文　献　紹　介

68-15. $\sqrt{x}$ の Newton-Raphson 計算のための最適出発値

D.G. Moursund: Opitimal Starting Values for Newton-Raphson Calculation for $\sqrt{x}$ 〔CACM, Vol. 10, No. 7, July, 1967, pp. 430～432〕 key: numerical method, approximation

区間 $[a, b]$ $(0<a<b)$ で，$\sqrt{x}$ を Newton-Raphson 法で求める際の出発値 $G(x)$ が，次の条件をみたすようにとることを考える．

“ある $m(m=0, 1, 2, \cdots)$ に対して，

$$x_0=G(x),\ x_{i+1}=\frac{1}{2}\left(x_i+\frac{x}{x_i}\right)\ (1\leqslant i\leqslant m-1)$$

として，適当な $G(x)$ の属す函数のクラスのうちで

$$\max_{x\in[a,\,b]}\left|\frac{\sqrt{x}-x_m}{\sqrt{x}}\right|$$

を最小にする”．

さて，$N[f(x)]=\frac{1}{2}\left[f(x)+\frac{x}{f(x)}\right]$ とし，

$e_0(x)=\sqrt{x}-G(x)$，$e_m(x)=\sqrt{x}-N^m[G(x)]$ $(m\geqslant 1)$ とおくと，$e_{i+1}=-e_i^2/2[\sqrt{x}-e_i]$ より，$e_0(x)=y$ とおくと，$e_m(x)$ は，x と y の函数，$e_m(x, y)$ である．そこで，

$$W_m(x, y)=\mathrm{sgn}(y)\ (e_m(x, y)/\sqrt{x})$$

を考えると，問題は

$$\max_{x\in[a,\,b]}|W_m(x, \sqrt{x}-G(x))|$$

を，最小にすることである．因子 sgn (y) は，著者による，重みつきチェビシェフ近似の一般論（SIAM J. Numer. Anal, Vol. 3. 1966, pp. 435-450)が，適用できるためである．W_m の具体例は，$W_0=y/\sqrt{x}$，$W_1=(\mathrm{sgn}y)y^2/2[x-y\sqrt{x}]$ などである．

この問題に関して，次の結果がえられる．

定理　多項式または有理函数の範囲での最適解は，$m=1, 2, \cdots$ の場合に共通である．

しかし，$m=0$ すなわち $\max_{x\in[a,\,b]}\left|\frac{\sqrt{x}-G(x)}{\sqrt{x}}\right|$ を最小にする $G(x)$ は，$m\geqslant 1$ では，最適とはいえないことが，$G(x)=$定数の範囲で示されている．

最後に，三次までの多項式の場合に定理で示された共通の最適解の，係数の数値例が示されている．

（牛島照夫）

68-16. 方程式の直接解に対する計算可能な誤差限界

B.A. Chartres and J.C. Geuder: Computable Error Bounds for Direct Solution of Linear Equations 〔JACM, Vol. 14, No. 1, Jan., 1967, pp. 63～71〕 key: numerical method, error evaluation

連立一次方程式の後向き（a posteriori）の誤差評価を考える．A を (n, n) 行例，b を n ベクトルとしたとき，$Ax=b$ の数値解 x が，$(A+\delta A)x=b$ の正確な解であると見なし，δA の要素 δa_{ij} の評価を求めるのが目的である．ガウスの消去法によるとして，(a) $A=LU$（$L=(l_{ij})$ は，対角線上が 1 の下三角行列，$U=(u_{ij})$ は，上三角行列），(b) $L^{-1}b=c$，(c) $x=U^{-1}c$ により順次変形する．以下 L, U および c で，対応する計算値を表わすことにする．ここで $A=LU-E$, $b=(L+\delta L)c$, $(U+\delta U)x=c$ をみたす，E, δL および δU が求まると

$$\begin{aligned}b&=(L+\delta L)(U+\delta U)x\\&=(A+E+U\delta L+L\delta U+\delta L\delta U)x\\\therefore\quad \delta A&=E+U\delta L+L\delta U+\delta L\delta U\end{aligned}$$

