University of Science and Technology of China

Optimization Algorithm

MATH5015P

Course Project Report

Author:

Jiazhi He: PB19151772

May 27, 2022

Contents

1 Research Background

Logistics distribution is aimed at the needs of customers, and at the same time, according to the order requirements of users, a series of sorting, coding, distribution and other tally work are carried out. The goods with determined quantity and specifications are delivered to users according to the agreed time and place.

Let's start with a simple courier itinerary: when we buy something online, its courier itinerary will go through multiple stations and various means of transportation to reach us.

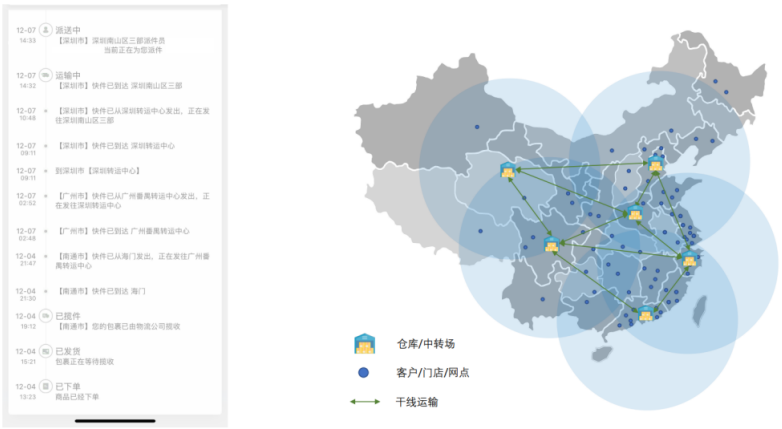

Figure 1: Logistics distribution

There are multiple transfer stations in each region, so which transfer station should you choose? Airplane, train, or car? How to choose between the delay of a single user and the efficiency of the overall logistics system?

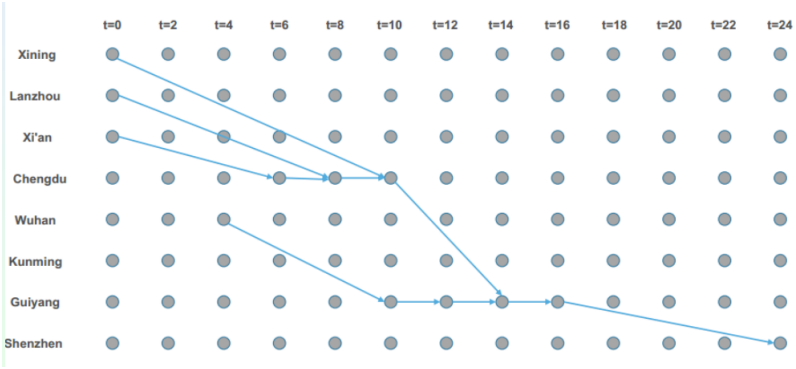

Figure 2: Path Selection

Logistics problems are mainly divided into the following three problems:

- Site selection
- Path planning problem
- Assignment problem

We will mainly focus on the path planning problem.

According to the objective function:

- lowest shipping cost
- minimum delay

According to the constraints:

- Time window constraints
- Vehicle Volume Constraints

So the model has the following classification:

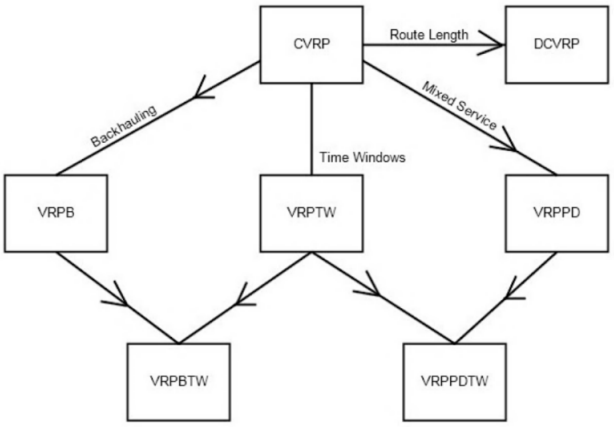

Figure 3: Model classification

2 Optimization Modeling

Traveling Salesman Problem, TSP: A commodity salesman is going to sell goods in several cities. The salesman starts from one city and needs to pass through all the cities before returning to the starting point. How should the travel route be chosen so that the total travel time is the shortest?

