講　座

設計自動化におけるグラフ理論と組み合わせ算法 (3)*

可児賢二** 大附辰夫***

4. 組み合わせ問題の複雑度

3章では，DA において我々が遭遇する問題の多くが，グラフの問題あるいはそれを自然に拡張した形での組み合わせ問題として表現できることについて述べてきた．以下の4～7章においては，これらの問題を計算機処理を前提として解くときの算法を中心に述べる．

さて，組み合わせ問題を大きく分類すると，ある性質を満足するものを列挙するという**数えあげ問題** (enumeration) と，組み合わせ的な拘束条件のもとで，ある目的関数を最小（最大）にするという**計画問題**（最適化問題）に分けられる．数えあげ問題においては，どんな算法を用いても少なくとも数えあげの対象となるものの数に比例するだけの計算の手間を要すると考えられる．例えば n 節点の完全グラフには n^{n-2} 個の木があり，すべての木を列挙するという問題においては，n が少し大きくなっただけで計算時間は非現実的な量に増してしまう．このような意味で以後数えあげ問題は対象としないこととする．実際，3章で述べた問題はすべて計画問題に属している．

ところで，計画問題は一般に次のように記述される．

$$\left.\begin{array}{ll} \text{目的関数} & f(x)\longrightarrow \text{最小(最大)} \\ \text{拘束条件} & x\in S\subset X \end{array}\right\} \qquad (4.1)$$

ここで X は対象とする空間，S はその部分集合で許容領域を示す．例えば最小木問題 (**2.4** 節参照) では，Xとして与えられたグラフの部分グラフの集合，S として木の集合，$f(x)$ として木に含まれる枝の重みの総和が相当している．実数の集合などを扱う通常の計画問題と組み合わせ計画問題との違いは，対象とする空間 X が連続集合であるか，離散集合であるかだけである．ところが組み合わせ計画問題は連続量を対象とする計画問題と違って，統一的，一般的に扱うことが困難である．その理由として次のようなことが挙げられる．

(A) 連続量を扱う計画問題において，重要な役割を果たしている微分，ラグランジェ乗数法などが殆んどの場合無力である．

(B) 局所最適解 (local optimum) が数多く存在する場合が多い．従って単純な傾斜法 (gradient 法) によって最適解が得られる可能性は低い．

(C) 問題の困難度の不連続性…条件，目的関数が少し変化すると急に難しくなり，同じ手法を使えない場合が多い．例えば最小木問題とスタイナ木問題の場合 (**2.4** 節参照)．

(D) 表現の多様性…同じ問題が数学的にはいろいろな形で表現できる．また逆に一見異なる問題が本質的には同じであったりする．

さて，組み合わせ問題の**複雑度** (complexity) は，問題の規模 n の増大に伴って計算の手間がどのように増大するかによって表わされる．計算の手間が例えば n の3次の多項式で評価できるとすれば，「その問題は $O(n^3)$ の問題である」とか「その問題の複雑度は n^3 である」などと表現する．特定の問題に対して，複雑度という言葉は現在までに知られている最良の算法を前提とし，組み合わせ的に最悪の場合の計算の手間という意味で用いられるのが普通である．ここで注意しなければならないことは，複雑度は計算の手間の増大の割合を表わす指標であって計算の手間の絶対量を表わすものではない，ということである．例えばグラフの一つの木を求める問題と非可分成分へ分解する問題の複雑度は共に m (m はグラフの枝の数) であるが，前者の方がはるかに簡単である．これは共に一次式であっても m の係数の大きさが異なることを意味する．

組み合わせ問題は，上に述べた複雑度の立場から**扱い易い** (tractable) 問題と**扱いにくい** (intractable) 問

* Graph Theory and Combinatoriol Algorithms for Design Automation by Kenji KANI (IC Division, Nippon Electric Co., Ltd.) and Tatsuo OHTSUKI (Central Research Laboratories, Nippon Electric Co., Ltd.)

** 日本電気 (株) 集積回路事業部
*** 日本電気 (株) 中央研究所

表-4.1 扱い易い問題（n：節点の数，m：枝の数）

問題	複雑度
平面性の判定	n
連結成分への分解	m
非可分成分への分解	m
強連結成分への分解	m
最小木	n^2
一点から全点への最短径路	n^3
2部グラフの最大マッチング	$n^{5/2}$
全節点間の最短径路	n^3
最小（枝）被覆	n^3

題とに区別して考えられる．扱い易い問題は，計算の手数が問題の規模nの多項式で評価できるものである．このクラスの問題においては，算法，データ構造，プログラミングの工夫によってnが数百，数千の大規模なものでも最適解が求められるのが普通である．2章あるいは3章で触れた問題の中で，このクラスに属する問題の複雑度を**表-4.1**に示した[6]（nを節点の数，mを枝の数とする）．

これに対して，計算の手数がnの階乗あるいは指数関数のオーダで増大するものを総称して扱いにくい問題とよぶ．実際に我々が遭遇する設計問題には種々様々な組み合わせ計画問題が介入し，その殆んどが扱いにくいものである（**表-4.2** 参照）．一方，扱いにくい計画問題の著しい特徴は，問題の規模がわずか増大しただけで急激に計算時間が増加し，容易に大型計算機の能力の限界を越してしまうということである．一例として巡回セールスマン問題（**2.3** 節参照）を例にとってみよう．n個の節点を持つ完全グラフには$(n-1)!/2$個のハミルトン閉路がある．$n=10$とすれば$(10-1)!/2=181,440$個のハミルトン閉路があるが，大型計算機を持ってすれば，全てのハミルトン閉路を虱潰し的に調べたとしても，計算時間はそれほど問題にならないであろう．ところが$n=20$とすると$(20-1)!/2\simeq 6\times 10^{16}$個のハミルトン閉路が存在し，一つのハミルトン閉路あたりの処理時間を10ナノ秒（超高速計算機を前提とした楽観的見積り）としても，

$$\frac{6\times 10^{16}\times 10^{-8}\text{秒}}{365\text{日}\times 24\text{時間}\times 3,600\text{秒}}\simeq 20\text{年} \qquad (4.2)$$

