マイクロコンピュータによる問題解決

——「騎士の漫遊」パズルとプログラム言語——

若林信夫

1. はじめに

一般に，問題解決は，しばしば，試行錯誤の連続であるといわれるように，ある公式や計算規則を忠実に守って結果を得るのではなく，いろいろと試行錯誤しながら結果を得ることがふつうである。もちろん，解決の過程においては，ある公式や計算規則を用い，発見しながら，結果を得る場合が多い。ゲームやパズルの問題解決は，紀元前の昔から，ゲーム愛好家や数学者等により挑戦されてきたが，その多くは，非常に根気を要する発見的探索法（Heuristics）に基いて，解決に到達してきた。

1940年代後半に電子的な計算機械が発明されると，新しく計算機科学者が誕生した。彼らのあるものは，それを用いて，数学のパズルやチェスゲームを解き，人間の不完全な知能を代替させようとした。それ以来，四色問題の証明や超越数の計算の成功例に見られるように，未解決な問題を解決したり，今まで手計算で得られていた解の再確認を行うことができるようになった。

1970年代後半になると，マイクロプロセッサ（集積回路）の価格が著しく低廉になり，計算機のハードウェアが一部の所有物から市中に出回り，パーソナルな環境で，計算機を使用できる，いわば，パーソナルコンピュータの時代がやってきた。それに伴い，計算機の利用技術（ソフトウェア）も格段に向上し，オペレーティングシステムとその処理系は質・量ともに豊かになってきた。

筆者は，7年程前，[16]において，「ハノイの塔」のパズルの解を探索す

受領日：昭和61年1月18日

るために，13種のプログラム言語[1]を用いて，プログラミングし，ベンチマークを試みた。そのときは，1語が32ビットの汎用の（当時は大型の）計算機の上で行い，「ハノイの塔」の問題解決には，Pascal 言語は，教育的にも良好な環境を与えることを示唆した。その後，前述のように，マイクロコンピュータ（多くは，1語が8ビット又は16ビットであるがメモリ量は殆んど同程度の）が飛躍的に発展したので，同種のベンチマークを新しい環境で試み，結論を再検討してみたい動機が働く。

本稿は，問題解決の例として，「騎士（チェスのナイト）の漫遊」パズルを考察する[2]。このパズルについては，次節で詳しく述べるが，「ハノイの塔」よりも問題解決としては複雑であり，非決定論的に見える。次節では，このパズルを定式化し，解法を論じる。また，それがグラフ理論やオペレーションズ・リサーチとどのような関連があるかを述べる。問題を解く手順には，種々のアルゴリズム（算法）があるが，計算の手間が多項式オーダーで解けるものが望ましい。実際に計算機で解くためには，そのデータ構造とプログラミング技法が効率を支配する。マイクロコンピュータ上で利用できる計算機用プログラム言語は，汎用機のそれよりも豊富になっているが，本稿は，手続き型言語の代表である BASIC, FORTRAN 77, Pascal, Icon, C の5種の異なる言語を選択した。第3節は，付録1のプログラムリストと相俟ってプログラミングのベンチマークを試みる。問題解決と教育的観点から，「アルゴリズム＋データ構造＝プログラム」（Wirth）論を批判的に検討する。 5つの言語はいずれもMS－DOSの上で動かすが，そのうち，Icon は SNOBOL 4 の後継言語で，文字列処理やバックトラックが非常に強力で，移植容易で，読み易く，書き易い言語である。日本ではまだ紹介記事すら殆ど現われていないので，付録2に

1) プログラム言語は英語の Programming Language であるために，しばしば，プログラミング言語といわれるが，本稿は，ＪＩＳ情報処理用語規定（C 6230）に従って，「プログラム言語」という。

2) 同類のものに「エイトクィーン（8人の女王妃）」や「ステーブルマリッジ（安定な結婚）」問題があり，プログラミングの例題でしばしば使用される。後者は，就職試験や大学入学試験の制度の検討や，労働経済学の応用にも使われている。

おいて，その構文，使用法を説明しておく。

最後に，第4節は，問題解決にとって，究極には，良いアルゴリズムの開発と選択の重要性は，プログラム言語の選択に勝ることを指摘し，結語的覚書とする。

本論に先立ち，使用したマイクロコンピュータの環境について触れておく。機種は，日本電気（ＮＥＣ）製の，ＮＥＣ PC-9801 F3（16ビット，512 ＫＢ ＲＡＭ，８ＭＨ，8087無し）を本体とする。ディスクドライブを内蔵し，フロッピーディスク装置１台とハードディスク装置（10メガバイト）１台がある。プリンタは，ＰＣ-101日本語シリアルプリンタ，ディスプレイはＰＣ-ＫＤ551を使う。オペレーティングシステムは，主にＭＳ－ＤＯＳ 2.11版，プログラムは，Turbo Pascal の中のエディタを用いて作成編集した。なお，オペレーティングシステムと各言語は登録商標を持つ。

2．「騎士の漫遊」パズル

「騎士の漫遊（Knight's Tour)」パズルは，チェス盤の上に，騎士（チェスのナイト）をひとつだけ置いて，騎士の移動可能ルールに則して，盤の全てのマス目を１回だけ通過させる遊びである。

このパズルの歴史については，ラウズボールの名著［1］に詳しいが，18世紀以来，著名な数学者であるオイラー（1707—1783），ドモアブル（1667—1754），ヴァンデルモンド（1735—1796）らが解の一部を発表している。しかし，このパズルは，チェス盤を使っているので，チェスの歴史[3]以来，口伝えで楽しまれたものと思われる。

このパズルには，後述するように，いくつかの変形があるが，ここでは，騎

3）チェスの歴史は古代インドに遡るが，現代のチェスは15世紀に確立された。ヨーロッパでは戦術を検討する陣中の必要具として王侯将帥の間に行われたという。チェスはフランスではエシェック，ドイツではシャッハという。日本や中国の将棋（象棋）は，９×９の盤面を使う。チェスの騎士に相当するのは桂馬であるが，前方２個所しか進めない。チェスの騎士を八方桂馬と称し，「騎士の漫遊」パズルを「桂馬道問題」と呼ぶものもある（例えば，小谷［15］）。

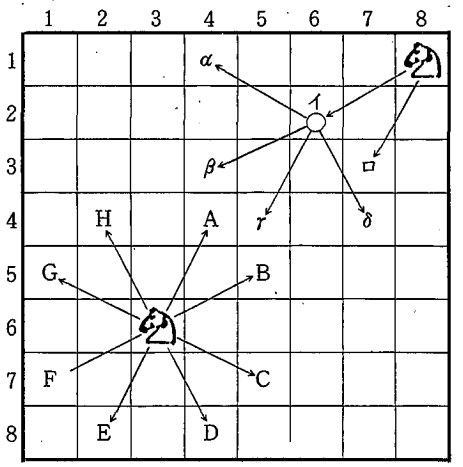

図1. 8×8のチェス盤と騎士の移動

士がチェス盤の任意のマス目から出発し，64個のマス目全部を1回ずつ漫遊するだけでよい，すなわち，必ずしも出発点に戻る経路でなくてもよいとする。出発点に戻る場合は，騎士の巡回，または，再入型の騎士の漫遊パズルと呼び，解の範囲を制限する。後者は，オペレーションズ・リサーチの「巡回セールスマン問題（ＴＳＰ）」と深い関係がある。

チェス盤は，図1のように，８×８のマス目を持ち，騎士はどの場所からも出発できるとする。騎士の移動可能ルールは，騎士が座標位置（６，３）から出発すれば，Ａ，Ｂ，…，Ｇ，Ｈの８通りの場所に進むことができるが，座標（１，８）なら，（２，６），（３，７）の２箇所にしか行けない。かりに，（２，６）を新しい出発点とすれば，今度は，（１，４），（３，４），（４，５），（４，７），（３，８）の５箇所（（２，６）へは戻れない）に進むことができる。このようにして，一足一足進んでいって全部のマス目を訪問したいのであるが，下手をすると，全部をカバーすることなく，「行き停り（八方ふさがり）」の状態に陥る。行き停りの状態を64手目まで延期するにはどう進んでいったらよいか，これが問題である。

