすべてのＭＩＳの集合を，それぞれ$\mathcal{M}_{i-1}$および$\mathcal{M}_i$と表わす．この節では，$\mathcal{M}_i$と$\mathcal{M}_{i-1}$との間にはどのような関係があるかについて考察する．ただし，以下では簡単のため，$G(W_i)$において頂点vに隣接している頂点の集合を$\Gamma_i(v)$（$\triangleq \Gamma_{G(W_i)}(v)$）で表わし，特に$G(W_i)$において$x \in W_i - W_{i-1}$に隣接している頂点の集合を$A$（$\triangleq \Gamma_i(x)$）で表わすことにする．

$\mathcal{M}_{i-1}$の部分集合$\mathcal{M}_{i-1}(x, \bar{A})$および$\mathcal{M}_{i-1}(x, A)$を，

$$\mathcal{M}_{i-1}(x, \bar{A}) \triangleq \{ M' \in \mathcal{M}_{i-1} \mid M' \cap A = \phi \} \qquad (4\text{-}7)$$

$$\mathcal{M}_{i-1}(x, A) \triangleq \{ M' \in \mathcal{M}_{i-1} \mid M' \cap A \neq \phi \} \qquad (4\text{-}8)$$

と定義すれば，$\mathcal{M}_{i-1}$は次のように分割される．

$$\mathcal{M}_{i-1} = \mathcal{M}_{i-1}(x, \bar{A}) + \mathcal{M}_{i-1}(x, A) \qquad (4\text{-}9)$$

ここで，$G(W_{i-1})$はGのすべての頂点を含み，xは$G(W_{i-1})$において孤立頂点（$\Gamma_{i-1}(x) = \phi$）であるから，すべてのＭＩＳ $M' \in \mathcal{M}_{i-1}$にxが含まれていることは明らかであろう．

［補題４．１］　集合$\mathcal{M}_i(\bar{x}, A)$を，

$$\mathcal{M}_i(\bar{x}, A) \triangleq \{ M = M' - \{x\} \mid M' \in \mathcal{M}_{i-1}(x, A) \} \qquad (4\text{-}10)$$

と定義すれば，次式が成立する．

$$\mathcal{M}_i \supset \mathcal{M}_{i-1}(x, \bar{A}) + \mathcal{M}_i(\bar{x}, A) \qquad (4\text{-}11)$$

（証明）　$\mathcal{M}_i \supset \mathcal{M}_{i-1}(x, \bar{A})$および$\mathcal{M}_{i-1}(x, \bar{A}) \cap \mathcal{M}_i(\bar{x}, A) = \phi$は明らかであるから，$\mathcal{M}_i(\bar{x}, A) \subset \mathcal{M}_i$を示す．そこで，$M \in \mathcal{M}_i(\bar{x}, A)$が$\mathcal{M}_i$の元でないと仮定すれば，明らかに$M$は$G(W_i)$のＩＳであるから，$M + X \in \mathcal{M}_i$なる集合$X$（$\neq \phi$）が存在する．$M + X$は$G(W_i)$のＭＩＳであるから，$G(W_{i-1})$のＩＳである．さらに，$M \cap A \neq \phi$であり，かつ$M + X \in \mathcal{M}_i$であるから，$x \notin X$

である．従って，$M+X+\{x\}$ は $G(W_{i-1})$ のＩＳとなり，$M+\{x\}$ が $G(W_{i-1})$ のＭＩＳであることに矛盾する．（証明終）

次に，$\mathcal{M}_i$ の中の $\mathcal{M}_{i-1}(x,\bar{A})+\mathcal{M}_i(\bar{x},A)$ 以外のＭＩＳに着目し，それらが $\mathcal{M}_{i-1}(x,A)$ の元から導き出されることを示そう．

［補題４．２］ 各 $M\in\mathcal{M}_i-\mathcal{M}_{i-1}(x,\bar{A})-\mathcal{M}_i(\bar{x},A)$ は x を含み，かつこのような M に対して，次の (i) および (ii) を満たす $M'\in\mathcal{M}_{i-1}(x,A)$ が存在する．

(i) $M\subset M'$ であり，かつ $M'-M\subset A$ $(=\Gamma_i(x))$ である．

(ii) 各 $y\in M'\cap A$ に対して，

(1) $\Gamma_{i-1}(y)=\phi$ であるか，あるいは

(2) すべての $z\in\Gamma_{i-1}(y)-A$ に対して $\Gamma_{i-1}(z)\cap(M'-A)\neq\phi$ であるか，

のいずれかが成立する．

（証明） 1°. まず，$M\in\mathcal{M}_i-\mathcal{M}_{i-1}(x,\bar{A})-\mathcal{M}_i(\bar{x},A)$ が x を含むことを示す．そこで，$x\notin M$ と仮定すれば，M は $G(W_i)$ のＭＩＳであるから，$M\cap A\neq\phi$ であり，さらに，任意の $z\in M$ に対して $\Gamma_i(z)\cap M\neq\phi$ であることに注意すれば，任意の $z\in M+\{x\}$ に対して $\Gamma_{i-1}(z)\cap M\neq\phi$ が成り立つ．従って，$M+\{x\}$ は $G(W_{i-1})$ のＭＩＳとなり，$M+\{x\}\in\mathcal{M}_{i-1}(x,A)$，すなわち $M\in\mathcal{M}_i(\bar{x},A)$ となるから，$M\in\mathcal{M}_i-\mathcal{M}_{i-1}(x,\bar{A})-\mathcal{M}_i(\bar{x},A)$ に矛盾する．

2°. 次に，(i) を満足する $M'\in\mathcal{M}_{i-1}(x,A)$ が存在することを示そう．明らかに，$M\in\mathcal{M}_i-\mathcal{M}_{i-1}(x,\bar{A})-\mathcal{M}_i(\bar{x},A)$ は $G(W_{i-1})$ のＩＳとなるから，M を含む $G(W_{i-1})$ のＭＩＳ M' が存在する．もし $M'=M$ であるならば，$x\in M$ および $M\cap A=\phi$ より，$M=M'$ は $\mathcal{M}_{i-1}(x,\bar{A})$ の元となり，M の仮定に反する．従って，M' は M を真に含む．そこで，ある頂点 $v\in A$ が $M'-M$ に含まれていたとすれば，$\Gamma_{i-1}(v)\cap M'=\phi$ であり，$v\in A$ に対しては $\Gamma_{i-1}(v)=\Gamma_i(v)$ が成り立つから，$\Gamma_i(v)\cap M'=\phi$ である．従って，$M'\supset M$ より，$\Gamma_i(v)\cap M=\phi$ となり，M が $G(W_{i-1})$ のＭＩＳであることに矛盾する．それゆえ，このような M' は，$\mathcal{M}_{i-1}(x,A)$ の元であり，(i) を満足する．

3°. そこで，この M' が (ii) も満足することを示そう．M' が $G(W_{i-1})$ のＭＩＳであることから，各 $y\in M'-M=M'\cap A$ に対して，

(1) $\Gamma_{i-1}(y)=\phi$ であるか，あるいは

(2) すべての $z\in\Gamma_{i-1}(y)$ に対して $z\in M\subset M'$ であるか，

のいずれかが成立する．(2) の場合に，$\Gamma_i(z)\cap M=\phi$ と仮定すれば，$M+\{z\}$ が $G(W_i)$ のＩＳとなり，$M\in\mathcal{M}_i$ に矛盾する．それゆえ，$\Gamma_i(z)\cap M=\Gamma_i(z)\cap(M'-A)\neq\phi$ であり，$z\in A$ に対しては $\Gamma_{i-1}(z)=\Gamma_i(z)$ であるから，すべての $z\in\Gamma_{i-1}(y)-A$ に対して，$\Gamma_{i-1}(z)\cap(M'-A)\neq\phi$ が成立する．（証明終）