The essence of this problem is to find a Hamilton circuit with the smallest weight in a weighted completely undirected graph. Commonly used algorithms:

- Exact solution algorithm: branch and bound method, linear programming method, dynamic programming method
- Approximate algorithm or heuristic algorithm: genetic algorithm, simulated annealing method, ant colony algorithm, tabu search algorithm, greedy algorithm and neural network, etc.

What we are looking for is the shortest path that visits all nodes, so we may be able to solve TSP by a method similar to solving the shortest path. As shown in the figure below, each column is the number of the node visited, and the shortest path starts from node 1 , goes through n steps, traverses through all nodes and returns node 1.

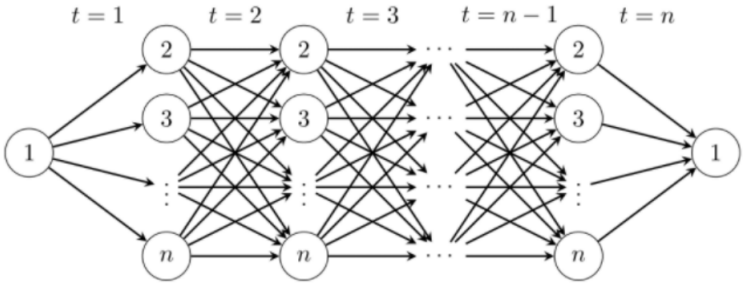

Figure 4: shortest route

For the shortest path model, the decision variable $x_{ij}^t$is a 0-1 variable, defined as follows:

$$x_{ij}^t = \begin{cases} 1 & \text{If path crosses arc}(i,t) \text{ and } (j,t+1) \\ 0 & \text{Otherwise} \end{cases}$$

$$i \in \{V\}, j \in V - \{i\}, t = 1, \cdots, n$$

Then the objective function is

$$\min \sum_{i \in V} \sum_{j \in V - \{i\}} c_{ij} \sum_{t=1}^{n} x_{ij}^t$$

First, refer to the network flow model of the shortest path problem to obtain the constraints related to the shortest path

$$\sum_{j \in V - \{1\}} x_{1j}^t = 1$$

$$\sum_{j \in V - \{1,i\}} x_{ij}^2 - x_{1i}^1 = 0, \forall i \in V - \{1\}$$

$$\sum_{j \in V - \{1,i\}} - \sum_{j \in V - \{1,i\}} x_{ji}^{t-1} = 0, \forall i \in V - \{1\}, t \in \{2, \cdots, n-1\}$$

$$x_{i1}^n - \sum_{j \in V - \{1,i\}} x_{ji}^{n-1} = 0, \forall i \in V - \{1\}$$

$$\sum_{i \in V - \{1\}} x_{i1}^n = 1$$

$$x_{1i}^1 + \sum_{t=2}^{n-1} \sum_{j \in V - \{1,i\}} x_{ji}^t \leq 1, \forall i \in V - \{1\}$$

It is required that all customers will not be visited repeatedly. In this way, we get the TSP integer programming model based on the shortest path.

Since there are n stages in this model (that is, $t = 0, t = 1$ and so on shown in the figure), this model is also called Time-staged model.

3 Solving Algorithm

This report uses the branch &bound algorithm (using lazy cut and user cut) to implement TSP problem modeling and solving based on GurobiPy.

3.1 DFJ Model

Define the decision variable as a 0-1 variable x_{ij}

$$x_{ij} = \begin{cases} 1 & \text{if goes from } i \text{ to } j \\ 0 & \text{otherwise} \end{cases} \quad (i,j) \in A$$

The objective function of the model can be written as

$$\min \sum_{(i,j)\in A} c_{ij} x_{ij}$$

The following two sets of constraints must be satisfied:

$$\sum_{j\in V,(i,j)\in A} x_{ij} = 1, \forall i \in V, \sum_{i\in V,(i,j)\in A} x_{ij} = 1, \forall j \in V$$

Only with this set of constraints, paths may contain subrings.

