巡回セールスマン問題への招待II

久保　幹雄

Indeed, I believe that vitrually every important aspect of programming arises somewhere in the context of sorting or searching.

Donald Knuth

(The Art of Computer Programming, Vol.3, 1973)

Indeed, I believe that vitrually every important aspect of combinatorial optimization arises somewhere in the context of the traveling salesman problem.

Mikio Kubo

(Invitation to the TSP II, 1994)

Oops! It's a recursion!

6. 簡単な最適解法を作ろう!

巡回セールスマン問題の最適解は、原理的には全ての解を列挙することによって求めることができる。しかし、n個の都市を巡回する可能な巡回路の数は、$(n-1)!$個(対称なら$(n-1)!/2$個)もあるので、このようなアプローチは、たちまち破綻をきたしてしまう。

あえて名前を出すことは差し控えるが、某有名新聞社の巡回セールスマン問題に関する記事に、「30都市になると、総当たり法は高性能計算機でも約3日かかる」とあったが、この見積もりは、多少楽観的過ぎる。これは、Douglas Hofstadterの言うところの数不感症(number-number: numbは痺れるという意味)の典型例である。封筒の裏(私の場合、書き損じた論文の裏が多い)の計算で、確認してみよう。

巡回路の総数29!を近似するためにJames Stirlingの便利な公式$n! \approx \sqrt{2\pi n}(n/e)^n$を使う。いま、某新聞社の言うところの高性能計算機が、想像を絶するほど高速で、1秒間に10億($=10^{10}$)個の巡回路を探すことができるものとしよう。計算時間の概算をするときに便利なAT & TのTom Duffによる誤差0.5%以内の格言「π秒は1ナノ(10^{-9})世紀」を使うと、$29!/10^{10}$(秒)$\geq \pi \times 10^{19}$(秒)$\approx 10^{10}$(世紀)であることが分かる。この数字は、宇宙の年齢より多少(ほんの5～10倍)大きい程度である。

巡回セールスマン問題の難しさを実感するために、列挙法のプログラムを作成しよう。列挙法のプログラムは、順列を生成するアルゴリズム(これは、プログラムの良い練習問題であり、多くのプログラミングの教科書に載っている)を使って簡単に書くことができる。

巡回路を表す順列を`tour[n]`、最短距離を`length`で表す。この2つの変数はglobal変数であり、`length`には、あらかじめ非常に大きな数を入れておく。C言語の場合は、配列の添字は、通常0からはじめるので、n都市の巡回路を表す点(都市)の列は、`tour[0]`, ⋯, `tour[n-1]`で表わされる。巡回路を構成するには、1つの点を始点および終点として固定しても構わないので、点$n-1$を固定し、$0,\cdots,n-1$の順列を生成することによって、全ての巡回路を生成する。

まず、配列`tour[]`が与えられたとき、その巡回路長を返す関数`cost_evaluate()`を記述する。ここで、`Dis()`は2点間の距離を返す関数である(§3参照)。

```
1 int cost_evaluate()
2 { int i, sum;
3   sum= Dis(n-1, tour[0]);
4   for(i=0; i<n-1; i++)
5       sum += Dis(tour[i], tour[i+1]);
6   return sum;
7 }
```

次に、順列生成のコア部分を記述する。関数`perm(i)`は、`tour`のi番目以降の順列を与える。アルゴリズムは再帰的に構成され、i番目の要素を固定したまま、$i+1$番目以降の順列を生成するか(4行)、i番目の要素を$j(>i)$番目の要素と交換した後で(6行)、$i+1$番目以降の順列を生成する(7行)。順列の添字が$n-1$に達したら、巡回路長を評価し(12行)、現在までに見つ

くぼ　みきお　東京商船大学　流通情報工学
〒135 東京都江東区越中島 2-1-6
e-mail: kubo@ship2.ipc.tosho-u.ac.jp

かった最短距離より小さかったら更新する(13行)。

```
 1 void perm(int i)
 2 { int j, tmp, cost;
 3   if (i < n-2){
 4     perm(i+1);
 5     for(j = i+1; j < n-1; j++){
 6       tmp = tour[i]; tour[i] = tour[j]; tour[j] = tmp;
 7       perm(i+1);
 8       tmp = tour[i]; tour[i] = tour[j]; tour[j] = tmp;
 9     }
10   }
11   else{
12     cost = cost_evaluate();
13     if (cost < length) length=cost;
14   }
15 }
```

