Solve TSP by ILP

Xinglu Wang *3140102282* *ISEE 1403, ZJU*

April 20, 2017

Contents

Abstract

In this report, I formulate Traveling Salesman Problem (TSP) as Integer Linear Program (ILP), then collect data for landmarks around HangZhou, use SageMath to solve the ILP and get the exact solution. Although limiting the number of landmarks to 20, I make some simple comparison with Heuristic Algorithm and visualize the routes on GMap. I find all the procedure quite interesting and exciting and learn a lot!

Keywords: TSP, ILP

1 TSP Problem

TSP is a combinatorial optimization problem to find the shortest possible Hamiltonian circuit for a complete weighted graph.

1.1 Notation

There are several key elements:

$\mathbf{s} = (s_i)$	At i step, the index of visited city is s_i, $0 \le i \le n-1$
$\mathbf{t} = (t_i)$	City i is visited at step t_i, $0 \le i \le n-1$
$\mathbf{X} = (x_{ij})$	Decision variable, $x_{ij} = \begin{cases} 1 & \text{the path goes from city } i \text{ to city } j \\ 0 & \text{otherwise} \end{cases}$
$\pi(\cdot)$	$\pi(i) = j$ if $x_{ij} = 1, 0 \le i, j \le n-1$
$\mathbf{C} = (c_{ij})$	Cost Matrix

Table 1: The notations that will be used

- Hamiltonian circuit means the travel route starting from vertex No. 0, ending at No. 0 and visiting all vertexes once and only once. Note that $x_{ij}|_{i=n-1,j=0}$ denote the distance go back from No.n-1 to No.0 while t_i will not consider the time consumed when going back since there is *no* $t_i|_{i=n}$.
- It is common for weights of the graph to become non-negative. But for a real-world problem, the weights do no necessarily be symmetric, because weights can represent time or cost. I choose *unsymmetrical* TSP to solve for its generality ans simplicity Thus, I should be careful that $\mathbf{C}$ is symmetric because the data I collect is distance while the solution $\mathbf{X}$ is unsymmetrical because it describe *directed* graph.

1.2 Formulation as ILP

We formulate this question step by step, first consider a relaxed version ILP of TSP with some necessary(but not sufficient) constrains:

$$\begin{aligned}
\text{Minmize} \quad & \sum_{i=0}^{n-1} \sum_{j=0,j\neq i}^{n-1} c_{ij}x_{ij} && && (1)\\
\text{subject to} \quad & \sum_{i=0,i\neq j}^{n-1} x_{ij} = 1 && j = 0, \dots, n-1; && (2)\\
& \sum_{j=0,j\neq i}^{n-1} x_{ij} = 1 && i = 0, \dots, n-1; && (3)\\
& x_{ij} \ge 0, x_{ij} \in \mathbb{Z} && i, j = 0, \dots, n-1. &&
\end{aligned}$$

It objective function (1) (and all the following equations throughout the problem), we simply drop x_{ii}, since $x_{ii} = 0$ is trivial. The constrain (2) means each city be arrived at from exactly one other city while equation (3) means from each city there is a departure to exactly one other city. We do not need to add $x_{ij} \le 1$ since the rests are enough to conclude it.

But these constraints are not sufficient conditions for TSP. This formulation is, in fact, the ILP formulation of Assignment Problem which belong to P hard class. We still need to add constraints to avoid subtour and we will find that TSP belong to NP hard class. Next we will add n auxiliary variables so that we just need to add $n(n-1)$ constrain.

$$
\begin{aligned}
\text{Minmize} \quad & \sum_{i=0}^{n-1} \sum_{j=0, j\neq i}^{n-1} c_{ij} x_{ij} & & \\
\text{subject to} \quad & \sum_{i=0, i\neq j}^{n-1} x_{ij} = 1 & & j = 0, \ldots, n-1; \\
& \sum_{j=0, j\neq i}^{n-1} x_{ij} = 1 & & i = 0, \ldots, n-1; \\
& t_j \geq t_i + 1 - n(1 - x_{ij})x & & 0 \leq i \leq n-1, 1 \leq j \leq n-1, i \neq j \qquad (4) \\
& x_{ij} \geq 0, x_{ij} \in \mathbb{Z} & & i, j = 0, \ldots, n-1.
\end{aligned}
$$

