『日本をスピード横断します!!』に関する考察

ー2千7百正年が一瞬になるまでー

1年D組　こほろぎ あさぴ

もくじ

1. はじめに

日本の県庁所在地を一回ずつ通る最短ルートについて、鈴木聖央氏[*1]は4380kmだと結論づけている。鈴木氏は試行錯誤によってこのルートを導いており、このルートが最短であるという証明は行っていない。

鉄道の世界では「最長片道きっぷ」というものがあり、知られている限り昭和36年の東大旅行研究会[*2]以降、数多くの人が実践している。しかし多くの人が経験的に最長ルートを求めていた。しかし、求めたルートが最長であるという証明は行っていなかったため、求めたルートが最長ではなかったことが後から分かった事例[*3]もある。求めたルートが最長であるという証明を行ったのは平成12年の葛西隆也氏[*4]が最初であり、以降、最長片道きっぷのルート算出は葛西氏の手法が使われている。これによる再計算の結果、昭和36年の東大旅行研究会のルートには誤りが見つかっている。

このように最長(最短)であると結論づけるためには証明が必要である。小学生の自由研究にそれを求めることは酷かもしれないが、証明を試みて無理でしたとか、証明が誤っていましたというのならともかく、数学(算数)のレポートとして、証明を試みていないのはいかがかと思う。

そこで今回は、改めて日本の県庁所在地を一回ずつ通る最短ルートを求め、最短ルートであることを証明することを試みる。

2. 準備

まずは各都道府県庁の緯度・経度の表を作った。これについては、mapfan webで各都道府県庁を検索し、その緯度・経度を利用した。

地球を半径rの球と仮定すると、地点A(経度x_1, 緯度y_1)、地点B(経度x_2, 緯度y_2)の距離dは、

$$d=r\arccos(\sin y_1\sin y_2+\cos y_1\cos y_2\cos(x_1-x_2))$$
$$=r\arccos(\sin y_1\sin y_2+\cos y_1\cos y_2\cos x_1\cos x_2+\cos y_1\cos y_2\sin x_1\sin x_2)$$

で求められる[*5]。地球の半径を6378.1366kmとして、それぞれの都道府県庁間の距離を計算し、テーブル化した。この数値を用いて鈴木氏の経路の距離を計算すると、4415.844355kmとなった。

3. 起終点の片方が那覇であることの証明

暗黙の了解で、札幌から那覇のルートになるとしているが、起終点の片方が那覇であることについては証明することができる。

まず終点(起点でもよいがとりあえず終点とする)が那覇でないと仮定する。すると那覇から1番近い県庁所在地は鹿児島、2番目に近い県庁所在地は宮崎なので、最低でも宮崎～那覇～鹿児島のルートを取らねばならなくなる。宮崎～那覇～鹿児島の距離は、731.334984km+657.9687

---

*1 算数の自由研究第2回作品コンクール小学校高学年の部塩野直道賞『日本をスピード横断します!!』白百合学園小学校4年・鈴木聖央

*2 http://www.desktoptetsu.com/saichohensen.htm#keifu

*3 種村直樹氏『さよなら国鉄　最長片道きっぷの旅』(実業之日本社)など

*4 当時東京大学大学院生　https://www.swa785.net/lop/index.html

*5 https://orsj.org/wp-content/corsj/or60-12/or60_12_701.pdf

95kmの札幌を起点とすると、札幌・鹿児島・那覇を除く44都府県の都府県庁所在地には2本の枝(出入りのルート)が接続されていることになる。最短ルートの都府県庁所在地に繋がる枝の長さをl_1、l_2、都府県庁所在地のすべての辺の中で最少の2辺をm_1、m_2とすると、$l_1+l_2 \geqq m_1+m_2$となるのは明白である。つまり、宮崎～那覇～鹿児島を通るルートは少なくも、
(札幌～青森の距離)+(札幌・青森・宮崎・鹿児島・那覇を除く44都府県の最少の2辺の総和)/2+(青森の最少の辺の長さ)/2+(宮崎の最少の辺の長さ)/2+(宮崎～那覇～鹿児島の距離)
よりは長くなる。これを計算すると4566.542267kmとなるが、すでに鈴木氏が4415.844355kmのルートを見つけているので、終点が那覇でないルートは解になり得ないことが分かる。ここから少なくとも起終点の片方は那覇であることが分かる。

4．全探索による計算(…できません)

これで起終点の片方が那覇であることが確定したので、残り46都道府県を並べて距離を比較して行けばいいことになった。

単純に以下のようなプログラムを書いて*6実行すればいいように感じるが…

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

#define MAX_PREF 47
#define N 46

char pref_name[47][9]={"札幌","青森","盛岡","仙台","秋田","山形",
    "福島","水戸","宇都宮","前橋","さいたま","千葉","東京","横浜",
    "新潟","富山","金沢","福井","甲府","長野","岐阜","静岡","名古屋",
    "津","大津","京都","大阪","神戸","奈良","和歌山","鳥取","松江",
    "岡山","広島","山口","徳島","高松","松山","高知","福岡","佐賀",
    "長崎", "熊本","大分","宮崎","鹿児島","那覇"};

double pref_dis[47][47]={
    (中略)
    };

int pref[MAX_PREF];
double min_d=1.79769e+307;
```

*6 順列生成部分については「C言語による最新アルゴリズム辞典」奥村晴彦著を参照した。