以下では，$M'\in\mathcal{M}_{i-1}(x,A)$ が補題4.2の (ii) の条件を満足するとき，M' は<u>条件A</u>を満足すると言う．

この補題4.2より，$\mathcal{M}_i - \mathcal{M}_{i-1}(x,\bar{A}) - \mathcal{M}_i(\bar{x},A)$ の元を見い出す方法として次のようなものを考えることができる：$\mathcal{M}_{i-1}(x,A)$ の元の中で，条件Aを満足するＭＩＳすべての集合を $\mathcal{C}_A$ とし，$\mathcal{C}_A$ の2つの元 M_1', M_2' に対して，1つの同値関係（≡）を次のように定義する．すなわち，「$M_1'-A = M_2'-A$ のときかつそのときに限り，$M_1' \equiv M_2'$ である」と定める．このとき，この同値関係によって $\mathcal{C}_A$ を類別し，各同値類からそれぞれ丁度1つずつ取り出した代表元の集合（いわゆる代表系）を $\tilde{\mathcal{C}}_A$ とする．この $\tilde{\mathcal{C}}_A$ を用いて集合 $\tilde{\mathcal{C}}_B$ を，

$$\tilde{\mathcal{C}}_B \triangleq \{ M = M'-A \mid M' \in \tilde{\mathcal{C}}_A \} \qquad (4\text{-}12)$$

と定義すれば，補題4.2より $\mathcal{M}_i - \mathcal{M}_{i-1}(x,\bar{A}) - \mathcal{M}_i(\bar{x},A) \subset \tilde{\mathcal{C}}_B$ が成立する：

さらに，次の補題を証明することができる．

［補題４．３］　次式が成立する．

$$\tilde{\mathcal{C}}_B = \mathcal{M}_i - \mathcal{M}_{i-1}(x,\bar{A}) - \mathcal{M}_i(\bar{x},A) \qquad (4\text{-}13)$$

（証明）　ここでは，$\tilde{\mathcal{C}}_B \subset \mathcal{M}_i - \mathcal{M}_{i-1}(x,\bar{A}) - \mathcal{M}_i(\bar{x},A)$ を示せばよいが，定義より明らかなように，$\tilde{\mathcal{C}}_B \cap \mathcal{M}_{i-1}(x,\bar{A}) = \phi$ および $\tilde{\mathcal{C}}_B \cap \mathcal{M}_i(\bar{x},A) = \phi$ であるから，$\tilde{\mathcal{C}}_B \subset \mathcal{M}_i$ を示せば十分である．そこで，$M \in \tilde{\mathcal{C}}_B$ が $G(W_i)$ のＭＩＳでないと仮定すれば，$\tilde{\mathcal{C}}_B$ の作り方から明らかなようにMは $G(W_i)$ のＩＳであるから，$M+X \in \mathcal{M}_i$ なる集合X（≠φ）が存在する．また，$\tilde{\mathcal{C}}_B$ の作り方から，$M'-A=M$ なる $G(W_{i-1})$ のＭＩＳ $M' \in \mathcal{M}_{i-1}(x,A)$ が存在するが，この M' に注目すれば，任意の $y \in M' \cap A$ に対して，次の(i)および(ii)が共に成立する．以下でそれを示そう．

(i)　$y \notin X$ 　　(4-14)

(ii)　$\Gamma_{i-1}(y) \cap X = \phi$ 　　(4-15)

$M+X$ が $G(W_i)$ のＭＩＳであり，かつ $x \in M$ であるから(i)は明らかである．そこで，任意の $z \in \Gamma_{i-1}(y)$ を考える．もし $z \in \Gamma_{i-1}(y) \cap A$ であれば，$x \in M$ および $M+X \in \mathcal{M}_i$ より，$z \notin X$ である．また，$z \in \Gamma_{i-1}(y) - A$ であれば，$\Gamma_i(z) = \Gamma_{i-1}(z)$ であり，かつ M' は条件Aを満足するから，$\Gamma_i(z) \cap M = \Gamma_{i-1}(z) \cap (M'-A) \neq \phi$ である．それゆえ，$M+X$ が $G(W_i)$ のＭＩＳであることから，$z \notin X$ である．従って，(i)，(ii)共に成立することがわかる．(i)および(ii)から，$M+X+(M'-A)$ は $G(W_{i-1})$ のＩＳとなり，これは $M' = M+(M'-A)$ が $G(W_{i-1})$ のＭＩＳであることに矛盾する．　（証明終）

以上の補題より，$\mathcal{M}_{i-1}$ のＭＩＳから $\mathcal{M}_i$ のＭＩＳを生成するためには，$\tilde{\mathcal{C}}_B$ の元を効率良く見い出せばよいことが解る．そこで以下では，$\mathcal{M}_{i-1}(x,A)$ から $\tilde{\mathcal{C}}_B$ を効率良く見い出すことを考えよう．

そこで，$A = \Gamma_i(x)$ の各元に対して，任意に1つの順序を与え，それを添字によって，

$$A \triangleq \{ y_1, y_2, \cdots, y_p \} \quad (p \triangleq |A|) \qquad (4\text{-}16)$$

と表わす．また，$\{y_j\}_{j=k}^{h}$ によって，添字が k から h までの頂点の集合

$$\{y_j\}_{j=k}^{h} \triangleq \{y_k, y_{k+1}, \cdots y_{h-1}, y_h\} \tag{4-17}$$

を表わすことにする．

［定義４．７］ $G(W_{i-1})$ の MIS $M' \in \mathcal{M}_{i-1}(x,A)$ に対し，どの $y_k \in M' \cap A$ に関しても，

(1) $\Gamma_{i-1}(y_k) = \phi$ であるか，あるいは

(2) すべての $z \in \Gamma_{i-1}(y_k) - \{y_j\}_{j=1}^{k}$ に対して，$\Gamma_{i-1}(z) \cap (M' - \{y_j\}_{j=1}^{k}) \neq \phi$ である，

のいずれか一方が成立するとき，この M' は条件Bを満足するという．

$M' \in \mathcal{M}_{i-1}(x,A)$ が条件Bを満足するということは，$\bigcup_{y \in M' \cap A} \Gamma_{i-1}(y) \subset V - M'$ の元に注目して言い換えれば，v が次の (i), (ii) の性質を満たすことを意味する．

(i) $v \notin A$ であれば，$G(W_{i-1})$ において，v は $M' - A$ の元に隣接している．

(ii) $v = y_j \in A$ であれば，$G(W_{i-1})$ において，v は $M' - A$ の元に隣接しているか，または，$k > j$ なる頂点 $y_k \in M' \cap A$ に隣接している．

$M' \in \mathcal{M}_{i-1}(x,A)$ が条件Aを満足している場合には，(i) は成立するが，必ずしも (ii) は成立しない．この (ii) の性質によって，$\mathcal{C}_A$ の各同値類から唯一つの元を選び出すことができ，以下の補題が成立する．

［補題４．３］ 集合 $\mathcal{M}_i'(x,\bar{A})$ を，

$$\mathcal{M}_i'(x,\bar{A}) \triangleq \{ M = M' - A \mid M' \in \mathcal{M}_{i-1}(x,A), \ M' \text{は条件Bを満たす} \} \tag{4-18}$$

