多商品流问题求解
——Dantzig-Wolfe 分解算法的程序实现

封钰震（1951362）

1 问题描述

已知城市路网的结构（即城市中的路段、交叉口等信息）和人们的出行需求（即从起点到终点的流量），每条路段上的容量有限，若不考虑路段上的流量对道路通行时间的影响（即拥堵效应），假设所有的车辆都由一个中央决策者进行调度（即不存在博弈），则如何安排每对起点-终点对（Origin-destination，OD）间的路径选择和每条路径上的流量，使得所有人出行的成本之和最小？

2 模型建立

这是一个多商品流问题，每一个 OD 对代表着一个商品。定义网络 $G(N,A)$，其中 N 为所有点的集合（往往代表着交叉口），A 为所有边的集合（往往代表着连接各个交叉口的路段），每条路段上的通行成本为 t_{ij}、最大通行容量为 c_{ij}，其中 $(i,j)\in A$。给定网络上需求的集合 W，该集合中的每一个元素 $w\in W$ 代表着一个起点 $o\in N$ 和终点 $d\in N$ 对 (o,d)；每一个 OD 对间有需求 λ_w，用户可以选择的路径集合为 R_w。

定义决策变量 $f_{ij}^w,(i,j)\in A,w\in W$ 为 OD 对 w 在边 (i,j) 上的流量。上述问题可以建模为以下模型：

$$\min \quad \sum_{w\in W}\sum_{(i,j)\in A} t_{ij}f_{ij}^w \tag{1}$$

$$\text{s.t.} \quad \sum_{w\in W} f_{ij}^w \leqslant c_{ij}, \forall (i,j)\in A \tag{2}$$

$$\sum_{(o,j)\in A} f_{oj}^w = \lambda_w, \forall w=(o,d)\in W \tag{3}$$

$$\sum_{(i,d)\in A} f_{id}^w = \lambda_w, \forall w=(o,d)\in W \tag{4}$$

$$\sum_{(i,n)\in A} f_{in}^w - \sum_{(n,j)\in A} f_{n,j}^w = 0, \forall n\in N-\{o,d\}, w=(o,d)\in W \tag{5}$$

$$f_{ij}^w \geqslant 0, \forall (i,j)\in A, w\in W \tag{6}$$

其中，目标函数（1）将各边上的通行成本进行累加，约束（2）限制了各边上的流量不能超过上限，约束（3）-（5）是各 OD 对的流量守恒约束，约束（6）规定了决策变量的定义域。

3 模型求解

3.1 初始化

为找到一个初始可行解开始迭代，我们采用大 M 方法，定义 M 为一个足够大的常数，且为每条路段 $(i,j) \in A$ 定义一个松弛变量 s_{ij} 和一个人工变量 a_{ij}。模型变为：

$$\min \quad \sum_{w \in W} \sum_{(i,j) \in A} t_{ij} f_{ij}^{w} + M a_{ij} \tag{7}$$

$$\text{s.t.} \quad \sum_{w \in W} f_{ij}^{w} + s_{ij} - a_{ij} = c_{ij}, \forall (i,j) \in A \tag{8}$$

$$\sum_{(o,j) \in A} f_{oj}^{w} = \lambda_w, \forall w = (o,d) \in W \tag{9}$$

$$\sum_{(i,d) \in A} f_{id}^{w} = \lambda_w, \forall w = (o,d) \in W \tag{10}$$

$$\sum_{(i,n) \in A} f_{in}^{w} - \sum_{(n,j) \in A} f_{n,j}^{w} = 0, \forall n \in N - \{o,d\}, w = (o,d) \in W \tag{11}$$

$$f_{ij}^{w} \geqslant 0, \forall (i,j) \in A, w \in W \tag{12}$$

$$s_{ij}, a_{ij} \geqslant 0, \forall (i,j) \in A \tag{13}$$

其中，目标函数（7）添加了对于人工变量 $a_{ij}, (i,j) \in A$ 的惩罚项；约束（8）通过松弛变量 $s_{ij}, (i,j) \in A$ 变为了等式，同时增加了人工变量 $a_{ij}, (i,j) \in A$；约束（9）-（12）保持不变；约束（13）为松弛变量和人工变量的定义域。