```
double cal_journey(int *p){
    int q;
    double sum=0.0;


    for(q=0;q<MAX_PREF-1;q++){
        sum=sum+pref_dis[p[q]][p[q+1]];
    }

    return sum;
}

void show_route(int *p){
    int q;
    double d;

    d=cal_journey(p);
    if(d<min_d){
        min_d=d;
        for(q=0;q<MAX_PREF;q++){
            printf("%s ",pref_name[p[q]]);
        }
        printf("¥n%fkm¥n",d);
    }
}

main(){
    int i,k,t,c[MAX_PREF];
    int count;


    count=0;
    for(i=0;i<MAX_PREF;i++){
        pref[i]=i;
    }
    for(i=1;i<=N;i++){
        c[i]=i;
    }
    k=1;
    while(k<N){
```

```
        if(k&1){
            i=c[k];
        } else {
            i=0;
        }
        t=pref[k];
        pref[k]=pref[i];
        pref[i]=t;
        show_route(pref);
        k=1;
        while(c[k]==0){
            c[k]=k;
            k++;
        }
        c[k]--;
    }
}
```

このプログラムは実行してはいけない。終了には恐ろしい時間がかかるのである。

ためしに手元のパソコンで N を 12 として実行したところ、75秒かかった。このプログラムは見てのとおり、N個の都道府県を並べてその距離を計算し、比較しているだけである。N個の都道府県を並べる並べ方はN!通りあるので、このプログラムの実行時間はN!にほぼ比例する。よって46都道府県で計算したときの推定所要時間は、8.6×10^{59}秒、年に直すと2.7×10^{43}年…2千7百正年!!。宇宙の歴史が139億年と言われているので、その20溝倍ということになる。うん、兆より上の命数を使うことになるとは思わなかった。

太陽の余命(>地球の余命)は50億年と言われているので、このプログラムは地球が滅びるより前に終了することはない。

実はこの問題は「最短ハミルトン路問題」と言って、NP困難(多項式時間で計算する手法が見つかっていない問題)ということになっている。こんな問題を自由研究に選ぶ小学生、恐るべし。

5. 緯度経度から距離計算方法の修正

この辺りで、計算に使用している都市間の距離が長すぎることに気付いた。例えば鹿児島から那覇の距離は、国土地理院[*7]によると655.7kmだが、私のプログラムでは657.968795kmである。原因として考えられるのは、地球を半径を6378.1366kmの球と仮定したことではと考えた。改めて調べるとこの6378.1366kmという数字は地球の赤道半径である。地球は赤道方向に膨らんだ回転楕円体なので、地球を半径を6378.1366kmの球と仮定すると南北方向の距離は長めに計算されてしまうことになる。

*7 https://www.gsi.go.jp/KOKUJYOHO/kenchokan.html

正確な距離を計算をするためにはどうしたらよいか調べたところ、国土地理院のページ[*8]が見つかった。…が、ぱっと見理解できない。ということで、330k INFOさんのページ[*9]の精度評価を参考に、Andoyer-Lambert[*10]という計算方法を使用することとした。
(余談だがこの330k INFOさん、自転車も趣味としており、大阪から東京を23時間37分で走破している。数々のアルゴリズムに関する記事も興味深い。)

```
double caldis(double x1,double y1,double x2,double y2){
    double a=6378.137;
    double b=6378.137-6378.137/298.257222101;
    double p1,p2;
    double x;
    double r;
    double f;

    p1=atan(b/a*tan(y1*PI/180.0));
    p2=atan(b/a*tan(y2*PI/180.0));


x=acos(cos(p1)*cos(x1*PI/180.0)*cos(p2)*cos(x2*PI/180.0)+cos(p1)*si
n(x1*PI/180.0)*cos(p2)*sin(x2*PI/180.0)+sin(p1)*sin(p2));

    f=(a-b)/a;

r=f/8.0*((sin(x)-x)*((sin(p1)+sin(p2))*(sin(p1)+sin(p2)))/(cos(x/2.
0)*cos(x/2.0))-(sin(x)+x)*((sin(p1)-sin(p2))*(sin(p1)-sin(p2)))/(si
n(x/2.0)*sin(x/2.0)));

