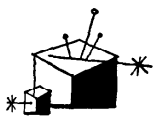
講　座



最近のグラフ理論とその応用(3)†

大　附　辰　夫††

5. グラフの問題の計算機による処理技法

グラフの問題は“組み合せ問題”の一種である．組み合せ問題とは有限個の要素から成る離散集合の性質に関する問題の総称であり，基本的には，①ある性質を満たす離散集合を列挙するという**数え上げ問題**，②組み合せ的な拘束条件の下である目的関数の値を最小(大)にするという**計画問題**および③ある性質を満足する離散集合が存在するか否かを判定するという**判定問題**の三つに分類される．ところで数え上げ問題においては，どんな解法を用いても，少なくとも数え上げの対象の数（例えば節点数nの完全グラフの木の数はn^{n-2}である）に比例するだけの計算の手間を要すると考えられる．したがって問題の規模が少し大きくなっただけで数え上げの対象が天文学的な量となってしまい，すぐに大型計算機の処理能力の限界を越えてしまう．このような意味で，数え上げ問題を議論の対象から除外することとする．

グラフの問題においては対象とする組み合せの数は有限であるので，手間をいとわなければ，すべての組み合せを“虱潰し”に調べることによって，必ず解が得られることになる．しかし，計算機の処理能力の限界を考慮すれば，個々の問題について「どの程度規模の大きなグラフまで処理できるか？」という立場からアルゴリズムの優劣を論ずることは実際的意味において重要である．一方，グラフの問題の中には，かなり大規模なグラフも扱えるようなアルゴリズムが存在するものと最適解を保証するアルゴリズムでは小規模なグラフしか扱えないものとがある．実際の応用面から抽出されるグラフ（ネットワーク）の問題の多くは後者に属する．したがって解の良さ（最適解への近さ）と処理時間あるいは扱えるグラフの大きさの限界との間のトレードオフを念頭においてアルゴリズムを工夫するということも重要になって来る．以下このような立場から議論を進めることにする．

5.1 アルゴリズムの複雑度

アルゴリズムの“良さ”を判断する際の最も基本的な要因は処理時間と記憶容量である．現在の大容量内部記憶を持つ計算機を前提とすると，解ける問題の規模の限界が記憶容量よりもむしろ処理時間によって規定されることが多い．又，多くのグラフの問題に対して，処理速度の面で良いアルゴリズムは所要の記憶装置の大きさの面でも良いことが多いので，主として処理時間を対象として議論を進めることにする．

処理時間が対象とする問題の規模nの関数$T(n)$で表わされるとき，$T(n)$を使用しているアルゴリズムの**時間複雑度** (time complexity) あるいは単に**複雑度**という．アルゴリズムの“良さ”を評価する際，$T(n)$そのものよりも，nの増大に対する$T(n)$の漸近的性質の方が重要な意味を持つことが多い．例えば，C_1, C_2, C_3を定数として$T(n)=C_0+C_1n+C_2n^2$ならば「そのアルゴリズムの(**漸近**)**複雑度**は$O(n^2)$である」あるいは「オーダn^2のアルゴリズムである」という表現を用いる．

与えられた問題に対して規模nを固定してもアルゴリズムの複雑度$T(n)$は一意に定まらない．何となれば，規模nの個別問題は無数に存在するからである．ここで**問題**とは，“グラフの木を求める”とか“グラフの彩色数を求める”などのことで，**個別問題** (an instance of the problem) とはそのような問題においてグラフの接続関係や節点，あるいは枝の重みなどの入力データを具体的に指定したもののことである．さて，複雑度$T(n)$を測るときの規準として，規模nのすべての個別問題に対する，① 平均値を取るという考え方と，② 最悪値を取るという考え方がある．どちらかといえば，前者の方が実用に則していることが多い．ところが，複雑度に関する従来の研究成果のほとんどは後者の評価規準を前提にしている．というのは，$T(n)$の平均値を求めるためには，入力データの確率分布を仮定しなければならず，現実的な仮定をすると理論的に扱うのが困難になるからである．

† Recent Development in Graph Theory and Its Applications (3) by Tatsuo OHTSUKI (Central Research Laboratories, Nippon Electric Co., Ltd.).

†† 日本電気(株)中央研究所

表-2 クラスPのグラフの問題

問題	漸近複雑度
平面グラフの判定	$O(n)$
グラフの連結成分への分解	$O(m)$
グラフの非可分成分への分解	$O(m)$
グラフの強連結成分への分解	$O(m)$
三角化グラフの判定	$O(m)$
区間グラフの判定	$O(m)$
重みの和最小の木	$O(n^2)$
1点から全点への最短径路	$O(n^2)$
2部グラフの最大マッチング	$O(n^{5/2})$
全節点間の最短径路	$O(n^3)$
節点の枝による最小被覆	$O(n^3)$
2節点間の最大フロー	$O(n^3)$

(n: 節点の数, m: 枝の数)

表-3 RAM, RASP プログラムの命令セット

	オペレーションコード	アドレス
1	LOAD	オペランド
2	STORE	
3	ADD	
4	SUB	
5	MULT	
6	DIV	
7	READ	
8	WRITE	
9	JUMP	レーベル
10	JGTZ	
11	JZERO	
12	HALT	な し

アルゴリズムの複雑度の概念を用いてグラフの問題を“やさしい”問題と“難しい”問題に分類することができる．“やさしい”問題とは，複雑度 $T(n)$ が問題の規模 n の多項式で与えられるような求解アルゴリズムが存在するもののことである．“難しい”問題とは，そのようなアルゴリズムが存在しない（と考えられている）もの，すなわち最上のアルゴリズムを持ってしても複雑度が $O(n!)$, $O(2^n)$ などとなってしまうもののことである．例えばグラフのオイラー閉路(2.2参照）の存在を判定する問題は“やさしい”問題であるのに対し，ハミルトン閉路（2.2 参照）の存在を判定する問題は“難しい”問題である．