Note that $j \geq 1$ in (4) means we do not consider salesman going back to No.0. In fact, if we let t_n denote the time step salesman go back to No.0, we have:

$$t_n = t_{n-1} + 1 = n \tag{5}$$

Then we prove that this ILP is equivalent to TSP. **(I).** cyclic permutation without subtour conclude constraint (4). Because (4) is equivalent to statement that if the next visited city for i is j ($x_{ij} = 1$) ,then $t_j = t_i + 1$. If $x_{ij} = 0$, then we add a trivial constraint $t_j \geq t_j + 1 - n$. The equivalent statement is satisfied. **(II).** constraint (4) eliminates all subtour. We can prove it by tricky reductio ad absurdum. Suppose the feasible solution $\mathbf{X}$ contain more than one subtour. Then there exist s $= (i_1, \ldots, i_r)$ not containing 0. Note that i_r in this circle will go back to i_1, so there is additional equation $t_{i_1} = t_{i_r} + 1$ quite different with (5). Go along this cycle, we find that $0 = r$, and the contradiction lies in the fact that the cycle do not containing 0.

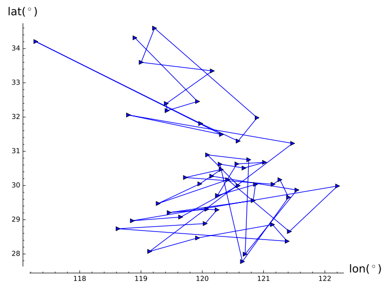
2 Experiments

The steps to solve the ILP include:

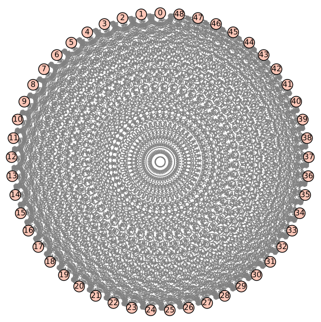
- Collect location data for landmarks around HangZhou.
- Transform from (latitude,longitude) to distance matrix $\mathbf{C}$. Following guide of Haversine_formula.
- Solve the ILP by MILP class. The equivalent code for ILP system (4) is more human-friendly and powerful compared with *'intlinprog'* function of Matlab. Ref. to A.2 on page 6. I put the code in the Appendix, hope do not impact the compactness of this report.
- Visualize on GMap.

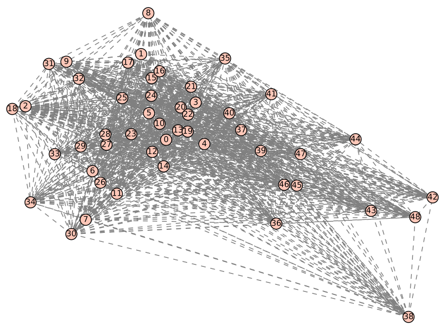
Apply ILP to a toy case with 20 landmarks around HangZhou, we get a optimal strategy to travel around HangZhou. It is quite interesting and exciting. Ref. to Fig 1d on the next page. Then I also use TSP solver in Sage to partially verify that my code is correct. The approximate optimal solution got by some heuristic algorithm is the same as the exact optimal solution. I make some try, and guess that when n is small, it is easier for heuristic algorithm to find exact optimal solution. Then I also try to travel around JiangZheHu Region(Yangtze River delta) and visualize it on the map, Ref. to Fig 2b on the following page.