と定義すれば，次式が成立する．

$$\mathcal{M}_i'(x,\bar{A}) \supset \tilde{\mathcal{C}}_B \tag{4-19}$$

（証明） $\mathcal{C}_A$ の各同値類に含まれる代表元唯一つが条件Bを満足することを示せば十分である．

1° まず，$\mathcal{C}_A$ の1つの同値類に含まれる相異なる2元 M_1', M_2' ($M_1' - A = M_2' - A$) が，同時に条件Bを満足することはないことを示す．そこで，頂点 $y_{k_1} \in M_1'$, $y_{k_2} \in M_2'$ を次のように定める．

y_{k_1} : $M_1' - M_2'$ ($\subset A = \Gamma_i(x)$) の元の中で，最大の添字を持つ頂点．

y_{k_2} : $M_2' - M_1'$ ($\subset A = \Gamma_i(x)$) の元の中で，最大の添字を持つ頂点．

以下では，$k_1 > k_2$ のとき，M_1' は M_2' より高いレベルにあると言うことにする．いま，M_1' が M_2' より高いレベルにあると仮定しても一般性を失わない．このとき，$y_{k_1} \in M_1' - M_2'$ かつ $M_2' \in \mathcal{M}_{i-1}(x,A)$ より，$\Gamma_{i-1}(y_{k_1}) \cap M_2' \neq \phi$ であり，$M_1' - A = M_2' - A$ および $k_1 > k_2$ であることから，$\Gamma_{i-1}(y_{k_1}) \cap M_2' \subset \{y_j\}_{j=1}^{k_1} \subset A$ が成立する．そこで，y_t を $\Gamma_{i-1}(y_{k_1}) \cap M_2'$ の元の中で，最大の添字を持つ頂点であるとすれば，この y_t ($\in M_2' \cap A$) に対して，y_{k_1} ($\in \Gamma_{i-1}(y_t) - \{y_j\}_{j=1}^{t}$

)が存在し，$\Gamma_{i-1}(y_{k_1}) \cap (M_2' - \{y_j\}_{j=1}^{+}) = \phi$ が成立する．それゆえ，M_2' は条件Bを満足しない．

2°. 次に，$\mathcal{C}_A$ の1つの同値類において，その同値類のどの元 M' よりも高いレベルにある元を M_m' とし，これが条件Bを満足することを示す．そこで，M_m' が条件Bを満足しないものと仮定する．すなわち，ある $y_k \in M_m' \cap A$ に対して，頂点 $z \in \Gamma_{i-1}(y_k) - \{y_r\}_{r=1}^{k}$ が存在し，

$$\Gamma_{i-1}(z) \cap (M_m' - \{y_r\}_{r=1}^{k}) = \phi \qquad (4\text{-}20)$$

が成り立つとする．今，$z \in \Gamma_{i-1}(y_k) - A$ であるとすれば，$M_m' \in \mathcal{C}_A$ は条件Aを満足するから，

$$\Gamma_{i-1}(z) \cap (M_m' - \{y_r\}_{r=1}^{p}) = \phi \qquad (4\text{-}21)$$

が成立し，(4-20)と矛盾する．従って，$z \in \{y_r\}_{r=k+1}^{p}$ であり，かつ(4-20)より次式が成立する．

$$\Gamma_{i-1}(z) \cap M_m' \subset \{y_r\}_{r=1}^{k} \qquad (4\text{-}22)$$

そこで，$y_j \triangleq z$ とし，次式で定められる集合Xを考える．

$$X \triangleq M_m' + \{y_j\} - (\Gamma_{i-1}(y_j) \cap M_m') \qquad (4\text{-}23)$$

Xは明らかに $G(W_{i-1})$ のISであるから，必要なら適当な集合Yを Xに付加して，$G(W_{i-1})$ のMIS $M_m'' \triangleq X + Y$ を作ることができる（図4-3参照）．以下で，この M_m'' に対して，(i) $M_m' - A = M_m'' - A$，および(ii) M_m'' は条件Aを満足する（$M_m'' \in \mathcal{C}_A$），の2つが成り立つことを示そう：

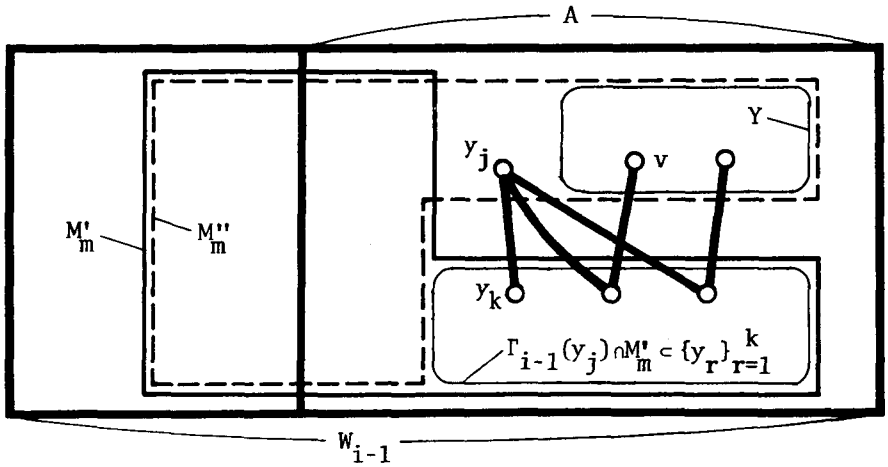


図4-3　証明のための説明図

(i) 今，Yの元 v に注目する．この v に対して，$\Gamma_{i-1}(v) \cap M_m' \neq \phi$ および $\Gamma_{i-1}(v) \cap X = \phi$ が成立するから，Xの定義より，$\Gamma_{i-1}(v) \cap M_m' \subset \Gamma_{i-1}(y_j) \cap M_m'$ が成り立つ．従って，(4-22)より，

$$\Gamma_{i-1}(v) \cap M'_m \subset \{y_r\}_{r=1}^{k} \subset A \qquad (4\text{-}24)$$

となる．そこで，$\Gamma_{i-1}(v) \cap M'_m$ の1つの元を $y_t \in M'_m \cap A$ として，$v \in \Gamma_{i-1}(y_t) - A$ と仮定すれば，M'_m が条件Aを満足することより，$\Gamma_{i-1}(v) \cap (M'_m - A) \neq \phi$ となり，(4-24)に矛盾する．従って，$v \in \Gamma_{i-1}(y_t) \cap A \subset A$ であり，M''_m の定義から，

$$M'_m - A = M''_m - A \qquad (4\text{-}25)$$

であることがわかる．

(ii) 任意の $y \in (M''_m \cap A) - \{y_j\} - Y$ に対して，$\Gamma_{i-1}(y) = \phi$ あるいはすべての $z \in \Gamma_{i-1}(y) - A$ に対して $\Gamma_{i-1}(z) \cap (M''_m - A) \neq \phi$ のいずれかが成り立つことは，$y \in M'_m \cap A$，$M'_m \in \mathcal{C}_A$，および(4-25)より明らかであろう．従って，M''_m が条件Aを満足しないと仮定すれば，ある $v \in Y + \{y_j\}$ に対して，補題4.2の(ii)の条件が成立しないことになる．すなわち，ある $v \in Y + \{y_j\} \subset M''_m \cap A$ に対して，頂点 $w \in \Gamma_{i-1}(v) - A$ が存在し，$\Gamma_{i-1}(w) \cap (M''_m - A) = \phi$ となる．それゆえ，(4-25)より，$\Gamma_{i-1}(w) \cap (M'_m - A) = \phi$ であり，かつ M'_m が条件Aを満足していることから，w は $M'_m \cap A$ のどの元にも隣接していない．従って，次式が成り立つ．

$$\Gamma_{i-1}(w) \cap M'_m = \phi \qquad (4\text{-}26)$$

一方，$w \notin A$，$w \notin M''_m$，および $M'_m - A = M''_m - A$ より，

$$w \notin M'_m \qquad (4\text{-}27)$$

であることがわかる．それゆえ，(4-26)および(4-27)より，$M'_m + \{w\}$ が $G(W_{i-1})$ のISとなり，M'_m が $G(W_{i-1})$ のMISであることに矛盾する．従って，M''_m は条件Aを満足する：