“やさしい”問題と“難しい”問題との類別に関する詳細な議論は 5.4 にゆずることにするが，代表的な“やさしい”グラフの問題とこれに対して最善とみなされているアルゴリズムの複雑度を表-2 に掲げておく[13),30)~39)]．興味あるグラフの問題は数多く存在するが，そのほとんどは“難しい”問題であると思っておけば無難である．

5.2 計算過程のモデル化

アルゴリズムの複雑度や問題の難しさについて，実際の計算機の処理と対応付けて議論すると極めて複雑になってしまう．そこで，議論の展開が現実と遊離しない範囲内で，現実の計算機を抽象化，単純化したモデルを導入する必要がでてくる．

現実の計算機の処理能力をほぼ忠実に反映した抽象モデルとして **RAM** (Random Access Machine), **RASP** (Random Access Stored Program Machine) などが提案されている[40)]．これらは表-3 に掲げる基本命令——アセンブラの経験者ならばこれらの意味は明らかであろう——を実行することができ，その実行時間はある定数値で抑えられるという性質を持った計算機モデルである．現実の計算機はもっと豊富な命令セットを持っているが，いずれの命令も表-3 にあるものを（データの値と独立な）定数回繰り返すことによってシミュレートできる．RAM と RASP の違いは，前者はプログラムを実行中にプログラム自身を変更できない代りに間接アドレスの機能を持つのに対して後者はその逆である，ということである．しかし一方を前提としたアルゴリズムから，漸近複雑度を変えることなしに他方を前提としたアルゴリズムに変換できるという意味で，両者は同等の能力を持つ．

RAM も RASP も現実の計算機と同じように，次のような意味で能力が限定されている．

i）入力データ，内部データ，出力データのいずれも2進化符号で表現される．

ii）並列処理できる演算レジスタの数は有限である．

iii）一定時間内に命令が実行できるデータのビット長は有限である．

結論として，RAM も RASP もアルゴリズムの漸近複雑度を論ずる限りにおいては，現実の計算機と同等の能力を持つような抽象モデルである．

アルゴリズムの複雑度を見積る際には，各々の命令を実行するために要する時間とそれらが実行される回数が本質的な要因となる．もし，対象とする問題で扱われる整数が1個のレジスタに格納できる程度の大きさならば，すべての命令の実行時間はある定数であると考えてよい．このような評価規準を**一様コスト** (uniform cost) という，十分大きな整数を扱う問題においては，表-3 の 1～8 の命令の実行時間は処理するビット系列の長さに比例すると考えるのが現実的である．ある整数 i の値が，

$$2^{n-1} \leqq |i| < 2^n \qquad (5.1)$$

の範囲にあれば,

$$n-1 \leqq \log_2 |i| < n \qquad (5.2)$$

となるので，命令 1～8 の実行時間は $\log_2|i|$ に比例するとみなされる．このような評価規準を**対数コスト** (logarithmic cost) という．対数コストによる評価の方が一様コストによる評価よりも厳密であるが，本質的な差が出ない場合――ほとんどのグラフの問題ではそうである――には，単純な一様コストによる評価を採用して差しつかえない*.

オートマタ理論で良く知られている**チューリング機械** (Turing Machine, 以後 **TM** と略す) は現実の計算機と“ほとんど同等”の処理速度に持つという前提のもとに，その機能を最大限に原始的にした代数的モデルである．ここで“ほとんど同等”とは，「TM を前提とした複雑度が問題の規模 n の多項式で与えられるとき，しかもそのときに限り，現実の計算機を前提とした複雑度も n の多項式で与えられる」[42] という意味である．

TM の厳密な定義や詳細な議論[43]については，本文の主題を論ずる上で必要ないので省略する．ただ，TM は，多項式オーダの二つのアルゴリズムの優劣を比較する上では不適当であるが，多項式オーダのアルゴリズムであるか否かを議論するためにはこれで十分であることを強調しておく．

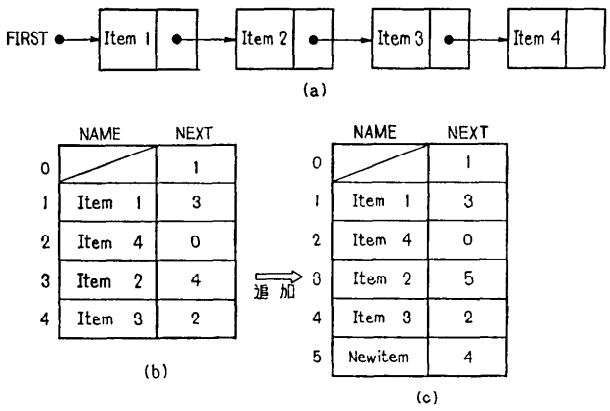
特定のアルゴリズムの能率を論ずる際には RAM または RASP を前提とするのが適切であるが，個々の命令をいちいち記述したのでは複雑になりすぎる．そこで，アルゴリズムを RAM (RASP) プログラムとして具体化する手順が明らかである限り，それをなるべく抽象的なレベルで記述できるような手段が要求される．このような意味から，アルゴリズム記述用の疑似言語として **Pidgin ALGOL**[42] が提案され，アルゴリズムを扱った論文などにおいて広く用いられるようになって来た．**図-28**(d)および**図-30** に例を挙げてある．Pidgin ALGOL のキーワードや文法は ALGOL のそれと同じであるが，集合やグラフを変数としたり，数式や自然言語をステートメントとして挿入したりすることもできる点が ALGOL と異なる点である．もちろん RAM (RASP) プログラムとして具体化する手順が明らかでなければならない．