の計算時間を要することになる．この問題は有限集合を対象としたものであり，有限の手間で最適解が求められるという数学的意味においては解けない問題ではないが，現実的意味においては“虱潰し法”では解けないと考えるべきであろう．

表-4.2 扱いにくい問題

問題	参照
○ハミルトン閉路	2.3 節
（巡回セールスマン問題）	2.3 節
○最小帰還枝（節点）集合問題	2.8 節
○スタイナ木問題	2.4 節
（多点間配線問題）	3.6 節
○節点彩色数問題	2.6 節
（ICのブロック列間の配線問題）	3.6 節
（短絡テスト最小化問題）	3.7 節
○行列の被覆問題	2.7 節
（ブール関数の簡単化）	3.4 節
（最小故障検出系列）	3.7 節
（最小故障診断系列）	3.7 節
分割問題	3.5 節
配置問題	3.6 節
グラフの平面化問題	2.5 節
（ICの平面配線問題）	3.6 節
○最大クリーク	2.6 節
（連立方程式のピボッティング）	3.3 節
同形性の判定	2.2 節

ここで最近明らかになった興味ある結果を紹介しておこう．扱いにくい組み合わせ問題のいくつかは，Cook-Karp クラスとして区別することができ，それに属する一つの問題に対して，問題の規模nの多項式のオーダで最適解を求め得る算法が存在すれば，他の問題もそのオーダで解くことができることがわかっている[63][64]．表-4.2 の問題のうち，○印は Cook-Karp クラスに属することが証明されている．“この表に掲げた扱いにくい問題の殆んどが多項式の手間で解ける”というようまい話はありそうにもないので，この事実は“多くの扱いにくい組み合わせ計画問題に対して，その最適解を追求することはあきらめた方が良い”ことを示唆している．このような意味で，5章で述べる一般的解法や6章で述べるヒューリスティック算法が，実際の設計問題を解決する上で必要不可欠なものとなっている．

5. 最適解を求める一般的解法

扱いにくい組み合わせ問題の算法としては，6章で述べるヒューリスティック算法が広く用いられているが，本章では，最適解を追求する立場からの一般的手法として，ⅰ）動的計画法，ⅱ）分枝限定法，ⅲ）整数計画法，の三つについて簡単に触れておく．これらは比較的広範囲の組み合わせ計画問題に適用でき，実用的にもある程度の成果を収めているものである．

i) 動的計画法

動的計画法(dynamic programming)は Bellman[65][66] によって開拓され，制御問題，配分問題，スケジューリング問題など，広範囲の応用が精力的に研究された

最適化手法である．動的計画法は「最適方策は，最初の状態および決定が何であろうとも，残余の決定は最初の決定から生じた状態に関して最適方策を構成しなければならない」という，いわゆる最適性の原理に基づいている．最適性の原理はあくまでも“原理”であって，与えられた問題に最適性の原理が成立するかどうか，あるいは成立してもその後の具体的計算手順をどのように定めるかは個々の問題に対して個別に考察する必要がある．

一例として巡回セールスマン問題（**2.3** 節参照）を取り上げてみよう．グラフの節点（都市）の数を n，出発点を 0 とする．点 u に到達して，k 個の節点 $v_1, v_2, \cdots, v_k$ が未だ残っている段階において，もし出発点 0 から中間点 u まで最適の径路を選んだとすれば，u から残りの k 個の節点を通って 0 に戻る径路も最短のものでなければならない．そこで，$f(u; v_1, v_2, \cdots, v_k)$ を節点 u から k 個の節点 $v_1, v_2, \cdots, v_k$ を通って出発点 0 に戻るための最短径路長を表わすものとすれば，

$$l=f(0; v_1, v_2, \cdots, v_{n-1}) \tag{5.1}$$

が所要の最短ハミルトン閉路の長さを与えることになる．この値は，

$$\begin{aligned} &f(u; v_1, v_2, \cdots, v_k) \\ &= \min_{1\leqq i\leqq k} \{d(u, v_i)+f(v_i; v_1, \cdots, v_{i-1}, v_{i+1}, \\ &\quad \cdots, v_k)\} \end{aligned} \tag{5.2}$$

を反復して適用することによって求められる．ここで，$d(u, v)$ は 2 節点 u, v 間の距離を表わす．(5.2) 式の反復計算は，まず

$$f(u; v)=d(u, v)+d(v, 0) \tag{5.3}$$

を求めることから始まり，これを基にして $f(u; v_1, v_2)$ が求まる．これをくり返せば最後に (5.1) 式の l の値が求められる．この考え方に基づけば，巡回セールスマン問題は $n^2 \cdot 2^{n-1}$ のオーダで解けることが知られている[67)68)]．虱潰し法では，$n!$ のオーダであることを思えば相当の進歩であるが，それでも一般に最適解が求められるのは今のところ $n=20$ 前後が限界である．