    return a*(x+r);
}
```

こうして計算し直したところ、鹿児島から那覇の距離は655.715096km、鈴木氏の経路の距離は4409.233271kmとなった。

---

*8 https://vldb.gsi.go.jp/sokuchi/surveycalc/surveycalc/algorithm/bl2st/bl2st.htm
*9 https://www.330k.info/essay/geodesic_distance_formula_comparison_2/
*10 https://web.archive.org/web/20160411010136/http://www2.nc-toyama.ac.jp/~mkawai/lecture/sailing/geodetic/geosail.html

6．動的計画法による計算

色々調べて見ると、動的計画法というので計算量が減らせることが分かった。

私なりに動的計画法を解釈すると…

集合S、頂点vを終点とする最短ハミルトン路は、集合Sからvを除いた集合でiを終点とする最短ハミルトン路とiからvの距離との合計のうち、最短距離のものである。

なんやら分かりませんね。ちょっと具体的に言うと、

那覇スタートで九州から山口へ行く最短経路は、

（
那覇スタートで九州内で鹿児島を終点とする最短経路と鹿児島から山口の距離の合計
那覇スタートで九州内で宮崎を終点とする最短経路と宮崎から山口の距離の合計
那覇スタートで九州内で大分を終点とする最短経路と大分から山口の距離の合計
那覇スタートで九州内で熊本を終点とする最短経路と熊本から山口の距離の合計
那覇スタートで九州内で長崎を終点とする最短経路と長崎から山口の距離の合計
那覇スタートで九州内で佐賀を終点とする最短経路と佐賀から山口の距離の合計
那覇スタートで九州内で福岡を終点とする最短経路と福岡から山口の距離の合計
）

の最短距離のものですよということ。

まずは那覇と鹿児島だけの集合の最短距離(明らかに那覇と鹿児島の距離そのもの)から順に計算して行って積み重ねて行けば、任意の都道府県の集合の最短距離を計算できるということになる。ということで、那覇から札幌まで、全都道府県庁所在地を通る最短距離を求めるプログラムは次のようになる。(ただし46都道府県だとlong形式のビット数を越えるためこのプログラムは実行できない。)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

#define MAX_PREF 47
#define MIN_PREF 1

char pref_name[47][9]={(中略)};

double pref_dis[47][47]={
    (中略)
    };

double dp[((long)1)<<(MAX_PREF-MIN_PREF)][MAX_PREF-MIN_PREF];

double cal_journey(int *p){
```

```
    int q;
    double sum=0;

    for(q=0;q<MAX_PREF-MIN_PREF;q++){
        sum=sum+pref_dis[p[q]-MIN_PREF+1][p[q-MIN_PREF+2]];
    }

    return sum;
}

void show_route(long i){
    long q2,q3;
    long q1=(((long)1)<<(MAX_PREF-MIN_PREF))-1;
    double min_d;
    long min;

    q2=i;
    printf("%s ",pref_name[MAX_PREF-q2-2]);
    while(1){
        min=MAX_PREF;
        min_d=1.79769e+307;
        for(q3=0;q3<(MAX_PREF-MIN_PREF);q3++){
            if((q1&(((long)1)<<q3))&&(q2!=q3)){
                if(abs(dp[q1][q2]-dp[q1-(((long)1)<<q2)][q3]-pref_dis
[MAX_PREF-q2-2][MAX_PREF-q3-2])<min_d){

min_d=abs(dp[q1][q2]-dp[q1-(((long)1)<<q2)][q3]-pref_dis[MAX_PREF-q
2-2][MAX_PREF-q3-2]);
                    min=q3;
                }
            }
        }
        q1=q1-(1<<q2);
        if(q1==0){
            printf("%s ",pref_name[MAX_PREF-1]);
            break;
        }
        q2=min;
        printf("%s ",pref_name[MAX_PREF-q2-2]);
    }
```

```
}

main(){
    unsigned long q1,q2,q3;
    int pref[MAX_PREF];
    double dis,min;
    long t;

    printf("%ld¥n",(((long)1)<<(MAX_PREF-MIN_PREF)));

    for(q1=1;q1<(((long)1)<<(MAX_PREF-MIN_PREF));q1++){
        for(q2=0;q2<(MAX_PREF-MIN_PREF);q2++){
            if(q1&(1<<q2)){
                if(q1==(1<<q2)){
                    dp[q1][q2]=pref_dis[MAX_PREF-q2-2][MAX_PREF-1];
                } else {
                    dp[q1][q2]=1.79769e+307;
                    for(q3=0;q3<(MAX_PREF-MIN_PREF);q3++){
                        if(q1&(((long)1)<<q3)){

dis=dp[q1-(((long)1)<<q2)][q3]+pref_dis[MAX_PREF-q2-2][MAX_PREF-q3-
2];
                            if(dis<dp[q1][q2]){
                                dp[q1][q2]=dis;
                            }
                        }
                    }
                }
            } else {
                dp[q1][q2]=1.79769e+307;
            }
        }
    }
    printf("%fkm¥n",dp[(((long)1)<<(MAX_PREF-MIN_PREF))-1][0]);
    show_route(0);
}
```