である．したがって，計算法に即して，E, δL, δU の評価をすればよい．

さて浮動小数点数 a と b の加算の結果 c は，

$$c=a(1+\varepsilon_1)+b(1+\varepsilon_2)\qquad |\varepsilon_i|<\beta_1$$

をみたす．β_1 は機械の丸め方に依存する定数である．(a), (b), (c) 各段階での中間結果は，高精度で計算されることが多いことを考慮して，$|\varepsilon_i|<\beta_2$ $(\beta_2\leqslant\beta_1)$ を仮定する．ここで

$$\beta_1'=\beta_1(1+\beta_1),$$

$$\beta_2'=n^{-1}\left(\exp\left(\frac{n\beta_2}{1-\beta_2}\right)-1\right)(1+\beta_1),$$

$$\beta_1''=\beta_1'+n\beta_2'',\ \beta_2''=\beta_1'(1+\beta_1+n\beta_2')$$

とおく．本論文で選択された計算法によれば

$$|\delta a_{ij}|<2(j+3)w_{ij}\beta_2''+\begin{cases}2|u_{ij}|\beta_1'', & i<j\\3|u_{jj}|\beta_1'', & i=j\\2|l_{ij}u_{jj}|\beta_1'', & i>j\end{cases}$$

なる評価を得る．ここで $w_{ij}=\sum_{k=1}^{\min(i,\,j)-1}|l_{ik}u_{kj}|$ であり，さらに，$\beta_2\ll\beta_1$ を仮定している．

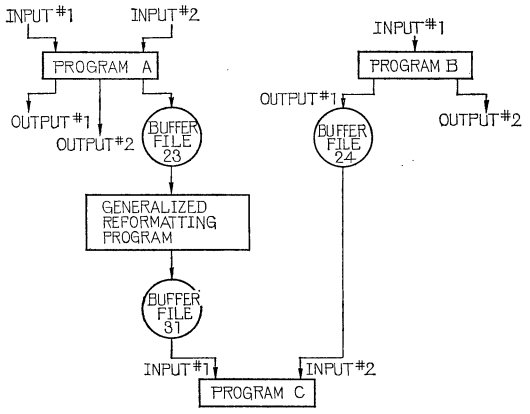
上の結果の応用として，右辺を L, U の計算と平行して行ない，あらかじめ設定した誤差範囲をこえたときに，他の計算法―たとえば精度を上げる―を採用することが考えられる．ガウス消去法の実行中に，積 $l_{ik}u_{kj}$ の値は得られるから，この判定には，約 $n^3/3$ 回の加算が余分に加わるだけで良い．　（牛島照夫）

68-17. 独立なプログラム間の交信を行なうための一方法

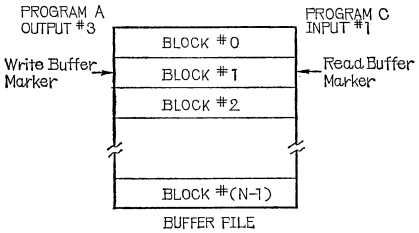
E. Morenoff and J.B. McLean: Inter-Program Communications, Program String Structures and Buffer Files 〔Proc. SJCC, 1967, pp. 175～183〕 key: programming system, program linkage

システムの変化に対し，プログラミング・システムを本質的にデザインしなおさずに，即応させる一般的な手段はプログラム・モジュラリティーを達成することである．

Program String Structure は各プログラム・モジュール（building block）を結合する一つの技術を提供する．結合のおもなる概念は各プログラムが実際に Combine or integrate を行なわれずに，しかも全体が一つのまとまりとして用いられることである（第1図参照）．つまり，結合の方法は一つのプログラムが全て終了してから他のプログラムへコントロールを渡すのではなく，データを移動させることによって，モジュール化した各プログラムを並列に動作させようとするものである．これを行なうために，一つのプログラムから他のプログラムへデータ・セットを渡す一般化された Buffer File Linkage Mechanism が用意される．この Linkage Mechanism はモニタの一部であり，第1図に示した Buffer File を通して，プログラム間のデータ・セットの流れを制御する．



第1図



第2図

おもなる Buffer File の制御方法は，Read/Write Buffer Marker を使う．たとえば，両 Marker の示す点が一致していて（Write Buffer Marker のカウントが Read Buffer Marker より1サイクルだけ余分に増加した場合を除く），かつプログラム C がデータ・セットを Read したいとき，Read できない禁止条件は Linkage Mechanism によって直ちに発見される．モニタのスケジューラはこの禁止条件を与えられることによって，プログラム A, C を delay の状態にする．

