第一问：基于Branch and cut求解的TSP/Pickup and delivery Problem

$$\min d+\sum_{k\in K}\sum_{i\in V}\sum_{j\in V}c_{ij}x_{ijk}$$

$$\text{s.t.}\begin{cases}d=\max\limits_{k\in K}\sum\limits_{i\in V}\sum\limits_{j\in V}c_{ij}x_{ijk}\\ \sum\limits_{j\in V}\sum\limits_{k\in K}x_{ijk}=1,\forall i\in V-\{0\}\\ \sum\limits_{j\in V}x_{ijk}-\sum\limits_{j\in V}x_{jik}=0,\forall i\in V,k\in K\\ \sum\limits_{j\in V}x_{0jk}=1,\forall k\in K\\ \sum\limits_{i\in S}\sum\limits_{j\in S}x_{ijk}\le|S|-1,\forall S\subset V-\{0\},S\ne\emptyset,k\in K\\ x_{iik}=0,\forall i\in V,k\in K\\ x_{ijk}\in\{0,1\}\end{cases}$$

在处理TSP问题中的子环问题时，常用的方法有基于Branch and bound的Branch and cut算法、基于Column Generation的Branch and Price算法和Lagrange Relaxation算法。考虑到该问题规模较小，可以得到精确解，故采用Branch and cut算法。

算法的思想是：......（此处省略）

使用Gurobi求解器进行求解。

首先，去除环约束的LIP问题的LP松弛问题的最优解为：

```
Optimize a model with 108 rows, 883 columns and 3446 nonzeros
Model fingerprint: 0x9585e90c
Coefficient statistics:
  Matrix range     [1e+00, 1e+02]
  Objective range  [1e+00, 1e+00]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+00, 1e+00]
Presolve removed 44 rows and 42 columns
Presolve time: 0.01s
Presolved: 64 rows, 841 columns, 3282 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    0.0000000e+00   1.002209e+02   0.000000e+00      0s
     124    1.4044933e+02   0.000000e+00   0.000000e+00      0s

Solved in 124 iterations and 0.01 seconds
```

```
Optimal objective  1.404493333e+02
```

将其设为该TSP问题的下限；

接着，求解去除环约束的LIP问题，得到最优解；

然后，计算图中的Strong connected components（方法见《算法导论》，大致是拓扑排序+inversed graph+DFS）；

最后，将识别出的环生成约束加入到模型中，开始新一轮迭代。

```
Explored 12482 nodes (142033 simplex iterations) in 7.05 seconds
Thread count was 4 (of 4 available processors)

Solution count 7: 591.17 591.17 593.982 ... 830.981

Optimal solution found (tolerance 1.00e-04)
Best objective 5.911699722749e+02, best bound 5.911699722749e+02, gap 0.0000%
```

附录

Python代码：

```
from gurobipy import *
import pickle
import networkx as nx

# define the constants
V_NUM = 21
K_NUM = 2
ORIGIN_IDX = 5
MAX = 1e5

# load and generate basic data
f = open("../data/distance", mode="rb")
distance = pickle.load(f)
f.close()
node_list = list(range(ORIGIN_IDX - 1)) + list(range(ORIGIN_IDX, V_NUM))
dist = {(i, j, k): distance[i][j] for i in range(V_NUM) for j in range(V_NUM) for k in
range(K_NUM)}

```

```python
dict_linear = {(i, j, k): i*V_NUM*K_NUM + j*K_NUM + k for i in range(V_NUM) for j in
range(V_NUM) for k in range(K_NUM)}
dict_3d = mi = dict(zip(dict_linear.values(), dict_linear.keys()))

# create a model
MODEL = Model()
# MODEL.setParam('OutputFlag', 0)

# add variables
x = MODEL.addVars(dist.keys(), obj=dist, vtype=GRB.BINARY, name='x')
d = MODEL.addVar(name="d")
MODEL.update()

# set the objective
MODEL.setObjective(d + quicksum(distance[i][j] / 1.5 * x[i, j, k] for i in
range(V_NUM) for j in range(V_NUM) for k in range(K_NUM)), GRB.MINIMIZE)

# add constraints
MODEL.addConstrs(quicksum(x[i, j, k] for j in range(V_NUM) for k in range(K_NUM)) == 1
for i in node_list)
MODEL.addConstrs(quicksum(x[i, j, k] for j in range(V_NUM)) - quicksum(x[j, i, k] for
j in range(V_NUM)) == 0 for i in range(V_NUM) for k in range(K_NUM))
MODEL.addConstrs(quicksum(x[ORIGIN_IDX - 1, j, k] for j in range(V_NUM)) == 1 for k in
range(K_NUM))
MODEL.addConstrs(d - quicksum(distance[i][j] / 1.5 * x[i, j, k] for i in range(V_NUM)
for j in range(V_NUM)) >= 0 for k in range(K_NUM))
MODEL.addConstrs(x[i, i, k] == 0 for i in range(V_NUM) for k in range(K_NUM))
MODEL.addConstrs(x[i, j, k] + x[j, i, k] <= 1 for i in range(V_NUM) for j in
range(V_NUM) for k in range(K_NUM))



# callback - use lazy constraints to eliminate sub-tours
def mycallback(model, where):
    if where == GRB.Callback.MIPSOL:
        vals = model.cbGetSolution(model._vars[:-1])
        edges = list((i, j, k) for i, j, k in dict_linear.keys() if
vals[dict_linear[(i, j, k)]] > 0.5)
        shortest_cycle, k = cycle(edges)
        if shortest_cycle is not None:
```

```python
            model.cbLazy(quicksum(x[i, j, k] for i in shortest_cycle for j in shortest_cycle) <= len(shortest_cycle) - 1)


# find the loops
def cycle(edges):
    node_vehicle = {}
    G = nx.DiGraph()
    for e in edges:
        G.add_edge(e[0], e[1])
        node_vehicle[e[0]] = e[2]
        node_vehicle[e[1]] = e[2]
    shortest_subtour = None
    min = V_NUM
    cycle: list
    for cycle in nx.simple_cycles(G):
        if cycle.count(ORIGIN_IDX - 1) == 0:
            if len(cycle) < min:
                min = len(cycle)
                shortest_subtour = cycle
    if shortest_subtour is None:
        k = None
    else:
        k = node_vehicle[shortest_subtour[0]]
    return shortest_subtour, k


MODEL._vars = MODEL.getVars()
MODEL.Params.lazyConstraints = 1
MODEL.optimize(mycallback)

for v in MODEL.getVars():
    if round(v.x, 0) == 1:
        print(v)
print(MODEL.getVarByName('d'))
```