一点から他のすべての節点への最短径路を求めるための最良の算法として知られている Dijkstra 法[32)]も同じ最適性の原理に基づいている．しかしこの方法は問題の性質を巧妙に利用しており，計算の手間は $O(n^2)$ に過ぎない．

ii) 分枝限定法

分枝限定法 (branch and bound method) の基本的発想は，直接解くことが難しい問題をより易しい何個かの問題に分解し，そのすべてを解くことによって間接的に元の問題を解こうとするものである[69)]．すなわち，(4.1) 式での問題の対象とする空間 X をいくつかの部分 $X_1, X_2, \cdots, X_m$ に分解し，それぞれの拘束条件を $S_k=S\cap X_k$: $k=1\sim m$ と考える．従って，

$$\left.\begin{array}{l} \text{目的関数} \quad f(x)\rightarrow\text{最小} \\ \text{拘束条件} \quad x\in S_k\subset X_k \end{array}\right\} \tag{5.4}$$

という問題をすべての k について解けば，元の問題が等価的に解かれたことになる．得られた部分問題がやはり大きすぎるときは，同様な分解操作が部分問題にも適用される．この操作を続けると，生成される部分問題は次第に小規模なものとなり，最終的にすべての部分問題が解かれる（**図-37** 参照）．例えば巡回セールスマン問題では，出発点を固定した場合に，これに隣接した節点のどれを 2 番目の節点に選ぶかによって部分問題に分解される．

ところで，分解操作を加えると部分問題の数がどんどん増加するという欠点がある．そこで，ある部分問題から原問題の最適解を得る可能性のないことが何らかの理由によって結論できれば，ただちにそれを考慮の対象から除く（**限定操作**）という考え方がとられる．すなわち，ある部分問題（(5.4) 式とする）に対して，その拘束条件 S_k を $\bar{S}_k$; $S_k\subset\bar{S}_k\subset X_k$ で置き代えて

$$\left.\begin{array}{l} \text{目的関数} \quad f(x)\rightarrow\text{最小} \\ \text{拘束条件} \quad x\in\bar{S}_k\subset X_k \end{array}\right\} \tag{5.5}$$

という緩和問題を考える．(5.4) 式，(5.5) 式の最適解をそれぞれ x, x^* とすれば，

$$f(x^*)\leqq f(x) \tag{5.6}$$

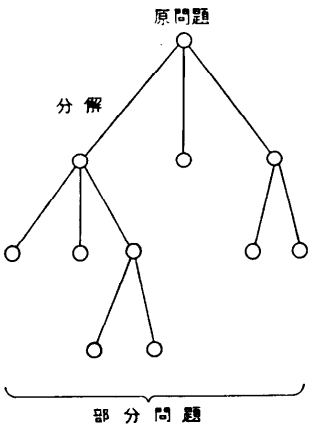

図-37 分枝限定法

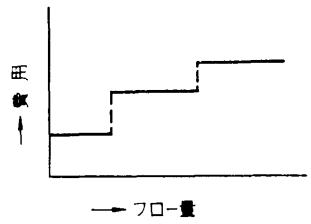

図-38 階段状費用特性

であることは明らかである．従って，$f(x^*)$ の値が今まで得られたどの部分問題の解よりも大きかったら，この部分問題を考慮の対象からはずすことができる．言いかえれば，分枝限定法により問題が能率良く解けるかどうかはこの限定操作をいかにうまく与えるかにかかっている．分枝限定法を IC のブロック列間の配線問題に適用した報告もある[70]．

iii) 整数計画法

整数計画 (IP: Integer Programming) 問題は，LP (Linear Programming) 問題に，一部あるいは全部の変数が整数値をとるという拘束条件を追加したものである．IP の応用を，その定式化の方法に従って2種に大別することができる．まず，取り扱う変数自体が分割を許さない場合，それを表わすには整数値を取らねばならない．もう一方は，複雑な（変則的な）目的関数あるいは拘束条件を線形不等式の形式に書くために整数変数を導入する場合である．例えば**図-38** のような費用特性をもつネットワークフロー問題を解くような場合に整数変数を導入して定式化することができる．このような意味で，IP は極めて広汎な問題に適用できる手法である．実際，商用のプログラム・パッケジもいくつか開発されている．最小被覆問題 (**2.7, 3.4, 3.9** 節参照) は，IP に適した組み合わせ計画問題の典型である[71]．

IP における難点は，収束速度が個々の問題の性質に大きく依存し，あらかじめ計算時間の予測をしにくいことである．一般的傾向としては，IP の収束速度は LP と異なり拘束条件の個数よりむしろ整数変数の個数に依存すると考えられている．大まかな目安として，100 個程度までの整数変数をもつ問題ならば，適当な工夫をすることによって実用的な意味で解けると考えられている．

さて，IP を実際に解くための算法の代表的なものとしては，**切除平面法** (cutting plane method)，**分枝限定法** (注：ある問題を直接分枝限定法で解くのと，IP 問題として定式化した後分枝限定法を適用することは区別した方が良い）などがあるが，その内容については例えば文献72)，73)などを参照されたい．

以上の3種類の手法は組み合わせ計画問題の一般的解法として広く用いられ，実用的にもかなりの成功を収めてきたが，これらはあくまでも中規模の問題を対象としたものである．ここで“規模”という言葉は，設計対象そのものの大きさのことではなく，解こうとしている問題の難しさを考えた上での大きさを指している．例えば100個の節点からなるグラフの最短径路問題ならば小規模であろうし，巡回セールスマン問題ならば大規模であると言える．