そのためのアプローチとしていくつか提案されてきたが，発見的探索法とグ

ラフ理論の応用に分類することができる。

2.1 発見的探索法

発見的探索法は，騎士の移動のルールに従って移動していくが，行き停ったら，一つ手前のステップに戻り，他の移動を試みる。それでも行き停りなら，他に移動するか，もう少し戻るというものである。この発見的探索法は，問題解決の専門用語では，縦型探索（深さ優先＝depth search）法といい，横型探索（幅優先＝breadth search）と区別する。縦型探索法は，バックトラック（後戻り）法[4] ともいい，多くのゲームやパズルで試みられている。しかし，実際に，行き停りに遭遇する前に「先読み（look-ahead)」を行って，バックトラックを最小限にする，いわゆる，「前処理」部分が重要である。「騎士の漫遊」パズルでは，バックトラックを不要にするアルゴズムが存在する。しかし，ここでは，最も素朴な発見的探索法から始める。

いま，「次の移動を試みる」という戦略を Pascal 言語風の手続きで表現すれば，図2のようになる。

```
procedure 次の移動を試みる
  begin 移動方向選択のための初期設定
    repeat 次の移動のリストから候補を選択
      if 騎士の道 then
        begin 移動を記録
          if 盤全部を覆っていない then
            begin 次の移動を試みる
          if 行き停り then 前の記録を消去
            end
        end
    until (移動がうまくいく)∨(もうすべて調べた)
  end
```

図2. バックトラックアルゴリズム

4) 「バックトラック法」は R. J. Walker [12] の命名による。現在，組合せ問題や定理証明の解の至る所で利用され，プログラミングの重要な技法の一つとなっている。プログラム言語の中には，Prolog や Icon のようにその機能をもつものもある。

上の手続きは，騎士の道でなければ，次の候補を探し，騎士の道でしかも盤全部を覆っていないならば，「次の移動を試みる」手続きを再帰的に呼んでいる。移動がうまくできれば，手続きを抜け，行き停りか判定する。

このようにして，騎士の漫遊の道を可能なら逐次的に追加して，行き停りのときは，削除し別の候補を調べる。最終的には，存在すれば，道を完成することができる。

上の手続きにおいて，「次の移動のリストから候補を選択する」文は，「前処理部」で，発見的探索法の効率を左右するカギである。

もっとも素朴な方法は，騎士の進める所を系統的に（例えば，時計回りに），とり出す方法である。しかし，盤面のすべてのマス目は等確率で訪問されない。例えば，（1，1）は，（3，2），（2，3）の2箇所からの訪問しか可能ではなく，盤の中心部に至るにつれ，訪問される確率は高い。したがって，これから選ぶ方向にウェイト付けをすることができる。各マス目に対し，騎士の移動場所数（連結度）を計算すれば，図3のようになる。この図は，盤のマス目のウエイト付けを与えるので騎士の漫遊の方向を定める。実際，ドイツの数学者，H. Warnsdorff は1823年，次の簡単なルールを提案した（[13]）。

	1	2	3	4	5	6	7	8
1	2	3	4	4	4	4	3	2
2	3	4	6	6	6	6	4	3
3	4	6	8	8	8	8	6	4
4	4	6	8	8	8	8	6	4
5	4	6	8	8	8	8	6	4
6	4	6	8	8	8	8	6	4
7	3	4	6	6	6	6	4	3
8	2	3	4	4	4	4	3	2

図3　騎士の各点における連結度

「次の移動が，1以上の最小数となる方向に移動せよ。もし，同じ最小数なら，どちらかを任意に選べ」

というものである。このルールを，上図をもとに具体的に述べよう。

いま，騎士が座標（1，1）から，出発する。騎士は，（3，2）と（2，3）に進めるが，それぞれの連結度はともに6であるから，（2，3）を任意に到着地とする，また，（2，4）から出発すれば，6方向のうち（1，2）が最小数ゆえ，到着地とせよ，というものである。しかし，図から明らかなように，同数になるマス目が多すぎるので，もう一つ先に行ったマス目の連結度まで考慮した方がよい。騎士の2段先の連結度は，騎士の一段目に可能なマス目の連結度を加え合せればよい。例えば，座標（6，3）の連結度は，許される一段目の座標が，（7，5），（7，1），（5，5），（5，1），（8，4），（8，2），（4，4），（4，2）であって，その連結度はそれぞれ，6，3，8，4，4，3，8，6であるから，合計42ということになる。この数字の先頭に小数点をつけ（．42），1段目に加えてできた指数が，図4である。タイブレークを避けるためには図3よりも図4を使う方が好ましい。この表は，上下，左右対称になっているので，対称性を利用すれば記憶する部分は節約できる。

	1	2	3	4	5	6	7	8
1	2.12	3.18	4.23	4.26	4.26	4.23	3.18	2.12
2	3.18	4.24	6.32	6.37	6.37	6.32	4.24	3.18
3	4.23	6.32	8.42	8.48	8.48	8.42	6.32	4.23
4	4.26	6.37	8.48	8.56	8.56	8.48	6.37	4.26
5	4.26	6.37	8.48	8.56	8.56	8.48	6.37	4.26
6	4.23	6.32	8.42	8.48	8.48	8.42	6.32	4.23
7	3.18	4.24	6.32	6.37	6.37	6.32	4.24	3.18
8	2.12	3.18	4.23	4.26	4.26	4.23	3.18	2.12

図4. 騎士の2段移動指数

Warnsdorff ルールの最も重要な点は，彼のルールに従えば，バックトラックの必要がなく，直接的に，64回（n×n盤なら，n^2 回）のサーチだけで解を得ることである。したがって，騎士の漫遊パズルは，多項式オーダーの手間で解ける。しかしながら，Warnsdorff ルールは，矩形盤に対し常に適用可能かどうかは，数学的に未解決である。また，再入型の騎士の漫遊パズルを直接的に解かない。

発見的探索法の効率を支配する他の要素は，盤のデータ構造と，移動方向選択のための初期設定である。それについては，プログラミングを扱う，第3節の話題である。

2.2 グラフ理論による解法

グラフ理論は，頂点（node, vertex）の集合と辺（arc, edge）の集合から作られたグラフを数学的に考究する学問である。8×8のチェス盤の上で，騎士の漫遊する経路を求めることは，騎士の通過できるマス目を線で結べば，グラフができることから，グラフ理論の応用である。いわゆる，ハミルトン路（閉路）は，グラフの各頂点をちょうど一回ずつ通る路（閉路）である。これ

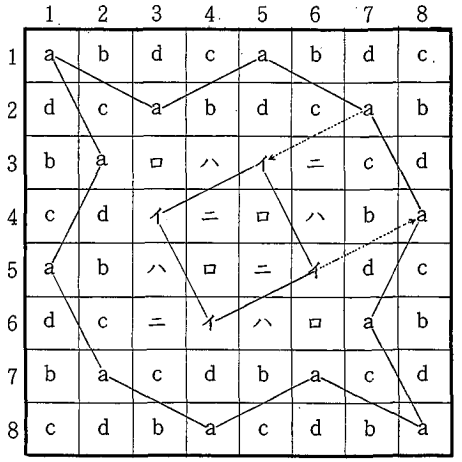

図5. グラフのファクターによるハミルトン閉路の構成

は，1859年にアイルランドの数学者ハミルトン（Sir William Rowan Hamilton, 1805-1865）が提案したことによる。ここでは，Berge［2］に従って，ハミルトン閉路すなわち，出発点に戻る再入型の閉路に限定しそのアルゴリズムを考えよう。

いま，8×8のチェス盤の対称性を利用して騎士の漫遊する8種類のハミルトン閉路を作ることができる（図5）。それらを，a，b，c，d，イ，ロ，ハ，ニとする。任意の2つの閉路をとる。一方の閉路の2つの連続する頂点が他方の閉路の2つの連続する頂点に隣接しているならば，この2つの閉路は結線することができる。例えば，上図でa—閉路とイ—閉路をとると両者にはその関係が成り立っているから2箇所で2つの点を結線することができる（図の2箇所の点線を見よ）。同じようにして，イとb，bとロ，ロとc，cとハ，dとニ，そして，ニとaについても可能であり，結局，全てのマス目を覆うハミルトン閉路（再入する「騎士の漫遊」巡路）を導くことができる（図6）。