以上，(i)および(ii)より，M'_m と M''_m は $\mathcal{C}_A$ の同じ同値類に入っていることが解る．さらに，M''_m は $y_j = z$ ($j > k$) を含み，かつ，(4-22)および(4-23)より $M'_m - M''_m \subset \Gamma_{i-1}(y_j) \cap M'_m \subset \{y_r\}_{r=1}^{k}$ であることから，M''_m が M'_m より高いレベルにあることになり，M'_m の選び方に矛盾する．

(証明終)

[補題4.5] $\mathcal{M}'_i(x, \bar{A}) \subset \tilde{\mathcal{C}}_B$ が成立し，補題4.4より，次式が成立する．

$$\mathcal{M}'_i(x, \bar{A}) = \tilde{\mathcal{C}}_B \qquad (4\text{-}28)$$

(証明) 条件Bを満足する $M' \in \mathcal{M}_{i-1}(x, A)$ は，同時に条件Aをも満足することを示せばよい．今，そのような M' が条件Aを満たさないと仮定すれば，ある $y_j \in M' \cap A$ に対して，頂点 $z \in \Gamma_{i-1}(y_j) - A$ が存在し，$\Gamma_{i-1}(z) \cap (M' - A) = \phi$ が成り立つ．そこで，そのような y_j の中で，最大の添字を持つ頂点を y_k とし，$z \in \Gamma_{i-1}(y_k) - A$ に対して次式が成立するものとする．

$$\Gamma_{i-1}(z) \cap (M' - A) = \phi \qquad (4\text{-}29)$$

一方，kの最大性より，

$$\Gamma_{i-1}(z) \cap (M' \cap \{y_r\}_{r=k+1}^{p}) = \phi \qquad (4\text{-}30)$$

が成り立つから，(4-29)より，$\Gamma_{i-1}(z) \cap (M' - \{y_r\}_{r=1}^{k}) = \phi$ となる．従って，M'は条件Bを満足しないことになり，M'の仮定に矛盾する．（証明終）

以上の補題によって，$\tilde{\mathcal{E}}_B$は$\mathcal{M}_{i-1}(x,A)$から条件Bを用いて直接導き出せること，さらに，その際，重複の起らないことが示された．また，補題4.3および4.5から，次の定理が導かれる．

［定理４．１］ 次式が成立する．

$$\mathcal{M}_i = \mathcal{M}_{i-1}(x,\bar{A}) + \mathcal{M}_i'(x,\bar{A}) + \mathcal{M}_i(x,A) \qquad (4\text{-}31)$$

（証明略）

この定理より，$G(W_i)$のすべてのＭＩＳを$G(W_{i-1})$のそれらから求める手順の概略として，図4-4に示すようなものが考えられる．

```
begin
   empty the set M_i ;
   for   each M' ∈ M_{i-1}   do
        if   M' ∩ Γ_i(x) = φ   then   put M' ∈ M_{i-1}(x,Ā) into M_i
        else   begin   ( in this case M' ∈ M_{i-1}(x,A) )
                  put M'-{x} ∈ M_i(x̄,A) into M_i ;
                  if   M' satisfies condition B   then
                       put M'-Γ_i(x) ∈ M_i'(x,Ā) into M_i
               end
end ;
```

図4-4　$G(W_i)$のすべてのＭＩＳを$G(W_{i-1})$のそれらから見い出す手順の概略

４．５　アルゴリズム

以下では，与えられたグラフ$G=[V,E]$のすべてのＭＩＳを生成するアルゴリズムを，図4-4の手順に基ずいて考察する．ところで，$G(W_i)$のすべてのＭＩＳを$G(W_{i-1})$のそれらから見い出す手順において，最も多くの演算時間を必要とするのは，$M' \in \mathcal{M}_{i-1}(x,A)$が条件Bを満足するか否かの判定である．そこでまず，この判定を効率良く行う手法について考察する．

４．５．１　条件Bの判定法

まず，次の写像を定義する．

[定義4．8] $G(W_i)=[V, E(W_i)]$ において，$\mathcal{M}_i$ と V の直積 $\mathcal{M}_i \times V$ から，非負の整数の集合 I の中への写像 $f_i : \mathcal{M}_i \times V \to I$ を，

$$f_i(M, v) \triangleq |\Gamma_i(v) \cap M| \tag{4-32}$$

と定義する．

定義より，$G(W_i)$ において，頂点 v が $M \in \mathcal{M}_i$ の元であるための必要十分条件が，$f_i(M, v)=0$ で与えられることは明らかであろう．さらに，定理4-1 およびこの写像から，次の関係は容易に導かれる．

Ⅰ．各 $M' \in \mathcal{M}_{i-1}(x, \bar{A})$ に対し，次の関係が成立する．

(i) どの $y_j \in A = \Gamma_i(x)$ に対しても，

$$f_i(M', y_j) = f_{i-1}(M', y_j) + 1, \tag{4-33}$$

(ii) どの $v \in V - A$ に対しても，

$$f_i(M', v) = f_{i-1}(M', v). \tag{4-34}$$

Ⅱ．各 $M' \in \mathcal{M}_{i-1}(x, A)$ に対し，次の関係が成立する．

(i) $$f_i(M'-\{x\}, x) = |M' \cap A| \tag{4-35}$$

(ii) どの $v \in V - \{x\}$ に対しても，

$$f_i(M'-\{x\}, v) = f_{i-1}(M', v) \tag{4-36}$$

Ⅲ．$\mathcal{M}_{i-1}(x, A)$ の中で，条件 B を満足する M' に関して，次の関係が成立する．

(i) どの $y_j \in A$ に対しても，

$$f_i(M'-A, y_j) = f_{i-1}(M', y_j) - |M' \cap A \cap \Gamma_{i-1}(y_j)| + 1 \tag{4-37}$$

(ii) どの $v \in V - A$ に対しても，

$$f_i(M'-A, y_j) = f_{i-1}(M', y_j) - |M' \cap A \cap \Gamma_{i-1}(y_j)|. \tag{4-38}$$

また，$M' \in \mathcal{M}_{i-1}(x, A)$ が条件 B を満足するということは，任意の $y_k \in M' \cap A$ に対して，

(1) $\Gamma_{i-1}(y_k)=\phi$ であるか，あるいは

(2) すべての $z \in \Gamma_{i-1}(y_k) - \{y_j\}_{j=1}^{k}$ に対して，$f_{i-1}(M', z) - |M' \cap \{y_j\}_{j=1}^{k} \cap \Gamma_{i-1}(z)| \neq 0$ である，

のいずれかが成立することであると言い直すことができる．従って，どのような $z \in \Gamma_{i-1}(y_k) \cap \{y_j\}_{j=1}^{k}$ に対しても，$f_{i-1}(M', z) - |M' \cap \{y_j\}_{j=1}^{k} \cap \Gamma_{i-1}(z)| + 1 > 0$ が成立することに注意すれば，次のような手順で $M' \in \mathcal{M}_{i-1}(x, A)$ が条件 B を満足するか否かを判定することができる．

0°. 各頂点 $v \in V$ に対して，整数変数 $IS(v)$ を割り当て，初期値として $IS(v) \leftarrow f_{i-1}(M', v)$ とする．

1°. $A=\{y_j\}_{j=1}^{p}$ の各元 y_j に対して，添字の順に 2° の操作を繰り返す．

2°. $y_j \in M'$ のときのみ，$\Gamma_{i-1}(y_j)$ のすべての元 z に対して，$IS(z) \leftarrow IS(z)-1$ とし，その後，$y_j \in M'$ のいかんにかかわらず，$IS(y_j) \leftarrow IS(y_j)+1$ とする．