5.3 基本的な処理技法

グラフの問題を扱うためのアルゴリズムとその中で参照するデータ構造は多種多様であり，問題ごとに専用の技法が工夫されている．しかし，それらはいくつかの基本的なデータ構造や処理手順を組み合せたり，変形したりしたものである．ここでは，多くのグラフの問題に共通に用いられている基本的な技法とそれらの一部の応用例について解説する．

5.3.1 リスト構造

有限集合の要素を追加したり削除したりする際に便利なデータ構造として**リスト** (list) がある．**図-28**(a)は 4 個の要素から成る集合を表現するためのリストの概念図であり，集合の各々の要素がその名前と次の要素の名前の記憶場所を示す**ポインター**とによって構成されている．これを計算機のデータ構造として具現するには，例えば図-28(b)に示すように二つの 1 次元配




```
procedure  INSERT (ITEM, FREE, POSITION) :
begin
        NAME [FREE] ←— ITEM ;
        NEXT [FREE] ←— NEXT [POSITION] ;
        NEXT [POSITION] ←— FREE
end
```

(d)

図-28　リストとその実現法

* 整数の行列を扱う組み合せ問題においては注意を要する[41].

列を用いれば良い，この構造を用いれば，リストの中の任意の位置に新しい要素を挿入したり，逆に任意の要素を削除したりするための処理時間は定数値で抑えられる*. 図-28(c)は新しい要素 Newitem を図-28(a)における Item 2の次の位置に挿入した際の結果を示す．一般に (ITEM という名前の) 新しい要素を POSITION という場所に記憶されている要素の (リスト上において) 次に挿入し，それを FREE という場所に記憶するためには，図-28(d)に示される手順を用いれば良い．例えば 図-28(b)の状態に対して INSERT (Newitem, 5, 3) を実行すれば，図-28(c)の状態となる．

特定の要素を削除するためには，ポインターを両方向に付けた方が便利なこともある．例えば，図-28(a)のリストから Item 2を削除したい場合，Item 1のポインターを Item 3が記憶されている場所を指すように変更しなければならないので，一つ手前の要素 (Item 1) を指す逆方向のポインターも必要になってくる．このような意味での削除を伴うアルゴリズムに対しては，図-28(b)において逆方向ポインターを記憶するための1次元配列を追加した**双方向リスト**による表現が適している．

5.3.2 グラフの表現法

グラフ——枝の数を m，節点の数を n とする——の接続関係を計算機の中に記憶する際に，接続行列や隣接行列 (**2.1** 参照) を用いることは，拙劣な方法である．というのは，その場合，$O(n^2)$ あるいは $O(mn)$ の大きさの記憶装置を必要とし，そのような行列を作ったり，参照したりするだけのために $O(n^2)$ あるいは $O(mn)$ の手間を要するからである．これに対して，下記のリストを応用したデータ構造を用いれば，記憶装置の大きさ，それを作成するための手間は共に $O(m+n)$ である．

図-29(a)の有向グラフにおいて，各々の節点から出ている枝 (あるいはその終点) の集合をリストで連結すれば**図-29**(b)のようになる．これを，二つの1次元アレイを用いて表現すれば**図-29**(c)のようになる．ここで節点および枝には通し番号が付けられていて**，記憶場所 l には枝 (または節点) l に関する情報が記憶されるものとする．例えば節点4から出ている枝を調べたい場合，NEXT [4]=8 によって先ず枝8が見つけられる．次に HEAD [8]=2 によってその終点は節点2であることがわかる．また NEXT [8]=9 によって，枝9も節点4から出ていることが分る．更に，NEXT [9]=0 によって，これ以上節点4から出ている枝はないことがわかる．

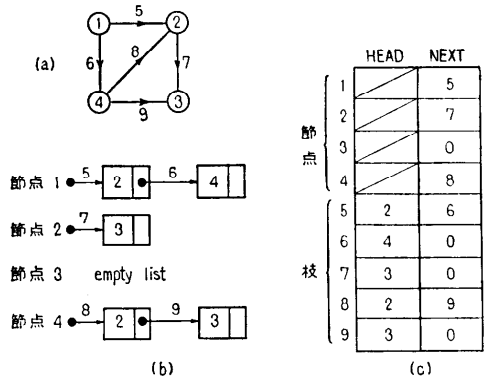


図-29 有向グラフのリフト表現

図-29 に示したデータ構造はグラフの接続関係を表現するための最も基本的な部分である．個々のグラフの問題に対しては，冗長であっても，処理能率を高めるためにより多くの1次元配列から成るデータ構造を用いることが多い．例えば，枝の削除を伴う問題では，各節点に接続している枝の集合を双方向リストによって表現したほうが都合が良い．又，各々の枝ごとに，その終点だけでなく，始点も直接参照できるための1次元配列を追加する場合もある．

無向グラフを表現する場合には，各々の枝を互いに逆方向の二つの仮想的な有向枝の並列接続とみなして上記のデータ構造を用いれば良い．そして，枝 i を表わす二つの仮想的な有向枝を記憶場所 i と $(m+i)$ に記憶するようにすれば，枝の削除の操作も容易に行える．

5.3.3 グラフの探索

グラフを扱うための能率良いアルゴリズムのほとんどは，「一つの節点から出発して，それに接続している枝をたどりながら，到達した節点にマークまたはラベルを付けて行く」という**探索** (search) を基本としている．出発点と同じ連結成分に属するすべての点はこの探索によって到達することができるので，マークを付けられていない節点が残ったら，その一つを新たな出発点として探索を繰り返すことによって，グラフの各連結成分を抽出することができる．また，未だマ