上記の性質は，グラフ理論ではファクターという概念に対応する。すなわち，ファクターとは，互いに共通な頂点を持たない初等閉路の集まりでしかもグラフのどの頂点もそれらの閉路のどれか一つに含まれているものである。ファクターの存在する盤ではハミルトン閉路を容易に作ることができるが，ファクターはどんな盤でも存在するわけではないから，グラフ理論による上記の解法は特

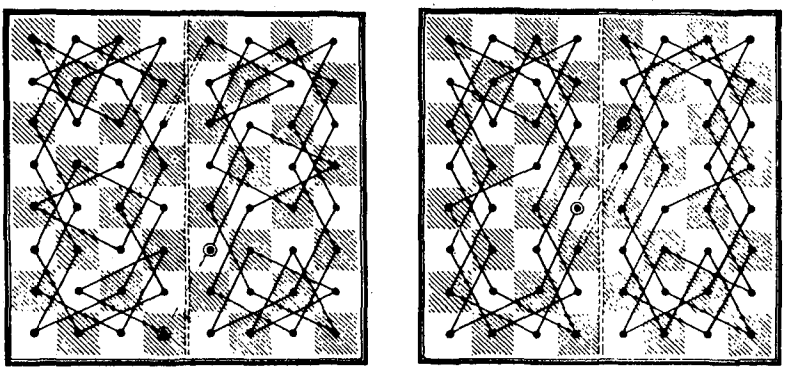

図6. 完成したハミルトン閉路（2種類）Berge [2, p. 109]

殊（ad hoc）である。

ハミルトン閉路を求めることは，オペレーションズ・リサーチの巡回セールスマン問題（Traveling Salesman Problem, TSP）と密接な関連がある。それゆえ，巡回セールスマン問題でこれまで開発された各種の解法を援用して，騎士の漫遊路を求めることができる。巡回セールスマン問題では，通常，頂点と頂点の間の辺のコスト（費用，時間，距離など）が可変であるが，騎士の漫遊路では，一定値（マスの辺の長さを1とすれば，可能な道は$\sqrt{5}$のコストであるが，プログラミング上では，0とする。）である点に違いがある。可能な64都市を一巡する巡回セールスマン問題は，ＴＳＰの解法のベンチマークの例題として，Held and Karp［5］により使われてきたが，難解な問題の一つである。線形計画に直せば，次のようになる。

$$c_{ij}=\begin{cases}0, \text{ もし, } i \text{ と } j \text{ が隣接している, } i, j=1, 2, \cdots 64\\ \infty, \text{ さもないとき}\end{cases}$$

として，

$$min \quad \sum_{1\leqq i<j\leqq 64} c_{ij}\ x_{ij}$$

s/t

(i) $\sum_{j>i} x_{ij}+\sum_{j<i} x_{ji}=2, \quad (i=2,\ldots, 63)$

(ii) $\sum_j x_{1j}=2,$

(iii) $\sum_{1\leqq i<j\leqq 64} x_{ij}=64,$

(iv) $\sum_{i<j\,i\,j\in S} x_{ij}\leqq |S|-1, \qquad S\subset\{2,3,\ldots,64\}$

(v) $x_{ij}\leqq 1,$

(vi) $x_{ij}\geqq 0,$

(vii) x_{ij} は整数，

である。ここで，制約（i）は，各頂点が，次数2をもち，（ii）は，出発点①は，次数2をもち，（iii）は，経路の長さは64になること，（iv）は，頂点のどんな真部分集合をとっても部分閉路は存在しないことを表わす。

騎士の漫遊パズルを巡回セールスマン問題に変換して解くことは，アルゴリズムの手間から得策ではないので，本稿では定式化にとどめる。

2.3　その他の「騎士の漫遊」パズル

「騎士の漫遊」パズルには，いくつかの変形があり，いくつかの性質が発見されているので整理しよう。

(1)　騎士の漫遊が得られる最小の盤は，(1×1を除けば，以下同様)，3×4の矩形盤であり，方形盤では，5×5の盤である（図7）。

(2)　5×5の盤には，1728通りの漫遊路がある。しかし，回転，逆転，反射を除けば，本質的には，112個の異なる漫遊路しかない。どの経路も隅から出発し，最後に別のマスに到着する。

(3)　8×8の（チェス）盤には，3825通りの漫遊路がある［1876年，フランスの Flye Sainte-Marie［11］は盤の対称形を利用して算出した］。

(4)　再入型の騎士の漫遊路（巡回路）が存在する最小の盤は，矩形では，3×10，方形では，6×6の盤である。

(5)　8×8の盤には，足跡番号をもとに，準魔方陣，すなわち，すべての行の和とすべての列の和がちょうど260になるものが存在する（図8）。

	1	2	3	4
1	1	4	7	10
2	12	9	2	5
3	3	6	11	8

	1	2	3	4
1	1	4	7	10
2	8	11	2	5
3	3	6	9	12

(a)　矩形では3×4盤が最小

	1	2	3	4	5
1	1	24	13	18	7
2	14	19	8	23	12
3	9	2	25	6	17
4	20	15	4	11	22
5	3	10	21	16	5

(b)　方形では5×5盤が最小

図7.　騎士の漫遊

	1	2	3	4	5	6	7	8
1	1	48	31	50	33	16	63	18
2	30	51	46	3	62	19	14	35
3	47	2	49	32	15	34	17	64
4	52	29	4	45	20	61	36	13
5	5	44	25	56	9	40	21	60
6	28	53	8	41	24	57	12	37
7	43	6	55	26	39	10	59	22
8	54	27	42	7	58	23	38	11

図 8. 準魔方陣となる騎士の漫遊路

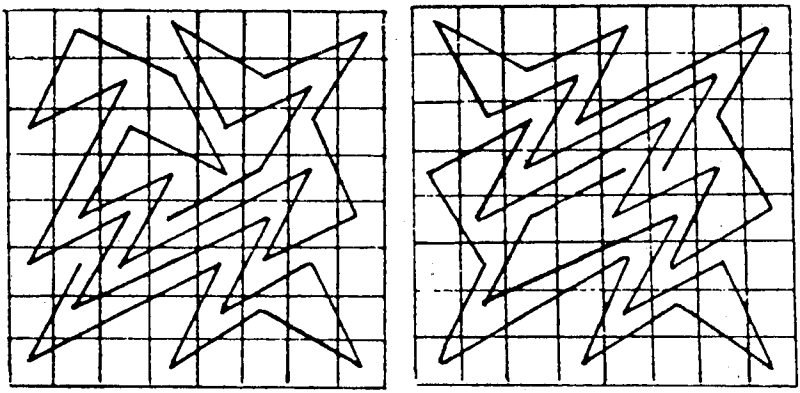

図 9. 交差しない最長の騎士の漫遊路 (Knuth [8])

(6) 8×8のチェス盤で，騎士が交差しない最長の経路は，長さが35で，2つある（図9）。9×9の将棋盤では47跳びである。

(7) 探索木（バックトラックトリー）は，指数的に増大する。6×6盤では，88,467点，7×7盤では，10,874,674点，8×8盤では，3 ＝205,891,132,094,649点になる。(Knuth [8])。

3. 発見的探索法のインプリメンテーション

本節は,「騎士の漫遊」パズルの発見的探索版を，主に，マイクロコンピュータ上で，プログラミングし，プログラム言語のベンチマークを行う。

計算機用プログラム言語は，標準規格の整ったものから，自分1人だけが使う言語まで非常に多数ある。そして各言語はそれぞれ独自の機能をもつので，ある一つのパズルを解くだけで言語のベンチマークを試みても意味のある結論を導出することは無理があろう。ここでは，あえて，言語の特徴を説明しながら，異ったプログラムを作成し，比較・対比し，管理実験（controlled experiment）を行う。

われわれは，下記の3種類のプログラムを作成し，その全リストを付録1に掲げた：

1．素朴なバックトラック法，

2．ヒューリスティック戦略，

3．Warnsdorff 戦略。

まず最初は，図2で説明した素朴なバックトラック型のアルゴリズムの精緻化である。これは再帰手続きであるが，BASIC や FORTRAN のために，非再帰型に変換したものを最初にテストする（Wirth [14]）。

次は，ヒューリスティック戦略で，盤面の移動可能方向をシミュレーションによって選択する。行き停りになると，盤面をリセットすることにより，再帰を避けている（Meissner 他 [9]）。