最後に、呼び出しルーチンを作る。まず、初期順列として$0,1,\cdots,n-1$を`tour[]`に入れておき(3,4行)、0番目以降の順列を生成する(5行)。再帰が終了した時点での`length`の値が最適巡回路長である。そのときの巡回路も、最良解を更新したときに保存しておくことによって容易に得られるが、ここでは省略している。

```
1 void enumerate()
2 { int i;
3   for(i=0; i<n; i++)
4         tour[i]=i;
5   perm(0);
6   printf( "%d", length );
7 }
```

ここで注意しておくが、上述の某新聞社の記事の真偽のほどを確かめるために、$n=30$の巡回セールスマン問題を作成し、上のプログラムを実行してはならない。結果は悲劇的であり、数十年待っても終了しないプログラムのために、貴重な計算機資源を失うことになる。くどいようだが、念を押しておく。

注意: nが13以上の巡回セールスマン問題に対して上のプログラムを使わないこと! 爆発します。

それでは、$n \geq 13$の巡回セールスマン問題の最適解を得ることは絶望的か、と言うとそうでもない。$n=16$程度までなら、動的計画法(Dynamic Programming: DP)を用いた美しいアルゴリズムが効率的に最適解を算出する。

まず、アルゴリズムを簡単に説明しよう。ある始点sから出発し、点の部分集合$S(\subseteq V)$を全て経由し、点$i(\in S)$にいたる最短路長を$f(i,S)$と書くことにする。簡単に分かるように、境界条件$f(i,\{i\})=d_{si}$および次の再帰方程式によって計算された$f(s,V)$が最適巡回路長である。

$$f(j,S\cup\{j\})=\min_{i\in S}[f(i,S)+d_{ij}]$$

このアルゴリズムの計算量は$O(n^2 2^n)$であり、必要な記憶容量は$O(n2^n)$である。このアルゴリズムの計算量も、やはり指数オーダーであるが、全列挙のアルゴリズムに比べて大幅な改善となっている。

上のアルゴリズムをプログラムとして実現させるためには、集合Sをどのように表現するかが鍵となる。列挙法のときと同じように、点を0から$n-1$で表し、始点sを$n-1$番目の点としよう。集合を整数で表し、点が集合に含まれているか否かを2進表現の各ビットで表すことにする。すると、Vの部分集合Sは、$0(S=\emptyset)$から$2^n-1(S=V)$までの整数で表すことができる。

C言語では、2^nは1をn回左にビットシフト(左ビットシフト演算子は`<<`)することによって得られるので、定数`SMAX`を`1<<n`としておく。はじめに、$f(i,S)$(プログラム内では`f[i][S]`)を大きな値に初期設定しておき(4-6行)、境界条件を入れる(7-8行)。ここで、`1<< i`は、1を左にi回ビットシフトした数であり、集合$\{i\}$を表す。

次に、再帰方程式を計算する(9-19行)。ここで、点iが集合Sに含まれているか否かは、`1<<i`とSのビットごとの論理積(AND: &)をとることによって判定できる(ちなみに、!はNOTを表す演算子)。また、$S\cup\{j\}$は、`S`と`1<<j`のビットごとの論理和(OR: |)をとることによって表すことができる。

```
 1 void DP()
 2 { int i, j, S, tmp;
 3   int  f[n][SMAX];
 4   for(S=1; S<SMAX; S++)
 5       for(i=0; i<n; i++)
 6           f[i][S] = 9999;
 7   for(i=0; i<n-1; i++)
 8     f[i][1<<i]  = Dis (n-1, i);
 9   for( S=1; S <SMAX; S++){
10     for(i=0; i<n; i++){
11       if( !( (1<<i) & S) ) continue;
12       for(j=0; j<n; j++){
13         if( (1<<j) & S ) continue;
14         tmp   =  f[i][S] + Dis(i, j);
15         if ( tmp < f[j][ (S | (1<<j)) ] )
16                 f[j][ (S | (1<<j)) ] = tmp;
17       }
```

```
18    }
19  }
20  printf( "%d", f[n-1][SMAX-1]);
21 }
```