---

* 要素の名前を記憶する1次元配列だけで集合を表わすと，追加あるいは削除の1回の操作に要する手間は集合の要素数に比例する．

** 節点や枝に (通し番号でない) 固有名が付けられていることがあるが，この場合は外部名と内部参照番号との対応表およびこれに基づく変換ルーチンが必要である．

ークされていなかった節点を新しくマークするたびに，その直前にたどった枝を区別するという操作を付加することによってグラフの一つの木が求められる．探索を最後まで行うためにはすべての枝を調べることになるが，**5.3.2** に述べたデータ構造を前提とすれば，各々の枝を調べる処理の手間が定数で抑えられるので，探索アルゴリズムの漸近複雑度は $O(m)$ である．

集合のリストによる表現，グラフの節点ごとの隣接リストによる表現およびここで述べた探索を応用すれば，オイラー閉路の作図（**2.2** の**図-4** 参照）を $O(m+n)$ の手間で実行することができる．先ず，同じ枝を2度通ることがないように，一度通った枝を取り除く操作が必要であるので，グラフの接続関係を双方向リストによって表現する必要がある．次に，$\hat{P}_1=P_1$, $\hat{P}_2=\hat{P}_1 \cup P_2$, $\hat{P}_3=\hat{P}_2 \cup P_3$, …などのオイラー閉路の一部を枝の系列として記憶するためにリストが用いられるが，これまでに出来た閉路と新しく抽出された閉路を合成して新しい枝の系列を生成するために，枝の系列を円順列として記憶する必要がある．そのためには，集合の中の最後の要素(枝)から最初の要素(枝)を指すためのポインターを付ければ良い．オイラー閉路の作図が $O(m+n)$ の手間で実行できることを実証するためには，より詳しい議論が必要であるが，以上述べたことが考え方の基本であるので，読者自身で確められたい．

探索の途中段階において，次に調べるべき枝はすでにマークが付けられた節点に接続している新しい枝ならどれでも良いという自由度があるが，特に最も早くマーク付けされた節点から出ている枝を優先するような探索を**横型探索** (breadth-first-search, 以後 BFS と略す) という．これに対して最も新しくマーク付けされた節点から出ている枝を優先するような探索を**縦型探索** (depth-first-search, 以後 DFS と略す) という．

図-30 は無向グラフ $G=(V, E)$ に DFS を適用して一つの木 $T \subset E$ を求める手順を Pidgin ALGOL によって記述したものである．ここでグラフ G の接続関係は，各々の節点 v についてそれに隣接している節点を結んだリスト $L(v)$ によって表現されているものとする．又，同じ枝を二度調べないために，各々の節点 $v \in V$ について，$L(v)$ をどこまで走査したかを憶えておくための整数変数を用意することを前提とする．

```
begin
    T←φ;
    for all v∈V do mark v "new";
    while there exists a "new" vertex v in V do
        DFS(v)
        procedure DFS(v):
        begin
            mark v "old";
            for each vertex w on L(v) do
              if w is "new" then
                 begin
                   add (v,w) to T;
                   DFS(w)
                 end
        end
end
```

図-30　DFS による木の生成法

BFS でも DFS でも，それ自体の能率はほとんど同じであるが，それを応用する問題によっては，一方が他方よりも有利であることがある．例えば，グラフが三角化グラフあるいは区間グラフであるか否かを $O(m)$ の手間で判定するアルゴリズム[38]は BFS を前提としている．一方グラフの平面性を $O(n)$ の手間* で判定するアルゴリズム[37]や $O(m)$ の手間でグラフを非可分成分や強連結成分に分解するアルゴリズム[32]は DFS を前提としている．

5.4　組み合せ問題の分類

5.1 に述べたように，グラフの問題は“やさしい”問題と“難しい”問題に分類されるが，ここではこの分類に対する理論的背景を解説した後に，この理論から，「“難しい”グラフの問題に対しては，グラフが大規模になったら，最適解を求めることは断念した方が良い」という悲観的結論が導かれる，ことを指摘する．

5.4.1　バックトラック

図-31(c)は**図-31**(a)のグラフのハミルトン閉路の存在を，あらゆる組み合せを虱潰しに調べることによって，判定する為の**探索木**である．図-31(a)は節点 u を出発点として順番に接続している枝を探索して行く様子を示している．図-31(c)に示す探索木の各々の分岐点において，選ぶ枝に関する自由度があるが，図-31(a)では枝1, 8, 5, 3 が選ばれて次に節点 y に接続している枝を選ぶ段階が示されている．次に選ぶことができる未探索の枝は2と6であるが，いずれを選んでも節点 z を含まない（すなわち，ハミルトン閉路でない）閉路が形成されて“行き止まり”となってしまう．しかし，“行き止まり”に達したからといっ

* 平面グラフにおいて，並列枝がないとすれば，$m \leq 3(n-2)$ という関係があるので，$O(m)$ と $O(n)$ は等価である．

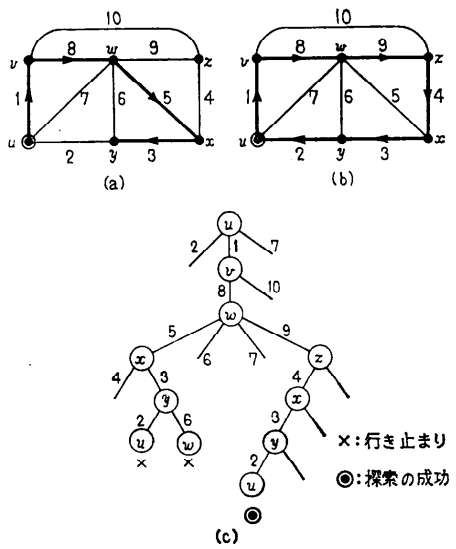


図-31　ハミルトン閉路の探索

て，ハミルトン閉路が存在しないことにはならない．例えば，探索木において分岐点wまでもどって節点wに接続している枝として9を選べば，図-31(b)に示すように，最後にハミルトン閉路，

