Solve TSP by ILP

Xinglu Wang   *3140102282*   *ISEE 1403, ZJU*

April 24, 2017

Contents

Abstract

In this report, I formulate *Traveling Salesman Problem* (TSP) as *Integer Linear Program* (ILP), crawl data for landmarks, solve the ILP by SageMath and visualize the routes on GMap. For 50 landmarks symmetric TSP, I try to formulate it as undirected graph model to keep the size of problem small-scale and successfully receive a solution in fast speed. I do not explore the method that detect and eliminate subtour dynamically at running-time, because I prefer to elegant closed-form ILP. I want to focus on symmetric formulation of symmetric TSP and hope to compose a elegant report. I find all the procedure quite interesting and exciting and I feel quite grateful for Dear Prof. Thomas Honold who gives us a lot help!

Keywords: Symmetric/Asymmetric TSP, ILP

1 TSP

TSP is a combinatorial optimization problem to find the shortest possible Hamiltonian circuit for a complete weighted graph. For the notation that will be used, Ref. to Tab 1 on the following page

1.1 Notation

$\mathbf{s} = (s_i)$	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 **Symmetric TSP**: $x_{ij} = 1$ when there is a path from No.i to No.j $0 \le i, j \le n-1, i \ne j$ **Aymmetric TSP**: $x_{ij} = 1$ when there is a connect between No.i and No.j $0 \le i < j \le n-1$
$\pi(\cdot)$	$\pi(i) = j$ if $x_{ij} = 1, 0 \le i, j \le n-1$
$\mathbf{C} = (c_{ij})$	Cost Matrix with arbitrary c_{ii} since we do not consider $x_{ij}\|_{i=j}$ **Symmetric TSP**: $\mathbf{C} = \mathbf{C}^T$ **Aymmetric TSP**: no certain relation between $\mathbf{C}$ and $\mathbf{C}^T$

Table 1: The notations that will be used

There are several key components in the definition of TSP:

- 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 **ending at No. 0** and **once and only once** are not contradict! Because we consider this in our notation: $x_{ij}|_{i=n-1,j=0}$ 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* $t_i|_{i=n}$.
- **Asymmetric TSP** is more general and easy to formulate. But it can increase the size of problem doubly, Ref. to 1.2 on page 4. In this case, 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.
- **Symmetric TSP** will be formulate by undirected graph model and reduce the number of variable by 50%. In this case, be careful that $\mathbf{C}$ and the solution $\mathbf{X}$ are both symmetric, which means the subindex of $\mathbf{X}$ will be in lexicographic order (*i.e.* $\mathbf{X} = (x_{ij}), i < j$)

1.2 Formulation as ILP

Asymmetric TSP

Let us formulate asymmetric TSP using undirected graph model straightly. We will add some constraints (*i.e.* n auxiliary variables and $n(n-1)$ constrain) to avoid subtours and we will find that TSP belong to NP hard class

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

In the objective function (1), we simply drop x_{ii}, because $c_{ii} = 0$ if we use **adjacent matrix C**, but if we use *'graph.CompleteGraph'* in Sage, we do not need to consider $j \neq i$, because there will not be self-loop.

Note that $j \geq 1$ in (2) 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:

$$t_n = t_{n-1} + 1 = n \tag{3}$$

Then we prove that this ILP is equivalent to TSP. **(I).** cyclic permutation without subtour conclude constraint (2). Because (2) 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 (2) eliminates all subtour. We can prove it by tricky reductio ad absurdum. Suppose the feasible solution **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 (3). Go along this cycle, we find that $0 = r$, and the contradiction lies in the fact that the cycle do not containing 0.

Symmetric TSP

However, I find it takes a long time to solve 50 landmarks TSP, Ref to Fig 3 on page 6. So I try to formulate it by directed graph model($G = (V, E), |V| = n - 1$).

If we use adjacent matrix, degree constraint becomes $\sum_{i=0}^{j-1} x_{ij} + \sum_{k=j+1}^{n-1} x_{jk} = 2, 0 \leq j \leq n-1$. If we use sparse incident matrix **A**, degree constrain becomes $\sum_{v=0}^{n-1} A_{ve} = 2, 0 \leq e \leq |E|$.

For subtour elimination constraint, I try to express it by

$$|t_j - t_i| \geq \begin{cases} 1 & x_{ij} = 1 \text{ and } j \neq 1 \\ 1 - n & x_{ij} = 0 \end{cases}$$

But it is difficult to transform into linear constraints.