さて前にも述べたように，扱いにくい組み合わせ計画問題の計算の手間は，**図-39** に示すように問題の規模が増大すると階乗あるいは指数関数のオーダで増大する．一方上記の一般的解法（特に分枝限定法）は本質的には虱潰し法であって，虱潰しを効率良く行うとか，最適解を与える可能性がないと明らかになった場合の虱潰しを省くなどの工夫によって計算時間を短縮したものである．従って定性的には，階乗のオーダの複雑度を指数関数のオーダに下げるとか，指数関数の係数を小さくする程度の効果しかない．極端に言えば，図-39 の（a）のカーブを（b）のカーブへと改良させて，指数関数の立上りを右にずらしたに過ぎない．従って扱いにくい組み合わせ計画問題においては，一般的解法により最適解を求め得る問題の大きさには限界があり，大規模の問題に対しては，次の6章で述べるヒューリスティック算法に頼るのが現実的であろう．

6. 近似最適解を求めるヒューリスティック算法

ヒューリスティック (heuristic) という形容詞は，ギ

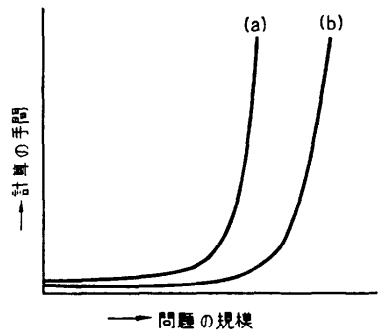

図-39 扱いにくい問題における問題の規模と計算の手間の関係

リシャ語の heúrēka が語源だとされ，「発見的」と訳されることが多い．しかし，近年計算機利用を前提とした算法の性質を論じるときには，「それが必ずしも最適解を与えるという保証はない」という意味で用いられることが多く，語源に基づいた「発見的」という訳は不適当であろう．このような“非理論的”ともいえるヒューリスティック算法の存在意義は，第1に，扱いにくい計画問題の厳密な最適解を求めることは不可能であるというところにあり，第2に，計画問題を実際の設計の立場から眺めれば次のようなことが言えることに注意すべきであろう．

(b) 設計のために組み合わせ計画問題を解くという意味は，最適解でなくとも最適解に十分近い解であれば，あるいは最適解とはかなり離れていても設計要求さえ満たせば，充分であるということである．

(b) 設計コスト…設計作業に伴う費用はできるだけ少なくしたい．従って設計の目的に応じて，妥当な計算コストの範囲内で解を求められることが要求される．

(c) 実際の設計問題は，単一の計画問題として表現できることは少なく，拘束条件も明確でなく，目的関数も複雑であるのが普通である．そこで問題を定式化する段階において既にある近似が行われることが多い．また複雑な問題は単純な問題に分解されて取扱われることが多い．そこで設計者の総合的な判断をとり入れるために複数個の近似最適解が求められる方が良いことがある．

ヒューリスティック算法においては，組み合わせ問題自体がそうであるが，個々の問題についてその特殊性を利用した手法は数え切れないほど提案されているが，統一的な研究は殆んどなされていない．多種多様なヒューリスティック算法を大ざっぱに分類すれば，下記の3種のアプローチが代表的なものである．

i) 直接構成法 (constructive method)

局所的に最適と思われる判断規準に基づいた処理を積み重ねて，これを全体の問題に対する近似最適解とみなす．Markowitz はスパース行列処理（**3.3** 節参照）において，計算時間最小の掃き出し順序を求める問題に対して n^3(n は行列の次数）のオーダの手間で近似最適解を求める手順を提案した[40]が，これは実用的に成功を収めた直接構成法の一例である．しかし一般的に言えば，この方法はヒューリスティック算法の中でも特に説得力に欠けるものである．というのは，扱いにくい組み合わせ計画問題では，局所最適解が数多く存在し，その一つが真の最適解に十分近いという保証は全然ないからである．

ii) 探索法 (search)

分枝限定法に代表されるような高度な虱潰し法を近似最適解を求める立場で利用する方法である．分枝限定法において，計算時間が限られていて，計算を途中で打ち切ったとしても，それまでに得られた最良の解，すなわち近似最適解が設計要求を満たすものとして受け入れられることもしばしばある．しかし許容できる近似最適解を与える部分問題が最後の方に探索されるように原問題が分解されることもあり，許容できる解が得られるか否かは運に作用される要素が大きい．これに対して，許容できる誤差の範囲 $\varepsilon>0$ をあらかじめ設定し，すべての部分問題をより短時間に探索するという方法も考えられる．すなわち，各々の部分問題の緩和問題（(5.5) 式）の解 x^* に対する限定操作において，

$$f(x)\leqq f(x^*)+\varepsilon \qquad (6.1)$$

ならばその部分問題を終端するように判定規準を変更する．ここで x はそれまでに得られた最良の解を表わす．この変更によって，誤差 ε の範囲内の近似最適解がより少ない手間で求められることになる．探索法では計算時間の見通しをたてにくいという欠点があるが，得られた近似最適解の誤差の範囲が明確にできるという大きな利点がある．

iii) 逐次改善法 (iterative improvement)