3°. y_j に対する2°の繰り返し過程で，ある $y_k \in M'$ に隣接する z に対して，$IS(z)-1=0$ となるならば，M' は条件Bを満足しない，そうでないならば，条件Bを満足すると判定することができる．

なお，M' が条件Bを満足した場合には，2°の繰り返しの終了時の各 $IS(v)$ の値は，$f_i(M'-A, v)$ の値になっていることに注意しよう．

4．5．2 アルゴリズム

以下では簡単のため，与えられたグラフ $G=[V,E]$ の各頂点は，1から $n \triangleq |V|$ までの整数で表現されているものとし（$V \triangleq \{1,2,\cdots,n\}$），グラフGは，各頂点に隣接する頂点のリスト $Adj(v)$（$=\Gamma_G(v)$）で表わされているものとする．さらに，各 i（$1 \leq i \leq n$）に対して，頂点の集合 W_i を $W_i \triangleq \{1,2,\cdots,n\}$ とし，$G(W_i)$ のMISの集合を $\mathcal{M}_i \triangleq \{M_1^i, M_2^i, \cdots, M_{\mu_i}^i\}$ とする．

ここで，アルゴリズムの説明を容易にし，それを理解する助けとなる有向グラフ $\mathcal{G}$ を定義する．

［定義4．9］ 与えられたグラフ $G=[V,E]$ に対して，次のようにして作られた有向グラフ $\mathcal{G}$ を，Gに付随したMIS有向グラフと言う．各 $\mathcal{M}_i$ のどの元 M_j^i についても，$\mathcal{G}$ の頂点 M_j^i を1対1に対応させ，$\mathcal{M}_{i-1}$ の元 M_j^{i-1} から図4-4の手順で $\mathcal{M}_i$ の元 M_k^i が生成されるときかつそのときに限り，頂点 M_j^{i-1} から頂点 M_k^i への有向辺 (M_j^{i-1}, M_k^i) を与える．

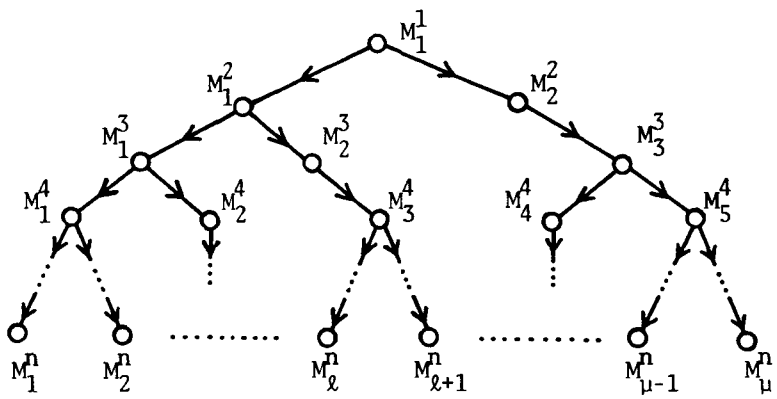


図4-5 MIS有向グラフ $\mathcal{G}$

MIS有向グラフgは，例えば図4-5に示されるような形状を持つ，$\mathcal{M}_1$のただ1つの元$M_1^1=V$を根（root）とする外向木（arborescence[27]）となる．さらに，ある頂点M_j^{i-1}から出て行くgの有向辺が2つあるということは，$M_j^{i-1}\in\mathcal{M}_{i-1}$が条件Bを満足したことを意味している．

図4-6に，与えられたグラフGのすべてのMISを生成するアルゴリズムをALGOL-likeな言語で記す．このアルゴリズムは，Gに付随するMIS有向グラフg上のすべての頂点を，$M_1^1\in\mathcal{M}_1$から始めてbacktrackingの技法によって探索するが，gの1つの頂点に対して，手続きBACKTRACKが1回呼び出される．

今，g上のある頂点M_j^{i-1}に対して，手続きBACKTRACK$(i-1)$が呼び出されたとすれば，$M_j^{i-1}\in\mathcal{M}_{i-1}$に対して次の操作が行われる．このとき，各IS$(\cdot)$の配列要素IS$(v)$は$f_{i-1}(M_j^{i-1},v)$の値となっている．これは，BACK-TRACKの呼び出し回数に関する帰納法で容易に証明することができる．

まず，ラベルC1において，カウンタcを用い$M_j^{i-1}\cap\Gamma_i(i)=\phi$か否かが調べられる．空の場合（c=0）には，B1において$M_j^{i-1}=M_k^i\in\mathcal{M}_i$に対応した$g$の頂点$M_k^i$に関してBACKTRACK$(i)$が呼び出される．空でない場合には，B2において$M_j^{i-1}-\{x\}\in\mathcal{M}_i$に対応してBACKTRACK$(i)$が呼び出された後，C2において，$M_j^{i-1}$が条件Bを満足するか否かが調べられる．その結果，論理変数fがtrue（すなわち，M_j^{i-1}が条件Bを満足する）場合のみ，B3において$M_j^{i-1}-\Gamma_i(i)\in\mathcal{M}_i$に対応してBACKTRACK$(i)$が呼び出される．なお，B1，B2，B3においてBACKTRACKが呼び出される際には，各IS(v)の値は$f_i(M_k^i,v)$の値に置き換えられており，かつ，それらの呼び出しが終了した後には，もとの$f_{i-1}(M_j^{i-1},v)$の値に復元されていることは容易に確められるであろう．ただし，ラベルL3およびL4の復元操作においては，各頂点iに割当てられたリストBucket(i)が用いられている．このBucket(i)には，$\Gamma_i(i)$の頂点の中で，C3の操作が行われた頂点が挿入され，これを用いることによって，IS$(\cdot)$の値を元の$f_{i-1}(M_j^{i-1},v)$の値に復元することができる．

この図4-6のアルゴリズムに対して，次の定理が成り立つことは，定理4.1およびそれ以後の説明から明らかであろう．

［定理4．2］　与えられたグラフGに対してこのアルゴリズムを適用すれば，GのすべてのMISを重複なく見い出すことができる．

（証明略）

```
procedure  MIS ;
     comment  Procedure MIS is a routine for generating all the MIS's of
        a graph G represented by adjacency lists Adj(·).   Integer n is
        a global variable denoting the number of vertices ;
 begin    integer list array  Bucket(n) ;    integer array  IS(n) ;
   procedure  BACKTRACK( integer value  i ) ;
     begin    integer  c, x ;    logical  f ;
       if   i < n   then
          begin
            x := i + 1 ;
            c := 0 ;
   C1:      for   y ∈ Adj(x) such that y ≤ i   do
               if   IS(y) = 0   then   c := c + 1 ;
            if   c = 0   then
              begin
   L1:          for   y ∈ Adj(x) such that y ≤ i   do   IS(y) := IS(y) + 1 ;
   B1:          BACKTRACK( x ) ;
   L2:          for   y ∈ Adj(x) such that y ≤ i   do   IS(y) := IS(y) - 1
              end
            else
              begin
                IS(x) := c ;
   B2:          BACKTRACK( x ) ;
                IS(x) := 0 ;
                f := true ;
   C2:          for   y ∈ Adj(x) such that 1 ≤ y ≤ i, in increasing order   do
                   begin
                     if   IS(y) = 0   then
                        begin
                          put y in Bucket(x) ;
   C3:                  for   z ∈ Adj(y) such that z ≤ i   do
                           begin
                             IS(z) := IS(z) - 1 ;
                             if   IS(z) = 0   then   f := false
                           end
                        end ;
                     IS(y) := IS(y) + 1
                   end ;
   B3:          if   f = true   then   BACKTRACK( x ) ;
   L3:          for   y ∈ Adj(x) such that y ≤ i   do   IS(y) := IS(y) - 1 ;
   L4:          for   y ∈ Bucket(x)   do
                  begin
   L5:              for   z ∈ Adj(y) such that z ≤ i   do   IS(z) := IS(z) + 1 ;
                    delete y from Bucket(x)
                  end
              end
          end
       else   output new MIS designated by IS(·)
     end  BACKTRACK ;
   for   j := 1   until   n   do
      begin
         IS(j) := 0 ;
         Bucket(j) := ϕ
      end ;
   BACKTRACK( 1 )
 end  MIS ;
```