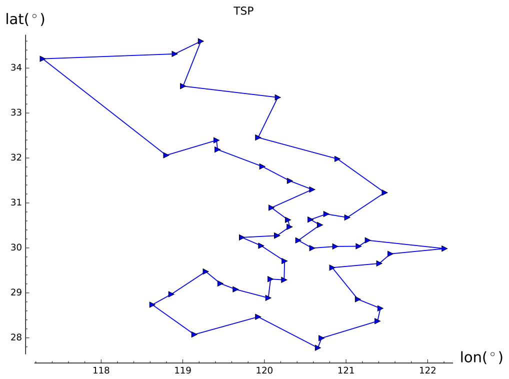
I just take 20 landmarks from 49 collected coordinate data. The reason lies in the fact that I find 50 points consume seemingly endless time. I also have some basic test on running



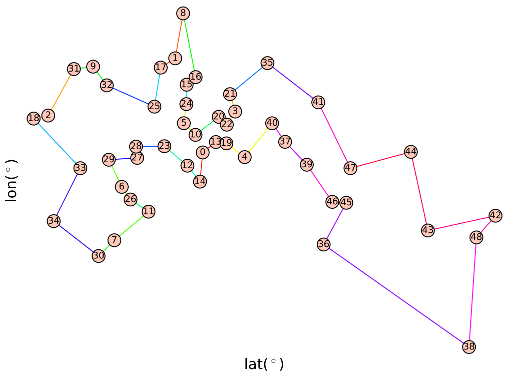
(a) Initial feasible solution for TSP


(b) Transform into complete weight graph


(c) Plot according to locations

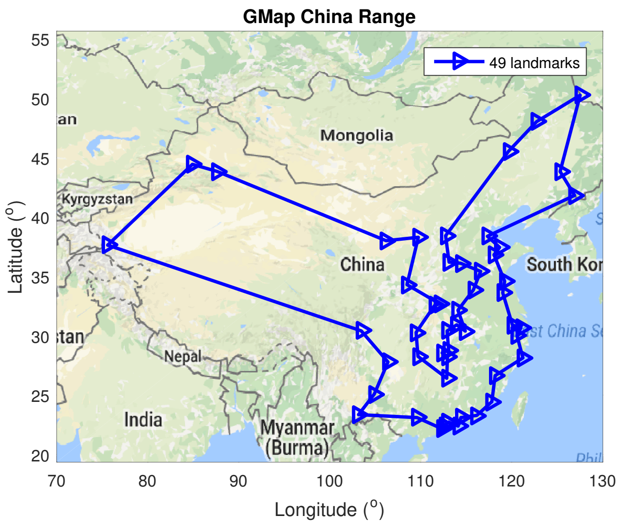


(d) The optimal solution find by ILP

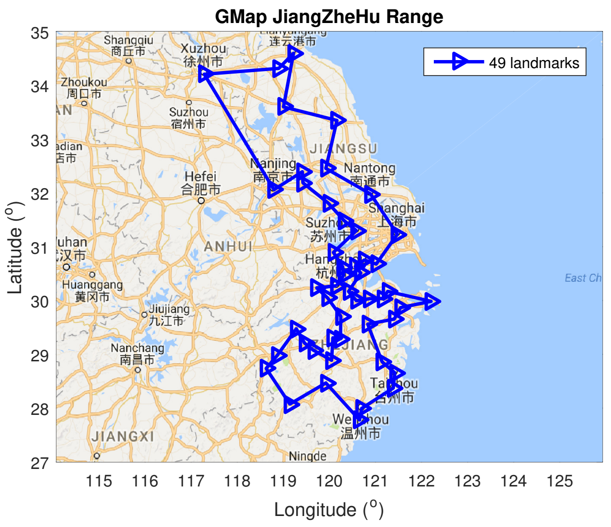


(e) Approximately optimal find by Sage

Figure 1: The plot of TSP class 1e only keep topological relations, but after simple transformation, we find for 20-pts, two solutions $\mathbf{X}$ are equivalent. (Here the transformation means change from undirected graph's $\mathbf{X} = (x_{ij})$ with $i < j$ to directed graph's $\mathbf{X}$)



(a) TSP on Hangzhou



(b) TSP on JiangZheHu

Figure 2: JiangZheHu is a wider range than Hangzhou. We limit number of landmarks to about 20 for speed.

time, and find it grow explosively(and unstably), Ref. to Fig 3. We can conclude that the time complexity of branch-bound method is of exponential time complexity.