最後に，Warnsdorff 戦略である。これは，盤面の移動可能方向を，二段先読みによって，バックトラックすることなく，直接的に解を得るものである（Gilpin [4]，Irwin [6]，Joshi [7]）。

最初の2つの戦略は，盤面と騎士の動きについて次の「データ構造」を用意する（Warnsdorff 戦略についてのデータ構造は，Icon の項で述べる）：

BOARD [1..8，1..8] ＝ 8×8 のチェス盤．初期には，未踏査ゆえ0をとる。

A [1..8], B [1..8] =騎士の基本移動パターンを示すX座標の乖離とY座標の乖離，常に（A, B）=（2, 1），（1, 2），（－1, 2），（－2, 1），（－1, －2），（1, －2），（－2, －1）の8通りが可能である。

プログラム言語は，BASIC, Pascal, FORTRAN 77, Icon, Cの5種類の手続き型言語を用いた。実験には，APL, LISP, LOGO, Prolog も使用したが，本稿では，焦点を絞るために割愛し，他稿に譲る。

（a）**BASIC**（Beginner's All-purpose Symbolic Instruction Code）

BASIC はマイクロコンピュータの代表的なプログラム言語である。BASIC は本来，コンパイル型でかつ対話型を意図して，米国ダートマス大学でケメニーとクルツにより作成された（1956－1963年）。しかし，今日，マイクロコンピュータで使われる多くのBASIC は，インタプリタ型で，オブジェクトコードを発生せずに解釈・実行してしまうために，スピードが遅い。しかし，最近になって BASIC コンパイラも使用可能になり，コンパイラ型はインタプリタ型よりどの程度，得策かを「騎士の漫遊」パズルでテストしてみた。その結果5×5の盤で一つの解を得るために，同一のプログラムを流しても，約6倍の差があった（表1の上部参照）。5分間とか30分間，計算結果を待つことは苦痛であるが，8×8の盤面では，コンパイラを使用しても24時間の実行時間内には答を得ることができなかった。次に，汎用の計算機（MELCOM-COSMO 700C）ではどの程度改良されるかをみた（表1の最下行）。MELCOM BASIC はインタプリタであるにも拘らず，たったの13秒で結果を得た。しかし，このことから，BASIC なら，MELCOM

表1. BASIC ベンチマーク (5×5盤)

種　　類	名　　称	実行時間[5]	比 率
インタプリタ	BASIC-86 Rev. 5.27	30分01秒	138.5
コンパイラ	N 88-BASIC (86) Compiler	5分06秒	23.5
汎用機インタプリタ	MELCOM BASIC	13秒	1

5) 本稿でいう「実行時間」はCRTへの出力時間や時間計測ルーチンの呼出し時間を含む。

BASIC を使えという即断は許されない。MELCOM BASIC は変数が一文字しか使えない。また，マイクロコンピュータの BASIC と違って，画面制御ルーチンがないので，バックトラックの状況を観察することが難しい。何よりも汎用の計算機は使用時間が限られ，実行ＣＰＵの制限時間に抵触する。８×８のチェス盤に対しては，マイクロコンピュータでは三昼夜経つと（82時間59分後）答を印刷するが，汎用計算機では１時間で実行時間打切りのメッセージを出し，１時間分の使用料がかかる。

再帰的プログラムが自由に使えないことも BASIC の欠点である。BASIC では，確かに自分自身のサブルーチンを GOSUB で呼ぶことが制限的に出来るが，引数を自動的に渡すことはできない。そのため，スタック（棚）を用意してパラメータをプッシュダウン，またはポップアップにより管理しなければならない。

(b) Pascal

Pascal 言語は，スイスの N. Wirth が1967年，米国スタンフォード大学のＣＳ236の講義のために教育目的と，Algol 60 の後継コンパイラ案として作成したものである。データ構造と制御構造の機能が豊富で，数百行程度のプログラムなら十分にその効果を発揮する。「騎士の漫遊」パズルを解く程度なら Pascal で十分であるが，分離翻訳の機能を持たないために，１箇所のプログラムの訂正でもはじめから翻訳し直さなければならない欠点がある。

Wirth は1980年，筆者がＥＴＨを訪問したとき，Modula-2 言語をデモンストレーションし，Pascal は使っていないと言ったことを思い出す。

現在，マイクロコンピュータ上では多数の Pascal コンパイラが利用可能であるが，われわれは，Borland International の Turbo Pascal 3.0 版を用いた。この製品は，価格が安く，コンパイル時間が速いメリットをもつ。さらに，そのスクリーンエディタはプログラムの作成，編集に重宝である。

プログラムは，BASIC 版と同じくバックトラック法によるもので，Wirthのオリジナルな再帰版による。５×５の盤で，１つの解を得るのに，39秒，６×６盤では18分30秒，７×７盤では８時間55分，８×８のチェス盤では

表2. Pascal での実行時間

盤面	マイクロコンピュータ Turbo Pascal	汎用機 Pascal 8000
5×5	39秒	11秒
6×6	18分30秒	5分25秒
7×7	8時間55分	アボート
8×8	10時間01分	アボート

10時間を要した（表2参照)。オリジナルな再帰版を単純に非再帰版に直して試みた（BASIC 版を Pascal 版にしたもの）が，所要時間に変わりはなかった。なお，MELCOM の Pascal 8000では，7×7盤以上では，BASIC 同様，ユーザ割当てのＣＰＵ時間（60分）では解くことができなかった。

実験の結果，次の4点を観察することができる。

1）Pascal は BASIC よりも実行時間がはるかに速い。Turbo Pascal は，汎用機の Pascal 8000よりも実行効率がよい。

2）再帰プログラムは非再帰版よりもプログラムステップ数が短かく，簡潔でわかり易い。実行時間も両者に差異はない。よって，再帰プログラムで実行時間がかかるものを単純に非再帰版に変換しても得策ではない。

3）汎用機で実行時間の制限にひっかかりそうなプログラムはマイクロコンピュータを利用した方がコスト的に有利である。

4）バックトラックプログラムは実行するのに指数的なオーダーで時間がかかる。Knuth［8］のように，予め，探索木の大きさを推定するか，より効率のよい直接的な算法を工夫すべきである。Wirth［14］の例は，再帰的アルゴリズムないし，バックトラックアルゴリズムの学習教材として理解すべきで実用的でないことを明言すべきであった。

（C）FORTRAN 77

FORTRAN（Formula Translator）は，1954年 IBM 704の発表のあと IBM の J. Backus らが設計・製作し，科学技術計算用に長く使われ

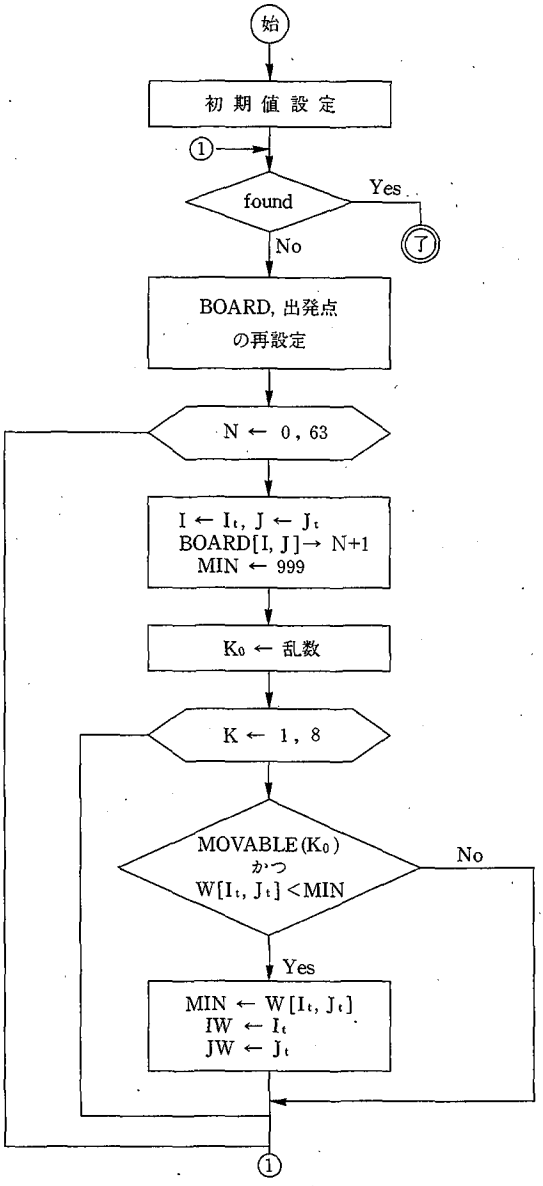

図 10. 発見的戦略のフローチャート

てきた。改訂も二，三回なされたが，Pascal やCのいわゆる現代プログラム言語の影響を受けて，1977年の標準版，すなわち，FORTRAN 77 が現在入手できる最新版である（FORTRAN- 8 X も検討されている)。これには，種々の新しい機能（例えば，CHARACTER タイプや，制御の流れの構文 $\begin{pmatrix}\text{IF}\sim\text{THEN}\\ \text{文 1}\\ \text{END IF}\end{pmatrix}$ が追加され，科学技術の応用分野を拡張している。また，FORTRAN IV までは使いにくかった，.LT.，.LE.，.GT.，.GE.，.EQ.，.NE. はそれぞれ，<，<=，>，>=，=，<>（><）によって代替できる。