図4-6　ＭＩＳ生成アルゴリズム

4．6　複雑度解析および計算結果

前節において提案したアルゴリズムを評価する次の定理が成立する．

［定理4．3］　このアルゴリズムの時間複雑度およびスペース複雑度は，それぞれ$O(n\cdot m\cdot\mu+n+m)$および$O(n+m)$で与えられる．ここで，n，m，およびμはそれぞれグラフの頂点，辺，およびMISの個数を示す変数である．

（証明）　スペース複雑度が$O(n+m)$であることは，手続きBACKTRACKの再帰呼び出しの深さがnであり，かつリスト$Adj(\cdot)$および$Bucket(\cdot)$に要するメモリスペースが$O(n+m)$であることに気が付けば，容易に確められる．そこで，時間複雑度を評価するために，g上の1つの頂点M_j^iに対して呼び出されたBACKTRACK(i)に要する演算時間を考える．いま，Gの頂点xの線度をd_xと表わせば，ラベルL1，L2，L3，L4，C1，およびC2の操作の繰り返し回数は，それぞれd_xを越えることはない．さらに，L4に含まれるL5およびC2に含まれるC3の操作の繰り返し回数は，BACKTRACK(i)全体で，どちらも各$y\in Adj(x)$の線度d_yの総和$\sum_{y\in Adj(x)} d_y$を越えることはないから，グラフの線度の総和$2m$を越えることはない．従って，手続きBACKTRACK(i)において，再帰呼び出しBACKTRACK(x)（B1，B2，およびB3）が行なわれている時間以外に要する演算時間は，定数k_1，k_2，k_3を適当に選べば，$k_1\cdot m+k_2\cdot d_x+k_3$を越えることはない．それゆえ，MIS1つ当りに要する演算時間は，gにおいて$M_1^1\in\mathcal{M}_1$から$M_j^n\in\mathcal{M}_n$に至る順路上の各頂点に対して呼び出された手続きBACKTRACKのこのような演算時間の総和を越えないから，高々$\sum_{x=1}^{n}(k_1\cdot m+k_2\cdot d_x+k_3)=$

$k_1\cdot n\cdot m+2k_2\cdot m+k_3\cdot n$であり，$O(n\cdot m)$である．さらに，出力に要する演算時間もMIS1つ当りに$O(n)$であるから，手続きMIS全体として要する演算時間は高々$O(n\cdot m\cdot\mu)$であり，入力に要する演算時間も考慮して，時間複雑度は$O(n\cdot m\cdot\mu+n+m)$となる．　（証明終）

このアルゴリズムをFORTRANを用いてプログラム1，NEAC 2200/700で幾つかの例題について実験を行った．

初めに，図4-1で示されるようなk個の3・クリークから成るグラフについて実験した．その結果を表4-1に示す．ただし，ここでは出力の量を少くなくするため，MISの各頂点のリストは出力せず，MISの個数のみを出力している．

表4-1の結果は，このアルゴリズムが$O(\mu)$の演算時間で結果を出力したことを示している．これは以下のように理論的に説明できる：　定理4.3の証明

表4-1　k個の3・クリークから成るグラフに対する実験結果

k	4	5	6	7	8
number of MIS's	81	243	729	2187	6561
CPU time per MIS (μsec)	555.6	551.4	558.3	547.3	554.8

k	9	10	11	12	13
number of MIS's	19683	59049	177147	531441	1594323
CPU time per MIS (μsec)	585.4	586.8	580.3	546.3	546.3

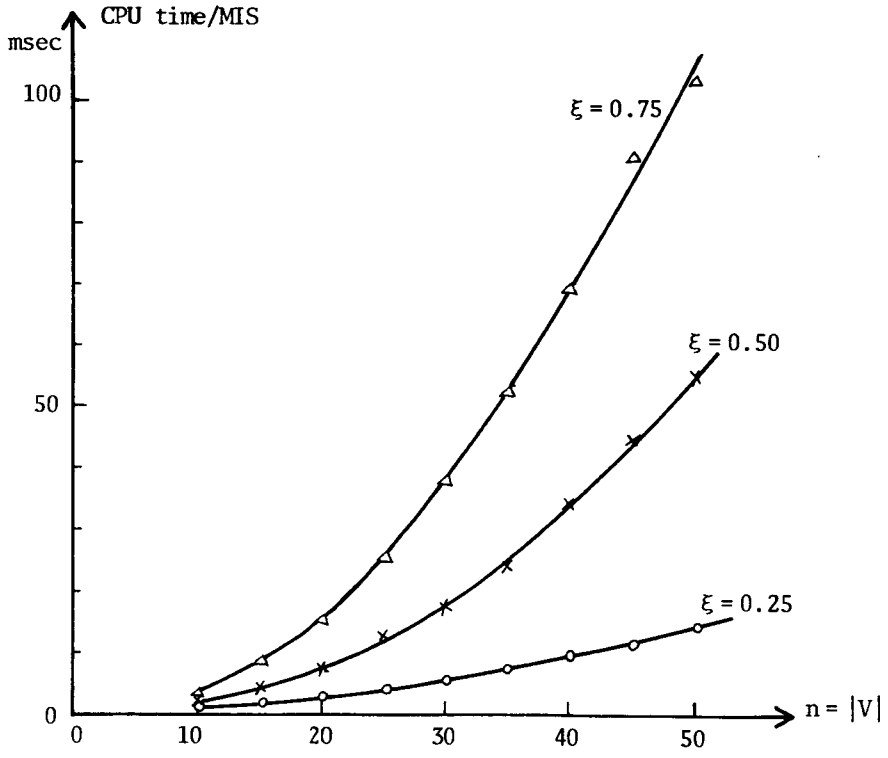


図4-7　ランダムに生成したグラフに対する実験結果1.

において示したように，G上の頂点M_j^i（$i<n$）に対して呼び出された手続きBACKTRACK(i)に要する演算時間は，再帰呼び出しの時間を除けば，$O(\sum_{y\in Adj(x)} d_y + d_x)$と書け，$k$個の3・クリークから成るグラフにおいては，どの頂点vの線度d_vも2であるから，このような演算時間は$O(1)$であることがわかる．さらに，$i=n$の場合，すなわちMISを出力する際にも，MISの各頂点を出力しないから，必要な演算時間は$O(1)$である．従って，Gの頂点の個数をζとすれば，$n=m=3k$であるから，全体として必要な演算時間は$O(\zeta+n)$である．ところで，このような$n=3k$のグラフGは3^k個のMISを持ち，3^kは頂点1個当りのMISの個数の最大値であることから[35]，このようなGに付随したMIS有向グラフgの頂点の個数は3^{k+1}を越えない．それゆえ，ζのオーダは$O(\mu)$である：

次に，$n=10$から50までのランダムに生成したグラフに対してこのアルゴリズムを適用した．各nの値に対して，mの値からξ（$\xi=2m/n(n-1)$）$=0.25$，0.50，および0.75であるような3種類のグラフを考え，それぞれに対して9つずつグラフを生成した．その結果をnおよびmに関して，図4-7