3.2 主问题

设 $y^1, y^2, \ldots, y^R$ 为解空间 $\left\{ f_{ij}^{w}, (i,j) \in A, w \in W | f_{ij}^{k} \text{满足约束（9）-（12）} \right\}$ 的极点，其中 $R = |\times_{w \in W} R_w|$。对于每一个极点 $y^r, r = 1, 2, \ldots, R$，有，

$$f_{ij}^{r} = f_{ij}(y^r) = \sum_{w \in W} f_{ij}^{w}, \forall (i,j) \in A \tag{14}$$

$$t^r = t(y^r) = \sum_{w \in W} \sum_{(i,j) \in A} t_{ij} f_{ij}^{w} \tag{15}$$

定义决策变量 $\mu^r, r = 1, 2, \ldots, R$ 为每个极点的权重，则主问题为：

$$\min \quad \sum_{r=1}^{R} t^r \mu^r + M a_{ij} \tag{16}$$

$$\text{s.t.} \quad \sum_{r=1}^{R} f_{ij}^{r} \mu^r + s_{ij} - a_{ij} = c_{ij}, \forall (i,j) \in A \tag{17}$$

$$\sum_{r=1}^{R} \mu^r = 1 \tag{18}$$

$$0 \leqslant \mu^r \leqslant 1, \forall r = 1, 2, \ldots, R \tag{19}$$

$$s_{ij}, a_{ij} \geqslant 0, \forall (i,j) \in A \tag{20}$$

3.3 子问题

设约束（17）、（18）对应的对偶变量分别为 $\eta_{ij},(i,j)\in A$ 和 α。检验数 σ^r 的计算如下：

$$\begin{aligned}\sigma^r &= \sum_{(i,j)\in A} f_{ij}^r\eta_{ij}+\alpha-t^r \\ &= \sum_{(i,j)\in A}\sum_{w\in W} f_{ij}^w\eta_{ij}+\alpha-\sum_{w\in W}\sum_{(i,j)\in A} t_{ij}f_{ij}^w \\ &= -\sum_{w\in W}\sum_{(i,j)\in A}(t_{ij}-\eta_{ij})f_{ij}^w+\alpha\end{aligned} \tag{21}$$

于是，子问题为：

$$\min \quad \sum_{w\in W}\sum_{(i,j)\in A}(t_{ij}-\eta_{ij})f_{ij}^w \tag{22}$$

$$\text{s.t.} \quad \sum_{(o,j)\in A} f_{oj}^w=\lambda_w,\forall w=(o,d)\in W \tag{23}$$

$$\sum_{(i,d)\in A} f_{id}^w=\lambda_w,\forall w=(o,d)\in W \tag{24}$$

$$\sum_{(i,n)\in A} f_{in}^w-\sum_{(n,j)\in A} f_{n,j}^w=0,\forall n\in N-\{o,d\},w=(o,d)\in W \tag{25}$$

$$f_{ij}^w\geqslant 0,\forall(i,j)\in A,w\in W \tag{26}$$

这相当于对每一个 OD 对 $w\in W$，在边的权重调整为 $t_{ij}-\eta_{ij},(i,j)\in A$ 的图 $G(N,A)$ 中，求从起点到终点的最短路径，并将所有的流量分配给这条最短路径。

由于 $\eta_{ij},(i,j)\in A$ 是受限主问题（Restricted Master Problem, RMP）的对偶变量，因此对于任意的 η_{ij}，这一列的检验数：

$$\eta_{ij}\leqslant 0,\forall(i,j)\in A \tag{27}$$

因此，

$$t_{ij}-\eta_{ij}>0,\forall(i,j)\in A \tag{28}$$

这意味着所有边的权重为正数，于是，对每一个 OD 对 $w\in W$ 求解最短路径使用 Dijkstra 算法即可。

4 数值实验

实验选取如图 1所示的路网，路网数据、需求数据可在仓库https://github.com/bstabler/TransportationNetworks/tree/master/SiouxFalls 中查看。网络共计 24 个节点、76 条边，网络上有 528 个 OD 对。为保证解可行，各条路段的容量在原数据集的基础上扩大至两倍；令 $M=10^6$。

模型使用 GUROBI 10.0 在 Intel(R) Core(TM) i5-5257U CPU @ 2.70GHz 处理器上求解，最大线程数为 4 个，算法共计迭代 453 次（即生成 453 列），花费 12.17 秒（不包括加载网络的时间）。其中，每次迭代后得到的目标函数如图 2所示，图分为两部分，左半部分为第 1 至 199 次迭代的变化情况，右半部分为第 200 至 453 次迭代的变化情况。