われわれは，NECの日本語 PC-FORTRAN を用いたが，FORTRAN 77 をほぼ満足している。

しかし，FORTRAN 77 になっても，FORTRAN にはスタックが存在しないので，再帰を許さない。もし再帰的プログラムを利用するならば，前述のように，プログラムでスタックを生成して繰返し型に変換しなければならない。ここでは，「騎士の漫遊」パズルを解くのに，(2)の「発見的戦略」によった（図10のフローチャートと，付録1のプログラムリストを参照)。

プログラムにおいて，第1カラムにCのある行は注釈行で日本語が書ける，RANDOM（0）は，0～1の擬似乱数列を発生し，次の移動方向をランダ

表3. モンテカルロ・シミュレーションによる探索

漫遊路の長さ	1000回試行	100回試行
64	100回	11回
63	723	68
62	12	0
61	103	3
60	47	16
59	0	2
58	15	0
実行時間	5分37秒	34秒

ムに選択している。それをサブルーチン ＭＯＶＥで用い，移動可能か調べている。このやり方は，いわば，モンテカルロ・シミュレーション法ともいえる。実行時間は，８×８盤の64個のマス目を漫遊するひとつの路を探すのに４秒かかり，100回当りは34秒，1000回当りは５分37秒かかった（表３）。

FORTRAN 77 は，再帰を許さない欠点はあるが，使用実績があるために，問題解決としては今後とも根強い支持がありそうである。

(d) Icon

Icon は，SNOBOL（StriNg Oriented symBOlic Language）の流れを汲む，非数値の処理を指向する新しいタイプのプログラム言語である。Icon 言語の詳細については，付録２に譲るが，付録１のプログラムリストから分かるように Pascal よりはＣ言語に近い。しかし，Prolog のように，式の評価が成功（success）か失敗（fail）かで制御が移る点，特色がある。

米国アリゾナ大学ではＰＤＰ-11 上で教育・研究用に７，８年使用されてきたが，1985年にはＭＳ-ＤＯＳ版がペンシルバニアの C. Wills により完成し，公開配布された。われわれは，ＭＳ-ＤＯＳ上の Icon を用いるが，汎用機上では利用可能ではない。

パズルのプログラムは，Warnsdorff の直接法による。この方法の実行効率をあげるためにデータ構造を改造する。

［データ構造］

「騎士の漫遊」パズルを計算機のプログラムに変換するとき，盤面と騎士の移動のデータ構造が重要であることは既に述べた。しかし，BASIC, Pascal, FORTRAN 77 いずれも８×８の盤に対して，BOARD［1..8，1..8］を用意し，未探査の場所を０に，騎士のたどる道順を，１，２，…と番号付けることをしてきた。そのとき，騎士の移動は，盤面の位置によって，動きが制限された。このことは，BOARD［-1..10，０..9］を用意することによって，８通りの対称な移動が可能になる。また，配列の添字計算には実行時間がかかるので，２次元配列よりは１次元配列にし，番号付けを０..119とする（図11）。また，未探査のマスは０，通行禁止のマスは１に初期設定する。配

	0	1	2	3	4	5	6	7	8	9
−1	0 1	1 1	2 1	3 1	4 1	5 1	6 1	7 1	8 1	9 1
0	10 1	11 1	12 1	13 1	14 1	15 1	16 1	17 1	18 1	19 1
1	20 1	21 0	22 0	23 0	24 0	25 0	26 0	27 0	28 0	29 1
2	30 1	31 0	32 0	33 0	34 0	35 0	36 0	37 0	38 0	39 1
3	40 1	41 0	42 0	43 0	44 0	45 0	46 0	47 0	48 0	49 1
4	50 1	51 0	52 0	53 0	54 0	55 0	56 0	57 0	58 0	59 1
5	60 1	61 0	62 0	63 0	64 0	65 0	66 0	67 0	68 0	69 1
6	70 1	71 0	72 0	73 0	74 0	75 0	76 0	77 0	78 0	79 1
7	80 1	81 0	82 0	83 0	84 0	85 0	86 0	87 0	88 0	89 1
8	90 1	91 0	92 0	93 0	94 0	95 0	96 0	97 0	98 0	99 1
9	100 1	101 1	102 1	103 1	104 1	105 1	106 1	107 1	108 1	109 1
10	110 1	111 1	112 1	113 1	114 1	115 1	116 1	117 1	118 1	119 1

図11. 縁付き8×8チェス盤

列を1次元にした結果，騎士は，8通りの方向全部を探査することができる。例えば，出発地を（1，1）＝（21）とすれば，13，33，42,40，(29)，(9)，0，2が，その対象である。そのとき，現在位置との乖離は，それぞれ，−8，12，21，19，8，−12，−21，−19であるから，配列名JUMPの各要素は±21，±19，±12，±8をとりうる。新位置は，

KT＋JUMP［TRY］

で求められる。

一般に，n×nの盤では，BOARD ベクタは，(n＋4)×(n＋2)の大きさになり，配列名JUMPの中身は

±(2n＋5)，±(2n＋3)，±(n＋4)，±n

で表わされる。

Warnsdorff 法のフローチャートを図12で示す。

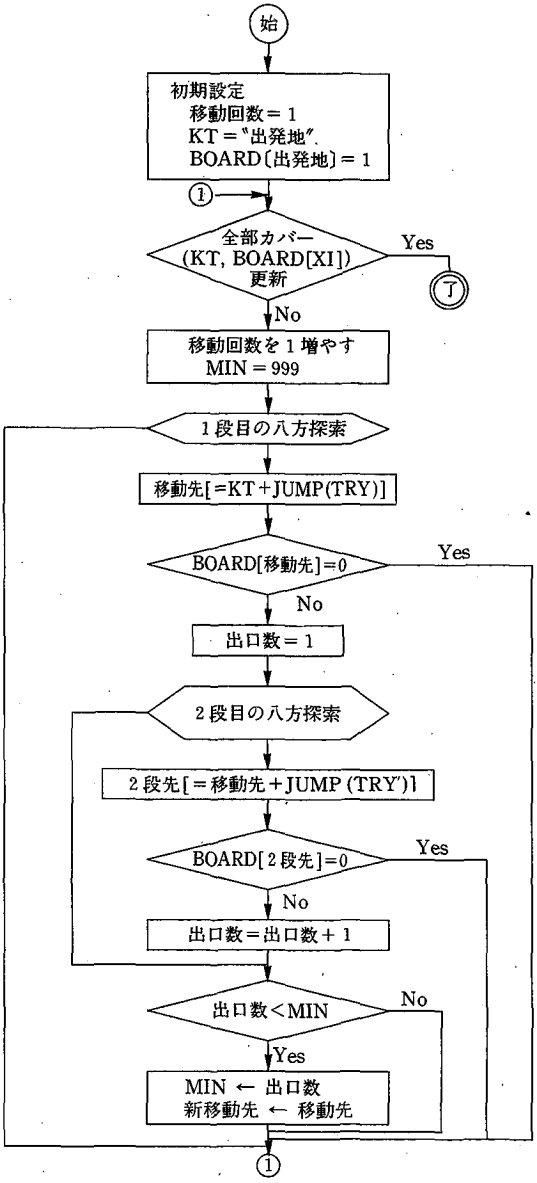

図 12. Warnsdorff 戦略のフローチャート

この方法は，再帰を含まず，二重のループ構造だけで直接的に解を得る。さらに，二段先読み表（図4）を用いれば，一重（8通り）のループ構造で，解を得る。出発地を移動させれば，すべての出発地に対して，一つまたはゼロ個の漫遊路を得る。しかし，再入の漫遊路（閉路）を導くものではないことに注意する。