$P=\{1, 8, 9, 4, 3, 2\}$

が求まる．この手法によってハミルトン閉路の存在を判定するためには，最悪の場合，探索木のすべての枝を調べる必要がある．グラフの節点の数をnとすれば，探索木の枝の数は少なくとも2^n個程度はあり，この数に比例する程度の処理時間を要することになる．

探索木の上でDFS (**5.3.3** 参照) を行いながら，あらゆる可能な組み合せを重複なく調べることによって組み合せ問題を解く手法を**バックトラック** (backtrack) という．アルゴリズムの能率のことを気にしなければ，すべての組み合せ問題において判定すべき場合の数は有限であるので，バックトラックを用いれば必ず解が求められることになる．

2.2 で述べたオイラー閉路を求めるための手順においても，本質的にはハミルトン閉路の問題におけるものと同じバックトラックを行っているのである．ところが，探索木の分岐点における枝分かれの自由度があっても，任意の枝を選びながら先に進み，"行き止まり"に達したらオイラー閉路が存在しないことを（定理 2.1 が）保証しているのである．このように，"やさしい"問題においては，"探索木の枝をすべて調べないでも解が求められる"ためその問題固有の性質が存在するのである．このような都合の良い性質が知られている問題，すなわち，"やさしい"問題はむしろ，"特殊"な部類に属する．

ハミルトン閉路の問題のように，本質的には探索木に沿った虱潰しをしなければならない．すなわち，処理時間が問題の大きさの指数関数のオーダで増大するような求解アルゴリズムしか知られていない問題を総称して，"難しい" (intractable) 問題という．

5.4.2　グラフの問題の符号化

アルゴリズムの複雑度を一般的に論じるために，与えられた問題を計算機に入力するとき，および処理結果を出力するときの形式を統一しておく．ここで対象とする問題は，入力として有限長のビット系列を与えて，"yes" 又は "no" の 1 bit の情報が出力されるような判定問題であるという前提を置く．もちろん，計画問題も扱うが，これは判定問題の繰り返しであると解釈する．例えば，節点数nのグラフの最小彩色問題は，$1<k\leq n$なる各々の整数kについて「グラフの節点をk色以内で色分けできるか？」を判定する問題の繰り返しであると解釈する．巡回セールスマン問題も，枝の重みが有理数である限り，「与えられた長さ以下のハミルトン閉路が存在するか？」を判定する問題の繰り返しと解釈できる．このようにほとんどの組み合せ計画問題は「片方を多項式オーダで解くアルゴリズムが存在すれば，しかも，そのときに限り，他方も多項式オーダで解ける」という意味で"等価な"判定問題に帰着できる．

判定問題の入力として，グラフ，整数のベクトル，集合，ブール代数表現などいろいろな構造が考えられるが，すべてビット系列として符号化できる．ここでは，具体的な符号化の方式を論ずる必要はない．有限長のビット系列の全体を$\mathcal{B}$とすれば，対象とする問題（およびその判定条件）を符号化した入力Lはその部分集合である．以後**問題**と言えば，ある有限長のビット系列の集合 $L\subset\mathcal{B}$ であると解釈する．問題Lに対する（正しい）アルゴリズムは任意の**個別問題** $x\in L$ に対して "yes" を出力し，任意の $x\in\mathcal{B}-L$ に対して "no" を出力する．

5.4.3　クラス P

前述のオイラー閉路の問題や表-2 に掲げた，いわゆる"やさしい"，問題のクラスは**クラス** P という概念によって特徴付けられる．ある問題 $L\in\mathcal{B}$ に対して，下記の性質を満たすアルゴリズムAが存在するとき，L は**クラス** P に属する（$L\in P$ と記す）という．

i) Aは任意の $x\in L$, $x\notin L$ に対して，それぞれ“yes”，“no”を出力する．

ii) 任意の $x\in L$ に対して，Aを用いて出力を得るのに要する時間が $T(|x|)$ 以下となるような多項式 T が存在する．ここで $|x|$ はビット系列xの長さである．

クラスPに対する厳密な定義は TM の概念を用いて与えられている[43]が，上記の表現と等価であり，この方が直観的に理解しやすい．また問題の規模とは，厳密には“符号化された入力の長さ”であるが，グラフの問題においては節点の数，枝の数，あるいはそれらの混合であると解釈しても大差はない．

5.4.4 問題相互の変換

数理計画の分野では，ある特殊な問題をより標準的な問題に変換してから処理するという技法がしばしば使われる．例えば，多種多様な組み合せ問題が0-1整数計画問題に変換されること[44]，更に，これが0-1ナップザック問題* に変換されること[45]などが知られている．このような“変換”の意味についての形式的な定義を考えてみよう．

二つの判定問題 L, $M\subset\mathcal{B}$ に対して，以下の性質を満たす写像 $f:\mathcal{B}\to\mathcal{B}$ が存在するとき LはMに(多項式時間で)**変換可能** (transformable) であるといい，$L\succ M$ と記す．

i) 任意の $x\in L$ に対して $f(x)$ の計算時間が $T(|x|)$ 以下となるような $|x|$ の多項式 T が存在する．

ii) $x\in L$ であるとき，しかもそのときに限り，$f(x)\in M$ となる．

この定義は，「$x\in L$ を判定する代りにまず $f(x)$ を計算してから $f(x)\in M$ を判定すればよい」という意味での変換 $L\succ M$ を規定している．又，「$L\succ M$ かつ，$M\in P$ ならば $L\in P$」であることも明らかである．この定義は抽象的過ぎるので，具体例を挙げよう．

n個のブール変数 $x_1, x_2, \cdots, x_n$ に対して $\{x_1, x_2, \cdots, x_n; \bar{x}_1, \bar{x}_2, \cdots, \bar{x}_n\}$ の各要素を**リテラル** (literal) といい，幾つかのリテラルの論理和を**クローズ**(clause)という．ここで幾つかのクローズ $C_1, C_2, \cdots, C_m$ の論理積，

$$f(x_1, x_2, \cdots, x_n; \bar{x}_1, \bar{x}_2, \cdots, \bar{x}_n)=C_1\wedge C_2\cdots\wedge C_m \tag{5.3}$$