达到最优时，路网中各路段上的流量 $\sum_{w\in W} f_{ij}^w,(i,j)\in A$ 分布情况如表 1所示。交通管理部门可以根据路段上的流量情况合理规划道路建设，判断是否需要采取扩建措施。

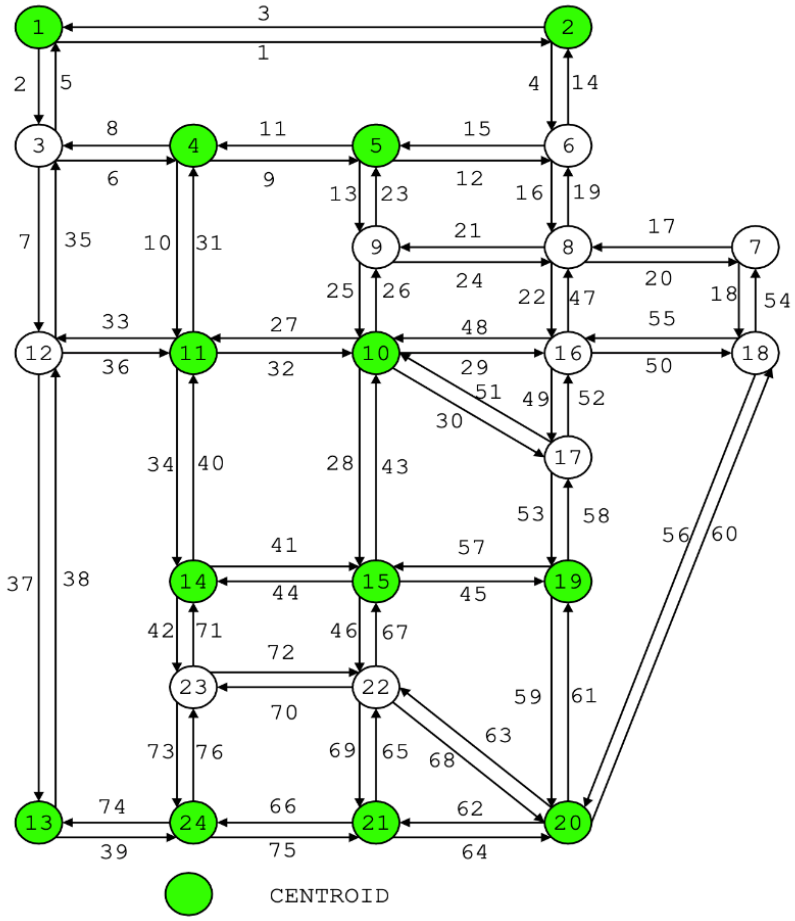

图 1: 路网拓扑结构

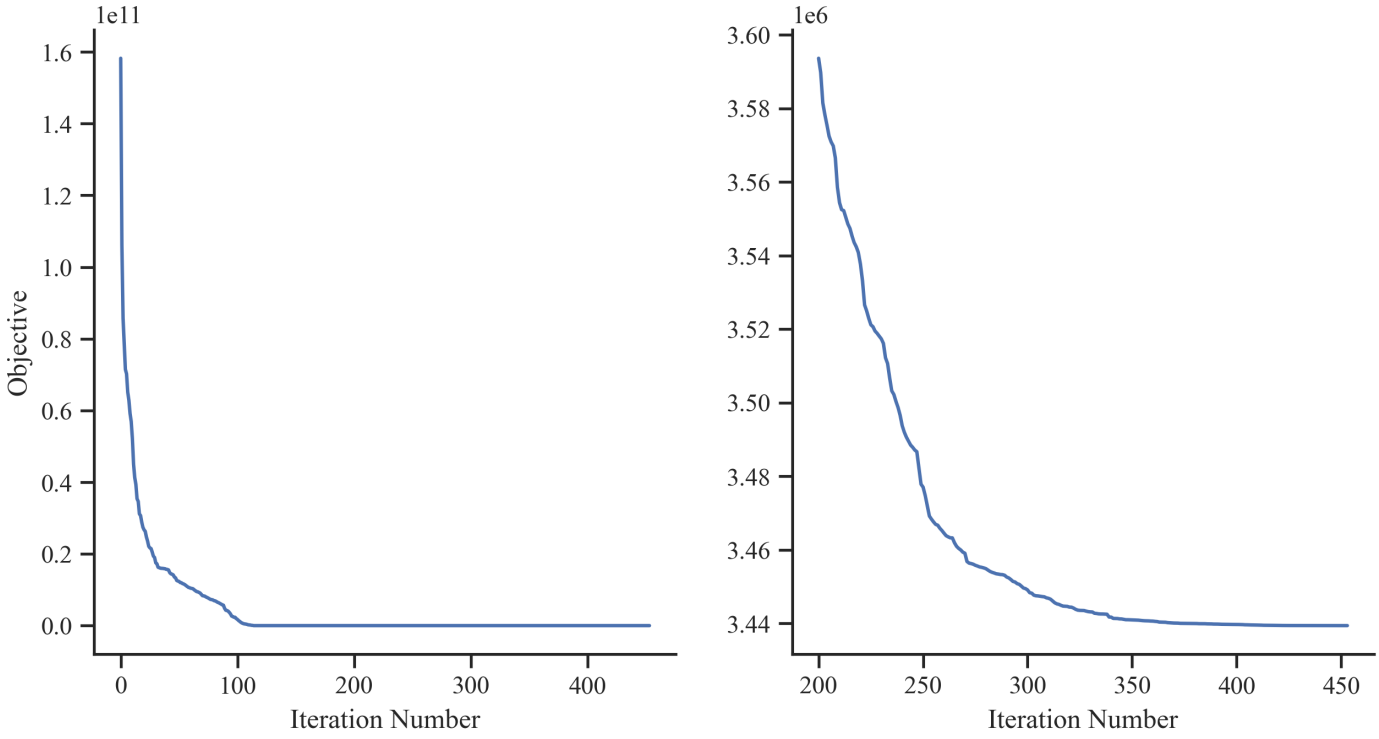

图 2: 目标函数变化情况

路段编号	起点	终点	流量	容量	路段编号	起点	终点	流量	容量
1	1	2	3100	51800.4	39	13	24	10182.5	10182.5
2	1	3	6700	46806.9	40	14	11	9753	9753
3	2	1	3100	51800.4	41	14	15	9345.3	10255.1
4	2	6	5900	9916.4	42	14	23	9849.6	9849.6
5	3	1	6700	46806.9	43	15	10	26987	27024
6	3	4	12199.8	34221	44	15	14	9345.3	10255.1
7	3	12	7100.2	46806.9	45	15	19	21253.1	29129.5
8	4	3	12199.8	34221	46	15	22	19198.4	19198.4
9	4	5	17899.8	35565.6	47	16	8	6108.4	10091.6
10	4	11	5500	9817.7	48	16	10	9709.8	9709.8
11	5	4	17899.8	35565.6	49	16	17	10459.8	10459.8
12	5	6	9896	9896	50	16	18	11441.2	39359.8
13	5	9	17302.7	20000	51	17	10	8699.4	9987
14	6	2	5900	9916.4	52	17	16	10459.8	10459.8
15	6	5	9896	9896	53	17	19	9647.9	9647.9
16	6	8	9797.2	9797.2	54	18	7	11459.8	46806.9
17	7	8	12359.8	15683.6	55	18	16	11549.4	39359.8