適当な初期近似解を与えて，これに逐次改善を加えながら局所最適解を求める方法である．(4.1) 式の表現において，x_i をある途中段階の近似解とすれば，次の近似解 x_{i+1} は拘束条件を満たし，かつ目的関数の値を改善させる必要がある．このような x_{i+1} は対象となる空間 X において，何らかの意味において x_i と近い距離にあるものが選ばれる．例えば巡回セールスマン問題においては，共有する枝の本数が多い二つのハミルトン閉路は近い距離にあると考えられる．この方法において，x_i から x_{i+1} を求めるという（拘束条件と改善度のチェックを含めた）単位処理が簡単であることと改善度の高いことが重要である．逐次改善法によって得られる計算結果は，初期近似解の良し悪しに大きく影響され，これを単独で用いることは危険である．そこで，乱数発生によって多数の初期近似解を生成し，それぞれについてこの方法を適用して多

くの局所最適解を求め，その最良のものを近似最適解として抽出するという方法が提案されている．この方法は比較的広い範囲の問題に適用でき，実用的にもかなりの成果を収めている[47)74)75)]という意味で注目に値する．

上に述べた三つの方法の他にも，主として補助的に利用されるヒューリスティック算法として，下記のようなものがある．

iv) 近 似 法 (approximation method)

与えられた計算問題が扱いにくいときに，あらかじめ扱い易い問題で近似してから，それの最適解を求めるという方法である．また拘束条件，目的関数が不明確なときにはなるべく扱い易い問題で近似するのが望ましい．

v) 乱数発生法 (random start)

乱数発生によってなるべく多くの拘束条件を満たす解を生成し，その中から最良のものを選ぶという考え方に基づく．これをそのまま用いて良い近似最適解が得られる可能性は低いが，逐次改善法との組み合わせによって極めて強力なものとなる[47)74)75)]．

vi) 特徴抽出法 (feature extraction method)

探索法，逐次改善法などによって比較的良い解を複数個求め，それらに共通した性質を抽出し，これに基づいて探索の範囲（(4.1)式における空間 X）をせばめるという方法である．さらに良い解を得るには，前と同じ手順をくり返すわけであるが，今度は問題が小さくなっているので，計算問題が短縮できることになる．例えば，巡回セールスマン問題において複数個の局所最適解が得られたとしよう．これらのハミルトン閉路のすべて，または大部分が特定のいくつかの枝を共有しているとすれば，真の最適解もそれらの枝を含んでいると予想される．そこで，もう一度同じ手順をくり返すときには，これらの枝をあらかじめ固定しておくのである．S. Lin はこの方法と逐次改善法と乱数発生法を組み合わせたものを，いくつかの組み合わせ計画問題に適用し，解き得る問題の大きさの範囲が大幅に広がったことを報告している[76)]．

以上いくつかのヒューリスティック算法を紹介したが，これらは排他的なものではなく，巧妙に組み合わせることによってより大きな効果をもたらすのである．例えば，探索法を逐次改善法の反復操作に適用して，計算時間が短縮できたという報告もある[75)]．

ところで，ヒューリスティック算法は，過去の経験やそれに基づく直感に頼って設計されるのが普通である．その有効性を評価したり，他人に説得しようとしても，できることは計算実験結果を示すぐらいで，論理的根拠に欠けるものである．しかし少くとも「何故ヒューリスティック算法に頼るか」という意味での説得力は必要である．表-4.1 に挙げたような扱い易い問題や，5章で述べたような手法を用いて最適解が妥当な計算時間内に求められる程度の規模の問題にヒューリスティック算法を適用するのは意味がない．ヒューリスティック算法は最後の手段として用いられるものである．

次に，与えられた問題に対して，それに適したヒューリスティック算法を設計する際にも，ある程度の論理性が必要である．ヒューリスティック算法では，最適解への収束は保証されないが，近似解の良さと計算時間とのトレードオフの意味での妥当性が必要である．逐次改善法と乱数発生法を組み合わせたヒューリスティック算法を巡回セールスマン問題に適用する場合を例にとって考えてみよう．局所最適解といってもいろいろ考えられるので，まず **λ-最適解**という定義を導入する[74)]．λ-最適解とは「λ 本以下の枝をとり代えてもそれより良い解は得られない」という条件を満たす局所最適解のことを指す．n-最適解（n は節点の数）が真の最適解に一致することは言うまでもない．λ が増加するにれつて，最適解の得られる可能性が高まる反面，計算時間も階乗のオーダで増加する．

乱数発生法では，得られた近似最適解の誤差の絶対量は出せないので，代わりに最適解が得られる期待値により解の良さを表わすものとする．S. Lin は数多くの例題について計算機実験を行い，上の意味での解の良さと計算時間のトレードオフは $\lambda=3$ で最適になるという結論を出している[74)]（もちろん巡回セールスマン問題に限ったことであるが）．また彼のプログラムで 3-最適解を得るための手間は n^3 のオーダである．

次に，何個の局所最適解を得る必要があるかという問題が生じる．一つの局所最適解が真の最適解となる確率を p とすれば，乱数発生法を利用して得られた k 個の局所最適解のどれかが真の最適解を与える確率は

$$\hat{p}(k)=1-(1-p)^k \qquad (6.2)$$

で与えられる．一方 S. Lin の経験的な見積りによれば，3-最適解に対する p の値はほぼ

$$p\simeq 2^{-n/10} \qquad (6.3)$$

であることが確かめられている．いま，$\hat{p}(k)\geqq 0.99$

の解の良さが要求されたとする．$n=20(p=0.25)$なら

$(0.75)^k \leqq 0.01$

すなわち，局所最適解を $k=16$ 回求める必要がある．$n=60(p=1/64)$ なら，$k=300$ 回求める必要がある．この評価は一例にすぎないが，計算時間と解の良さとのトレードオフをある程度見積ることができることを示している．

