

16.4.1 两阶段法实现 Dantzig-Wolfe 分解算法介绍

首先将问题初始化，一般情况下我们需要用启发式算法生成一些初始的列。但是如果我们没有初始的列怎么办呢？文献（kalvelagen, 2003）提供了一种用两阶段法求解的思路。两阶段法是线性规划求解中为人熟知的方法。

由于有凸组合约束和中心约束，因此我们刚开始添加的列可能会违反中心约束。因此我们可以在第1阶段引入人工变量，目标函数是最小化人工变量的和。引入人工变量的目的是判断该问题是否有可行解，如果第1阶段最小化人工变量的和的目标函数到达了0，就说明该问题有解。我们就可以删去人工变量，进行第2阶段的算法，最终得到原问题的最优解。

本问题中，中心约束为

$$\sum_{i \in I} \alpha_i \mu_i + \sum_{j \in J} \beta_j \lambda_j \leq b \quad (16.13)$$

的形式，我们可以引入一个单独的人工变量 $x_a \geq 0$ ，并且将约束改写为

$$\sum_{i \in I} \alpha_i \mu_i + \sum_{j \in J} \beta_j \lambda_j - x_a \leq b \quad (16.14)$$

设置第1阶段的目标函数为

$$\min x_a \quad (16.15)$$

此时，子问题产生的变量在目标函数中的系数全部为0。当第1阶段的目标函数（16.15）为0时，我们开启第2阶段的算法。将人工变量的值固定为0，继续迭代算法即可。

16.4.2 第1阶段

初始化问题的时候模型是空列。我们添加人工变量 s ，将主问题初始化为

$$\max z = s \quad (16.16)$$

$$\text{s.t. } -s \leq 80 \rightarrow \pi_1 \quad (16.17)$$

$$\text{Null} = \text{Null} \rightarrow \pi_2 \quad (16.18)$$

$$\text{Null} = \text{Null} \rightarrow \pi_3 \quad (16.19)$$

$$s \geq 0 \quad (16.20)$$

$$(16.21)$$

1. Iteration 0

模型（16.16）中，人工变量 s 就是为了算法能够进行下去才引入的，2个凸组合的约束也需要初始化。我们先将其初始化为 Null。

相应的初始化代码如下。

DWInstancePhase1Step0

```
RMP = Model('DW Master Problem')
# RMP.setParam("OutputFlag", 0)

#----- start initialize RMP-----
# initialize column : Null column
row_num = 3

# because vars in RMP is added into RMP dynamically, thus we creat an array
# to store the variable set
rmp_vars = []
rmp_vars.append(RMP.addVar(lb=0.0
                           , ub=1000
                           , obj=1
                           , vtype=GRB.CONTINUOUS # GRB.BINARY, GRB.INTEGER
                           , name='s_1'
                           , column=None
                           ))
# rmp_cons = []
var_names = []
col = [-1, 0.001, 0.001]
var_names.append('s_0')

# column to add constraints into RMP
rmp_cons.append(RMP.addConstr(lhs = rmp_vars[0] * col[0]
                               , sense= "<="
                               , rhs= 80
                               , name='rmp_con_central'
                               )
                )
# rmp_cons.append(RMP.addConstr(lhs = rmp_vars[0] * col[1]
#                               , sense= "=="
#                               , rhs= 1
#                               , name='rmp_con_convex_combine_mu'
#                               )
#                 )
```

```

41 rmp_cons.append(RMP.addConstr(lhs = rmp_vars[0] * col[2]
42     , sense= "<="
43     , rhs= 1
44     , name='rmp_con_convex_combine_lam'
45     )
46
47
48
49 RMP.setAttr("ModelSense", GRB.MAXIMIZE)
50 #----- RMP initialize end -----
51 RMP.chgCoeff(rmp_cons[1], rmp_vars[0], 0.0)
52 RMP.chgCoeff(rmp_cons[2], rmp_vars[0], 0.0)
53 rmp_cons[1].setAttr('RHS', 0)
54 rmp_cons[2].setAttr('RHS', 0)
55
56 RMP.write('DW.lp')
57 RMP.optimize()
58 rmp_dual = RMP.getAttr("Pi", RMP.getConstrs())
59 print('rmp_dual = ', rmp_dual)

```

运行上述模型，得到对偶变量如下。

DWInstancePhase1Step0Output

```

Optimal objective 1.000000000e+03
rmp_dual = [0.0, 0.0, 0.0]