このようにプログラムの入出力は常に Buffer File を通して行なわれるため，他のプログラムと同期をとることを考えないでよい．

Buffer File を導入したことによるその他のおもな利点は

（1） オンライン・コンピューター・システムの場合，Read できない禁止条件が解かれると直ちにプログラムC（第2図）はデータを読みとってプログラムAが終了しないうちに，利用できる結果の一部を端末のユーザに渡すことができる．

（2） Buffer File の大きさを調節することによって，より複雑なスケジューリング・アルゴリズムが可能になる．

（3） 独立なプログラム間の結合を可能にしたことによって，実際に並列でプログラムが動作する Multi-Computer environment において有利である．

（4） プログラムの State information を渡すのに使用すると，より便利である（両 Marker の機能による）．

などである．

最後に，他のシステム，DM-1, OS/360, Multics 645/1, などとの技術の比較が行なわれている．

（藤井狷介）

68-18. リスト処理言語における高速および低速の記憶装置の使用

Cohen: A Use of Fast and Slow Memories in List-Processing Languages〔CACM, Vol. 10, No. 2, Feb., 1967, pp. 82～86〕key: list processing, secondary memory

リスト処理においては,計算時間を犠牲にしてでも,大きな記憶場所を必要とする問題がよくある.そのような問題に対処するためには,アクセスタイムは短いが高価なゆえに,大きさに制限のあるコア・メモリを,garbage collection によってだましだまし使っても,たいてい,記憶場所の不足に悩まされるものである.この論文では,アクセスタイムは長いが比較的容易に手に入る大容量の補助記憶装置(ドラムまたはディスク)を,リスト構造をしまうための記憶場所として使用する(決して新しくないが),能率のよい方法について述べている.その方法は,コア・メモリにおいてリスト構造をしまうために利用できる領域および補助記憶装置(たとえばディスク)を,適当な大きさのページに分割し,プログラムによって参照されたリスト・セルがコアになければ,コアの中で一番不活発なページをディスクに戻し,その場所に参照されたリスト・セルを持つページをディスクから持って来るというものである.ここで最も問題になることは,コア・メモリにおいて利用できるページ数および一番不活発なページを決めるための規準であるが,それらについて実験的に次のような結論を与えている:コアメモリにおいて利用できるページの数が多ければ計算時間にとって有利なことは明らかであるが,200 のリスト・セルを含むページ(この数は実験に使用した金物に依存するものである)が 10 より多くあっても,大多数の問題に対して計算時間の得にならない.コア・メモリにおける一番不活発なページとは,一番長い間使用されなかったページであると決めれば十分である.

このように変項を決めたときには,コア・メモリよりも 10^4 倍もアクセスタイムの長いディスクを使用しても,計算時間の増大を(コアだけを使用したときに較べて)3倍程度におさえることができた.

(紹介者注) このような方法で補助記憶装置を使用すると,garbage collection によって,小さなリストが多くのページに分散するように free list が作られたときには,その後の計算が非常に遅くなる.garbage collection の際に,free list が特定のページにつまるようにつめかえればそのような問題はないように思えるが,今度は garbage collection に多くの時間がかかることになる.どのような場合につめかえをするのが有利であるかは,Jacques Cohen and Laurent Trilling "Remarks on Garbage Collection Using a Two-Level Storage" BIT 7 (1967) で考察されている.

(二村良彦)

68-19. いろいろなリスト構造における能率のよい Garbage Collection 手続

H. Schorr and W.M. Waite: An Efficient Machine-Independent Procedure for Garbage Collection in Various List Structure〔CACM, Vol. 10, No. 8, Aug., 1967, pp. 501～506〕key: list processing, garbage collection

リスト処理システムにおいて,使い捨てられた記憶場所を再生利用する,すなわち free list にもどす方法は,次の三つに大別される.(1)プログラマが,自分のプログラムのすべてのリスト構造についてそれが使用されているかどうかを覚えていて,使用されていないものをシステムの命令を用いて free list にもどす.(2)リスト構造にそれをサブ・リストとするリスト構造の数を覚えておくカウンタがあって,そのカウンタが 0 になったときにシステムが自動的に free list にもどす.ただし,不必要なメイン・リストをもどす操作はプログラマがシステムの命令を用いて行なわれなければならない.(3)free list がつくされると,システムが自動的に使い捨てられたリスト構造を集めて free list にもどす.プログラマは,リスト構造が使用されているかいないかという問題に関与しない.3番目の方法は,garbage collection と呼ばれ,LISP などにおいて使用されている.garbage collection の特長は,プログラマがリスト構造を再生利用する問題に関与しないですむので,プログラム上の負担が軽くなることである.