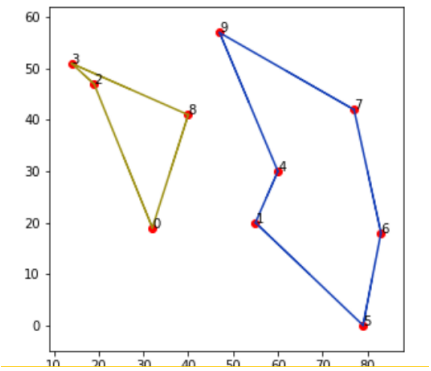

Figure 5: There may be subrings

To eliminate subrings, we add the following constraints.

For a proper subset S, it contains at least one edge to its complement.

$$\sum_{j \notin S, i \in S, (i,j) \in A} x_{ij} \geq 1, \quad \forall s \subset V, 2 \leq |S| \leq n-1$$

For a proper subset S, the number of internal edges is less than $|S|$

$$\sum_{i,j \in S, (i,j) \in A} x_{ij} \leq |S| - 1, \quad \forall s \subset V, 2 \leq |S| \leq n-1$$

However, for the $O(n^2)$ level 0-1 decision variable, the order of magnitude of the constraint is $O(2^n)$, affecting the performance of the above model in the actual scene, we need to add constraints dynamically.

3.2 MTZ Model

The problem of increasing the number of constraints is solved by adding a set of auxiliary decision variables. u_i represents the number of steps to reach the node i, $u_1 = 1$.

$$u_i + 1 \leq u_j + M(1 - x_{ij}), \quad i, j = 2, \cdots, n \in V, (i,j) \in A$$

$$1 + u_i \leq n - 1, \quad i \in 2, \cdots, n \in V$$

$O(n^2)$ level 0-1 decision variables, $O(n)$ level integer variables, the order of magnitude of constraints is $O(n^2)$. The number of constraints is less than the previous models, but the constraints are not as compact as the previous models. Integer optimization is good at optimizing a large number of 0-1 variables, and hates large M.

3.3 Branch &Bound

The branch and bound algorithm is the most basic exact algorithm for solving mixed integer optimization problems.

- Use a tree structure to divide the solution space into subspaces, $x_1 \leq 2$ or $x_1 \geq 3$.
- For each subspace, after ignoring the integer constraints, the problem becomes an ordinary linear programming, which can be solved by the simplex method. For the minimization problem, a lower bound of the original problem is obtained by ignoring the integer conditions.
- If the real number solution is larger than the existing optimal integer solution, it is impossible to find a better integer solution for the subspace.
- If the real number solution satisfies the integer constraint, the integer optimal solution of the subspace is obtained. It is only necessary to compare with the existing feasible solutions to determine whether to update the current optimal solution.
- If the real number solution does not satisfy the integer condition, select a variable x_i to branch, if $x_i = 0.3$, divide into two subspaces $x_i \leq 0$ or $x_i \geq 1$ (adding more constraints)

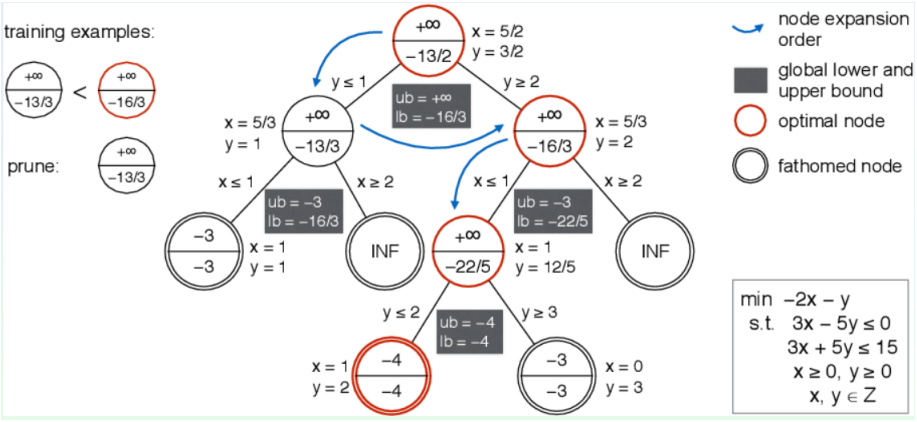

Figure 6: Branch &Bound Example

The modeling of the MTZ model is not compact enough, and the modeling of the DFJ model is compact, but the number of constraints is exponential. Below we use **Lazy Cut** and **User Cut** to dynamically add constraints to remove sub-rings during the constraint process.