上のプログラムは、最短巡回路長を出力するが、対応する巡回路は、15,16行目で最小値を更新したときのiの値を記憶しておくことによって再現できる。

7. 近似解法雑感

最近では、多面体論の優雅さと線形計画法のパワーを利用した分枝カット法と呼ばれる最適化手法の洗練化により、数千都市のユークリッド巡回セールスマン問題を解くことが可能になった。原稿執筆中における世界記録は、William Cook, David Applegate, Vasek Chvátal, Robert Bixby による4461都市問題であり現在7397都市問題に挑戦中である。また、ランダムに作られた非対称巡回セールスマン問題は、50万都市の問題さえ解かれている。それではもう近似解法は必要ないのか? 幸いなことに、巡回セールスマン問題の応用では、巨大な問題を高速に解く必要がある場合が多いのである(§2参照)。最適解から数%の近似解をパソコンで数分で得られるのに、最適解を得るためにスーパーコンピュータで数時間かける人は、世の中にはそうざらにはいないだろう。したがって、今後も近似解法の重要性は、損なわれるどころか、ますます増大すると考えられる。

巡回セールスマン問題に対する近似解法は大きく分けて、構築法とローカルサーチ(改善法)に分けられる。構築法は何もないところから解を作っていく方法であり、ローカルサーチは何らかの方法で得られた解をもとに、さらに良い解を探していく方法である。ここでは幾つかの構築法を紹介し、ローカルサーチについては次章以降で詳しく述べることにする。なお、以下では対称巡回セールスマン問題に絞ってアルゴリズムを考える。

最も簡単にコード化できる構築法は、ランダムな順列を近似巡回路とする方法である。これは解法とは呼べないものかも知れないが、強力なローカルサーチの初期解として用いるのなら最も良い構築法の1つである。

効率的な近似解法を発明するための基本は、人間の最も自然な摂理(貪欲性)に基づいたアイディアを使うことである。貪欲性に基づいた、すなわち先のことなど考えず目先の利益だけを追求する解法を2つあげる。

Nearest Neighbor 法: 適当な点から出発し、まだ訪問していない点で現在地点から最も近い点へ移動する。全ての点を訪問したら出発地点へ戻る。

Greedy 法: 最小木問題に対する有名なKruskalのアルゴリズムのように、枝を短い順に加えていく。このとき、全ての点を通過する前に閉路ができたり、点の次数(接続している枝の本数)が2を超えてしまうときは枝を加えない。

次に、点を順次挿入していくタイプの構築法をあげる。このアイディアも、何度も手で巡回セールスマン問題を解いた人ならば誰でも自然に思いつくものであり、またコード化もたやすい。

Farthest Insertion 法: まず、適当な点を選び、その点のみを経由する退化した巡回路(セルフループ)を作る。次に、まだ巡回路に入っていない点の中で現在の巡回路から最も遠い点を選び、その点を巡回路の連続した2つの点に間に距離の増加が最も小さくなるように挿入する。この操作を巡回路を得るまで繰り返す。

上にあげたいくつかの解法は、自然ではあるがあまり芸がない。解法に芸を入れるには、得られた解に何らかの保証を施すことである。以下に最悪の場合の保証の点で優れた2つの解法を示す。

Double 最小木法: 最小木の枝を2重にすることによりEuler閉路を作る。Euler閉路をたどるとき、1度通過した点は通過せず近道をとることによって巡回路を得る。

Christofides' Algorithm (§4参照): 最小木と最小木における奇数の次数を持つ点に対する最小マッチングを合わせることにより、Euler閉路を作る。後はDouble最小木法と同じ操作によって巡回路を得る。

上の2つの解法は、三角不等式を満たす巡回セールスマン問題に対して最悪の場合の保証が定数で抑えられるという特徴を持つ。保証の導出はいとも簡単である。最小木は最短巡回路長以下であり、また最小マッチングは最短巡回路長の半分以下である。また、近道を通るとき、三角不等式の条件から距離が増えることはない。よって、最悪の場合でも近似解は最適解の2倍以下(Double 最小木)または1.5倍以下(Christofides' Algorithm)の長さを持つことが分かる。

Christofides' Algorithmは、最悪の場合の保証の観点から見ると現在では最良の近似解法であるが、平均的な観点から見るとあまり推奨できない。計算時間