この論文では LISP などで用いられる一方向リストおよび SLIP などで用いられる二方向リストの garbage collection について,簡単で記憶場所をあまり使用しなくてすむ方法について述べている.たとえば,一方向リストの場合には,使用されているリスト構造をたどって,すべてのリスト要素に印をつける際に,分岐点を憶えるために工夫をし,わずか三つのレジスターを一時記憶場所として用いるだけですませている.この方法は比較的機械依存度が低いので,作成しようとするリスト処理言語にいくつかの簡単な機能を追加すれば,その言語を用いて記述できると主張している.

(二村良彦)

68-20. ブロック向きシステム設計における考察

D.H. Gibson: Considerations in Block-Oriented Systems Design 〔Proc. SJCC, 1967, pp. 75～80〕 key: simulation, operating system, blocking

メモリと CPU（中央処理装置）との間を，いくつかのワードをまとめるブロック単位で伝送することの得失を調べることが本論文の主題である．すなわち，余分のワードを伝送することが CPU にとって有益か否かということで，IBM 7000 シリーズのデータによりシミュレーションを行なった．調査項目は，（1）1ブロック当たりのワード数，（2）CPU が局所的にストアすべきブロック数，（3）局所的にストアされているブロック群の一つを CPU が移しかえたいときのアルゴリズム，（4）ブロック単位伝送の有利さに影響を及ぼすようなプログラムの型，などについてである．ブロックの大きさは2のベキ乗で変化させ，1ブロック当たり4ワードから1ブロック当たり 4096 ワードまで試みた．

局所メモリの大きさは32ワードから8192ワードまで変えた．ブロック群の一つを移しかえるアルゴリズム（これを以後，置換アルゴリズムという）は約15種類を試みた．CPU からメモリを参照する順序をアドレスパターンとよぶことにすると，それらの両極端としてはランダムなものと連続的なものとがある．局所メモリ内に CPU が所望するあるワードが見つからない確率を P とすると，ランダムの場合には

$$P=1-\frac{\text{局所メモリの大きさ}}{\text{全メモリの大きさ}}$$

であり，連続的の場合には $P=\dfrac{1}{\text{1ブロックの大きさ}}$ である．IBM 7000 シリーズのデータでのアドレスパターンは，ランダムでも連続的でもない．局所メモリの大きさを2048ワード，ブロックの大きさを16ワードとするときの，アドレスパターンの 76% が $P\leqq 0.0275$ であった．1ブロック内の余分のワード数の CPU への貢献度はブロックの大きさが大きくなるにつれて減少する．すなわち，他のものを一定にしたとき，ブロックの大きさは小さいほどよい．しかし1ブロックが4ワード以下では役立たない．

局所的にストアされるべきブロック数としては，理論的には最小限4ブロックでよいが，実例では効率よいオペレーションのためには4ブロックよりかなり大きい量が必要であることを示した．1ブロックを 16 ワードとして，128ブロック以上を局所メモリにストアすべきである．しかしオペレーションの効率は 128 以上のブロック数では一定値に収束していくので局所メモリの大きさは1ブロックを 16 ワードとして 128 ブロック，すなわち2キロワードが適当である．いかなるアドレスパターンに対しても最善であるような置換アルゴリズムはない．置換アルゴリズムは，(i) 合同写像アルゴリズム (ii) 先着順 (iii) 参照頻度，の三つに分類される．プログラムの大きさは，ブロック単位伝送の効率を決定するときのアドレスパターンに対して2次的な重要さしか認められない．

最後に，局所メモリに見出されないワードを参照する回数をシステムの評価数として，三つのシステムの効率を比較して，大容量メモリ ⟷ 局所メモリ ⟷ CPU のシステムがすぐれているとしている．結論として，（1）ブロックの大きさは小さい方がよく4ワードから16ワードがふさわしく，とくに16ワードが最適である．（2）局所メモリの大きさは，2キロワードから4キロワードが適当である．（3）置換アルゴリズムにより速度を改善できることもある．（4）メモリに情報を出し入れする活発さはプログラムの大きさと無関係である．（5）大容量メモリ⟷局所メモリのシステムがすぐれているといえる．（6）各種プログラムのアドレスパターンは，メモリに情報を出し入れする活発さに最も重要な変化を与える．