実行時間は8×8のチェス盤で一つの解を得るまで3秒足らずである。

(e) C

1972年，DECのPDP−11用に，D. Ritchie（ベル研究所）が設計・製作したC言語は，もともと，システム記述言語であり，UNIXオペレーティングシステムの補完言語であったが，今日，われわれのMS−DOS上でも全ゆる問題領域のプログラムを書くことができるようになった。Cは，Pascalほど高級言語ではなく，デバッグに時間がかかるが，コンパクトで強力なスタイルをもつので，パズルやゲームのプログラミングに人気がある。われわれは，マイクロコンピュータ上では，Lattice Cの2.04版を，汎用の MELCOM機では，UOS−C（岡山理科大学，木村宏氏提供）2.6版を用いた。

プログラムは，付録1のリスティングにあるように，Icon と同じく Warnsdorff 戦略を用いた。実行時間は，Icon 同様，3秒足らずで解を得たが，プログラム言語の全般的主観的評価は Icon の方が優る。

4．結語的覚書

われわれは，発見的探索法（試行錯誤法）による問題解決の問題として有名な，「騎士の漫遊」パズルをとりあげ，マイクロコンピュータ上で管理実験を試みた。

マイクロコンピュータは比較的安価で，パーソナルな環境で，好きな時間だけ利用でき，豊富なプログラム言語が使えるので，アルゴリズムの設計，開発，評価には非常に便利な道具である。

筆者は，20年前，オペレーションズ・リサーチやグラフ理論の例題において，「騎士の漫遊」パズルに初めて接した。プログラムの技法として再帰的アルゴ

リズムがあり，その例題になっていることを知ったのは最近である。プログラミングを進めるうちに，プログラムの技法やプログラム言語の選択もさることながら，より効率的なアルゴリズムの開発またはその選択が重要であること，またその好例が，このパズルを含むバックトラック法であることがわかった。

マイクロコンピュータは今日，日進月歩で，ますます身近になり，機能も強化されているが，問題解決のためには，一方では，計算機の環境の整備のために，他方ではより本質的なアルゴリズムの開発と選択のために考究しなければならないと思う。

付録1. プログラムリスト

```
type kight.bas
5 TIME$="00:00:00"
10 ' ============================================================
20 '            Knight's Tour
50 ' ============================================================
60 '  initialize  a(8),b(8),h=board
70 DIM C(25),H(5,5)
80 N=5:N1=N*N
100 ' ------- main routine -------
110 GOSUB 1000
120 GOSUB 2000
130 GOSUB 3000
135 PRINT TIME$;
140 STOP
1000 ' set up .....
1060 FOR K=1 TO 8: READ A(K),B(K): NEXT
1090 DATA 2,1,1,2,-1,2,-2,1,-2,-1,-1,-2,1,-2,2,-1
1100 X=1:Y=1
1120 RETURN
2000 ' ------------------------------
2010 '   try sub.
2020 '     entry : n, n1, a(k), b(k), x,y
2030 '     modify: k,p,u,v,i,x,y,c(i)
2040 '     exit  : h(u,v)
2060 '   -------------------- initialize
2070 P=N: I=2: H(X,Y)=1: K=0
2110 '  ============== selection of moves
2120 IF I<=0 OR I> N1 THEN 2430
2130 K=K+1: C(I)=K
2150 IF K > 8 THEN 2330
2190   U=X+A(K)
2200   IF U < 1 OR U > N THEN 2110
2210   V=Y+B(K)
2220   IF V < 1 OR V > N THEN 2110
2230   IF H(U,V) > 0 THEN 2110
2240    '=>=>=>=>=>=>=>=>=>=>=>=> forward move =>=>=>
2250    H(U,V)=I
2270    I=I+1
2280    IF I>N1 THEN 2310
2290    X=U:Y=V
2310    K=0
2320    GOTO 2110
2330    '<-<-<-<-<-<-<-<-<-< backtrack <-<-<-
2350    H(X,Y)=0
2360    I=I-1
2370    IF P < I THEN 2390
2380    P=I
2390    K=C(I)
2400    X=X-A(K): Y=Y-B(K)
2420  GOTO 2110
2430 RETURN
2440 '
3000 REM --------------------------
3010 '     output sub.
3020 '     h(I,m) =. output device
3030 '
3050  FOR M=1 TO N
3060  PRINT USING "#### #### #### #### #### ";H(1,M),H(2,M),H(3,M),H(4,M),H(5,M)
3130 NEXT
3140 RETURN
3150 END
```

リスト1. BASIC

```
type b:knp8.pas
program knp;
const n=8; (* n is no. of sides *)
      nsq=64; (* nsq=n*n        *)
      x=1; y=1;  (* start position (x,y)=(1,1) *)
type  index=1..n;
      TimeString=string[8];
var
      q: Boolean;
      s: set of index;
      a, b: array[1..8] of integer;
      cb:    array[index,index] of integer;
procedure init;
var i,j: index;
begin s:=[1,2,3,4,5,6,7,8];
   a[1]:=  2;  b[1]:=  1;  a[5]:=-2; b[5]:=-1;
   a[2]:=  1;  b[2]:=  2;  a[6]:=-1; b[6]:=-2;
   a[3]:= -1;  b[3]:=  2;  a[7]:= 1; b[7]:=-2;
   a[4]:= -2;  b[4]:=  1;  a[8]:= 2; b[8]:=-1;
   for i:=1 to n do
     for j:=1 to n do cb[i,j]:=0;
   cb[1,1]:=1;
end;
procedure try(i: integer; x,y: index; var q: Boolean);
var k,u,v: integer; q1: Boolean;
begin k:=0;
   repeat k:=k+1; q1:=false;
      u:=x+a[k]; v:=y+b[k];
      if (u in s)and (v in s) then
      if cb[u,v]=0 then
      begin cb[u,v]:=i;
         if i<nsq then
         begin try(i+1,u,v,q1);
            if not q1 then cb[u,v]:=0
         end else  q1:=true
      end
   until q1 or (k=8);
   q:=q1
end;
procedure result;
var i,j: index;
begin if q then
   for i:=1 to n do begin
      for j:=1 to n do write(cb[i,j]:5);
      writeln
   end else writeln(' No Solution for the start (',x:2,y:2,')');
end;{$I current.pas }
begin { main }writeln(time);
   init;
   try(2,1,1,q);
   result;writeln(time)
end.
```