(最小マッチングを求める部分に$O(n^3)$かかる)のわりには得られる解があまり良くないからである。

次に、点(都市)が2次元ユークリッド平面上に分布している問題に対する近似解法を考える。

Karpの分割算法(§4参照): 平面を各領域に含まれる点の数がθ個以下になるように分割し、各領域に対する最適巡回路を(例えば)動的計画法を用いて解く。得られた領域ごとの巡回路をつなぐことによってEuler閉路を得る。後はDouble最小木法と同じ操作によって巡回路を得る。

この近似解法を使うと、パラメータθを適当に決めることによって$n \to \infty$のときに最短巡回路長に収束する近似値をnに対する多項式時間で得ることができる(詳細については巡回セールスマン問題I, 文献[4,§6]を参照)。

空間充填曲線法(Spacefilling Curve Method): 2次元の領域を満たす曲線†上に、全ての点を写像し、曲線上での順番にしたがって点を訪問する。

この解法はコード化も容易であり、驚くほど速い。詳細については[1]を参照: この論文にはBASICで書かれたわずか31行(メインになる部分は6行)の簡単なプログラムがついている。悪い解でも良いから速く解が欲しいときには推奨できる。

さて、上の解法の評価を行うには系統的な数値実験が必要である。ここでは、紙面の都合上、実験結果の全ては載せることができないので、AT & TのDavid Johnson [2]によって行われた実験的解析の一部を表1に示す(2-opt,3-opt,Lin-Kernighanについては§§8,9参照)。得られた解は、構築法の中ではChristofides' Algorithmが最も良く、続いてFarthest Insertion法、Greedy法となっている。しかし、前述のようにChristofides' Algorithmの計算時間は膨大であり、実用的とは言えない。最も実用的な解法は次章以降で述べるローカルサーチ(2-opt,3-opt,Lin-Kernighan)を用いた解法である。

したがって、構築法を評価するときには解の良さだけで判断するのでなく、ローカルサーチとの相性も重要な問題になる。Nearest Neighbor法やGreedy法は、解法の最終段階では極端に長い枝を選択してしまうので解の値は悪いが、部分的には最適解と同じ枝を多く含む傾向がある。一方、Farthest Insertion法は、この解法だけから得られる解は比較的良い値を示すが、部分的に見ると最適解からほど遠く、ローカルサーチにかかりにくい構造を持っていると考えられる。ちなみに、私のささやかな経験から推奨できる解法は、Nearest Neighbor法またはGreedy法による解(またはランダムな解)を初期解に用いて、ローカルサーチを行う方法である。

†イタリアの代表的数学者Giuseppe Peano (1858–1932)が1890年に初めて導出した空間を埋めつくす曲線. 従来の曲線の定義を超えるものであったので、怪物曲線とも呼ばれていた. 当初, Peanoは数学的記号のみを用いた存在証明しか与えなかったが, Hilbert (§4参照)によって翌年, 図が与えられた. 現在では, この曲線はBenoit Mandelbrotによって導入されたFractal曲線の1つとして, 怪物というよりも観賞用の絵画になっている.

表1: 近似解法の比較: 100問のランダムに作られたユークリッド巡回セールスマン問題の平均, $n = 1000$, Karpの分割算法のパラメータθは10, 解の良さは(近似値 − Held and Karpの解法による下界) / 下界 ×100 (%), 2-opt,3-optの初期解はNearest Neighbor, Lin-Kernighanの初期解はランダム.

解法名	計算時間(秒)	解の良さ(%)
Nearest Neighbor	9.8	26.5
Greedy	24.4	16.9
Farthest Insertion	56.0	12.5
Double 最小木	32.1	39.0
Christofides	4053.1	9.9
Karpの分割	42.9	56.1
空間充填曲線	2.7	31.2
2-opt	5.0	6.4
3-opt	6.6	3.5
Lin-Kernighan	16.7	2.1

8. 実用的な近似解法を作ろう!