Gurobi结果：

```
Gurobi Optimizer version 9.1.2 build v9.1.2rc0 (mac64)
```

```
Thread count: 2 physical cores, 4 logical processors, using up to 4 threads
Optimize a model with 990 rows, 883 columns and 5168 nonzeros
Model fingerprint: 0x9d1c1028
Variable types: 1 continuous, 882 integer (882 binary)
Coefficient statistics:
  Matrix range     [1e+00, 1e+02]
  Objective range  [1e+00, 1e+02]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 1e+00]
Presolve removed 504 rows and 42 columns
Presolve time: 0.01s
Presolved: 486 rows, 841 columns, 4130 nonzeros
Variable types: 1 continuous, 840 integer (840 binary)

Root relaxation: objective 4.213480e+02, 101 iterations, 0.00 seconds

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

     0     0  421.34800    0   46          -  421.34800      -     -    0s
     0     0  491.68100    0   33          -  491.68100      -     -    0s
     0     0  496.11700    0   16          -  496.11700      -     -    0s
     0     0  496.11700    0   22          -  496.11700      -     -    0s
     0     0  496.11700    0   16          -  496.11700      -     -    0s
     0     2  496.11700    0   22          -  496.11700      -     -    0s
*  208   210              75     830.9806667  539.29700  35.1%  11.1    0s
H  268   256                    816.9993333  539.29700  34.0%  12.0    0s
H  319   229                    609.7846487  539.29700  11.6%  11.7    0s
H  349   224                    596.8272752  539.29700  9.64%  11.5    0s
H  366   215                    593.9819873  539.47579  9.18%  11.5    0s
H  793   440                    591.1700000  553.88163  6.31%  12.8    1s
H  946   440                    591.1699946  554.50419  6.20%  14.3    2s
H 2406   431                    591.1699723  578.61373  2.12%  14.0    3s
  6245   870     cutoff   47       591.16997  583.96929  1.22%  12.3    5s

Cutting planes:
  Gomory: 16
  Flow cover: 6
  Inf proof: 1
```

```
  Zero half: 28
  Mod-K: 2
  RLT: 12
  Lazy constraints: 2

Explored 12482 nodes (142033 simplex iterations) in 7.05 seconds
Thread count was 4 (of 4 available processors)

Solution count 7: 591.17 591.17 593.982 ... 830.981

Optimal solution found (tolerance 1.00e-04)
Best objective 5.911699722749e+02, best bound 5.911699722749e+02, gap 0.0000%

User-callback calls 26798, time in user-callback 0.19 sec
<gurobi.Var x[0,1,0] (value 0.9999995803730946)>
<gurobi.Var x[1,4,0] (value 0.9999995803565548)>
<gurobi.Var x[2,3,1] (value 1.0)>
<gurobi.Var x[3,10,1] (value 1.0)>
<gurobi.Var x[4,2,1] (value 0.9999995001976029)>
<gurobi.Var x[4,6,0] (value 0.9999995002405051)>
<gurobi.Var x[5,4,1] (value 0.9999995002630798)>
<gurobi.Var x[6,20,0] (value 0.9999995002374565)>
<gurobi.Var x[7,9,0] (value 0.999999580358502)>
<gurobi.Var x[8,18,0] (value 0.9999995806292076)>
<gurobi.Var x[9,8,0] (value 0.9999995805623223)>
<gurobi.Var x[10,11,1] (value 1.0)>
<gurobi.Var x[11,12,1] (value 1.0)>
<gurobi.Var x[12,13,1] (value 1.0)>
<gurobi.Var x[13,14,1] (value 1.0)>
<gurobi.Var x[14,5,1] (value 1.0)>
<gurobi.Var x[15,16,0] (value 0.9999995002658683)>
<gurobi.Var x[16,17,0] (value 1.0)>
<gurobi.Var x[17,0,0] (value 0.9999995001668063)>
<gurobi.Var x[18,15,0] (value 0.9999995002324273)>
<gurobi.Var x[19,7,0] (value 1.0)>
<gurobi.Var x[20,19,0] (value 0.9999990804497377)>
<gurobi.Var d (value 201.24197185085075)>

Process finished with exit code 0
```