（杉藤芳雄）

68-21. スタック・オートマトンとコンパイリング

S. Ginsburg, et al.: Stack Automata and Compiling 〔JACM, Vol. 14, No. 1, Jan., 1967, pp. 172～201〕 key: automaton, compile, stack automaton

スタックと記号表を持つコンパイラの数学的モデルであるスタック・オートマトンを形式化し，今までにプログラム言語に関して研究された他のオートマトン（有限オートマトン，プッシュダウン・オートマトン，linear bounded オートマトン，チューリング機械）との関係を述べている．スタック・オートマトンは有限の内部状態，入力テープおよびスタックを持つ機械であり，入力テープからどちらの方向（二方向）にも記号を読み込んでいけるし，かつスタックの中にあるどの記号も読むことができる．ただしスタックに書き込む場合は，一番上の記号だけしか書きかえることができない．スタック・オートマトンに出力構造をつけ加

えると，

```
begin real XY; integer JK; Boolean IK;
JK:=0; XY:=JK; end
```

のような ALGOL プログラムを認識し出力プログラムを生成するようなコンパイラを作ることができる．このとき，宣言された変数の名前と型と記憶場所を示す情報はスタックに置かれ，代入文の分析の際に参照される．

スタック・オートマトンとプッシュダウン・オートマトンとの違いは，前者ではスタックの中の任意の記号を読むことができ，後者ではスタックの先頭の記号しか読むことができない（last-in-first-out）ということだけである．スタック・オートマトンでスタックの中の任意の記号を書きかえ可能にするとチューリング機械と同等になる．このようにスタック・オートマトンはプッシュダウン・オートマトンとチューリング機械の間の受理能力を持っている．スタック・オートマトンによって受理される集合は帰納的であること，および context-sensitive 言語は deterministic なスタック・オートマトンによって受理されることも示されている．（二村良彦）

68-22. Context Free 文法の認識装置の能率について

T.V. Griffiths and S.R. Petrick: On the Relative Efficiencies of Context-Free Grammer Recognizers [CACM, Vol. 8, No. 5, May, 1965, pp. 289～300] key: context free grammar, recognition

Context Free 言語の文を認識するアルゴリズムがいくつか知られており，各々の言語の文の文法に従って適当に選ばれて使用されている．この論文では，それらのアルゴリズムを，与えられた文の parsing tree を作る大体の方向に従って，top-to-bottom, bottom-to-top, left-to-right, および right-to-left に分類し，おのおのの類に関して行なった認識能率の比較実験の結果を示している．比較の公正を期するために，各認識アルゴリズムを，与えられた Context-Free 文法からその文を認識するためのチューリング機械を構成するための手続きとして定義し，構成されたチューリング機械が，文のすべての構造的記述（structural description）を見出すために必要とする命令の数を以って能率を計っている．一つの認識アルゴリズムの能率は，対象となる言語の文法の特性，たとえば左分岐型，右分岐型，自己埋蔵型など，および決定論的か否かに依存するものである．しかし決定論的な場合を除けば一般的には，selective-top-to-bottom アルゴリズムよりも，selective-bottom-to-top アルゴリズムの方が優れていると結論している（残りのアルゴリズムに関してはデータ不足のために明確に結論を出していない）．

selective-bottom-to-top アルゴリズムとは，文の parsing tree を構成する際に端末記号列（bottom）から出発し，可能な構文単位を組立ててゆき，最後に目的の構文単位，すなわち「文」（top）に至る手続であって，しかも低いレベルの構文単位を高いレベルの構文単位に結びつける際に，無駄な構造を避けるために特定の選出条件を入れたものである．

（紹介者注：この論文の結論に対して「それは実験に用いられた特定の文法についてのみいえることであって，一般的には正しいとは限らない」という反論が R.A. Brooker により "Top-to-bottom Rehabililated?" CACM, Vol. 10, No. 4, April, 1967 で述べられている．）