```

对偶变量是 $\pi^T = [0, 0, 0]$ 。这里有一个细节。我们得到的对偶变量全为 0，这样就会使得子问题的目标函数为 0。为了刚开始能够产生新列，我们需要将两个 Null 的凸组合约束 (16.18) 和 (16.19) 的对偶变量 π_2, π_3 初始化为一个非零的数。这里我们初始化为 $\pi_2 = \pi_3 = -1$ 。因此对偶变量为 $\pi^T = [0, -1, -1]$ 。

接下来我们需要构建子问题。子问题是获得检验数为正的列。但是在该阶段，我们的目标函数只能是 $\max s$ ，因此所有的即将新加入的列在目标函数的系数全为 0。

对于子问题 1 产生的列，检验数 (Reduced cost) 的表达式就可以写为

$$\begin{aligned} c_j - c_B^T B^{-1} N_j &= 0 - c_B^T B^{-1} \cdot \begin{bmatrix} 8x_1^i + 6x_2^i \\ 1 \\ 0 \end{bmatrix} \\ &= 0 - \pi^T \cdot \begin{bmatrix} 8x_1^i + 6x_2^i \\ 1 \\ 0 \end{bmatrix} \end{aligned}$$

$$\begin{aligned} &= 0 - \begin{bmatrix} 0 & -1 & -1 \end{bmatrix} \begin{bmatrix} 8x_1^i + 6x_2^i \\ 1 \\ 0 \end{bmatrix} \\ &= 1 \end{aligned}$$

因此，子问题 1 的模型可以写成

$$\begin{aligned} \max \quad & 1 \\ \text{s.t.} \quad & 3x_1^i + x_2^i \leq 12 \\ & 2x_1^i + x_2^i \leq 10 \\ & x_1^i, x_2^i \geq 0 \end{aligned}$$

求解上述模型的代码如下。

DWInstancePhase1Step0SP1

```

SP1 = Model('SubProblem 1')
x = {}
for i in range(2):
    x[i+1] = SP1.addVar(lb=0, ub=GRB.INFINITY, vtype=GRB.CONTINUOUS, name=
        'x' + str(i+1))

SP1.setObjective(0 * x[1] + 0 * x[2] + 1, GRB.MAXIMIZE)
# SP1.setObjective(90 * x[1] + 80 * x[2], GRB.MAXIMIZE)
SP1.addConstr(3 * x[1] + x[2] <= 12)
SP1.addConstr(2 * x[1] + x[2] <= 10)
SP1.optimize()
print('Objective = \t', SP1.ObjVal)
for var in SP1.getVars():
    print(var.varName, '= \t', var.x)

[Out]:
Objective =      1.0
x1 =      0.0
x2 =      0.0

```

目标函数 $1 > 0$ ，因此有新列加入。由于此时是第 1 阶段，因此目标函数的系数为 0。然后我们构造列

$$\begin{bmatrix} 0 \\ 8x_1^i + 6x_2^i \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 8 \times 0 + 6 \times 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} \text{obj} \\ \text{central constraint} \\ \text{convex constraint 1} \\ \text{convex constraint 2} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

此时主问题变化为

$$\max z = s \quad (16.22)$$

$$-s \leq 80 \quad (16.23)$$

$$\mu_1 = 1 \quad (16.24)$$

$$\text{Null} = \text{Null} \quad (16.25)$$

$$\mu_1, s \geq 0 \quad (16.26)$$

然后我们求解子问题 2。类似地，子问题 2 的定价问题可以写成（假设可行域极点的集合为 J ）。

$$\begin{aligned} \max & 1 \\ \text{s.t.} & 3x_3^j + 2x_4^j \leq 15 \\ & x_3^j + x_4^j \leq 4 \\ & x_3^j, x_4^j \geq 0 \end{aligned}$$

求解该问题的代码如下。

DWInstancePhase1Step0SP2

```

1 SP2 = Model('SubProblem 2')
2 xx = {}
3 for i in range(2):
4     xx[i+1] = SP2.addVar(lb=0, ub=GRB.INFINITY, vtype=GRB.CONTINUOUS, name='xx' + str(i+1))
5
6 SP2.setObjective(0 * xx[1] + 0 * xx[2] + 1, GRB.MAXIMIZE)
7 SP2.addConstr(3 * xx[1] + 2 * xx[2] <= 15)
8 SP2.addConstr(xx[1] + xx[2] <= 4)
9 SP2.optimize()
10 print('Objective = \t', SP2.ObjVal)
11 for var in SP2.getVars():
12     print(var.varName, '= \t', var.x)
13
14 [Out]:
15 Objective = 1.0
16 xx1 = 0.0
17 xx2 = 0.0