ヒューリスティック算法の設計は，問題の性質に大きく依存するので，一般論として述べることはできない．しかしある特定の問題を通して得られた経験を，異なった種類の問題あるいはより規模の大きい問題に対して有効に活用するためにも，適切な評価を行い，説得力をもたせる努力をすることは有益であろう．最後に，ヒューリスティック算法に対する最近の考え方を説いたものとして文献 77)を紹介しておこう．

7. グラフ的構造の表現法

本章では，グラフあるいはグラフ的構造（ハイパーグラフなど）に関する情報を計算機の記憶装置の中にどのような形式で表現するかという**データ構造** (data structure) と，表現されたデータをどのようにして参照するかという**アクセス手法** (access method) の問題について触れてみたい．この問題は一見些細なことがらのようであるが，実は種々の組み合わせ問題に対する算法の具体的手順を工夫することと表裏一体をなした重要な問題なのである．かっては直観的な記述だけでグラフやネットワークの理論や手法を理解したつもりになっていた時代があったが，“充分大きな規模の問題では，どの程度の大きさの記憶装置とどの程度の手間を要するか？”という問題意識のもとで解法の“良さ”を厳しく追求するというのが最近の研究の傾向となっている[6)78)79)]．例えば，グラフの平面性を判定するための新しい算法[22)]はデータ構造の工夫に依存するところが大である．

（有向）グラフ（枝の数を m，節点の数を n とする）の接続関係を表現するには，

（A） 各枝の両端点を与える表（$2m$ 程度の記憶装置の大きさが必要）．

あるいは

（B） 各節点に接続している枝の集合を与える表．

のいずれかがあれば原理的には十分であるが，実際の問題の算法では“節点から枝へ”と“枝から節点へ”の両方の検索が必要である場合が多いので，冗長ではあるが，(A)と(B)を併用するのが便利である．(B)の具体的なデータ構造の代表的なものとして次の二つの方式が考えられる．

(B1) 各節点を始点（終点）とする枝の番号を大きさ m の一次元配列の中に格納する．ただし，各節点を始点（終点）とする枝の集合は一つのブロックを成すように一まとめにし，各節点に対応するブロックの先頭の位置は大きさ n のもう一つの配列によって指されるようにしておく．

(B2) 各節点を始点（終点）とする枝をリストでつなぐ．最も簡単な方式としては，大きさ n の一次元配列 $\vec{\alpha}_+(*), \vec{\alpha}_-(*)$ と大きさ m の一次元配列 $\vec{\beta}_+(*), \vec{\beta}_-(*)$ を用意すれば良い．この場合，$j=\vec{\alpha}_+(i)$ は節点 v_i を始点とする枝の一つが e_j である（$j=0$ ならば節点 v_i を始点とする枝が存在しない）ことを示す．さらに，$k=\vec{\beta}_+(j)$ は節点 v_i を始点とする他の枝 e_k（$k=0$ ならばリストの終り）を示す．このようにして各節点について，それを始点とする枝を検索することができる．各節点を終点とする枝も，$\vec{\alpha}_-(*)$ と $\vec{\beta}_-(*)$ を用いて同様な手順で検索される．グラフの変形を伴う問題では，さらに四つの一次元配列 $\overleftarrow{\alpha}_+(*), \overleftarrow{\alpha}_-(*), \overleftarrow{\beta}_+(*), \overleftarrow{\beta}_-(*)$ を用意して，各節点に接続している枝のリストを逆方向にも検索できるようにした方が便利である．

図-40 に (A),(B1),(B2) の例を示す（次頁参照）．

グラフを計算機に入力するには，枝ごとにその両端点の番号を並べたデータを次々と読み込んで，これから上記のようなデータ構造を作り出せば良い．例えば**図-41**（次頁参照）のグラフに対して (B2) を作る場合，枝 e_1〜e_3 がすでに処理されていて新たに枝 $e_4=(v_1, v_4)$ が読み込まれたときには，$\vec{\alpha}_+(1)$ と $\overleftarrow{\alpha}_+(1)$ の内容（それぞれ1, 3 である）を参照した後に，$\vec{\beta}_+(3)=4$, $\vec{\beta}_+(4)=0$, $\overleftarrow{\alpha}_+(1)=4$, $\overleftarrow{\beta}_+(4)=3$ とデータを更新すれば，枝 e_4 の始点に関する情報が完成する．終点に関しても同様の処理を行えば良い．この例から明らかなように，一つの枝に関する処理時間はある定数でおさえられるので，上記のデータ構造を作成するのに要する手間は $O(m)$ である．両方向リストを用いるデータ構造は枝を削除する場合にも便利である．**図-42** (P. 589 参照) は一つの枝を削除したときのデータの変更の様子を示す．