に対して，適当に変数 $x_1, x_2, \cdots, x_n$ に“0”又は“1”を代入することによって f の値を“1”とすることができるか否かを判定する問題を**充足可能性** (satisfiability) 問題という．例えば，

$$f=(x_1\vee x_2\vee\bar{x}_4)\wedge(\bar{x}_1\vee\bar{x}_3)\wedge(\bar{x}_2\vee x_3)\wedge x_4$$

ならば，$x_1=x_4=$“1”，$x_2=x_3=$“0”と置くことによって $f=$“1”となるので，このfは充足可能である．

無向グラフ G および互いに分離した節点対の集合 $\{(s_1,t_1), (s_2,t_2), \cdots, (s_m,t_m)\}$ が与えられて，互いに節点を共有せずに (s_j,t_j); $j=1,2,\cdots,m$ を結ぶm個の道が存在するか否かを判定する問題を**離散的同時フロー** (discrete multicommodity flow) 問題という．**図-32** は，このような $m=3$ 個の道が存在する例を示す．

充足可能性問題が離散的同時フロー問題に変換可能なことは比較的容易に証明できる[41]．証明の基本は，各々のクローズ C_j; $j=1, 2, \cdots, m$ が道の端点の対 (s_j,t_j) に対応するような無向グラフ G を導入することである．又，各々のブール変数 x_i; $i=1, 2, \cdots, n$ は次のような G_i の部分グラフに対応する．例えば，変数 x_i がクローズ C_2, C_4 の中に表われ，$\bar{x}_i$ がクローズ C_1, C_3, C_5 の中に表われるとすれば，部分グラフ G_i の構造は**図-33** のようになる．前者の問題において $x_i=$“1”，“0”とすることは，それぞれ後者の問題において図-33 の2本の水平な道，3本の垂直な道を選ぶことに対応する．ここで同じ部分グラフにお

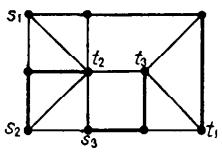


図-32 離散的同時フロー問題の解

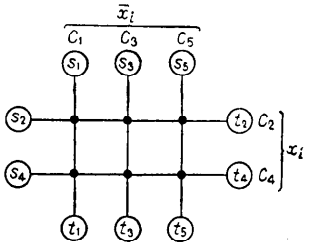


図-33 変数 x_i に対応する部分グラフ G_i

* 正の整数から成る二つのベクトル $(a_1, a_2, \cdots, a_m)$, $(c_1, c_2, \cdots, c_m)$ および正の整数 d を与えて，

$$\sum_{i=1}^{m} a_i x_i\to\max.;\ \sum_{i=1}^{m} c_i x_i\leqq d$$

とするような 0-1 ベクトル $(x_1, x_2, \cdots, x_m)$ を見つける問題を**0-1 ナップザック問題**という．

いて水平な道と垂直な道は必ず交わることを注意しておく．全体のグラフ G は，すべての G_i における各々の (s_j, t_j) の対を一つの対にまとめることによって得られる．前者が後者に変換可能であることを証明するためには，①ブール関数が充足可能であるとき，しかもそのときに限り，グラフ G が条件を満たす m 個の道の組を持つこと，および②前者の入力から多項式オーダの時間内に後者の入力を求めるアルゴリズムが存在することを証明すればよい．

5.4.5 クラス *NP*

我々が対象とするほとんどの組み合せ問題は“難しい”問題であるといわれているが，これらの問題の多くは**クラス *NP*** (nondeterministic polinomial-time) という概念によって特徴付けられる．厳密な定義は**非決定性 TM** の概念を用いて与えられる[43]が，必要以上に複雑であるので，ここではバックトラックの概念を用いて直観的に理解しやすい表現を与える．

判定問題 $L\subset\mathcal{B}$ が**クラス *NP*** に属する ($L\in NP$ と記す) とは，$x\in L$ か否かの判定をバックトラックによって行うことができ，その探索木の深さが $|x|$ の多項式で抑えられることである．すなわち“並列処理を無限に行えるような計算機があれば，多項式時間内で判定できる”ということと等価である[41]．

Cook はクラス NP の概念に関して次のような画期的な事実を導いた[46]．

[定理 5.1] クラス NP に属するすべての問題は充足可能性問題に変換可能である．

この定理によれば，もし充足可能性問題を多項式時間で解くアルゴリズムが発見されれば，クラス NP に属するすべての問題も多項式時間内で解けるという（信じられないような）結果が実現することになる．一方，$P\subset NP$ であることは定義から明らかである．したがって，次の結果が導かれる．