```

目标函数 $1 > 0$ ，因此有新列加入。同样地，由于此时是第一阶段，因此目标函数的系

数为 0，然后我们构造列

$$\begin{bmatrix} 0 \\ 7x_3^j + 5x_4^j \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 7 \times 0 + 5 \times 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \text{obj} \\ \text{central constraint} \\ \text{convex constraint 1} \\ \text{convex constraint 2} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

此时主问题变化为

$$\max z = s \quad (16.27)$$

$$-s \leq 80 \quad (16.28)$$

$$\mu_1 = 1 \quad (16.29)$$

$$\lambda_1 = 1 \quad (16.30)$$

$$\mu_1, \lambda_1, s \geq 0 \quad (16.31)$$

2. Iteration 1

我们求解更新后的主问题代码如下。

DWInstancePhase1Step1RMP

```

new_columns = [[0, 0, 1, 0]
               ,[0, 0, 0, 1]
               ]
for i in range(2):
    rmp_col = Column(new_columns[i][1:], rmp_cons)
    rmp_vars.append(RMP.addVar(lb=0.0
                                , ub=GRB.INFINITY
                                , obj=new_columns[i][0] # obj=new_obj_coef
                                , vtype=GRB.CONTINUOUS
                                , name='y' + str(i+1) # name='y\'' + str(len(rmp\_
                                vars)))
                                , column=rmp_col
                                ))
# RMP.remove(rmp_vars[0])
# del rmp_vars[0]
RMP.update()
RMP.write('DW.lp')
RMP.optimize()
rmp_dual = RMP.getAttr("Pi", RMP.getConstrs())
print('rmp_dual = ', rmp_dual)
print('Objective = \t', RMP.ObjVal)
for var in RMP.getVars():

```

```

22     print(var.varName, ' = \t', var.x)
23
24 [Out]:
25 rmp_dual = [0.0, 0.0, 0.0]
26 Objective = 1000.0
27 s_1 = 1000.0
28 y1 = 0.0
29 y2 = 0.0

```

新的对偶变量为 $\pi^T = [0, 0, 0]$, 目标函数为 1000, 因此还需要继续进行第 1 阶段的迭代。此时两个凸组合约束已经非空, 此时我们就使用真实的对偶变量的值即可。

此时根据对偶变量更新子问题 1 的目标函数。子问题 1 的目标函数更新为

$$\begin{aligned}
c_j - c_B^T B^{-1} N_j &= 0 - c_B^T B^{-1} \cdot \begin{bmatrix} 8x_1^i + 6x_2^i \\ 1 \\ 0 \end{bmatrix} \\
&= 0 - \pi^T \cdot \begin{bmatrix} 8x_1^i + 6x_2^i \\ 1 \\ 0 \end{bmatrix} \\
&= 0 - [0 \ 0 \ 0] \begin{bmatrix} 8x_1^i + 6x_2^i \\ 1 \\ 0 \end{bmatrix} \\
&= 0
\end{aligned}$$

因此子问题变化为

$$\begin{aligned}
\max \quad &0 \\
\text{s.t.} \quad &3x_1^i + x_2^i \leq 12 \\
&2x_1^i + x_2^i \leq 10 \\
&x_1^i, x_2^i \geq 0
\end{aligned}$$