試しに名古屋を起点に近畿・中国・四国・九州の24府県庁を通る最短経路を計算したところ、名古屋-津-大津-京都-奈良-大阪-神戸-和歌山-徳島-高松-岡山-鳥取-松江-高知-松山-広島-山口-大分-福岡-佐賀-長崎-熊本-宮崎-鹿児島-那覇

が最短ルートであることが分かった。またこの結果から、鈴木氏のルートより短いルートが存在する(つまり鈴木氏のルートは最短ではない)ことが明らかになった。日本海側の松江から太平洋側の高知に一気に行くルートが最短ルートであることが興味深い。試行錯誤ではこういうルートはまず思いつかない。試行錯誤の限界である。なお、計算時間は80秒であった。前述の全探索のプログラムで24府県分を計算すると約31億年かかる計算になるので、大幅にスピードアップできたことになる。本来は地球上に光合成生物(シアノバクテリア)が誕生した頃から計算し続けてようやく計算完了する内容を80秒で計算出来たのだからすごい。

ちなみにこの方法の46都道府県で計算したときの推定所要時間は39年なので、かなり現実的になる。ただし、メモリが$2^{46}\times4$バイト…281TB必要になる。こちらもかなり現実的な範囲内になった。プログラムを読める人は分かるだろうが、確保したメモリの半分は無駄にしているので、プログラムを上手く作れば141TBのメモリで済む。

7・仮定を入れた上での最短経路

と言っても39年も待っていられないので、以下の仮定を入れる。

○ 片方の起終点は那覇で確定だが、もう一方の起終点は札幌である。

○ 経路は北海道・東北・関東・中部ブロックと近畿・中国・四国・九州ブロックに分けられ、ブロック間を行き来するルートは取らない。

○ 両ブロックのルートは名古屋-津で接続する。

つまり、札幌-(東北・関東・中部の最短ルート)-名古屋-津-(近畿・中国・四国・九州の最短ルート)-那覇が最短ルートになると仮定するのである。両ブロックとも、都道府県の数は24以下なので、現実的な計算時間(1分少々)で計算できるはずである。

こうして計算したルートは以下のようになった。

札幌-青森-秋田-盛岡-仙台-山形-福島-新潟-宇都宮-水戸-千葉-さいたま-東京-横浜-静岡-甲府-前橋-長野-富山-金沢-福井-岐阜-名古屋-津-大津-京都-奈良-大阪-神戸-和歌山-徳島-高松-岡山-鳥取-松江-高知-松山-広島-山口-大分-福岡-佐賀-長崎-熊本-宮崎-鹿児島-那覇

総距離は4360.391749kmとなり、鈴木氏のルートより48.841522kmマイナスとなった。鈴木氏は50kmを1cmとした地図で計ったようなので、約1cm分短縮したことになる。先の松江-高知のルートに加え、東日本部分でも首都圏のルートが変わっているほか、鈴木氏が

札幌 ……新潟-前橋-宇都宮 ……甲府-長野 ……那覇

としているところを、

札幌 ……新潟-宇都宮 ……甲府-前橋-長野 ……那覇

として短縮している。地図を見ても前橋は新潟-宇都宮を結ぶ線分よりも甲府-長野を結ぶ線分に近いので、明らかに鈴木氏の見落としである。新潟県と栃木県、山梨県と群馬県はいずれも接していないので新潟-宇都宮、甲府-前橋というルートが思い浮かばなかったのか、または前橋は関東地方という先入観から逃れられなかったのか…。

ちなみに、新潟-(関東・中部・近畿)-和歌山の最短ルートも、

新潟-宇都宮-水戸-千葉-さいたま-東京-横浜-静岡-甲府-前橋-長野-富山-金沢-福井-岐阜-名古屋-津-大津-京都-奈良-大阪-神戸-和歌山

であることは確認出来たので、おそらくこれが日本全国の都道府県庁所在地を通る最短ルートで

あると考えられる。

このルートが最短であると断言するには、このルートが最短であることを証明する必要がある。このためには、より効率の良い探索アルゴリズムを探すか、または北海道・東北・関東・中部ブロックと近畿・中国・四国・九州ブロックという問題分解が正しいことを証明する必要がある。

8．整数計画ソルバーによる計算

前述の葛西氏は、最長片道きっぷのルート計算に整数計画ソルバーを使用している。

整数計画とは何か？については、葛西氏のページ[*11]が大変分かりやすい。簡単に言えば、「問題を一次式の連立不等式で表せれば、専用のソフトウェアが解いてくれますよ。」ということらしい（少なくとも素人の私はそう理解した）。そのソフトウェアがソルバーというようで、Excelにも搭載されている。私もソルバーの使用は考えたが、身近で使えるソルバーであるExcelソルバーは仕様制限で、変数（変化させるセル）は200以内しか使用できない[*12]。後で分かるとおり、変数は ${}_{47}C_2+47=1128$ 個使うことになるので手元では出来ないと判断した。（当初は制約式の数が仕様制限に引っかかると考えていたが、こちらの方は大丈夫なようである。）