3.4 Lazy cut

For the child nodes of the tree, when a real number solution is obtained, a variable is branched. When an integer solution is obtained, the next vertex from each vertex can already be known. If starting from a vertex, it does not pass through all vertices in the end, back to the starting point; a sub-ring is found, and the constraint of removing the sub-ring is added.

3.5 User cut

How can we quickly find the violated subring constraints when the real solution is obtained?

$$\sum_{j \notin S, i \in S, (i,j) \in A} x_{ij} \geq 1, \quad \forall s \subset V, 2 \leq |S| \leq n-1$$

We can consider the minimum cut problem, which divides the vertices into two disjoint sets so that the weight of the edge between the two sets is the lowest.

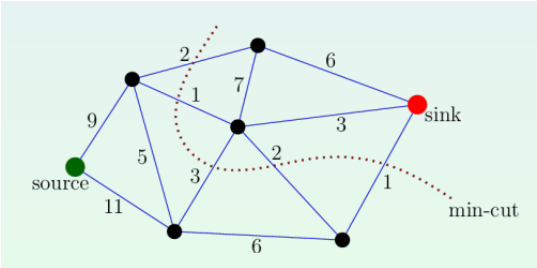

Figure 7: Minimum Cut

Consider the undirected symmetric TSP problem, use $x_{[ij]}$ to indicate whether the ij edge is selected; ij does not distinguish the order. It only represents the specified edge. There are a total of $\frac{n(n-1)}{2}$ variables. The constraint that the in-degree and out-degree

are respectively 1 becomes for the node i, all edges' passing through i sum is 2. We have the following problem

$$\begin{aligned} \min \quad & \sum_{i,j\in V, j\neq i} c_{[ij]} x_{ij} \\ \text{such that} \quad & \sum_{j\in V, j\neq i} x_{[ij]} = 2 \quad \forall i \in V \\ & x_{[ij]} \in \{0,1\} \end{aligned}$$

We need to find the set S_1 such that

$$\sum_{i\in S_1, j\notin S_1} x_{[ij]} \geq 2$$

In the minimum cut problem without source and sink nodes, the node set needs to be divided into two disjoint sets, so that the sum of the weights of the edges between the two sets is minimized. For non-integer nodes, solve the minimum cut problem. If the minimum cut is less than 2, the set S_1 that does not satisfy the constraint is found.

4 Input and Output

4.1 Lazy Cut

See the file Project1-lazycut.ipynb for the implementation of lazy cut.
Compile the functions `distance`,`subtour`,`subtoureli` first. Function `distance` is used to calculate the Euclidean distance between two points. Function `subtoureli` is the main function of optimization. Function `subtour` is used to get optimal path.

Just take part of the data provided by the teaching assistant as an example.

```
load C:/Users/mac/Desktop/Optimization/TSP data/20_0.py
```

output using the following code

```
city=list(range(len(points)))
coordinates={} ##dictionary
for i in city:
coordinates[i]=(float(points[i][0]),float(points[i][1]))
dist={(c1,c2):distance(c1,c2) for c1,c2 in combinations(city,2)}
```

```
m=gp.Model()
vars=m.addVars(dist.keys(),obj=dist,vtype=GRB.BINARY,name='x')
for i,j in vars.keys():
vars[j,i]=vars[i,j]
cons=m.addConstrs(vars.sum(c,'*')==2 for c in city)
m._vars=vars
m.Params.lazyConstraints=1
%time m.optimize(subtoureli)
vals=m.getAttr('x',vars)
selected=gp.tuplelist((i,j) for i,j in vals.keys() if vals[i,j]>0.5)
tour=subtour(selected)
assert len(tour)==len(city)
tour
```

The output result (using tspData100 _2) is

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

4.2 User Cut

See the file Project1-usercut.ipynb for the implementation of User Cut, Compile the functions `ipTSP` first.

Enter as follows

```
from tspData.tsp20_1 import points
a = list(range(len(points)))
points_dict=dict(zip(a,points))
```

The output can be done by using the main function `ipTSP`.

```
dist = {(c1, c2): distance(c1, c2) for c1 in range(len(points)) for c2
                                        in range(len(points)) }
```

```
ipTSP(points_dict, dist,fml='DFJ_User',outputFlag=True)
```

The output result (using tspData100 _2) is

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

Consistent with the results in Lazy Cut.