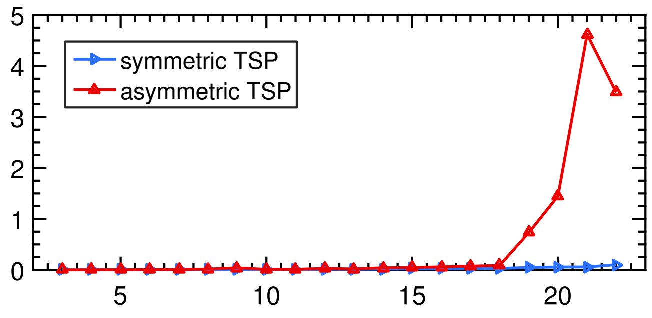


Figure 3: X-axis: number of landmarks, Y-axis: elapsed time in seconds. Running time starts to grow explosively at $n = 17$ and becomes $200s$ at $n = 19$ suddenly.

.

Appendices

Appendix A Lab Code

My codes are somewhat lengthy, so I just put some important code, hope not to impact the compactness of this report.

A.1 Crawling Data of CHINA Landmarks

```
import urllib,json,cPickle

f=open('city_name_pinyin.txt')
city_name=f.readlines()
city_name=[name.strip(' \t\r\n') for name in city_name]
f.close()
print city_name

key='AIzaSyDw12ZWkEypDDOJMbT3UPVouVQLLppeimM'
# address='Beijing'
location=[]
for address   in city_name:
    my_url = (
        'https://maps.googleapis.com/maps/api/geocode/json?address=%s&key=%s'
        ) % (address, key)
    print my_url
    try:
        response=urllib.urlopen(my_url)
    except urllib.error.HTTPError:
        print "Check Internet"
    else:
        print "Unkown"
    json_data=json.loads(response.read())
    print json_data
    loc=[json_data['results'][0]['formatted_address'],
        json_data['results'][0]['geometry']['location']]
    print loc
    location.append(loc)

print len(location),location
with open("location.pkl",'w') as f:
    cPickle.dump(location,f)
```