ここでは、巡回セールスマン問題に対するローカルサーチ(改善法)を実際に作ってみる。

ローカルサーチの基本構造は、極めて簡単であり、暗い夜道を懐中電灯をたよりに山登りをする人に例えられる。それゆえ、ローカルサーチは、しばしば丘登り法(Hill Climbing Method)と呼ばれる。

まず、夜道は暗くて山の頂上は見えないので、今、自分のいる場所の回りを懐中電灯で照らしてみる。現在地点よりも高い方向が見つかったら、その方向へ移動する。照らした範囲内に、高い地点が見つからなかったら、あきらめて寝袋を出して寝る。そのときの標高が登山者の得点になる。

簡単に分かるように、この方法では、いつも山の頂上で眠れるとは限らない。我々の考えている山は、幾つもの小高い丘がある(専門的には、多峰性を持つと言う)。夜道では、小高い丘と頂上の区別がつかない

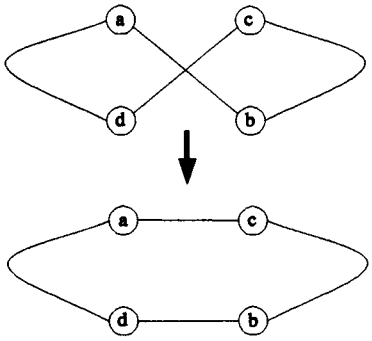

図 6: 2-opt

のである。ちなみに、懐中電灯の照らせる範囲は、専門的には近傍(neighborhood)と呼ばれ、小高い丘は局所最適解、山の頂上は大局的最適解と呼ばれる。

いま、現在地点をx、その標高を$c(x)$、近傍を$N(x)$と書く。次の関数$improve(x)$を用いることによって、ローカルサーチを疑似コードで記述することができる。

$$improve(x) = \begin{cases} \text{any } x' \in N(x) & \text{with } c(x') > c(x) \\ & \text{if such an } x' \text{ exists} \\ \emptyset & \text{otherwise} \end{cases}$$

```
procedure local search
 1 x := some initial feasible solution
 2 while improve(x) ≠ ∅ do
 3     x := improve(x)
 4 return x
```

巡回セールスマン問題に対する最も簡単なローカルサーチが2-optである。2-opt法とは、次のような方法である。図6の上図のような巡回路において、枝abおよびcdを除き、かわりにacとbd加えることを考える。このとき、巡回路の長さが減少していれば、新しいものに交換する。この操作を長さを減少させる交換が見つからなくなるまで続ける。

上のアルゴリズムを単純に実現するだけでは、大規模問題に対する効率的な解法は構築できない。実現の仕方によって、アルゴリズムの効率は大きく変わってしまうのである。

原データにおいては、各点間に枝があるものとしているが、巡回路に含まれる枝は、それほど長いものは含まれないものと考えられる。そこで、各点ごとに、近い順に数個(10程度)の点の間にだけ枝があるものとし、あとの枝はすべて消去してしまう。

どの点とどの点が隣接しているかという情報は、計算の途中で変化しないので、Foward-Starと呼ばれるグラフの表現法をとることにしよう。(C言語では、ポインタを使っても表現できるが、他の言語(FORTRAN,BASIC等)にも容易に変更できるように、ここではポインタを用いないFoward-Starを採用した。)

まず、`adja[]`と`head[]`の2つの配列を用意する。点iに隣接する点は配列`adja[]`の`head[i]`から`head[i+1]-1`番目に、点iに近い順に保存する。これは、ユークリッド巡回セールスマン問題に対しては、計算幾何学の手法[‡]を用いることによって$O(n \log n)$時間で実現可能であるが、ここでは長くなるので省略する。

巡回路を表すデータ構造として、前と同じように1次元配列`tour[]`を使う。`tour[i]`にはi番目に通過した点の番号を保存する。2-opt法は以下の関数を必要とする(これらの記述は、読者に任せる)。

`Next()`: 点の番号を入力すると、現在の`tour[]`上での次の点を返す。

`Flip(a,b,c,d)`: 4つの点a,b,c,dで`b=Next(a),d=Next(c)`を満たすものを入力すると、現在の巡回路における枝ab,cdをac,bdに置き換える。このとき、2点b,c間の全ての点を逆順にするか、2点d,a間の全ての点を逆順にする必要がある。