日本の県庁所在地を一回ずつ通る最短ルートについて、「7．仮定を入れた上での最短経路」を求めた時点で葛西氏にメールしたところ、予想通り、「整数計画法なら厳密解を得られる可能性があるかも」という回答をいただいた。Excelソルバーの制限の範囲内では収えない旨を伝えたところ、SCIPという整数計画ソルバーを紹介された。そこで「SCIP 入門」で検索すると、宮代氏の文書[*13]が見つかった。これまた大変分かりやすい。葛西氏や宮代氏の文章を見ると、本当に頭がいい人は難しいことを素人に分かりやすく説明出来る人なのだと感じる。ちなみに宮代氏は葛西氏の共同研究者であり、線形計画法による最長片道きっぷのルート計算は葛西氏と宮代氏の共同研究である。

ソルバーを使うためには、この問題を連立不等式で書き表さなければならない。これも葛西氏が重要なヒント（というかほぼ答え）をくれた。

> 定式化としては、
> 各頂点（都道府県庁所在地）に対しダミーの頂点を1つずつ設け、
> リアル頂点と、対応するダミー頂点の間に距離0のダミーの枝を1本ずつ設けて、
> ・リアル頂点から出る枝は必ず2本使用する……(1)
> ・ダミーの枝はグラフ全体でちょうど2回使用する ……(2)
> ・minimize Σ(枝の使用状況(0/1) × 枝の距離)……(3)
> といった感じかなと…

---

*11 https://www.swa785.net/lop/lop_ip01.html#whatsip

*12 https://support.microsoft.com/ja-jp/office/excel-%E3%81%AE%E4%BB%95%E6%A7%98%E3%81%A8%E5%88%B6%E9%99%90-1672b34d-7043-467e-8e27-269d656771c3#IDOEDBD=Newer_versions

*13 https://web.tuat.ac.jp/~miya/miyashiro_ORSJ.pdf

まず、都道府県庁どうしを結ぶ枝(線)に対して$p_{m,n}$(m、nは都道府県番号。$m<n$とする。)という0-1変数を定義する。例えば、札幌(都道府県番号1)と青森(都道府県番号2)を結ぶ枝は$p_{1,2}$という変数になる。0-1変数というのは、その枝を通らないときは0、通るときは1とする変数である。

リアル頂点と、対応するダミー頂点の間に距離0のダミーの枝を1本ずつ設けて…というのが目から鱗である。起点と終点を除いて、その都道府県に来る枝と出て行く枝で、通る枝は必ず2本あるはずだが、起点と終点については通る枝が1本しかない。前述のとおり、起点終点の片方は那覇であることは明らかなのだが、もう一方については不明(おそらく札幌)のままであったので、「起点は札幌」という仮定を加えなければならないと思ったのだが、各頂点(都道府県庁所在地)に距離0のダミーの枝を入れることにより、起点についてもダミーの枝と出て行く枝で2本使用すると書き表すことができる。各都道府県庁所在地のダミーの枝に対して、$p_{m,m}$(mは都道府県番号。)という0-1変数を定義する。

すると(1)については、

$$p_{1,1}+p_{1,2}+p_{1,3}+\cdots\cdots+p_{1,45}+p_{1,46}+p_{1,47}\geqq 2$$
$$p_{1,2}+p_{2,2}+p_{2,3}+\cdots\cdots+p_{2,45}+p_{2,46}+p_{2,47}\geqq 2$$
$$p_{1,3}+p_{2,3}+p_{3,3}+\cdots\cdots+p_{3,45}+p_{3,46}+p_{3,47}\geqq 2$$
$$\vdots \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \vdots$$
$$p_{1,45}+p_{2,45}+p_{3,45}+\cdots\cdots+p_{45,45}+p_{45,46}+p_{45,47}\geqq 2$$
$$p_{1,46}+p_{2,46}+p_{3,46}+\cdots\cdots+p_{45,46}+p_{45,46}+p_{46,47}\geqq 2$$
$$p_{1,47}+p_{2,47}+p_{3,47}+\cdots\cdots+p_{45,47}+p_{46,47}+p_{47,47}\geqq 2$$

と表せる。(後からよく考えれば「$\geqq 2$」は「$=2$」で充分であったが、不等式という思いが強すぎて、「$\geqq 2$」としてしまった。当たり前だが、同じ都道府県庁所在地を2度以上通るルートが最短になるはずがない。)

(2)については、

$$p_{1,1}+p_{2,2}+p_{3,3}+\cdots\cdots+p_{45,45}+p_{46,46}+p_{47,47}\leqq 2$$

と表せる。(後からよく考えればこれも「$\leqq 2$」は「$=2$」で充分であったが「$\leqq 2$」としてしまった。当たり前だが、0や1の場合はそれぞれ0の字形、6の字形のルートになっており、このようなルートが最短になるはずがない。)