So I look into Sage's code[2]. We can introduce auxiliary variables $r_{ij}, 0 \leq i, j \leq n-1, i \neq j$ with different meaning.

$$r_{ij} + r_{ji} \geq x_{ij} \qquad , \text{for } 0 \leq i < j \leq n-1 \tag{4}$$

$$\sum_{i=0, i\neq j}^{n-1} r_{ij} \leq 1 - \frac{1}{2n} \qquad , \text{for } 0 \leq j \leq n-1 \tag{5}$$

We give (r_{ij}) some physical meaning. We want to move $\frac{1}{n}$ from every node to 1 through now network flow (r_{ij}). There is a straight sense that it can be done if the circle we find is Hamilton Circle.

Then let me prove the correctness of these constrains strictly: **(I).** If existing a Hamilton Circle, then exists (r_{ij}) satisfies constraints. Even if the original graph is not a complete graph, so we still can simply construct a **virtual** network flow. Construct in these way: if No.i is connect with No.0 in the Hamilton circle we find, then modified $r_{i0} = 1 - \frac{1}{2n}$, and $r_{0i} = \frac{1}{2n}$. If No.i is not directly connect with No.0, then $r_{i0} = -\frac{1}{2n}$, $r_{0i} = \frac{1}{2n}$. Then I think r_{ij} will satisfies all constrains. **(II).** If existing a subtour not containing No.0, then constrains will not be satisfied. Sum along the subtour, we will get $k - \frac{k}{2n} \geq k$, which is contradictory.

Finally I want to formulate TSP use language of Graph which become elegant and straight in Sage. For $G = (V, E)$:

$$\begin{aligned}
\text{Minmize} \quad & \sum_{(\mu,\nu)\in E} c_{\mu\nu}x_{\mu\nu} & \\
\text{subject to} \quad & \sum_{\mu\in N_G(\nu)} x_{\mu'\nu'} = 2 & \text{for all } \nu \in V \\
& r_{\mu\nu} + r_{\nu\mu} \geq x_{\mu\nu} & \text{for all } (\mu,\nu) \in E \\
& \sum_{\mu\in N_G(\nu)} r_{\mu'\nu'} \leq 1 - \frac{1}{2n} & \text{for all } \nu \in V \wedge \nu \neq 0 \\
& x_{\mu\nu} \geq 0, x_{\mu\nu} \in \mathbb{Z} & \text{for all } (\mu,\nu) \in E.
\end{aligned}$$

Note that $(\mu, \nu) \in E$ makes sure $\mu < \nu$, and μ', ν' in above satisfies $\begin{cases} \mu' = \min(\mu,\nu) \\ \nu' = \max(\mu,\nu) \end{cases}$ to make sure $\mu' < \nu'$. I write in this way to keep optimization equalities and inequalities elegant.

Relation between Symmetric/Asymmetric TSP

For example[4], adjacent matrix for Symmetric TSP can transform into Asymmetric TSP by adding some *ghost nodes* A', B', C'. And the link weight between original node and ghost node can be set to small negative number.

	A	B	C
A		1	2
B	6		3
C	5	4	

$\rightarrow$

	A	B	C	A′	B′	C′
A				-w	6	5
B				1	-w	4
C				2	3	-w
A′	-w	1	2			
B′	6	-w	3			
C′	5	4	-w			

Apparently, the equivalent symmetric formulation of a asymmetric TSP have double variables. Since exact algorithm(ILP) for TSP can only be fast for small-scale problem, it is important to capture symmetric structure for 50 landmarks TSP to keep problem small-scale. Thanks a lot for Dear Prof. Honold's advice!

2 Experiments

The steps to solve the ILP include:

- Crawl location data for landmarks around HangZhou and China.
- Transform from (latitude,longitude) to distance matrix **C**. Following guide of Wiki/Haversine_formula[3].
- Solve the ILP by MILP class. The equivalent code for ILP system (2) is more human-friendly and powerful compared with *'intlinprog'* function of Matlab. I put the code in the Appendix, hope do not impact the compactness of this report.
- Visualize on GMap.