いよいよ、2-opt法の記述に入ろう。入力として、任意の初期巡回路`tour[]`と、その長さ`length`を与える。アルゴリズムは、極めて簡単であり、2つの基本ループの中で解の改良をチェックし、改良されているなら、`Flip()`を呼ぶことによって、解を変えていくだけである。

```
 1 void two_opt()
 2 { int i, a, b, c, d, j, tmp;
 3   two_opt_start:
 4   for(i=0; i<n; i++){
 5     a = tour[i];
 6     b = Next(a);
 7     for (j = head[a]; j < head[a+1]; j++){
 8       c    = adja[j];
 9       if (b==c) break;
10       d    = Next(c);
11       tmp = Dis(a,b)+ Dis(c,d)−Dis(a,c) −Dis(b,d);
12       if (tmp>0){
```

[‡]Delauney三角網(Voronoi図の双対)上で広がり優先のグラフ探索を行うか、$K-d$木(2分探索木のK次元版)を用いる方法が提案されている.

```
13            length -= tmp;
14            Flip (a, b, c, d);
15            goto two_opt_start;
16        }
17    }
18  }
19 }
```

このアルゴリズムが、驚くほど速い理由は、9行目にある。7から17行のループで、点aに隣接している点を近い順に探索しているが、この行があるために、枝acが枝abより長いときには探索を終了することになる。これは、次の簡単な事実に基づいている。

> 枝abおよびcdを除き、かわりにacとbd加えるとき、$ab < ac$または$cd < bd$の少なくとも一方が成立しないと、全体として改良できない。

さて、2-opt法の速さと解の良さについて考えよう。まず、最悪値解析と呼ばれる理論的な接近を試みる。2-opt法は、最悪の場合では指数オーダーの改良回数を要する場合があることが示されており、解の良さに関しては、最適解との比が、いくらでも悪くなるような例を作ることができる。これらの理論的な結果は、2-opt法を実際に動かしたことがある人からみると、全く見当違いのように思えるだろう。実際には、2-opt法は驚くほど速く、また算出された解もそれほど悪くはない。このことは、上のプログラムを使った系統的な実験的解析によって、いとも簡単に明らかになる。

実験的解析は、理論的な解析に比べて弱いものであるという考え方があるが、私はこの意見に反対である。周到に計画された実験的解析は、美しくかつ力強いものであり、現実に我々が直面している問題を解決するための有力な道具である。実験的解析に対する不評は、この方法論の基準が、まだ確立していないために発生していると思われる。

実験的解析の実施に際して、特に注意しておきたいことは、実験に用いたプログラムの詳細(データ構造、実現のための工夫、できればコード自身)、実験に用いたデータの詳細(できればベンチマーク問題を使う)、パラメータの設定方法、実験環境などを明確にすることである。これらのことは、他の分野における実験では当然のことであるが、ORや計算機科学の分野では、まだ十分に浸透していない。

実験科学におけるもう1つの基本原則として、実験の公正さがあげられる。しばしば公正さに欠ける実験結果を見受けるが、これは、否定的な結果は論文として受理されにくいからであると考えられる。他の科学の諸分野においては、否定的な結果も重要な成果として受け入れているのに対して、ORにおける論文の基準は、肯定的な成果に多少かたよっていると思われる。

今後、ORが実務にさらに浸透するためには、実験的解析も考慮したバランスのとれた研究体制が重要であると考えられる。上の意見が私の独断でないことを示すための幾つかの引用を今回の締めにしよう。

> *Anyone who thinks that empirical science is nontheoretical should take a look at quantum electrodynamics.*
>
> *Negative results are as important as positive results in other emperical sciences. But the OR communitity do not judge the publishability of results.*
>
> **J. N. Hooker**
> "Needed: An Empirical Science of Algorithms"
> (dimacs.rutgers.edu /pub/challenge/contributed)

> *So how should one analyze algorithms, by experiments or by theory? My suggestion is simple: by whichever is more effective for the problem at hand.*
>
> *Don't experimet when you should think; don't think when you should experiments!*
>
> **John Bentley**
> "Experiments on Geometric Traveling Salesman Heuristcs" (AT & T Technical Report, No. 151)

参考文献

[1] J. J. Bartholdi III and L. K. Platzman. An O($n \log n$) planar traveling salesman heuristic based on spacefilling curves. *Operations Research Letters*, 1:121–125, 1982.

[2] D. S. Johnson. Local optimization and the traveling salesman problem. In *Proc. 17-th Colloquium on Automata, Languages, Programming*, pages 446–461. Springer-Verlag, 1990.