グラフの節点や枝に（通し番号でない）固有名がつけられていることがあるが，この場合は外部名と内部

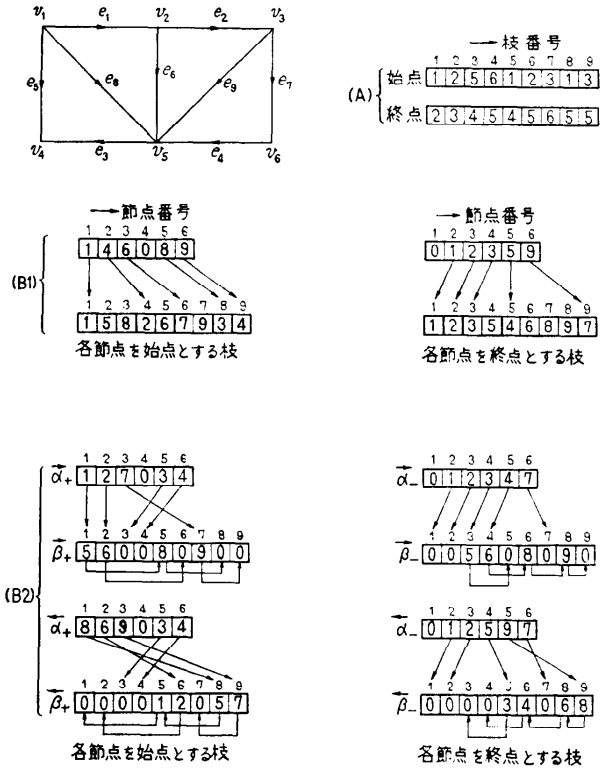

図-40 グラフの接続関係を表わすデータ構造

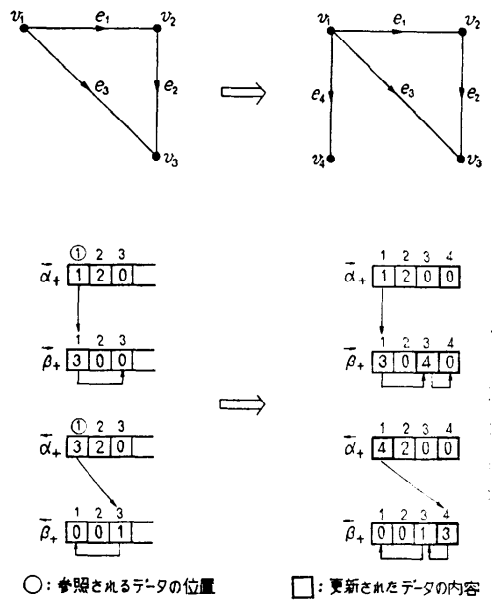

図-41 枝の追加にともなうデータの更新

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．

上記のデータ構造を用いれば，記憶装置の大きさ，作成するための手間は共に $O(m+n)$ であるが，これに対してグラフの接続関係を隣接行列や接続行列(**2.2** 節参照)をそのままの形で用いるのは，現在では最も拙劣な方法とされている．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．

グラフあるいはネットワークの問題において，最も基本的な操作は一つの木を求めることである．この操作には，一つの節点から出発して次々と新しい節点と枝を探索していくという，いわゆる**探索法** (search)[18]が用いられる．データ構造としては，(B1) を基本にしたものを用いると便利である．これによって，各節点についてそれに接続している枝を重複せずに走査することができ，$O(m+n)$ の手間で一つの木を求めることができる．最近開発されたグラフの連結性に関する問題の能率良い算法[18)80)81)]は前記のデータ構造とこの探索法を基本にしたものである．

さて上記のデータ構造は，“各枝の端点は二つしかない”というグラフの性質を利用しており，そのままではハイパーグラフあるいは一般の 1−0 行列を表現することはできないが，(B1) あるいは (B2) における考え方を応用することができる．すなわち，各々の“1”要素について，行番号，列番号，“1”要素を行(列)方向に走査するためのポインタなどの情報をもったテーブルを作り，それを一次元配列の中に並べておくという方式が考えられる．このようなデータ構造を作るには，“1”要素の数（ハイパーグラフと考えれば各々の枝の節点数の総和）に比例する大きさの記憶装置があれば十分である．1−0 行列に関するデータ構造については，最近スパース行列処理技法の進歩に関連していろいろ工夫されている[37)38)]．

ところで，DA のプログラムにおいては，接続関係を表現するデータ構造を実行時にアクセスする方式と，あらかじめ前処理の段階でアクセスしながら能率の良い目的プログラムを発生しておき，これを実行時にランさせる方式とがある[34)]．前者はテーブル・ドリ

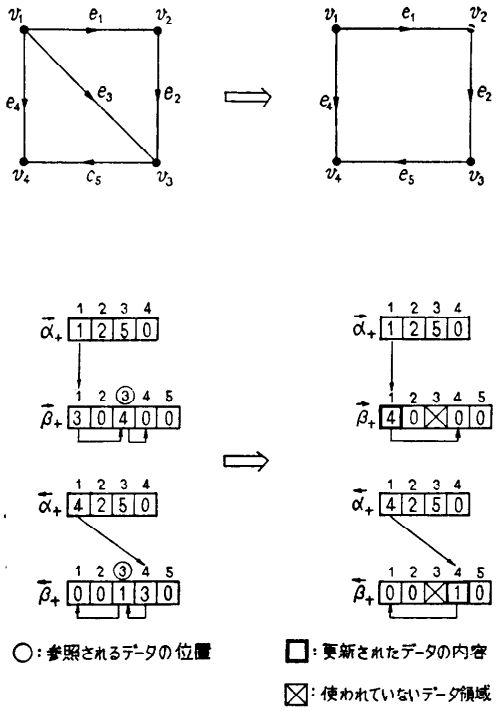

図-42　枝の削除にともなうデータの更新

ブン (table driven) 方式と呼ばれ，論理シミュレーションなどのように繰返し計算の内容が毎回異なる場合に多く用いられる．後者はコンパイル (compile) 方式あるいはコード発生 (code generation) 方式と呼ばれ，電子回路解析などのように同内容の計算が何回も繰り返される場合に用いられる[39]．前者は使用記憶装置の大きさの面で，後者は実行時間の面で優れている．一般には，使用する計算機システムによる制限と，対象とする問題の性質を考えた上で，どちらの方式が良いか，あるいは中間的な方式が良いかを決定しなければならない．