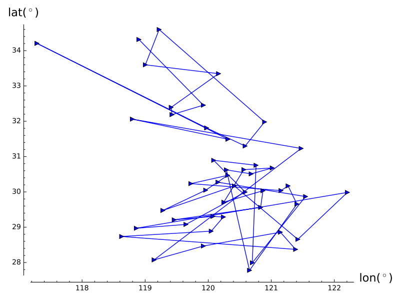
Apply ILP to a toy case with 50 landmarks around China, we get a optimal strategy to travel around China. It is quite interesting and exciting. Ref. to Fig 1d on the next page. I also try to travel around JiangZheHu Region(Yangtze River delta) and visualize it on the map, Ref. to Fig 2b on the following page.

I have some test about time of different formulations. Ref. to Fig 3 on the next page. We can observe that Symmetric formulation can solve 50 landmarks TSP while Asymmetric formulation can solve 23 landmarks TSP. We can expect them to grow explosively after 50 and 23.

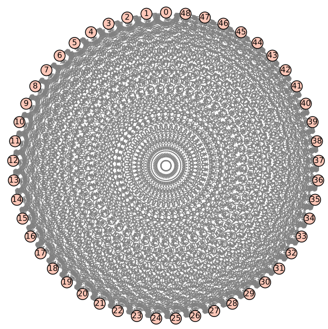
Appendices

Appendix A Figure

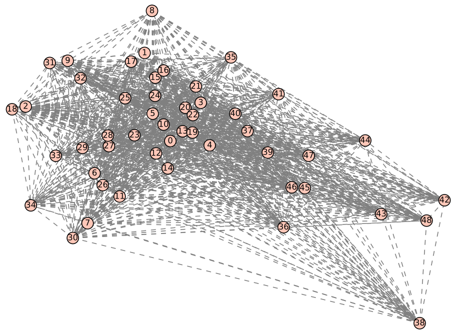
I put all figures in appendices to keep description of problem compact.



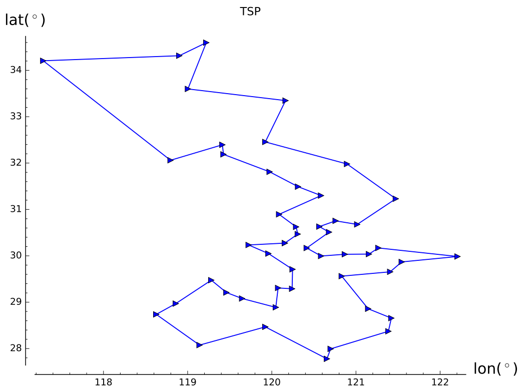
(a) Initial feasible solution for TSP



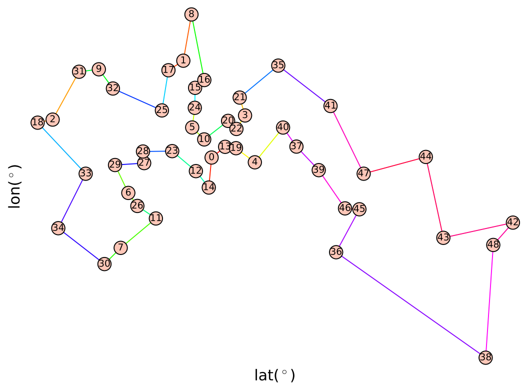
(b) Transform into complete weight graph