18	7	18	11468	46806.9	56	18	20	17002.5	46806.9
19	8	6	9797.2	9797.2	57	19	15	21253.1	29129.5
20	8	7	12368	15683.6	58	19	17	9647.9	9647.9
21	8	9	7790.5	10100.4	59	19	20	10005.2	10005.2
22	8	16	6000.2	10091.6	60	20	18	17002.5	46806.9
23	9	5	17302.7	20000	61	20	19	10005.2	10005.2
24	9	8	7690.5	10100.4	62	20	21	6145	10119.8
25	9	10	27831.6	27831.6	63	20	22	7004.9	10151.4
26	10	9	27831.6	27831.6	64	21	20	6039.2	10119.8
27	10	11	18884.2	20000	65	21	22	9733.4	10459.8
28	10	15	26887	27024	66	21	24	9770.7	9770.7
29	10	16	9709.8	9709.8	67	22	15	19198.4	19198.4
30	10	17	8699.4	9987	68	22	20	7010.8	10151.4
31	11	4	5600	9817.7	69	22	21	9627.6	10459.8
32	11	10	18684.2	20000	70	22	23	10000	10000
33	11	12	9817.7	9817.7	71	23	14	9849.6	9849.6
34	11	14	9753	9753	72	23	22	9900	10000
35	12	3	7100.2	46806.9	73	23	24	7509.9	10157
36	12	11	9817.7	9817.7	74	24	13	10182.5	10182.5
37	12	13	11211	51800.4	75	24	21	9770.7	9770.7
38	13	12	11311	51800.4	76	24	23	7409.9	10157