グラフあるいはネットワークの問題を扱うプログラムを作成するには，データ構造の設計，算法の記述だけでも，かなり複雑な作業が介入する．グラフ処理用の問題向き言語と称するものがいくつか発表されている[82)~85)]が，これらはいずれも近年の算法理論の進歩に追従できるものではない．今後の問題として，算法の記述が容易で詳細に至るまで自由に表現ができ，かつ計算機内での実行過程の複雑さ（記憶場所の大きさと処理速度）を明確に反映できるような言語の開発が望まれる．

おわりに

3回にわたって，設計自動化 (DA) におけるグラフ理論と組み合わせ算法について述べてきた．筆者の力不足のため表現のまずい点，重要な問題についての書き落しなどが多々あると思われるが御容赦戴きたい．特に3章で述べた個々の問題に対して試みられている種々の算法の比較，あるいはヒューリスティック算法が DA において実用的に用いられている例などについては紙数の関係上省略させて戴いたが，各々の問題の専門家による適切な解説をお願いしたい．

最後に，グラフに興味ある人がより一層 DA に，そして DA に興味ある人がグラフにより一層の関心を寄せてくれることを願って本講座の結びとしたい．また，原稿の段階で多くの誤りを御指摘戴いた査読者ならびに当社集積回路事業部 CAD 部川西宏氏，吉沢仁氏に深く感謝致します．

参考文献

(既出分のへ追加)

63) S. A. Cook: The Complexity of Theorem-Proving Procedures, Proc. 3rd Annual ACM Symp. on the Theory of Computing, pp. 151～158 (1971).

64) R. M. Karp: Reducibility among Combinatorial Problems, in "Complexity of Computer Computations (R. Miller & J. Thatcher, Eds.)", Plenum Press, New York, pp. 85～103 (1972).

65) R. E. Bellman: Dynamic Programming, Princeton Univ. Press, Princeton (1957).

66) R. E. Bellman and S. E. Dreyfus: Applied Dynamic Programming, Princeton Univ. Press, Princeton (1962).

67) R. E. Bellman: Dynamic Programming Treatment of the Traveling Salesman Problem, J. ACM, Vol. 9, No. 1, pp. 61～63 (1962).

68) M. Held and R. M. Karp: A Dynamic Programming Approach to Sequencing Problems, J. SIAM, Vol. 10, pp. 196～210 (1962).

69) E. L. Lawler and D. E. Wood: Branch-And-Bound Method: A Survey, Operations Research, Vol. 14, No. 4, pp. 699～719 (1966).

70) B. W. Kerninghan, et al.: An Optimum Channel-Routing Algorithm for Polycell Layouts of Integrated Circuit, 1973 DA Workshop pp. 50～59 (1973).

71) A. Geoffrin: Integer Programming by Implicit Enumeration and Balas' Method, SIAM Review, Vol. 9, pp. 178～190 (1967).

72) A. Geoffrin and R. Marsten: Integer Programming Algorithms: A Framework and State-Of-The-Art Survey, Management Science, Vol. 18, No. 9, pp. 465～491 (1972).
73) 茨木俊秀: 整数計画法Ⅰ～Ⅴ, オペレーションズリサーチ, 日科技連 (昭和45年10月～昭和46年1月).
74) S. Lin: Computer Solutions of the Traveling Solesman Problem, B. S. T. J., Vol. 44, pp. 2245～2269 (1965).
75) S. Lin and B. W. Kerninghan: An Efficient Heuristic Algorithms for the Traveling Salesman Problem, Operations Research, Vol. 21, No. 2, pp. 498～516 (1973).
76) S. Lin: Heuristic Techniques for Solving Large Combinatorial Problems on a Computer, in "Theoretical Approaches to Non-Numerical Problem Solving", Proc. 4th. Systems Symp. pp. 410～418, Springer-Verlag (1970).
77) S. Lin: Heuristic Programming as an Aid to Network Design, Networks, Vol. 5, No. 1 pp. 33～43 (1975).
78) 伊理正夫: グラフ的構造を有する情報の処理技法について, 昭47電気四連大, No. 217, p. 850 (1972).
79) J. E. Hopcroft and R. E. Tarjan: Efficient Algorithms for Graph Manipulation——Algorithm 447, Comm. ACM, Vol. 16, No. 6, pp. 372～378 (1973).
80) D. G. Corneil: A N^2 Algorithm for Determing the Bridges of a Graph, Information Processing Letters, Vol. 1, No. 2, pp. 51～55 (1971).
81) J. E. Hopcroft and R. E. Tarjan: Deviding a Graph into Triconnected Components, SIAM J. Comput., Vol. 2, No. 3, pp. 135～158 (1973)
82) S. Crespi-Reghizzi and R. Morpurgo: A Language for Treating Graphs, Comm. ACM, Vol. 13, No. 5, pp. 319～323 (1970).
83) W. C. Rheinboldt, et al.: On a Programming Language for Graph Algorithms, BIT, Vol. 12, No. 2, p. 220 (1972).
84) R. C. Read: Teaching Graph Theory to a Computer, in "Recent Progress in Combinatorics (W. T. Tutte, Ed.)", pp. 161～173, Academic Press, New York (1969).
85) T. W. Pratt and D. P. Friedman: A Language Extension for Graph Processing and its Formal Semantics, Comm. ACM, Vol. 14, No. 7, pp. 460～467 (1971).

(昭和50年4月15日受付)