目标函数为 0, 因此没有新列加入。

对于子问题 2, 目标函数也为 0。两个子问题都没有新的列产生。因此, 第 1 阶段结束。然后将主问题中的人工变量删除, 或者将其设置为 0。

16.4.3 第 2 阶段

首先我们设置第 1 阶段结束时的主问题中的人工变量为 0, 主问题变化为

$$\max z = s \quad (16.32)$$

$$-s \leq 80 \quad (16.33)$$

$$\mu_1 = 1 \quad (16.34)$$

$$\lambda_1 = 1 \quad (16.35)$$

$$\mu_1, \lambda_1, s \geq 0 \quad (16.36)$$

$$s = 0$$

首先求解该问题, 代码如下。

DWInstancePhase2Step1RMP

```

new_columns = [[0, 0, 1, 0]
               ,[0, 0, 0, 1]
               ]
for i in range(2):
    rmp_col = Column(new_columns[i][1:], rmp_cons)
    rmp_vars.append(RMP.addVar(lb=0.0
                                , ub=GRB.INFINITY
                                , obj=new_columns[i][0] # obj=new_obj_coef
                                , vtype=GRB.CONTINUOUS
                                , name='y'+ str(i+1) #name='y_ '+ str(len(rmp_vars))
                                , column=rmp_col
                                ))
# RMP.remove(rmp_vars[0])
# del rmp_vars[0]
rmp_cons[1].setAttr('RHS', 1)
rmp_cons[2].setAttr('RHS', 1)
RMP.update()
RMP.write('DW.lp')
RMP.optimize()
rmp_dual = RMP.getAttr("Pi", RMP.getConstrs())
print('rmp_dual = ', rmp_dual)
print('Objective = \t', RMP.ObjVal)
for var in RMP.getVars():
    print(var.varName, ' = \t', var.x)

[Out]:
rmp_dual = [0.0, 0.0, 0.0]
Objective = -0.0
s_1 = 0.0
y1 = 1.0
y2 = 1.0

```

新的对偶变量为 $\pi^T = [0, 0, 0]$ 。

第2阶段的子问题形式有所变化。假设工厂1对应的子问题1的约束的可行域的极点(2维)有 I 个, 其中每一个极点 $i \in I$ 的坐标为 (x_1^i, x_2^i) , 我们知道, 子问题1的可行域内的任何一点都是由极点集合 I 的坐标线性组合而成的, 那么最优解 (x_1^*, x_2^*) 同样也是由极点集合 I 中的所有点的线性组合表示的。继续上面的过程, 用 μ_i 表示子问题1中的第 i 个极点在最优解中贡献的比例。则子问题1对应的比例 μ_i 对(16.32)的目标函数系数的贡献为

$$90x_1^i + 80x_2^i = \text{对应决策变量在目标函数中的系数}$$

且对中心约束的约束系数贡献为

$$8x_1^i + 6x_2^i = \text{对应决策变量在中央约束中的约束系数}$$

因此, 检验数(Reduced cost)的表达式就可以写为

$$\begin{aligned} c_j - c_B^T B^{-1} N_j &= (90x_1^i + 80x_2^i) - c_B^T B^{-1} \cdot \begin{bmatrix} 8x_1^i + 6x_2^i \\ 1 \\ 0 \end{bmatrix} \\ &= (90x_1^i + 80x_2^i) - \pi^T \cdot \begin{bmatrix} 8x_1^i + 6x_2^i \\ 1 \\ 0 \end{bmatrix} \end{aligned}$$

因此, 子问题1的模型可以写成

$$\begin{aligned} \max \quad & (90x_1^i + 80x_2^i) - \pi^T \cdot [8x_1^i + 6x_2^i, 1, 0]^T \\ \text{s.t.} \quad & 3x_1^i + x_2^i \leq 12 \\ & 2x_1^i + x_2^i \leq 10 \\ & x_1^i, x_2^i \geq 0 \end{aligned}$$

其中, π 是主问题约束的对偶变量, $\pi^T = [\pi_1, \pi_2, \pi_3]$ 。同理, 子问题2的定价问题可以写成(假设可行域极点的集合为 J):

$$\begin{aligned} \max \quad & (70x_3^j + 60x_4^j) - \pi^T \cdot [7x_3^j + 5x_4^j, 0, 1]^T \\ \text{s.t.} \quad & 3x_3^j + 2x_4^j \leq 15 \\ & x_3^j + x_4^j \leq 4 \\ & x_3^j, x_4^j \geq 0 \end{aligned}$$

1. Iteration 1

新的对偶变量为 $\pi^T = [0, 0, 0]$ 。我们根据对偶变量更新子问题1。子问题1更新为

$$\begin{aligned} \max \quad & (90x_1^i + 80x_2^i) - [0, 0, 0] \cdot [8x_1^i + 6x_2^i, 1, 0]^T \\ & = 90x_1^i + 80x_2^i \\ \text{s.t.} \quad & 3x_1^i + x_2^i \leq 12 \\ & 2x_1^i + x_2^i \leq 10 \\ & x_1^i, x_2^i \geq 0 \end{aligned}$$