表 1: 各路段上的流量 $\sum_{w\in W} f_{ij}^{w}, (i,j) \in A$ 分布情况

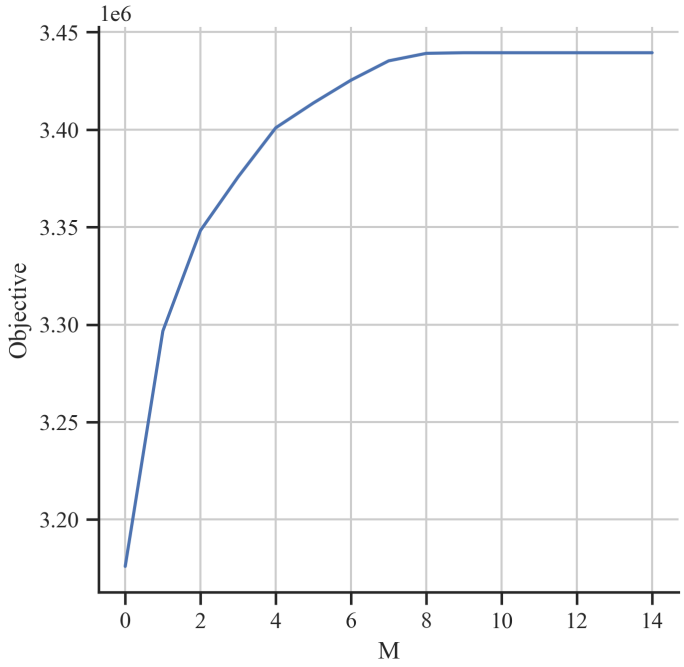

图 3: 目标函数与价格参数 M 的关系

假设交通管理部门可以以一定的价格扩建道路，价格为 M 每单位流量，此时模型不需要更改，目标函数（即总通行成本）与价格参数 M 的关系如图 3所示。当 $M = 0$ 时，即忽略路段容量限制、所有 OD 对都采用最短路径时的总通行成本。随着价格的上涨，起初，当 $1 \leqslant M \leqslant 9$ 时，扩建道路可以带来成本的减少；但是，当 $M > 9$ 后，扩建道路的成本太高，并不能带来成本的减少。此时，对偶价格 $|\eta_{ij}|$ 的最大值为 9。

A 程序说明

本节将说明程序中的几个关键函数。完整程序可见 GitHub：https://github.com/Feng-Yz/Multi-Commodity-Network-Flow。