(3)については、

$$253.832205p_{1,2}+373.601018p_{1,3}+\cdots\cdots+729.095400p_{45,47}+655.715096p_{46,47}$$

と表せる($p_{m,n}$の係数は都道府県庁所在地間の距離)。これを最小化すればよい。

クロマニヨン人程度の知能があれば(道具を作るための道具を作るという概念を持っているという意味である。実際にクロマニヨン人は毛皮の服を作るための針を作る等が出来たと言われる。)、これを記述したLPファイルを作るプログラムを書くことができる。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
```

```
double pref_dis[47][47]={
    (中略)
    };

main(){
    int q1,q2;
    FILE *fl;

    fl=fopen("journey.lp","w");

    fprintf(fl,"minimize¥n");

    for(q1=0;q1<46;q1++){
        for(q2=q1+1;q2<47;q2++){
            fprintf(fl,"+ %f p(%d,%d) ",pref_dis[q1][q2],q1+1,q2+1);
        }
    }

    fprintf(fl,"¥n");

    fprintf(fl,"subject to¥n");

    for(q1=0;q1<47;q1++){
        fprintf(fl,"c%d: ",q1+1);
        for(q2=0;q2<47;q2++){
            if(q1<q2){
                fprintf(fl,"+ p(%d,%d) ",q1+1,q2+1);
            } else {
                fprintf(fl,"+ p(%d,%d) ",q2+1,q1+1);
            }
        }
        fprintf(fl,"> 2¥n");
    }

    fprintf(fl,"c48: ");

    for(q1=0;q1<47;q1++){
        fprintf(fl,"+ p(%d,%d) ",q1+1,q1+1);
    }
    fprintf(fl,"< 2¥n");
```

```
    fprintf(fl,"binary¥n");

    for(q1=0;q1<47;q1++){
        for(q2=q1;q2<47;q2++){
            fprintf(fl,"p(%d,%d)¥n",q1+1,q2+1);
        }
    }

    fprintf(fl,"end¥n");

    fclose(fl);
}
```

このプログラムを実行すると、以下のようなLPファイルが生成される。

```
minimize
+ 253.832205 p(1,2) + 373.601018 p(1,3) +  ……
    (中略)
 …… + 90.818901 p(45,46) + 729.095400 p(45,47) + 655.715096 p(46,47)
subject to
c1: + p(1,1) + p(1,2) +  ……(中略) …… + p(1,46) + p(1,47) > 2
c2: + p(1,2) + p(2,2) +  ……(中略) …… + p(2,46) + p(2,47) > 2
    (中略)
c46: + p(1,46) + p(2,46) +  ……(中略) …… + p(46,46) + p(46,47) > 2
c47: + p(1,47) + p(2,47) +  ……(中略) …… + p(46,47) + p(47,47) > 2
c48: + p(1,1) + p(2,2) +  ……(中略) …… + p(46,46) + p(47,47) < 2
binary
p(1,1)
p(1,2)
    (中略)
p(46,47)
p(47,47)
end
```

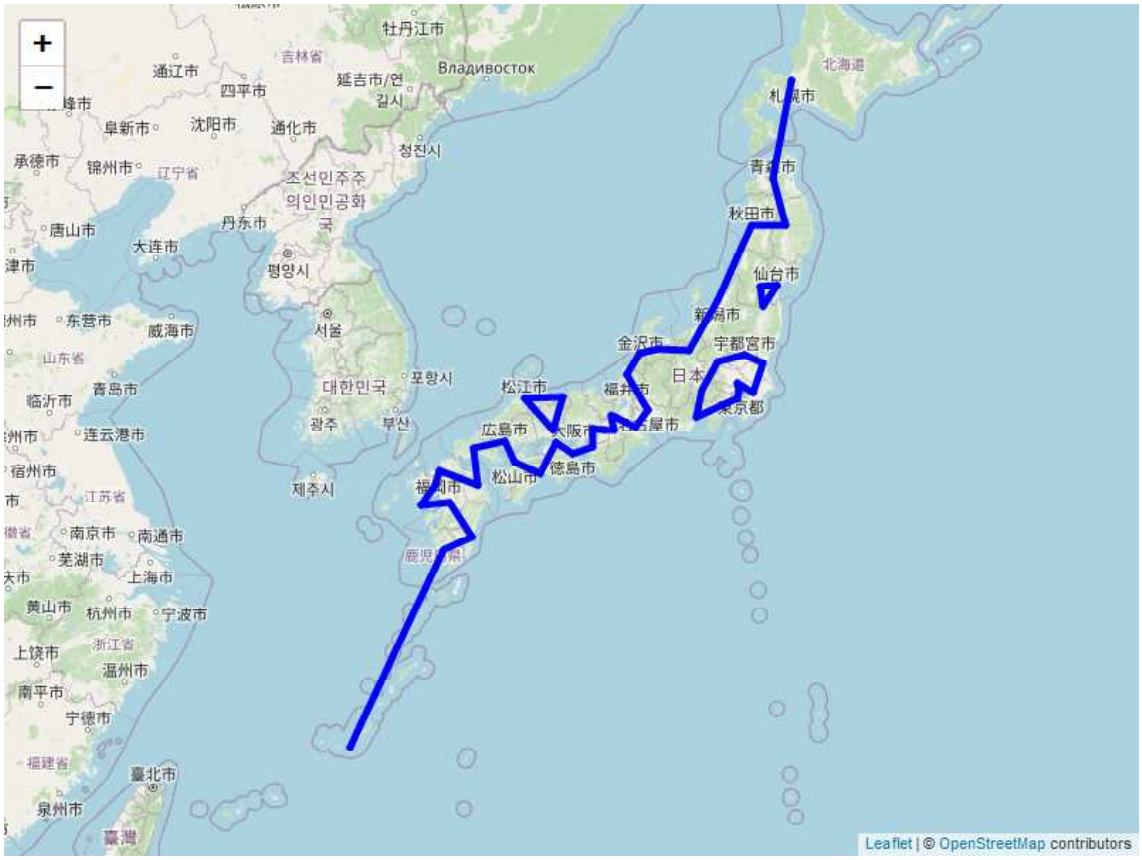
>2 、<2 は>=2 や<=2 でなくて良いのかと思われるかもしれないが、宮代氏の文書によると「> は >= と，< は <= と同じ意味である」らしい。(そもそも本来はどちらも=2 で良かった。)

これをSCIPで解くと、以下のような解が出てきた。いつ計算したの？というスピードである。2千7百正年は何だったのだろうか？