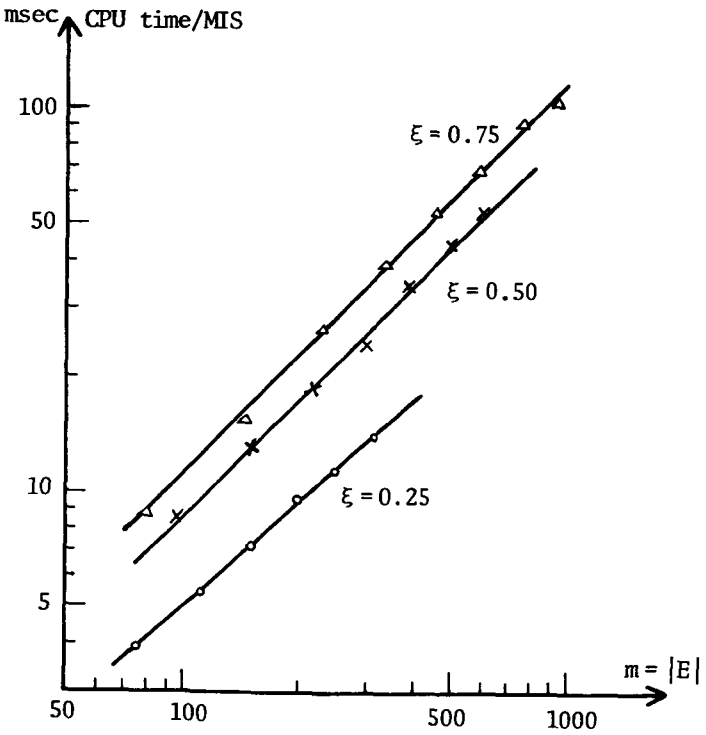


図4-8　ランダムに生成したグラフに対する実験結果2.

および図4-8に示す．これらの結果は，このアルゴリズムに要するＭＩＳ１つ当りの演算時間が実際にはほぼ$O(m)$であることを示している．

4．8　結　言

この章では，与えられたグラフのすべてのＭＩＳを見い出す新しいアルゴリズムを提案した．このアルゴリズムは，基本的には部分グラフの系列を用いた手法であるが，制限付きのbacktrackingの技法を用いた手法であるとも考えることができ，その時間およびスペース複雑度は，それぞれ$O(n \cdot m \cdot \mu + n + m)$および$O(n+m)$で表わされる．このオーダの時間およびスペース複雑度は，従来のアルゴリズムでは得られなかったものであり，このような時間およびスペース複雑度のアルゴリズムが得られた理由は，次のような性質を持つ$\mathcal{M}_{i-1}$と$\mathcal{M}_i$との関係が見い出せたためである．

(i)　各$M_j^{i-1} \in \mathcal{M}_{i-1}$は，１つあるいは２つの$M_k^i \in \mathcal{M}_i$を重複なく生成する．

(ii)　$M_j^{i-1} \in \mathcal{M}_{i-1}$から生成されるどのような$M_k^i \in \mathcal{M}_i$も，$M_j^{i-1}$と$G(W_i)$の接続関係にのみ依存し，$G(W_{i-1})$あるいは$G(W_i)$の他のＭＩＳには依存しない．

第5章 結論

本研究で得られた結果と残された問題点について以下にまとめる．

第2章においては，グラフの列挙問題に対するアルゴリズムの効率の良さを評価する方式について述べた．ここで述べた最悪複雑度およびオーダ(order)を用いた評価方式は，広く一般的に用いられているものであり，理論的には妥当なものであるが，最悪例というものは多くの場合まれなものであり，かつ実際に処理すべき問題の規模には制限があるであろうから，実用的には完全な評価方式であるとは言えない．このうち，最悪複雑度評価の欠点を補うものとして平均複雑度による評価方式があるが，この評価方式は，グラフの問題においては現在のところ最短路問題に対して行なわれているだけである[36]．従って，この評価方式に関する研究およびその実践は，今後残された重要な問題の1つである．なお，理論的評価は悪いが実際に処理すべき問題の規模内では速いアルゴリズムと，理論的評価は良いがその規模内では遅いアルゴリズムとはどちらが良いアルゴリズムであるか，さらに，それをどのような手法で判定するかという問題については，そのアルゴリズムが適用される個々の問題において独自に議論されるべきであろう．

第3章においては，有向グラフのすべてのサイクルを列挙する新しいアルゴリズムを提案した．n，m，およびcをそれぞれ有向グラフの頂点，有向辺，およびサイクルの個数とすれば，ここで提案したアルゴリズムは$O((n+m)\cdot(c+1))$の時間複雑度および$O(n+m)$のスペース複雑度を持っており，かつ，サイクルを見い出す際に探索すべき辺の個数（探索範囲）が従来のアルゴリズムよりはるかに制限されている．この探索範囲を制限することは，より良い時間複雑度$O(\gamma+n+m)$を持つアルゴリズムを見い出す1つの接近法であったが，ここで提案したアルゴリズムではこのオーダの時間複雑度を得ることはできなかった．ただし，γは各サイクルの長さの総和である．従って，この$O(\gamma+n+m)$の時間複雑度を持つアルゴリズムを見い出すことは，今後に残された問題である．このようなアルゴリズムを得るための接近法としては，探索範囲を制限するという方法以外に，3.3.1節において述べた[21]のアルゴリズムによる方法が考えられる．[21]のアルゴリズムには誤りがあったが，それをblocking-unblocking技法以外の技法で解決することができれば，$O(\gamma+n+m)$のアルゴリズムへの1つの手がかりが得られるであろう．

第4章においては，無向グラフのすべての極大独立頂点集合を列挙する新しいアルゴリズムを提案した．n，m，およびμをそれぞれ無向グラフの頂点，無向辺，および極大独立頂点集合の個数とすれば，ここで提案したアルゴリズムは$O(n\cdot m\cdot\mu+n+m)$の時間複雑度および$O(n+m)$のスペース複雑度を持

っており，このオーダの複雑度は従来のアルゴリズムでは得られなかったものである．また，このアルゴリズムによって，有向グラフのサイクルの列挙問題と同様，列挙問題に対する制限付きbacktracking技法の有効性が示された．ところで，列挙すべき各極大独立頂点集合の元の個数の総和をνとすれば，サイクルの列挙問題同様，$O(\nu+n+m)$の時間複雑度を持つアルゴリズムが最良であるか，グラフの接続構造を調べるためには，どうしても辺の探索が必要であることから，このオーダのアルゴリズムを得ることは極めて困難であると考えられる．従って，まず$O((n+m)\cdot(\mu+1))$の時間複雑度のアルゴリズムを見い出すことが，今後に残された問題であろう．

さて，本文において考察しなかった列挙問題のうち，まだ効率の良いアルゴリズムが見い出されておらず，かつ工学的にも有用な問題として，無向グラフにおけるカットセットの列挙問題を挙げることができる[37]．この問題は，通信網の信頼度計算などに応用を持ち，この問題に対してもbacktracking技法を効率的なものにする適切な制限条件が見い出せるか否かは極めて興味深い．

謝　辞

本研究の全過程を通じて，直接理解ある御指導を賜わり，つねに励ましていただいた尾崎弘教授，樹下行三助教授ならびに白川功助教授に衷心より感謝の意を表する．

大学院前期，後期両課程において電子工学一般および各専門分野に関し御指導，御教示を賜わった電子工学教室小山次郎教授，中井順吉教授，寺田浩詔教授，児玉慎三教授，電子ビーム研究施設裏克己教授，塙輝雄教授，産業科学研究所松尾幸人教授，中村勝吾教授，角所収教授ならびに喜田村善一名誉教授に深謝する．