[系] [充足可能性問題]$\in P$ であるとき，しかもそのときに限り $P=NP$ である．

5.4.6 クラス *NP* 完全

判定問題 $L\subset\mathcal{B}$ が次の性質を満たすとき，L は NP 完全 ($L\in NP$ 完全と記す) であるという[46),47)]．

i) $L\in NP$

ii) 任意の $M\in NP$ は L に変換可能 ($M\succ L$) である．

この定義と定理 5.1 より，次の結果が得られる[47)]．

[定理 5.2] NP 完全問題はすべて P に属するか，又はどれも P に属さないかのいずれかである．更に，前者が成立することは $P=NP$ と等価である．

すなわち P, NP, NP 完全の間の包含関係を図示すれば，**図-34**(a)，(b)のいずれか片方のみが可能ということになる．

定理 5.1 より充足可能性問題は NP 完全である．又，充足可能性問題がある問題 $L\in NP$ (例えば，離散的同時フロー問題) に変換可能であれば，L も NP 完全である．一般に関係 $\succ$ は「$L_1\succ L_2$ 且つ $L_2\succ L_3$ ならば $L_1\succ L_3$ である」という性質を満たすので，既に NP 完全であることが証明されている問題 M を適当に選んで $M\succ L$ であることが示せれば ($L\in NP$ はたいてい自明である)，L も NP 完全であることになる．充足可能性問題を出発点として，既に多くの組み合せ問題が NP 完全であることが“いもづる式”に証明されている．**図-35** はその導出過程の一部を示している．これ以外にも NP 完全問題が文献 41), 42), 45)～49) などに列挙されており，又，次々と新しい

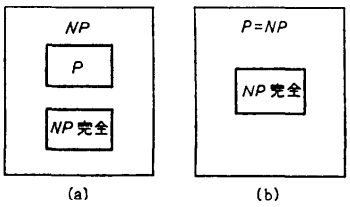


図-34 *P*, *NP*, *NP* 完全の間の包含関係

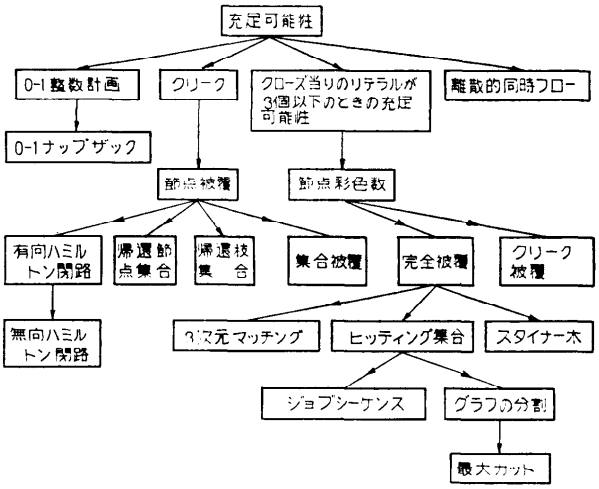


図-35 *NP* 完全問題の導出過程

NP 完全問題が発見されつつある.

定理 5.2 によれば，一つの NP 完全問題に対して多項式オーダのアルゴリズムが発見されれば，NP に属するすべての問題（ほとんどすべての組み合せ計画問題，判定問題）が多項式オーダで解けることになる．常識的に考えれば，このようなことは信じられない．すなわち定理 5.2 は「NP 完全問題に対しては多項式オーダのアルゴリズムが存在しない」ことに対する情況証拠を与えていると考えてもよい．言い換えれば，「ほとんどの組み合せ問題を解くことはあきらめるべきである」ということにもなる．

理論的には，$P=NP$ 又は $P\subsetneq NP$ のいずれかであるが，そのどちらであるかについては未解決である．しかし，「恐らく後者であろう」というのが多くの研究者の常識的見解である．

5.5 "難しい" 組み合せ問題に対するアプローチ

システムやネットワークの設計において介入する組み合せ問題のほとんどはクラス NP，すなわち "難しい" 問題であるが，比較的広範囲の組み合せ計画問題に適用できる一般的手法として，**動的計画法**[50]，**分枝限定法**[51]，**整数計画法**[52]などが利用されていて，実用的にもある程度の成果を収めている．これらの方法では，バックトラックによる "虱潰し" における無駄な探索を最大限省いて大幅な処理時間の短縮が成されているが，扱える問題の規模という意味ではすぐに限界に達してしまう．というのは，いくら無駄を省いても本質的には "虱潰し" であって，処理時間が問題の規模の多項式で抑えられるわけではないからである．すなわち，扱っている問題そのものがクラス NP に属するという本質的な原因に帰着されることになる．

前述の組み合せ問題に対する悲観的側面は次の前提条件の上に立っている．

i）アルゴリズムの複雑度は平均ではなく最悪の場合について評価される．

ii）計画問題において，真の最適解以外は受入れない．

iii）アルゴリズムはいかなる個別問題に対しても正しく働かなければならない．

しかし，実際の応用面で組み合せ問題を解く場合は，上記の前提条件をある程度くずしても支障がないことが多い．例えば，線形計画問題を解くためのシンプレックス法の手間は，最悪の場合は問題の規模の指数関数のオーダで増大するが，平均値の意味での計算の手間は多項式オーダであることが（理論的根拠は示されていないが）経験的に実証されている．すなわち，シンプレックス法は前提条件 i）をくずしても実用的には満足できるアルゴリズムの例である．ところが，これは特殊な例であって，多くの組み合せ問題においては，平均の意味においても能率良いアルゴリズムが知られていない．

前提条件 ii）に関して妥協した方法としては，**ビンパッキング問題***（NP 完全である）に対して，最悪相対誤差 $\varepsilon\leqq 2/9$ の範囲内の近似解を $O(n\log n)$ の手間で求めるアルゴリズムが提案されている[53]．しかし，グラフの節点の彩色問題を始めとして，多くの NP 完全な問題において，十分大きな誤差を許してもやはり NP 完全であるということも証明されている[54),55]．