```
solution status: optimal solution found
objective value:                        4304.932018
p(1,1)                                         1   (obj:0)
p(1,2)                                         1   (obj:253.832205)
p(2,3)                                         1   (obj:129.304018)
(以下略)
```

2行目の objective value: 4304.932018 は4304.932018kmのルートが見つかりましたよという意味、3行目の p(1,1) 1 は起点は札幌(都道府県番号1)ですよという意味、4行目の p(1,2) 1 は札幌(都道府県番号1)と青森(都道府県番号2)の枝を通りますよという意味…(5行目以降略)…と思われる。

すわ、4360.391749kmより短いルートが存在したのか？と早とちりしてはいけない。イメージがつきにくいので、これを地図に表してみる。これもさくっとプログラムを作った。



はい。独立したループが出来ているのである。葛西氏のページを読んでいれば想定の範囲内である。この後、独立したループを除去するためには場当たり的に制約式を追加していくことになる。例えば、上の地図では仙台(都道府県番号4)-山形(都道府県番号6)-福島(都道府県番号7)-仙台というループがあるが、これを除去するためには、

$p_{4,6}+p_{4,7}+p_{6,7}\leqq 2$

(仙台-山形、仙台-福島、山形-福島という枝を通るのは2回以内ですよ)

という制約式、つまり、

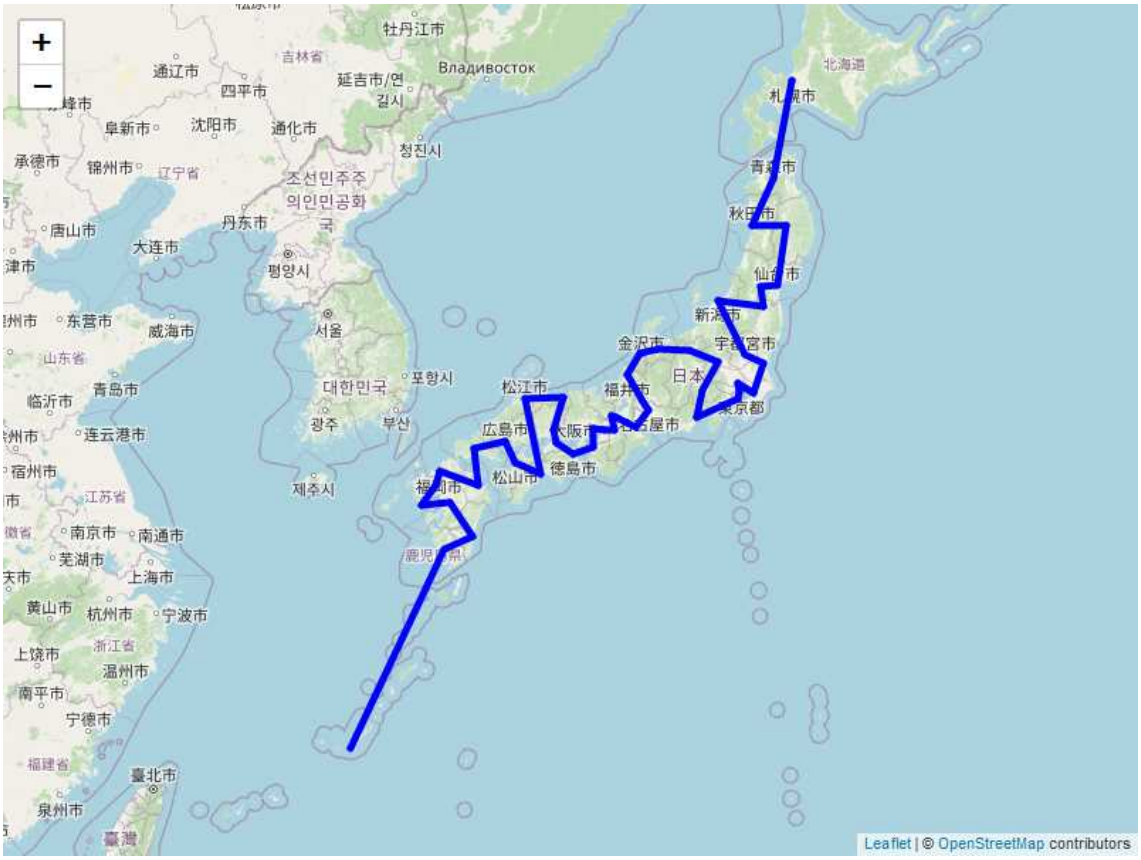
c51: p(4,6) + p(4,7) + p(6,7) < 2

という行を加える。以下、水戸-宇都宮-前橋-甲府-静岡-横浜-東京-さいたま-千葉-水戸のループ、鳥取-松江-岡山-鳥取のループも同様である。これは手作業である。(プログラムを書いても良いが。)

再び計算して、ループを除去して…をくり返すこと3回、以下の制約式を追加したところでループが無くなった。(たった3回であるが、手作業でいつ終わるか分からない作業をくり返すのはきつかった。)

c51: p(4,6) + p(4,7) + p(6,7) < 2
c52: p(8,9) + p(8,12) + p(9,10) + p(10,19) + p(11,12) + p(11,13) + p(13,14) + p(14,22) + p(19,22) < 8
c53: p(31,32) + p(31,33) + p(32,33) < 2