用Gurobi求解该子问题, 代码及结果如下。

DWInstancePhase2Step1SP1

```
SP1 = Model('SubProblem 1')
x = []
for i in range(2):
    x[i+1] = SP1.addVar(lb=0, ub=GRB.INFINITY, vtype=GRB.CONTINUOUS, name='x' + str(i+1))

SP1.setObjective(90 * x[1] + 80 * x[2], GRB.MAXIMIZE)
SP1.addConstr(3 * x[1] + x[2] <= 12)
SP1.addConstr(2 * x[1] + x[2] <= 10)
SP1.optimize()
print('Objective = \t', SP1.ObjVal)
for var in SP1.getVars():
    print(var.varName, ' = \t', var.x)

[Out]:
Objective =      800.0
x1 =      0.0
x2 =     10.0
```

子问题目标函数 $800 > 0$, 因此可以产生新列。新列的目标函数系数为 $90x_1^i + 80x_2^i = 80 \times 10 = 800$ 。然后我们构造列

$$\begin{bmatrix} 90x_1^i + 80x_2^i \\ 8x_1^i + 6x_2^i \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 800 \\ 60 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} \text{obj} \\ \text{central constraint} \\ \text{convex constraint 1} \\ \text{convex constraint 2} \end{bmatrix}$$

所以, 主问题中就可以将该列加入, 如下:

$$\max z = 800\mu_2 + s \quad (16.37)$$

$$60\mu_2 - s \leq 80 \quad (16.38)$$

$$\mu_1 + \mu_2 = 1 \quad (16.39)$$

$$\lambda_1 = 1 \quad (16.40)$$

$$\mu_1, \lambda_1, s \geq 0 \quad (16.41)$$

$$s = 0$$

然后接着求解如下子问题 2:

$$\begin{aligned} \max \quad & (70x_3^j + 60x_4^j) - [0, 0, 0] \cdot [7x_3^j + 5x_4^j, 0, 1]^T \\ \text{s.t.} \quad & 3x_3^j + 2x_4^j \leq 15 \\ & x_3^j + x_4^j \leq 4 \\ & x_3^j, x_4^j \geq 0 \end{aligned}$$

用 Gurobi 求解该子问题，代码及结果如下：

```
DWInstancePhase2Step1SP2

1 SP2 = Model('SubProblem 2')
2 xx = []
3 for i in range(2):
4     xx[i+1] = SP2.addVar(lb=0, ub=GRB.INFINITY, vtype=GRB.CONTINUOUS, name='xx' + str(i+1))
5
6 SP2.setObjective(70 * xx[1] + 60 * xx[2], GRB.MAXIMIZE)
7 SP2.addConstr(3 * xx[1] + 2 * xx[2] <= 15)
8 SP2.addConstr(xx[1] + xx[2] <= 4)
9 SP2.optimize()
10 print('Objective = ', SP2.ObjVal)
11 for var in SP2.getVars():
12     print(var.varName, '=', var.x)
13
14 [Out]:
15 Objective = 280.0
16 xx1 = 4.0
17 xx2 = 0.0
```

子问题目标函数 $280 > 0$ ，因此可以产生新列。新列的目标函数系数为 $70x_3^j + 60x_4^j =$

$70 \times 4 = 280$ 。然后我们构造列

$$\begin{bmatrix} 70x_3^j + 60x_4^j \\ 7x_3^j + 5x_4^j \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 280 \\ 28 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \text{obj} \\ \text{central constraint} \\ \text{convex constraint 1} \\ \text{convex constraint 2} \end{bmatrix}$$