第４章について有益な御助言，御援助をいただいた愛媛大学有吉弘教授，三菱重工業株式会社高砂研究所井手幹生氏，さらに，スペース複雑度に関して有益な御示唆をいただいたスタンフォード大学Ｒ．Ｅ．Tarjan助教授に厚く感謝する．

本研究に関し，福井大学谷口慶治教授，産業科学研究所翁長健次助教授，基礎工学部柏原敏伸助手，長崎造船大学川端信賢助手，徳島大学坂本明雄助手，日本電信電話公社武蔵野電気通信研究所松田潤博士，住友金属株式会社中央研究所山村春夫氏には本学大学院在学中に有益な御助言，御討論をいただき，心から謝意を表する．

また，琉球大学喜屋武盛基教授ならびに佐賀大学高松雄三助教授には，いろいろ御助言，御援助をいただき，厚く御礼申し上げる．

筆者の属している尾崎研究室の藤原秀雄助手，河田亨助手，戸松重一技官，大学院学生笹尾勤氏，千葉徹氏，トラン・ディン・アム氏，久保登氏，また同研究室の藤田基美子嬢には種々の面で御協力いただいた．ここに記して感謝する次第である．

参考文献

[1] J.Edmonds, "Maximum matching and a polyhedron with 0,1-vertices," J. Res. Nat. Bur. Standards, Vol. 69B, Nos. 1 and 2, pp. 125-130 (1965).

[2] D.G.Corneil, "The analysis of graph theoretical algorithms," Proc. 5th S-E Conf. Combinatorics, Graph Theory, and Computing, pp. 3-38 (1974).

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

[4] E.Lawler, "Introduction to the complexity of algorithms," Applied Computation Theory (D.T.Yeh, ed.), Prentice-Hall, N.J., pp. 61-81 (1976).

[5] N.Christofides, "Graph Theory: An Algorithmic Approach," Academic Press, N.Y. (1975).

[6] 伊理, "グラフの理論とその応用[I]," 信学誌, Vol. 54, No. 12, pp. 1704-1710 (1971).

[7] S.A.Cook, R.A.Reckhow, "Time bounded rondom access machines," J. Comp. System Sci., Vol. 7, No. 4, pp. 354-357 (1973).

[8] 白川, "組合せ問題における計算複雑度解析," システムと制御, Vol. 20, No. 8, pp. 395-404 (1976).

[9] 樹下, "オートマトン入門," 朝倉書店, 東京, Ch. 2 (昭48).

[10] R.M.Karp, "Reducibility among combinatorial problems," Complexity of Computer Computations (R.E.Miller, J.W.Thatcher, eds.), Plenum Press, N.Y., pp. 85-103 (1972).

[11] P.Naur (editor), "Revised report on the algorithmic language ALGOL 60," Comm. ACM, Vol. 6, No. 1, pp. 1-17 (1963).

[12] 尾崎, 白川, "グラフとネットワークの理論," コロナ社, 東京, (昭48).

[13] N.Deo,"Graph Theory with Applications to Engineering and Computer Science," Prentice-Hall, N.J. (1974).

[14] T.Kamae, "Asystematic method of finding all directed circuits and enumerating all directed paths," IEEE Trans. Circuits Theory, Vol. CT-14, No. 2, pp. 166-171 (1967).

[15] 有吉, "有向道および有向閉路の一算法," 信学論(A), Vol. 52-A, No. 2, pp. 101-102 (昭44).

[16] J.C.Tiernan, "An efficient search algorithm to find the elementary circuits of a graph," Comm. ACM, Vol. 13, No. 12, pp. 722-726 (1970).

[17] H.Weinblatt, "A new search algorithm for finding the simple cycles of a finite directed graph," J. ACM, Vol. 19, No. 1, pp. 43-56 (1972).

[18] R.E.Tarjan, "Enumeration of the elementary circuits of a directed graph," SIAM J. Comput., Vol. 2, No. 3, pp. 211-216 (1973).

[19] A.Ehrenfeuchet, L.D.Fosdick, L.J.Osterweil, "An algorithm for finding the elementary circuits of a directed graph," Tech. Rept. No. CU-CS-024-73, Dept. of Computer Sci., Univ. of Colorado (1973).

[20] 築山, 白川, 尾崎, "有向グラフのすべてのサイクルを求めるための一手法," 信学論(A), Vol. 58-A, No. 4, pp. 196-203 (1975).

[21] J.L.Szwarcfiter, P.E.Lauer, "Finding the elementary cycles of a directed graph in O(n+m) per cycle," Tech. Rept. No. 60, Computing Lab., Univ. of Newcastle upon Tyne (1974).

[22] D.B.Johnson, "Finding all the elementary circuits of a directed graph," SIAM J. Comput., Vol. 4, No. 1, pp. 77-84 (1975).

[23] R.C.Read, R.E.Tarjan, "Bounds on backtrack algorithms for listing cycles, paths, and sppanning trees," Networks, Vol. 5, No. 5, pp. 237-252 (1975).

[24] S.Tsukiyama, I.Shirakawa, H.Ozaki, "A survey: Generating all the Cycles of a digraph," Proc. 2nd USA-JAPAN Computer Conf., pp. 92-96 (1975).

[25] P.Mateti, N.Deo, "On algorithms for enumerating all the circuits of a graph," SIAM J. Comput., Vol. 5, No. 1, pp. 90-99 (1976).

[26] R.E.Tarjan, "Depth-first search and linear graph algorithms," SIAM J. Comput., Vol. 1, No. 2, pp. 146-160 (1972).

[27] C.Berge, "The Theory of Graphs and Its Applications," John Wiley and Sons, N.Y. (1962).

[28] S.R.Das, D.K.Banerji, A.Chattopadhyay, "On control memory minimization in microprogrammed digital computers," IEEE Trans. Computer, Vol. C-22, No. 9, pp. 845-848 (1973).

[29] M.C.Paull, S.H.Unger, "Minimizing the number of states in incompletly specified sequential switching functions," IRE Trans. Electronic Computers, Vol. EC-8, No. 9, pp. 356-367 (1959).

[30] P.M.Marcus, "Derivation of maximal compatibles using boolean algebra," IBM J., Vol. 8, No. 5, pp. 537-538 (1964).

[31] J.G.Auguston, J.Minker, "An analysis of some graph theoretical cluster techniques," J. ACM, Vol. 17, No. 4, pp. 571-588 (1970).

(Correction)

G.D.Mulligan, D.G.Corneil, "Corrections to Bierstone's algorithm

for generating cliques," J. ACM, Vol. 19, No. 2, pp. 244-247 (1972).

[32] E.A.Akkoyunlu, "The enumeration of maximal cliques of large graphs," SIAM J. Comput., Vol. 2, No. 1, pp. 1-6 (1973).

[33] C.Bron, J.Kerbosch, "Finding all cliques of an undirected graph — Algorithm 457," Comm. ACM, Vol. 16, No. 9, pp. 575-577 (1973).

[34] R.E.Osteen, "Clique detection algorithms based on line addition and line removal," SIAM J. Appl. Math., Vol. 26, No. 1, pp. 126-135 (1974).

[35] J.W.Moon, L.Moser, "On cliques in graphs," Israel J. Math., Vol. 3, No. 1, pp. 23-28 (1965).

[36] P.M.Spira, "A new algorithm for finding all shortest paths in a graph of positive arcs in average time $O(n^2 \log^2 n)$," SIAM J. Comput., Vol. 2, No. 1, pp. 28-32 (1973).

[37] A.Martelli, "A gaussian elimination algorithm for the enumeration of cut sets in a graph," J. ACM, Vol. 23, No. 1, pp. 58-73 (1976).