5 Comparing the effect of Lazy Cut and User Cut on solution rate

Although intuitively, the rate of User cut is faster than that of Lazy Cut, in fact, Lazy Cut always has a solution time close to 0, while the solution rate of User Cut is unstable. Some comparison results are shown as follows

```
Root relaxation: objective 6.978058e+00, 150 iterations, 0.01 seconds (0.00 work units)

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0    6.97806    0   18          -    6.97806      -     -    0s
     0     0    7.17112    0   16          -    7.17112      -     -    0s
     0     0    7.17695    0   16          -    7.17695      -     -    0s
     0     0    7.18644    0   30          -    7.18644      -     -    0s
     0     0    7.18644    0   30          -    7.18644      -     -    0s
...
Best objective 7.525137319930e+00, best bound 7.525137319930e+00, gap 0.0000%

User-callback calls 3349, time in user-callback 0.32 sec
Wall time: 1.06 s
```

Figure 8: Data100-1,Lazy

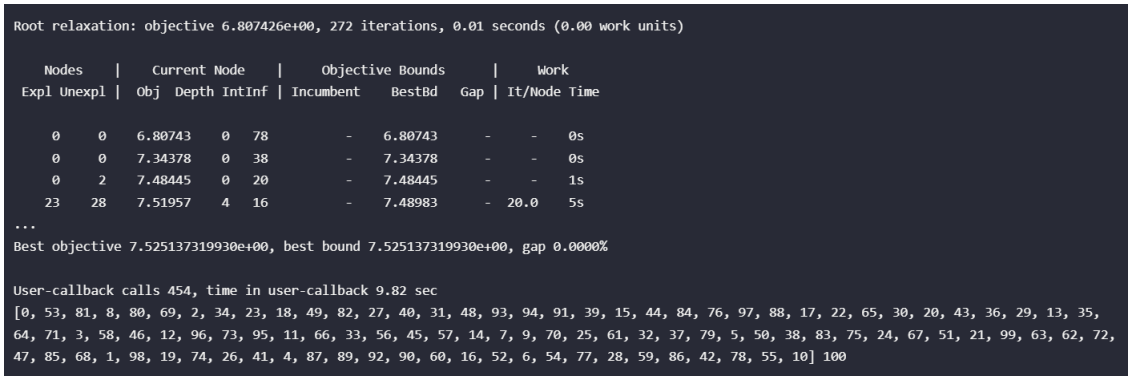

Figure 9: Data100-1,User

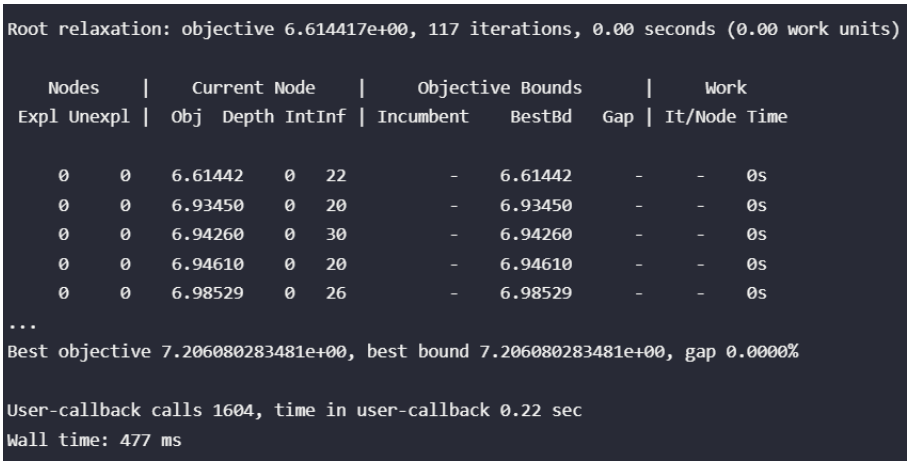

Figure 10: Data80-2,Lazy

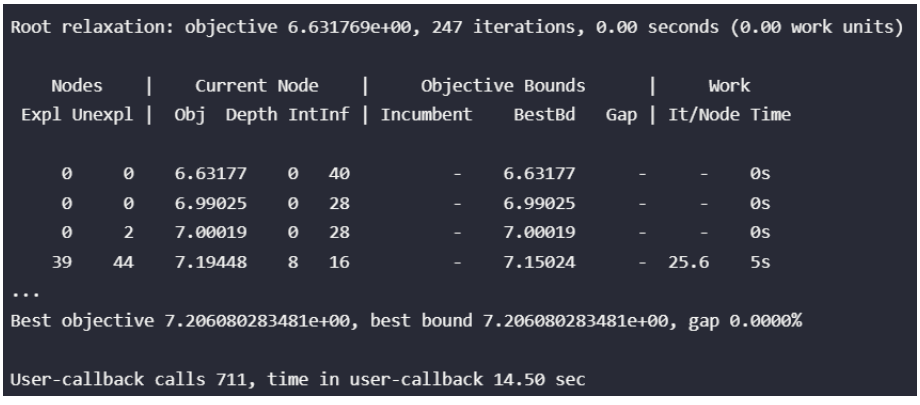

Figure 11: Data80-2,User

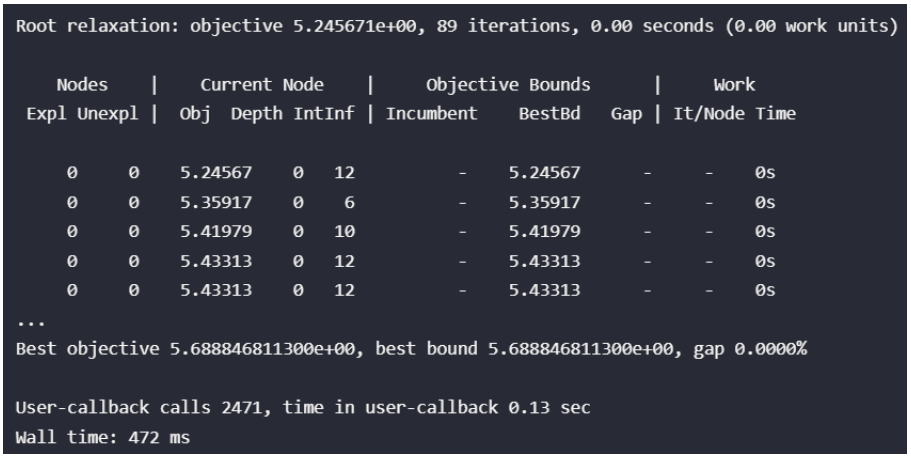

Figure 12: Data60-2,Lazy

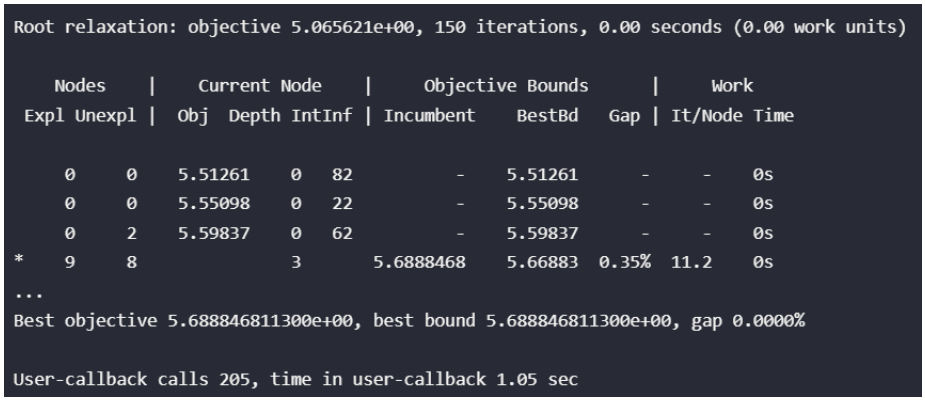

Figure 13: Data60-2,User

Therefore, the following conclusions are drawn

- The solution rate of Lazy Cut is more stable and the time is close to 0
- The solution rate of User Cut is not stable, and it is not necessarily faster than Lazy Cut.

My code is open source and available on Github[1].

[1] https://github.com/He-jiazhi/Optimization-Algorithms-Course-Project/tree/main/Project1-TSPproblem