(c) Plot according to locations

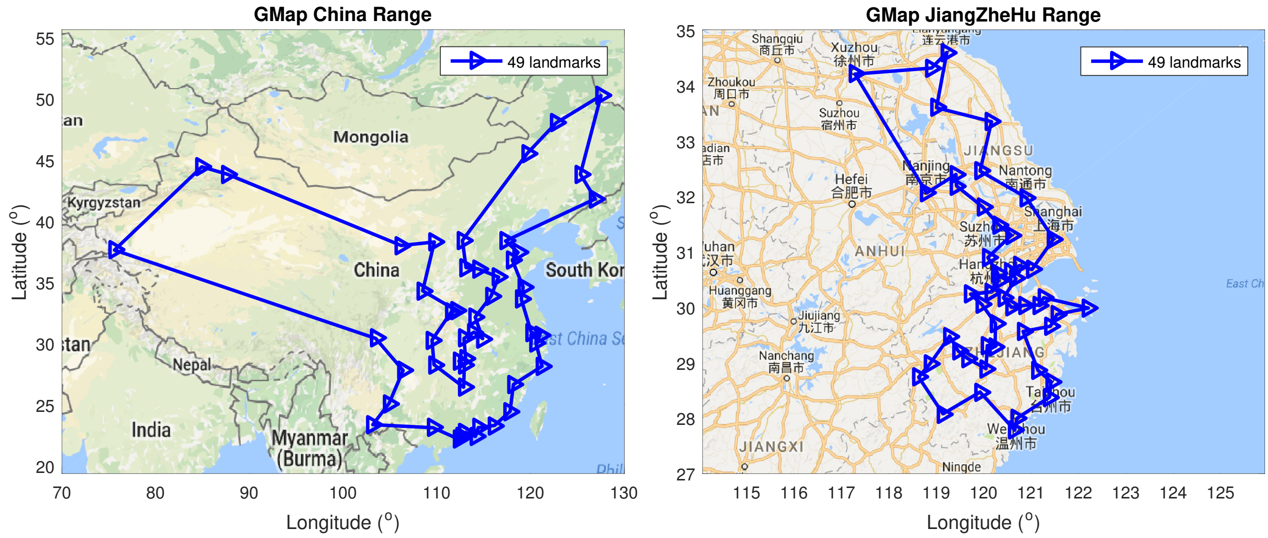


(d) The optimal solution find by ILP



(e) The optimal solution find by Sage(also formulate problem as ILP to solve)

Figure 1: Fig 1d is equivalent to Fig 2b on the following page with correct (longitude, latitude) as coordinate. But when set the position information for Graph I use (latitude, longitude) mistakenly. We can see the solution find by B.2 on page 7 is the same as Sage. In fact, I look into Sage's Source code, inspired by it and write new formulation of symmetric TSP.



(a) TSP on Hangzhou (b) TSP on JiangZheHu

Figure 2: JiangZheHu is a wider range than Hangzhou. I am quite happy to solve a TSP on China range.

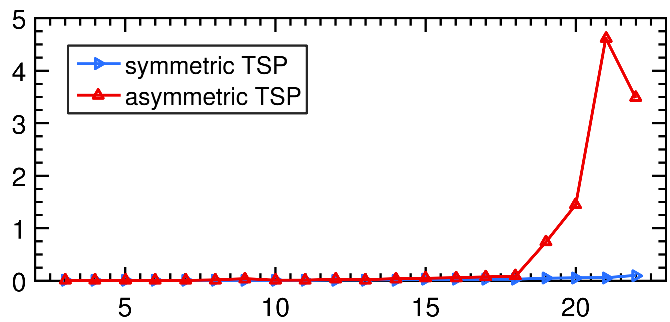


Figure 3: X-axis: number of landmarks, Y-axis: elapsed time in seconds. For Asymmetric TSP, running time starts to grow explosively at $n = 19$. We can expect Symmetric TSP starts to grow explosively after $n = 50$.

.

Appendix B Lab Code

Here let me put some important code(the part for different ILP formulation of TSP).

B.1 Symmetric Formulation

Here I use adjacent matrix.

```
# For small  problem:
#dist=Matrix([[0,1,2],[2,0,1],[1,2,0]])
dist=Matrix(dist)
p=MixedIntegerLinearProgram(maximization=False) #,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()
```

B.2 Asymmetric Formulation

Here I use language of graph.

```
p = MixedIntegerLinearProgram(maximization = False)

f = p.new_variable()
r = p.new_variable()

eps = 1/(2*Integer(g.order()))
x = g.vertex_iterator().next()

R = lambda x,y : (x,y) if x < y else (y,x)

# returns the variable corresponding to arc u,v
E = lambda u,v : f[R(u,v)]

# All the vertices have degree 2
for v in g:
    p.add_constraint(sum([ f[R(u,v)] for u in g.neighbors(v)]),
                     min = 2,
                     max = 2)
# r is greater than f
for u,v in g.edges(labels = None):
    p.add_constraint( r[(u,v)] + r[(v,u)] - f[R(u,v)], min = 0)
# defining the answer when g is not directed
tsp = Graph()
# no cycle which does not contain x
weight = lambda l : l if (l is not None and l) else 1
for v in g:
    if v != x:
        p.add_constraint(sum([ r[(u,v)] for u in g.neighbors(v)]),max = 1-eps)
p.set_objective(sum([ weight(l)*E(u,v) for u,v,l in g.edges()]) )
p.set_binary(f)
```

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

[3] https://en.wikipedia.org/wiki/Haversine_formula

[4] https://en.wikipedia.org/wiki/Travelling_salesman_problem#Solving_by_conversion_to_symmetric_TSP