所以，主问题中就可以将该列加入。主问题模型变化为

$$\max z = 800\mu_2 + 280\lambda_2 + s \quad (16.42)$$

$$60\mu_2 + 28\lambda_2 - s \leq 80 \quad (16.43)$$

$$\mu_1 + \mu_2 = 1 \quad (16.44)$$

$$\lambda_1 + \lambda_2 = 1 \quad (16.45)$$

$$\mu_1, \mu_2, \lambda_1, \lambda_2, s \geq 0 \quad (16.46)$$

$$s = 0$$

求解更新后的主问题的代码如下：

PythonGurobiDWExampleStep2RMP

```
new_columns = [[800, 60, 1, 0]
               ,[280, 28, 0, 1]
               ]
for i in range(2):
    rmp_col = Column(new_columns[i][1:], rmp_cons)
    rmp_vars.append(RMP.addVar(lb=0.0
                                , ub=1
                                , obj=new_columns[i][0] # obj=new_obj_coef
                                , vtype=GRB.CONTINUOUS
                                , name='y' #name='y' + str(len(rmp_vars))
                                , column=rmp_col
                                ))
RMP.remove(rmp_vars[0]) # remove slack variable
del rmp_vars[0]
RMP.update() # update model
RMP.write('DW.lp')
RMP.optimize()
rmp_dual = RMP.getAttr("Pi", RMP.getConstrs())
print('rmp_dual = ', rmp_dual)

[Out]:
rmp_dual = [10.0, 200.0, 0.0]
```

```

23 Objective =      1000.0
24 s_1 =      0.0
25 y1 =      0.0
26 y2 =  0.2857142857142858
27 y5 =      1.0
28 y6 =  0.7142857142857143

```

新的对偶变量为 $\pi^T = [10, 200, 0]$ 。

2. Iteration 2

继续更新子问题 1，首先，目标函数更新为

$$(90x_1^i + 80x_2^i) - \begin{bmatrix} 10, & 200, & 0 \end{bmatrix} \cdot \begin{bmatrix} 8x_1^i + 6x_2^i, & 1, & 0 \end{bmatrix}^T = 90x_1^i + 80x_2^i - 80x_1^i - 60x_2^i - 200 \\ = 10x_1^i + 20x_2^i - 200$$

子问题 1 更新为

$$\begin{aligned} \max \quad & 10x_1^i + 20x_2^i - 200 \\ \text{s.t.} \quad & 3x_1^i + x_2^i \leq 12 \\ & 2x_1^i + x_2^i \leq 10 \\ & x_1^i, x_2^i \geq 0 \end{aligned}$$

求解该子问题的代码及结果如下：

DWInstancePhase2Step2SP1

```

1 SP1 = Model('SubProblem 1')
2 x = {}
3 for i in range(2):
4     x[i+1] = SP1.addVar(lb=0, ub=GRB.INFINITY, vtype=GRB.CONTINUOUS, name=
        'x' + str(i+1))
5
6 SP1.setObjective(10 * x[1] + 20 * x[2] - 200, GRB.MAXIMIZE)
7 SP1.addConstr(3 * x[1] + x[2] <= 12)
8 SP1.addConstr(2 * x[1] + x[2] <= 10)
9 SP1.optimize()
10 print('Objective = \t', SP1.ObjVal)
11 for var in SP1.getVars():
12     print(var.varName, '= \t', var.x)
13
14 [Out]:
15 Objective =      40.0
16 x1 =      0.0
17 x2 =      4.0

```

因为目标函数为 0，也就是检验数为 0，所以不会产生新列。之后的迭代中，可以直接跳过子问题 1。

接着求解子问题 2。子问题 2 的目标函数更新为

$$(70x_3^j + 60x_4^j) - \begin{bmatrix} 10, & 200, & 0 \end{bmatrix} \cdot \begin{bmatrix} 7x_3^j + 5x_4^j, & 0, & 1 \end{bmatrix}^T = 10x_4^j$$

子问题 2 更新为

$$\begin{aligned} \max \quad & 10x_4^j \\ \text{s.t.} \quad & 3x_3^j + 2x_4^j \leq 15 \\ & x_3^j + x_4^j \leq 4 \\ & x_3^j, x_4^j \geq 0 \end{aligned}$$

求解该子问题的代码及结果如下：

PythonGurobiDWExampleStep1SP2

```

SP2 = Model('SubProblem 2')
xx = {}
for i in range(2):
    xx[i+1] = SP2.addVar(lb=0, ub=GRB.INFINITY, vtype=GRB.CONTINUOUS, name=
        'xx' + str(i+1))

SP2.setObjective(0 * xx[1] + 10 * xx[2], GRB.MAXIMIZE)
SP2.addConstr(3 * xx[1] + 2 * xx[2] <= 15)
SP2.addConstr(xx[1] + xx[2] <= 4)
SP2.optimize()
print('Objective = \t', SP2.ObjVal)
for var in SP2.getVars():
    print(var.varName, '= \t', var.x)

[Out]:
Objective =      40.0
x1 =      0.0
x2 =      4.0