（二村良彦）

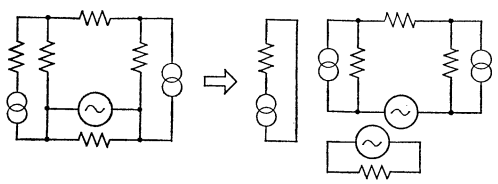
28-23. AEDNET: 非線型回路網のためのシミュレータ

J. Katzenelson: AEDNET: A Simulator for Nonlinear Networks [Proc. IEEE, Vol. 54, No. 11, Nov., 1966, pp. 1536～1552] key: programming system, simulation, network

AEDNET は線型と非線型電子回路のシミュレーションのためのディジタル計算機プログラムのシステムである．このシステムの主な特徴として二つあげている．

1) 回路網の素子として非線型 time-varying capacitor, resistor, inductor を許す．

ただし回路網はあるトポロジカルな条件を満足する必要があり，素子の特性（たとえば $v=R(i)$ のような関数で表わす）は適当なリプシッツ条件を満足する必要がある．トポロジカルな条件について，resistive branch に着目した場合を述べる．まず capacitor, inductor をそれぞれ適当な電圧源，電流源で置きかえることによってできた resistive 回路網を分割できる最大数の部分回路網に分割する（図参照）．



こうすることによって三つの型の部分回路網が得られる．各型の部分回路網に対するトポロジカルな条件は次のようなものである．

（1） open branch（電流源と並列にある単一の，branch）は current-controlled でなければならないが単調増加関数でなくてもよい．

（2） self-loop（電圧源と直列にある単一の branch）は voltage-controlled でなければならないが単調増加関数でなくてもよい．すなわち（1），（2）とも negative slope を許す．

（3） 一つ以上の素子を含む分割できない回路網の各素子は単調増加で1対1の特性を持たなければならない．

回路網の応答の計算は resistor のみの回路網（元の回路網の各電圧源と各 capacitor をショートし，各 inductor と各電流源を取り除く），capacitor のみの回路網（各電圧源をショートし，各 resistor, inductor, 電流源を取り除く），inductor のみの回路網（各電圧源 capacitor, resistor をショートし，各電流源を取り除く）をそれぞれ，一つのブロックと考え，one-element-kind net work を解くことにより行なう．まず初期時 t_0 において初期状態として capacitor の電荷 q と inductor のフラックス ϕ を与え，電圧源，電流源の t_0 における値を用いて capacitor ブロック，inductor ブロックを解きその出力を resistor ブロックに与えこれを解く．こうしてすべての必要な，branch 電流と電圧が得られる．次に $\dot{q}$, $\dot{\phi}$ を計算して，積分ブロック（predictor-corrector 法による）で q, ϕ の時間 t に関する微分方程式を解くことにより $t_0+\Delta t$ に対する q と ϕ の値が求まる．このプロセスを繰り返えすことにより解を得る．

（2） ディジタル計算機を on-line で使用する．使用する計算機は Project MAC time-sharing computer である．入力情報はオッシロスコープディスプレー装置にライトペンで回路図を画くことにより与えられ，また素子の特性をディスプレー装置に画くか，またはコンソールタイプライターでタイプインすることにより与えられる．出力は各部の波形が時間関数として，ディスプレー装置に画かれる．回路網が指定され，応答が計算され，ディスプレーされた後，使用者は，回路網の素子とトポロジーを変え再び計算しディスプレーさせることができる．　（千葉道種）

68-24. DDC システムにおけるディジタルバックアップ

J.M. Lombardo: The Place of Digital Backup in the Direct Digital Control System [Proc. SJCC, 1967, pp. 771～778] key: control, dual processor

DDC システムの有利さは，その制御のアルゴリズムの自由度が大きく，かつ経済モデルに従って，プロセスを自由に運転できる点にある．しかし，従来のように CPU の故障の際はアナログ計器のバックアップによってクリティカルなループをコントロールするという方式では，CPU の故障中は，その有利さが失なわれるばかりか，そのプロセスの運転に高度の演算またはバッチ処理を必要とするプロセスの場合はそれが中断されることによりかなり不都合のこととなることがあるため，DDC システムの導入がためらわれている場合が多い．

しかし，各ループに共通の回路のバックアップとしてシステムをデュアル構造とすることにより，その解決ができる．たとえば単一の CPU で MTBF が 2000 時間，修理に要する時間を8時間とすれば，その稼働率は 95.2% であるが，デュアル構造では 99.996% となり，信頼性の上では実用上問題がなくなる．