c54: p(8,9) + p(8,12) + p(9,10) + p(10,19) + p(11,13) + p(11,14) + p(12,13) + p(14,22) + p(19,22) < 8
c55: p(31,32) + p(31,37) + p(32,33) + p(33,37) < 3
c56: p(8,9) + p(8,11) + p(9,10) + p(10,19) + p(11,13) + p(12,13) + p(12,14) + p(14,22) + p(19,22) < 8
c57: p(31,32) + p(31,33) + p(32,37) + p(33,37) <3

結果的に求まったルートは以下のとおりである。



その距離は

objective value: 4360.391749

つまり「7．仮定を入れた上での最短経路」で求めた経路が最短ルートであると証明された。

9．計算結果と感想

日本の県庁所在地を一回ずつ通る最短ルートは、

札幌-青森-秋田-盛岡-仙台-山形-福島-新潟-宇都宮-水戸-千葉-さいたま-東京-横浜-静岡-甲府-前橋-長野-富山-金沢-福井-岐阜-名古屋-津-大津-京都-奈良-大阪-神戸-和歌山-徳島-高松-岡山-鳥取-松江-高知-松山-広島-山口-大分-福岡-佐賀-長崎-熊本-宮崎-鹿児島-那覇

であり、その距離は4360.391749kmであることが証明できた。

時速15km/hの自転車で走ると291時間…12日と2時間41分34秒かかることになる。ちなみに時速15km/hというのは鈴木氏の100m走よりも遅いが、全速力で100km走るのは不可能だが、自転車で時速15km平均で100km走るのは容易である。なお、330k INFOさんが自転車で走り続けると201時間…8日と9時間28分58秒かかることになる。キャノンボール(東京～大阪を自転車で1日で走ることをいう)をやる人なら本当に走りそうな範囲内である。

現実には、新潟-宇都宮、甲府-前橋を結ぶ直線に近いルートを通ることは地形上、ほぼ不可能であるし、神戸-和歌山、松江-高知など、海路の関係上不可能なところもあるので、本当に全都道府県庁所在地を最短で回りたいときにはこのルートは全く使い物にならない。

10．今後の展望

距離のテーブルを置き換えるだけで、鉄道による最短ルート、国道による最短ルートなども計算することができる。鉄道の場合、新潟-宇都宮や甲府-前橋といったルートは取れないので、どんなルートになるか興味深いところである。

個人的には、都道府県庁所在地に関わらず、全都道府県を回る最短ルート(この場合、端の都道府県は都道府県境に立つだけでよい)を求める計算方法も考えてみたい。

11．謝辞

葛西隆也さん

ヒマな私の自由研究に的確なアドバイスをくださり、ありがとうございました。

学士会Webサロンの皆様

私が張り合いたくなるような自由研究の木々を提供してくださり、ありがとうございました。

12．参考文献

該当箇所の脚注に表示しています。