下面的函数为 Dantzig-Wolfe 分解算法的核心。

```
def dwDecomposition(self, M=1e6, epsilon=1e-6, output=0):
    # 极点的列表（对应DW算法中的列）
    self.solutions = []
    # 使用最短路算法找到一个极点（对应一列）
    solution = self.generatePathForDemand()
    self.solutions.append(solution)
    # 找到有容量上界的边
    self.boundedEdges = {}
    for edgeId, edge in self.edges.items():
        edge: Edge
        if edge.capacity < np.inf:
            self.boundedEdges[edgeId] = edge
    # 初始化模型
    masterProblem = gp.Model()
    masterProblem.Params.LogToConsole = 0
    masterProblem.Params.LogFile = "./log.txt"
    # 添加变量，包括lambda、松弛变量和人工变量
    lams = masterProblem.addVars(len(self.solutions), lb=0, ub=1, obj=[s["cost"] for s in self.
        solutions], name="lam[0]")
    slacks = masterProblem.addVars(len(self.boundedEdges), lb=0, obj=0, name='s') # 松弛变量
    surpluses = masterProblem.addVars(len(self.boundedEdges), lb=0, obj=M, name='a') # 人工变量
    # 添加约束，包括容量约束和sum(lambda)=1的约束
    masterProblem.addConstrs((gp.quicksum(self.solutions[i]["flow"][k] * lams[i] for i in range(
        len(lams))) + \
        slacks[j] - surpluses[j] == self.boundedEdges[k].capacity for j, k in enumerate(self.
            boundedEdges.keys())), name="capacity")
    masterProblem.addConstr(gp.quicksum(lams[i] for i in range(len(lams))) == 1)
    # 记录开始时间
    startTime = time.time()
    # 求解受限主问题RMP
    masterProblem.optimize()
```

```
    # 得到对偶变量的值
    dualVars = masterProblem.getAttr(gp.GRB.Attr.Pi, masterProblem.getConstrs())
    # 迭代开始，迭代次数设置为0
    iterNum = 0
    # 根据对偶变量的值求解子问题（SP），得到检验数，同时新的极点会添加到极点列表里
    reducedCost = self.subproblem(dualVars)
    # 输出
    if output: print("{:.0f}\t\t\t{:.2f}\t\t\t{:.2f}".format(iterNum, masterProblem.getObjective
        ().getValue(), reducedCost))
    while reducedCost > epsilon and iterNum < 2000:
        # 取最新添加的极点
        s = self.solutions[-1]
        # 计算该极点对应的新列的系数
        colCoeff = [s["flow"][k] for k in self.boundedEdges.keys()]
        colCoeff.append(1) # 别忘了lambda对应的系数1
        # 生成新列添加进模型
        column = gp.Column(colCoeff, masterProblem.getConstrs())
        masterProblem.addVar(lb=0, ub=1, obj=s["cost"], name="lam["+str(iterNum+1)+']', column=
            column)
        # 求解新的受限主问题RMP
        masterProblem.optimize()
        # 得到对偶变量的值
        dualVars = masterProblem.getAttr(gp.GRB.Attr.Pi, masterProblem.getConstrs())
        # 迭代次数加1
        iterNum += 1
        # 根据对偶变量的值求解子问题（SP），得到检验数，同时新的极点会添加到极点列表里
        reducedCost = self.subproblem(dualVars)
        # 输出
        if output: print("{:.0f}\t\t\t{:.2f}\t\t\t{:.2f}".format(iterNum, masterProblem.
            getObjective().getValue(), reducedCost))
    # 记录结束时间
    endTime = time.time()
    # 模型保存和输出
    self.objModel = masterProblem
    if output: print("Iteration time: {:.2f}s".format(endTime - startTime))
    for v in self.objModel.getVars():
        if v.X != 0:
            varName = v.VarName
            varValue = v.X
            if output: print(varName + '\t' + str(varValue))
    # 得到各OD的路径和流量信息
    self.retrieveSol(output)
```

以下是求解子问题的函数。

```
def generatePathForDemand(self):
    # 储存该极点的信息
    solution = {"routes": {}}
    # 总成本（通行成本）
    totalCost = 0
    # 检验数
    reducedCost = 0
    # 对于每一个OD对，计算在调整权重后的路网上的最短路径，并分配流量
    for demandId, demand in self.demands.items():
        demand: Demand
        # 得到最短路径
        sp, _ = self.dijkstra(demand.o, demand.d)
        # 极点信息——路径
        solution["routes"][demandId] = sp
        # 根据路径计算总成本和检验数
        lastNode = demand.o
        for node in sp[1: ]:
            node: Node
            # 根据路径列表中的点找到对应的边
            edgeId = self.edgeDict[(lastNode.id, node.id)]
            edge: Edge = self.edges[edgeId]
            # 改变边上的流量
            edge.useCapacity(demand.quantity)
            # 总成本
            totalCost += demand.quantity * edge.initWeight
            # 检验数
            reducedCost += demand.quantity * edge.weight
            lastNode = node
```

```
    # 极点信息
    solution["cost"] = totalCost # 总成本
    solution["reducedCost"] = reducedCost # 检验数（只有这一次迭代有用）
    solution["flow"] = {} # 各边上的流量
    # 计算各边上的流量
    for edgeId, edge in self.edges.items():
        edge: Edge
        solution["flow"][edgeId] = edge.capacityUsed
    # 返回新极点（及其信息）
    return solution

def subproblem(self, dualVars: list):
    # 调整路网上各边的权重
    i = 0
    for edgeId, edge in self.boundedEdges.items():
        edge: Edge
        edge.weight = edge.initWeight - dualVars[i]
        i += 1
    # 重置图上各边的流量
    self.resetEdgeCapacity()
    # 得到一个新极点（对应一列）
    solution = self.generatePathForDemand()
    # 计算检验数
    reducedCost = -solution["reducedCost"] + dualVars[i]
    # 如果检验数为正，就将该极点添加到极点列表中
    if reducedCost > 0:
        self.solutions.append(solution)
    # 返回检验数
    return reducedCost
```