デュアル構造とする場合，その故障の診断および交互の CPU 間の信号の伝達をいかにうまく行なうかが重要な問題である．そのためには，タイムシェアリングで使用している部分をブラックボックスとみて，模擬の入力に対する出力を診断する方法が最もうまい方法であろう．このようにすることによりかなり高価なシステムとはなるが，多くのループにアナログ計器をバックアップとして用意することと比較すれば，むしろ安価でかつ有利であるといえる．　（甲斐忠道）

68-25. On-line デバックと Off-line デバックの経験を基にした比較

E.E. Grant: An Empirical Comparison of ON-LINE And OFF-LINE Debugging [SDC. 18, May 1966] key: on-line debugging, time-sharing system

この論文には 1965 年から 1966 年の間に System Development Corporation でおこなった実験の結果が書かれている．

実験には 12 人のプログラマが参加し，各々には二つの問題が与えられ Simulated closed shop と SDC's Time-Sharing System（TSS）を使ってデバッキング

能率の比較が行なわれた.

問題の一つは代数方程式を解くもので，他の一つは 20×20 の cell maze のうち single path をみつける問題である.

それぞれの問題に対しては6個のプログラムはTSSを用いて ON-LINE で，他の6個は， closed shop (turn-around-time：2時間) を用いて OFF-LINE でデバックされた.

デバッキング能率の比較のため使用した CPU の時間と man hours (デバックを始めて終るまでに経過した時間ではなく，デバックするのに要した実際の時間) が測定された. その結果を表で示すと次のようになる.

		ON-LINE	OFF-LINE
代数方程式	CPUの時間	3	1
	man hours	1	1+α
Fathfinder	CPUの時間	1+α	1
	man hours	1	3

表中の数字は列間での割合を示し，α は微少な正数を示す.

以下この論文では上記の結果に関して検討がなされ次の点が明らかにされている.

1. OFF-LINE シミュレーションの妥当性と現象性.
2. 実験の際，割り当てたグループの不平等性,
3. 実験に使った2種類の問題の本質的な相違.
4. ON-LINE でデバックするプログラマの低能率な習慣の採用. (朝日洋次)

68-26. ハミルトン・パスおよびナイト・ツァーを見出す一方法

I. Pohl: A Method for Finding Hamilton and Knight's Tours 〔CACM, Vol. 10, No. 7, July, 1967, pp. 446～449〕 key: Hamilton path, knight's tour problem

ハミルトン・パスとは，グラフにおいてすべての点を1個，しかも，ただ1回に限り通るようなパスである. この名前は，ハミルトンが12面体の各頂点をただ一度通り，陵をその構成要素とするパスについて考察したことに由来する. また，チェス板上の任意の点からナイト（これは八方桂馬である）をすべての目を一度通るように動かすというナイト・ツァーの問題も，ハミルトン・パスの問題として有名である. これに関しては Warnsdorff の手順がある. すなわち,

> さらに続けて移動できる数が（0でないような場所が，幾つかあれば）最小となるような場所へ動かす. もし，その最小値をもつ場所が幾つか存在すればそのうちの一つを任意に選ぶ.

これは，しかし，経験則であり，一般のグラフ，または板の形式を変えた一般化されたチェス板の場合は必ずしも成功しない. そこで，この論文で，筆者は上記の手順を， k 位 (order k) の手順として一般化している. すなわち,

> k 回の移動により得られるすべてのパスを考え各パスに対して残りの結合の数を勘定する. そして，まず上の数が最大になり，ゆきづまりとならないような手を選ぶ. もし，いくつかの手がこの条件を同時に満たせば，$(k+1)$ 位の手を調べる.

ということである. この方法で， k がグラフの頂点の数より1少なくなれば，実際上総てのパスを調べていることになる. 原論文では， $k=1$ とおいて，いくつかのグラフを計算機を用いて調べている. 計算時間はだいたい，グラフのエッジ数に比例し，エッジ数が大きくなると， Fortet の方法よりも有効であると結論している.

なお，出発点以外の各点を一度，しかも，ただ一度に限り通り，最後に出発点にもどるようなパスをハミルトン・サーキットという. ハミルトンが最初に考案したのもこのハミルトン・サーキットである.

また，以上とよく似た問題に，オーイラー・サーキットの問題がある. これは，いわゆる一筆書の問題である. (塚本完治)