前提条件 iii）に関して妥協した方法としては，巡回セールスマン問題や無向ハミルトン閉路問題などの解を確率的な意味で "ほとんど確実に" 求めるアルゴリズムが提案されている[56]が，このアプローチも極く一部の問題に対してしか成功していない．更に，「理論の中で仮定しているところの対象とする個別問題の確率分布が実体に合うか？」という疑問も含んでいる．

以上述べたように，"難しい" 問題に対するアプローチは種々試みられたが，決め手となるようなものは出現していない．一方では，システムやネットワークの設計問題に介入する個々の組み合せ問題に対しては，人間の経験と知恵によって問題固有の性質を利用して，結構能率が良くてかつ最適解に近い解を与えるようなアルゴリズムが考察されている．このようなアルゴリズムを総称して**ヒューリスティック算法**[57),58]という．現状では，多くの設計の場においてヒューリスティック算法が主体となっている．

今後の研究課題は，ヒューリスティック算法における誤差と計算の手間の評価に対して理論的根拠を付けること，および特定の問題ごとにヒューリスティック算法を考案するのではなく，ある程度統一的に扱えるようなアプローチを与えることなどであろう．

* $0\leqq a_i\leqq 1$ の体積を持つ n 個の物体 $a_1, a_2, \cdots, a_n$ を，容器の容量は 1 として，最小個の容器につめる問題を**ビンパッキング問題**という．

参考文献

(既出分への追加)

30) Seppänen, J. J.: Algorithm 399: spanning tree, Commun. ACM, Vol. 13, p. 621 (1970).

31) Whitney, V. K. M.: Algorithm 422: minimal

spanning tree, Commun. ACM, Vol. 15, p. 273 (1972).
32) Tarjan, R. E.: Depth-first search and linear graph algorithms, SIAM J. Comput., Vol. 1, p. 146 (1972).
33) Gavril, F.: Algorithms for minimum coloring, maximum clique and maximum independent set of a chordal graph, SIAM J. Comput., Vol. 1, p. 180 (1972).
34) Hopcroft, J. and Karp, R. M.: A $n^{5/2}$ algorithm for maximum matchings in bipartite graphs, SIAM J. Comput., Vol. 2, p. 225 (1973).
35) Hopcroft, J. and Tarjan, R. E.: Efficient algorithms for graph manipulation——Algorithm 447, Commun. ACM, Vol. 16, p. 372 (1973).
36) Karzanov, A. V.: Determining the maximal flow in a network by the method of preflows, Sov. Math. Dokl., Vol. 15, p. 434 (1974).
37) Hopcroft, J. and Tarjan, R. E.: Efficient planarity testing, J. ACM, Vol. 21, p. 549 (1974).
38) Rose, D. J. and Tarjan, R. E.: Algorithmic aspects of vertex elimination on graphs, SIAM J. Comput., Vol. 5, p. 266 (1976).
39) Booth, K. S. and Lueker, G. S.: Testing for the consecutiveones property, interval graphs and graph planarity using PQ-tree algorithms, J. Comput. & Syst. Sci., Vol. 13, p. 335 (1976).
40) Cook, S. A. and Reckhow, R. A.: Time-bounded random access machines, J. Comput. & Syst. Sci., Vol. 7, p. 354 (1973).
41) Karp, R. M.: On the computational complexity of combinatorial problems, Networks, Vol. 5, p. 45 (1975).
42) Aho, A. V., Hopcroft, J. E. and Ullman, J. D.: The design and analysis of computer algorithms, Addison-Wesley (1974).
43) Hopcroft, J. E. and Ullman, J. D.: Formal languages and their relation to automata, Addison-Wesley (1969).
44) Dantzig, G. B.: On the significance of solving linear programming problems with some integer variables, Econometrica, Vol. 28, p. 30 (1960).
45) Bradley, G. H.: Equivalent integer programs and canonical problems, Manage. Sci., Vol. 17, p. 354 (1971).
46) Cook, S. A.: The complexity of theorem-proving procedures, Proc. 3 rd ACM Symp. on Theory of Computing, p. 151 (1971).
47) Karp, R. M.: Reducibility among combinatorial problems in Complexity of Computer Computations (R. E. Miller and J. W. Thatcher, eds.), Plenum Press, p. 85 (1972).
48) Garey, M. R., Johnson, D. S. and Stockmeyer, L.: Some simplified *NP*-complete problems, Proc. 6th ACM Symp. on Theory of Computing, p. 47 (1974).
49) Sahni, S.: Computationally related problems, SIAM J. Comput., Vol. 5, p. 262 (1974).
50) Bellman, R. E.: Dynamic Programming, Princeton Univ. Press, Princeton (1957).
51) Lawler, E. L. and Wood, D. E.: Branch-And-Bound Method: A Survey, Operations Research, Vol. 14, No. 4, pp. 699-719 (1966).
52) Geoffrin, A. and Marsten, R.: Integer Programming Algorithms: A Framework and State-Of-The-Art Survey, Management Science, Vol. 18, No. 9, pp. 465-491 (1972).
53) Johnson, D. S.: Approximation algorithms for combinatorial problems, J. Comput. & Syst. Sci., Vol. 9, p. 256 (1974).
54) Sahni, S. and Gonzalez, T.: P-complete approximation problem, J. ACM, Vol. 23, p. 555 (1976).
55) Garey, M. R. and Johnson, D. S.: The complexity of near-optimal graph coloring, J. ACM Vol. 23, p. 43 (1976).
56) Karp, R. M.: The probabilistic analysis of some combinatorial search algorithms, ERL-Memo., No. 581, Univ. Calif., Berkeley (1976).
57) 大附：大規模組合せ問題におけるヒューリスティック算法，信学誌，Vol. 58, 4, p. 416 (昭50-04).
58) Lin, S.: Heuristic programming as an aid to network design, Networks, Vol. 5, p. 33 (1975).

(昭和55年1月21日受付)