リスト2. Pascal

```
type k b:knf2.for
C ===== Knight's Tour by heuristic strategy.
      CHARACTER TT*8
      LOGICAL MOVE,FOUND
      INTEGER BOARD, HIST(64)
      REAL W(8, 8)
      COMMON BOARD(8,8),K0
      DATA (HIST(L), L=1,64)/64*0/
C Initialize  table of weights.
      DATA W/2.12,3.18,4.23,2*4.26,4.23,3.18,2.12
     1 ,3.18,4.24,6.32,2*6.37,6.32,4.24,3.18,
     2  4.23,6.32,8.42,2*8.48,8.42,6.32,4.23,
     3  4.26,6.37,8.48,2*8.56,8.48,6.37,4.26,
     4  4.26,6.37,8.48,2*8.56,8.48,6.37,4.26,
     5  4.23,6.32,8.42,2*8.48,8.42,6.32,4.23,
     6  3.18,4.24,6.32,2*6.37,6.32,4.24,3.18,
     7  2.12,3.18,4.23,2*4.26,4.23,3.18,2.12/
   66  format(1h )
       CALL TIME(TT)
       write(6,69) TT
   69  FORMAT(1H ,A)
        FOUND = .FALSE.
C       RESET BOARD=0 (IT, JT)=(1, 1)
    1     DO 11 I=1,8
       DO 11, J=1,8
   11     BOARD(I,J)=0
       IT=1
       JT=1
       DO 3, N=1,63
         I=IT
         J=JT
         BOARD(I,J)=N
C        ... Generate random starting direction for next move.
         K0=INT(8.0*RANDOM(0))+1
C        ... Find minimum weight among legal moves.
         Wmin=99.0
         DO 4,K=1,8
           IF(MOVE(I,J,K,IT,JT)) THEN
           IF(W(IT,JT) <= Wmin) THEN
            Wmin=W(IT, JT)
            IMW=IT
            JMW=JT
          END IF
          END IF
    4      CONTINUE
           IF(Wmin >= 98.0) GOTO 7
           IT=IMW
           JT=JMW
    3    CONTINUE
C       .... Tour is complete. N=63.
       FOUND= .TRUE.
       WRITE(6, 6)((BOARD(I,J),J=1,8),I=1,8)
    6  FORMAT(1H ,8I3)
       WRITE(6, 66)
    7  HIST(N)=HIST(N)+1
       IF( .NOT. FOUND)GOTO 1
       WRITE(6, 6) HIST
       CALL TIME(TT)
       WRITE(6,69) TT
       END
       LOGICAL FUNCTION MOVE(I,J,K,IT,JT)
       INTEGER BOARD
       COMMON  BOARD(8,8), K0
       INTEGER IM(8), JM(8)
       DATA IM/ 1,2,2,1,-1,-2,-2,-1/
       DATA JM/ 2,1,-1,-2,-2,-1,1,2/
       K1=MOD(K+K0-2, 8)+1
       IT=I+IM(K1)
       JT=J+JM(K1)
        IF( ((IT >=1) .AND. (IT <= 8)) .AND.
     1        ((JT >=1) .AND. (JT <= 8))) THEN
        IF (BOARD(IT,JT) = 0) THEN
         MOVE= .TRUE.
         ELSE
         MOVE= .FALSE.
        END IF
        END IF
        RETURN
        END
```

リスト3. FORTRAN 77

```
procedure main() # Knight's  tour  by Icon language
 static jump, board
 initial { board:=list(120,1)
           jump:=list(8,0)
         }
         jump[1]:=-19; jump[2]:=-8; jump[3]:= 12; jump[4]:= 21
         jump[5]:= 19; jump[6]:= 8; jump[7]:=-12; jump[8]:=-21
         every j:=0 to 7 do  every i:=22+10*j to 29+10*j do board[i]:=0
         kt:=22; board[kt]:=1;move:=1;
# search by Warnsdorff algorithm
  repeat { move+:=1; min:=99;
           every try:=1 to 8 do
           { nex:=kt+jump[try]
             if board[nex]=0 then
             { exits:=1
               every try2:=1 to 8 do
               { testsq:=nex+jump[try2]
                 if board[testsq]=0 then exits+:=1
               }
               if exits < min then  { min:=exits; nextsq:=nex }
             }
           }
           if nextsq ~= kt then { kt:=nextsq; board[kt]:=move }
           else {
                 every i:=0 to 7 do {
                 every  write(board[10*i+22]," ",board[10*i+23]," ",
                 board[10*i+24]," ",board[10*i+25]," ",board[10*i+26]," ",
                 board[10*i+27]," ",board[10*i+28]," ",board[10*i+29]) }
                 break
                }
         }
 end
```

リスト 4. Icon

```
main()  /* Knight's Tour by C programming Language */
{       /*       Warnsdorff algorithm                 */
 int i,j,kt,move,try,try2,exits,nextsq,testsq,min,fd;
 static int jump[]={-19,-8,12,21,19,8,-12,-21};
 static int board[]={1,1,1,1,1,1,1,1,1,1,  1,1,1,1,1,1,1,1,1,1,
  1,0,0,0,0,0,0,0,0,1,  1,0,0,0,0,0,0,0,0,1,  1,0,0,0,0,0,0,0,0,1,
  1,0,0,0,0,0,0,0,0,1,  1,0,0,0,0,0,0,0,0,1,  1,0,0,0,0,0,0,0,0,1,
  1,0,0,0,0,0,0,0,0,1,  1,0,0,0,0,0,0,0,0,1,  1,1,1,1,1,1,1,1,1,1,
  1,1,1,1,1,1,1,1,1,1,  1,1,1,1,1,1,1,1,1,1,
  1,1,1,1,1,1,1,1,1,1};
 kt=21; board[kt]=move=1; printf(" Knight's Tour for 8x8 ¥n");
 while(1)
 { move+=1; min=999;
 for(try=0; try<=7; try++)
 { fd=kt+jump[try];
   if(board[fd]==0)
   { exits=1;
     for(try2=0; try2<=7; try2++)
     { testsq=fd+jump[try2];
       if(board[testsq]==0) exits+=1;
     }
     if(exits<min){ min=exits; nextsq=fd; }
   }
 }
 if(nextsq != kt) { kt=nextsq; board[kt]=move;}
 else
 { for(i=2; i<=9; i++)
   { for(j=1; j<=8; j++) printf("%2d ",board[10*i+j]); printf("¥n"); }
   break;
 }
}
}
```

リスト 5. C

付録2．Icon 言語の概説と使用法

1．は じ め に

Icon は，文字列と構造体からなる「非数値」の処理を意図した高級プログラム言語であり，その名前は，文字通り，プログラム言語の「聖像」になる願いが込められている。

それは，1978年春に，米国アリゾナ大学計算機科学科の Tim Korb, R. Griswold および D. Hanson の3名により設計・作製された。

Icon は最初に述べた主旨と設計者の名前から判断されるように，SNOBOL 4の後継言語である。周知のように，SNOBOL は，文字列の操作に重点をおいた記号言語であり，1962年頃，R. Griswold が，Bell 電話研究所で Polonsky らと設計・作製したことに始まる。SNOBOL は，文字，文字列の演算，再帰関数，テーブル構造，ユーザ定義のデータ型，再定義可能な演算があるので，コンパイラ作成，記号処理，文献検索，自然言語翻訳，言語学，グラフ処理のようなさまざまな分野に有用な言語ツールとなってきた。しかし，インタプリタであること，制御構造が古めかしく，プログラムの流れがとらえにくいなど多くの問題点があり，ＳＬ５の発表後，１年足らずで，Icon が登場した。

Icon は，Pascal やＣに見られる現代的な制御構造を豊富に持つが，「文」ベースの構文ではなく，「式」ベースの構文に基礎をおく。SNOBOL 4 から式の評価の成功と失敗，記憶領域の管理やガーベジコレクションの自動化を継承し，文字列処理の困難さを克服している。移植容易なことも相俟って，Icon の主要な適用領域をSNOBOL 4 以上に拡大させつつある。

なお，Icon の処理系は当初，FORTRAN IV のプリプロセッサである Ratfor によって作製されたが，現在はＣ言語によって第 5．10版（ＭＳ-ＤＯＳでは5.9版）まで改訂されている。

2．Icon の文法

Icon の基本記号は，名前，フィールド名，キーワードおよびリテラルからなる。名前とフィールド名は，英字又は下線記号から始まって任意個の英字，数字，下線記号の組合せで作られる。大文字と小文字は厳密に区別される。名前をカンマ（,）で区切ったものは，名前の並びという。キーワードは，下記の予約語29個と，キーワード22個からなる。予約語は，すべて小文字で，それ自身の目的にのみ使われ，名前やフィールド名には使うことができない。

break	**external**	**record**
by	**fail**	**repeat**
case	**global**	**return**
create	**if**	**static**
default	**initial**	**suspend**
do	**local**	**then**
dynamic	**next**	**to**
else	**not**	**until**
end	**of**	**while**
every	**procedure**	

キーワード22個は，第一文字に&（アンパサン）がつき，そのあとに小文字の名前が続き，ジェネレータの役割をする。

&ascii	**&input**	**&source**
&clock	**&lcase**	**&subject**
&cset	**&level**	**&time**
&date	**&main**	**&trace**
&dateline	**&null**	**&ucase**
&errout	**&output**	**&version**
&fail	**&pos**	
&host	**&random**	