A.2 Solve by Sage

```
import numpy as np
import matplotlib.pyplot as plt
import sys, os, \
    glob, cPickle, \
    argparse, errno, json,\
    copy, re,time,datetime
import numpy as np
import os.path as osp
import scipy.io as sio
from pprint import pprint
def lldistkm(latlon1,latlon2):
    '''
     Distance:
     d1km: distance in km based on Haversine formula
     (Haversine: http://en.wikipedia.org/wiki/Haversine_formula)
    '''

    radius=6371
    lat1=latlon1[0]*pi/180
    lat2=latlon2[0]*pi/180
    lon1=latlon1[1]*pi/180
    lon2=latlon2[1]*pi/180
    deltaLat=lat2-lat1
    deltaLon=lon2-lon1
    a=sin((deltaLat)/2)^2 + cos(lat1)*cos(lat2) * sin(deltaLon/2)^2
    c=2*atan2(sqrt(a),sqrt(1-a))
    d1km=radius*c    #Haversine distance
    return d1km
def lldistkm_o(l1,l2):
    l1=vector(l1)
    l2=vector(l2)
    return norm(l2-l1)
root='/home/xlwang/l-opt'
os.chdir(root)
print os.getcwd()
with open('location.pkl','r') as f:
    locations =cPickle.load(f)

for m_limits  in range(3,23):
    old_t = time.time()

    location=locations[:m_limits]

    latlons=[ [latlon[1]['lat'],latlon[1]['lng']] for latlon in location ]
    names=[latlon[0].split(',')[0] for latlon in location]
    n_pts=len(latlons)
    tab=table( \
    rows=[(latlons[i][0],latlons[i][1]) for i in range(n_pts)],\
    header_column=names,\
    frame=True\
    )
    str=latex(tab)
    with open("city_data.tex",'w') as f:
        f.write(str)
    dist=np.zeros((n_pts,n_pts))
    for i in range(n_pts):
        for j in range(n_pts):
            dist[i,j]=N(lldistkm(latlons[i],latlons[j]),32)
    dist

    # For small  problem:
    #dist=Matrix([[0,1,2],[2,0,1],[1,2,0]])
    dist=Matrix(dist)
    p=MixedIntegerLinearProgram(maximization=False,solver="GLPK") #,solver="PPL"
    x=p.new_variable(nonnegative=True,integer=True)
    t=p.new_variable(nonnegative=True,integer=True)
    n=dist.nrows()
    obj_func=0
    for i in range(n):
        for j in range(n):
            obj_func+=x[i,j]*dist[i,j] if i!=j else 0
    p.set_objective(obj_func)
    for i in range(n):
        p.add_constraint(sum([x[i,j] for j in range(n) if i!=j ])==1)
    for j in range(n):
        p.add_constraint(sum([x[i,j] for i in range(n) if i!=j ])==1)
    for i in range(n):
        for j in range(1,n):
            if i==j :
                continue
            p.add_constraint(t[j]>=t[i]+1-n*(1-x[i,j]))
    for i in range(n):
        p.add_constraint(t[i]<=n-1)

    #p.show()
    p.solve()
    elaspe=time.time() - old_t
    print m_limits, elaspe
```

```
def lldistkm(latlon1,latlon2):
    '''
     Distance:
     d1km: distance in km based on Haversine formula
      (Haversine: http://en.wikipedia.org/wiki/Haversine_formula)
    '''

    radius=6371
    lat1=latlon1[0]*pi/180
    lat2=latlon2[0]*pi/180
    lon1=latlon1[1]*pi/180
    lon2=latlon2[1]*pi/180
    deltaLat=lat2-lat1
    deltaLon=lon2-lon1
    a=sin((deltaLat)/2)^2 + cos(lat1)*cos(lat2) * sin(deltaLon/2)^2
    c=2*atan2(sqrt(a),sqrt(1-a))
    d1km=radius*c    #Haversine distance
    return d1km
```

```
def lldistkm_o(l1,l2):
    l1=vector(l1)
    l2=vector(l2)
    return norm(l2-l1)

g=Graph(Matrix(dist))
tsp = g.traveling_salesman_problem()
#tsp.plot(edge_style='dashed',color_by_label=True,edge_labels=False).save("tsp_sage.pdf")
tsp.plot(edge_style='dashed',color_by_label=True,edge_labels=False)

g=Graph(Matrix(dist))
#g.plot(edge_style='dashed', edge_color='gray').save("topology.pdf")
g.plot(edge_style='dashed', edge_color='gray')
```

A.3 Draw on GMap

```
ccc
addpath(genpath('..\'));
all_pnt=load('res_latlon.txt');
figure,

lat=all_pnt(:,1);
lon=all_pnt(:,2);
lat_new=[];
lon_new=[];

for i=1:length(lat)
    longitude=lon(i);
    latitude=lat(i);
%     if ~(longitude<120.13 || longitude>120.21)
%         if ~(latitude<30.26 || latitude>30.3)
            lat_new(end+1)=lat(i);
            lon_new(end+1)=lon(i);
%             continue;
%         end
%     end

end
lat=lat_new;
lon=lon_new;

plot(lon,lat,'MarkerSize',8,'Marker','>',...
    'LineWidth',2,...
    'Color',[0 0 1])

plot_google_map
set(gcf,'Color','White');
legend('21_landmark')

% create xlabel
xlabel('Longitude (^o)');

% create ylabel
ylabel('Latitude (^o)');
title('GoogleMap JiangZheHu Range')
export_fig .\l-opt\GMap_Jzh.pdf
```

References

[1] http://www.openstreetmap.org/export#map=16/30.2802/120.1700

[2] https://github.com/sagemath/sagelib/blob/master/sage/graphs/generic_graph.py