44種類の組込み関数がある。関数の本体が呼ばれる以前に，関数の引数が左から右に評価される。引数のどれかが「失敗」するならば関数は失敗する。いくつかの関数は結果を複数個生成する。下に Icon の組込み関数をアルファベット順に列挙するがその詳細は，Griswold［4］を参照のこと。

abs(n)	**any(c, s, i, j)**	**bal(** c_1**,** c_2**,** c_3**, s, i, j)**	**center(** s_1**, i,** s_2**)**
close(f)	**copy(x)**	**cset(x)**	**display(i, f)**
exit(i)	**find(** s_1**,** s_2**, i, j)**	**get(a)**	**image(x)**
integer(x)	**left(** s_1**, i,** s_2**)**	**list(i, x)**	**many(c, s, i, j)**
map(s_1**,** s_2**,** s_3**)**	**match(** s_1**,** s_2**, i, j)**	**move(i)**	**numeric(x)**
open(s_1**,** s_2**)**	**pop(a)** **pos(i)**	**pull(a)**	**push(a, x)**
put(a, x)	**read(f)**	**reads(f, i)**	**real(x)**
repl(s, i)	**reverse(s)**	**right (** s_1**, i,** s_2**)**	
sort(a)	**sort(t, i)**	**stop (** x_1**,** x_2**,･･･, xn)**	
string(x)	**system(s)**	**tab(i)**	**table(x)**
trim(s, c)	**type(x)**	**upto(c, s, i, j)**	
write(x_1**,** x_2**, ･･･, xn)**	**writes (** x_1**,** x_2**, ･･･, xn)**		

Icon プログラムは，宣言の集まりである。宣言の代表的な例は，手続き宣言である。これは，

procedure　名前（名前の並び）；

　　ローカル　　初期値節　本体　**end**

の形式をとる。名前の並び，ローカル，初期値節は無い場合もある。ローカルは，

local　　名前の並び

static　　名前の並び

dynamic　名前の並び

のいずれかの形をとる。これらを同一行に書くときのみ，セミコロン（；）で区切る。この部分は BCPL の思想を継承している。**local** は局所変数の宣言，**static** は静的変数の宣言，**dynamic** は動的変数の宣言である。

初期値節は，

initial 式

で表わされ，初期値を式で与える。そのときの左辺は **static** 等で宣言が必要である。

本体は，プログラムの中枢部で，式の集まりである。同一行に複数の式が現れるときのみセミコロンで区切る。

Icon では，Pascal やCで文といっているものも式と呼び，Pascal やC以上に，非常に豊富な構文をとる。式は，ある優先順位と連結性のもとに評価され，成功か失敗かを返す。

以下，式の名称と具体的な形を掲げる。

丸かっこ式	(式)
中かっこ式（複合の式）	{式の連なり}
カギかっこ式（リストの式）	[式の並び]
点つき式（フィールド参照の式）	式．フィールド名
添字式	式[式]
（s[i]はi番目，s[i:j]は，iからj番目，s[i±:j]はiからj，i±1からj，…番目の要素を指す）	
呼出し式	式（式の並び）
前置式	**not** | ! ＊ + − • ／ = ? ＼ ～ @ ∧
限定式	式＼式　（＼は¥で代替可）
転送式	式@式
指数式	式∧式
乗算類の式	＊　／　％　＊＊
加算類の式	+　−　++　−−
結合式	||　|||
比較式	<<=　=　>=　>～= <<　<<=　==　>>=　>>　～==　===　～==== 代替式　式|式
to−by式	式 **to** 式 **by** 式
代入式	:=　<−　:=:　<−>　+:=　−:=　＊:=　／:=　％:=

```
∧:=   <:=   <=:=   =:=   >=:=   >:=
~=:=   ++:=   --:=   **:=   ||:=
<< :=   <<=:=   =:=:=   >>=:=   >>:=
~==:=   ?:=   |||:=   ===:=   ~===:=
&:=   @:=
```

走査式	式?式
連接式	式 & 式
生成式	**create** 式
return 式	**return** 式
	suspend 式
	fail
break 式	**break**
next 式	**next**
case 式	**case** 式 **of** {式:式;
	default 式
	}
if-then-else 式	**if** 式 **then** 式 **else** 式
loop 式	**repeat** 式
	while 式 **do** 式
	until 式 **do** 式
	every 式 **do** 式

最後に Icon に現れる符号はＡＳＣＩＩ文字であり，いくつかを補足説明する。まず，#は引用符（“ ”）の中以外で使われると，注釈行を表わす。ストリング（“ ”）の最後尾に—（下線）が使われると次行との文字の連続を要求する。

例えば，

```
cons:= "bcd—
        efg"
```

は，cons:= “bcdefg” と同じである。バックスラッシュ文字（＼）はＮＥＣでは¥で代替するが機能切換文字となる。

＼bは後退，＼d削除，＼e　エスケープ，＼f　フォームフィード，＼l

ラインフィード，＼n　改行，＼r　キャリジリターン，＼t　水平タブ，＼v　垂直タブ，＼‘　単引用符，＼“　二重引用符，＼＼　バックスラッシュ，＼ddd　八進コード，＼xdd　十六進コード，＼^c　コントロールコードを表わす。

3．Icon プログラムの実行手順（MS-DOS）

（イ）　Icon のソースプログラムをエディタで作り，ソースファイル名の最後が .icn で終るようにする（例えば，ex1．icn）

（ロ）　Icon のソースプログラムを，トランスレータ（icont．exe）により，中間言語に変換する。その形式は，

```
icont  ex1．icn
```

である。結果はex1というファイル名で格納されるから，インタプリタ（iconx．exe）により実行する。その形式は，

```
iconx  ex1
```

である。トランスレートとインタープリットは一つのコマンドによっても可能である。

```
icont  ex1．icn  —x
```

とする。もし，入力データex1．datをファイルから得て，プリンタに出力したいなら，

```
icont  ex1．icn  —x  <ex1．dat  >prn
```

とする。コマンドには次のようないくつかのオプションをおける。

—s	メッセージを抑制
—t	トレース（追跡）を指定
—u	リンク時に未宣言の名前を検出
—o	出力ファイルの別名を—oの後に与える
—x	実行プログラムに渡す
—c	リンクを抑制する

参 考 文 献

[1] Ball, W. W. Rouse and H. S. M. Coxeter, *Mathematical Recreations and Essays*, 12th ed., University of Toronto Press, 1974.

[2] Berge, Claude, *The Theory of Graphs and its Applications*, John Wiley and Sons, 1962, 107-118.

[3] Griswold, Ralph E. and Madge T. Griswold, *The Icon Programming Language*, Prentice-Hall, Inc., 1983.

[4] Gilpin, Michael, "Computer-Generated Knight Tours," *Two-Year College Mathematics Journal*, Vol. 13, No. 4 (Sept. 1982), 252-259.

[5] Held, Michael and Richard M. Karp, "The Traveling-Salesman Problem and Minimum Spanning Trees: Part II," *Mathematical Programming*, 1 (1971), 6-25.

[6] Irwin, Donald P., "The Knight's Tour," *Creative Computing*, Vol. 11, No. 5 (May 1985), 64-69.

[7] Joshi, Bhairay, "The 'Knight's Tour' Problem," *Creative Computing*, Vol. 6, No. 8 (Aug. 1980), 78-83.

[8] Knuth, Donald Erwin, "Estimating the Efficiency of Backtrack Programs," *Mathematics of Computation*, Vol. 29, No. 129 (Jan. 1975), 121-136.

[9] Meissner, Leon P. and Elliot I. Organick, *FORTRAN 77*, Addison-Wesley, 1980, Chap. 10.

[10] Pohl, Ira, "A Method for Finding Hamilton Paths and Knight's Tours," *Communications of the ACM*, Vol. 10, No. 7 (July 1967), 446-449.

[11] Sainte-Marie, M. C. Flye, "Note sur un problème relatif á la marche du cavalier sur l'échiquier," *Bulletin Société Mathématique en France*, 5 (1876), 144-150.

[12] Walker, R. J., "An Enumerative Technique for a Class of Combinatorial Problem," *Proc. Sympo. Appl. Math.*, Vol. 10 (1960), 91-94.

[13] Warnsdorff, H. C., "Des Rösselsprunges einfachste und algemeinste Lösung," *Schmalkaden*, 1823.

[14] Wirth, NiKlaus, *Algorithms + Data Structures = Programs*, Prentice-Hall, Inc., 1976, Chap. 3.

[15] 小谷 善行「マイコンとパズルの世界」 産報出版, 1981, 第7章.

[16] 若林 信夫「問題解決とプログラム言語」 商学討究, 第29巻第4号 (1979年2月).