```

子问题目标函数 $40 > 0$ ，因此可以产生新列。新列的目标函数系数为 $70x_3^j + 60x_4^j = 60 \times 4 = 240$ 。然后构造列

$$\begin{bmatrix} 70x_3^j + 60x_4^j \\ 7x_3^j + 5x_4^j \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 240 \\ 20 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \text{obj} \\ \text{central constraint} \\ \text{convex constraint 1} \\ \text{convex constraint 2} \end{bmatrix}$$

所以，主问题中就可以将该列加入。主问题模型变化为

$$\max z = 800\mu_2 + 280\lambda_2 + 240\lambda_3 + s \quad (16.47)$$

$$\text{s.t.} \quad 60\mu_2 + 28\lambda_2 + 20\lambda_3 - s \leq 80 \quad (16.48)$$

$$\mu_1 + \mu_2 = 1 \quad (16.49)$$

$$\lambda_1 + \lambda_2 + \lambda_3 = 1 \quad (16.50)$$

$$\mu_1, \mu_2, \lambda_1, \lambda_2, \lambda_3, s \geq 0 \quad (16.51)$$

$$s = 0$$

下面求解更新后的主问题，代码及结果如下：

DWInstancePhase2Step2RMP

```

1 new_columns = [[240, 20, 0, 1]
2     ]
3 for i in range(1):
4     rmp_col = Column(new_columns[i][1:], rmp_cons)
5     rmp_vars.append(RMP.addVar(lb=0.0
6         , ub=GRB.INFINITY
7         , obj=new_columns[i][0] # obj=new_obj_coef
8         , vtype=GRB.CONTINUOUS
9         , name='y'+ str(i+1) #name='y_ ' + str(len(rmp_vars))
10        , column=rmp_col
11    ))
12 # RMP.remove(rmp_vars[0])
13 # del rmp_vars[0]
14 RMP.update()
15 RMP.write('DW.lp')
16 RMP.optimize()
17 rmp_dual = RMP.getAttr("Pi", RMP.getConstrs())
18 print('rmp_dual = ', rmp_dual)
19 print('Objective = \t', RMP.ObjVal)
20 for var in RMP.getVars():
21     print(var.varName, '= \t', var.x)
22
23 [Out]:
24 rmp_dual = [10.0, 0.0, 40.0]
25 Objective = 840.0
26 s_1 = 0.0
27 y1 = 0.25
28 y2 = 0.0
29 y1 = 0.75

```



新的对偶变量为 $\pi^T = [10, 0, 40]$ 。

3. Iteration 3

由于之前子问题 1 已经没有新列产生，我们直接来看子问题 2。

子问题 2 的目标函数更新为

$$(70x_3^j + 60x_4^j) - [10, 0, 40] \cdot [7x_3^j + 5x_4^j, 0, 1]^T = 10x_4^j - 40$$

子问题 2 更新为

$$\max 10x_4^j - 40$$

$$\text{s.t. } 3x_3^j + 2x_4^j \leq 15$$

$$x_3^j + x_4^j \leq 4$$

$$x_3^j, x_4^j \geq 0$$

求解该子问题的代码及结果如下：

PythonGurobiDWExampleStep3SP2

```

SP2 = Model('SubProblem 2')
xx = {}
for i in range(2):
    xx[i+1] = SP2.addVar(lb=0, ub=GRB.INFINITY, vtype=GRB.CONTINUOUS, name=
        'xx' + str(i+1))

SP2.setObjective(0 * xx[1] + 10 * xx[2] - 40, GRB.MAXIMIZE)
SP2.addConstr(3 * xx[1] + 2 * xx[2] <= 15)
SP2.addConstr(xx[1] + xx[2] <= 4)
SP2.optimize()
print('Objective = \t', SP2.ObjVal)
for var in SP2.getVars():
    print(var.varName, '= \t', var.x)

[Out]:
Objective = -0.0
xx1 = 0.0
xx2 = 4.0

```

没有新列产生。迭代终止。最终的主问题变成

$$\max z = 800\mu_2 + 280\lambda_2 + 240\lambda_3 + s \quad (16.52)$$

$$60\mu_2 + 28\lambda_2 + 20\lambda_3 - s \leq 80 \quad (16.53)$$

$$\mu_1 + \mu_2 = 1 \quad (16.54)$$

$$\lambda_1 + \lambda_2 + \lambda_3 = 1 \quad (16.55)$$

$$\begin{aligned} \mu_1, \mu_2, \lambda_1, \lambda_2, \lambda_3, s &\geq 0 \\ s &= 0 \end{aligned} \quad (16.56)$$

求解该主问题的代码及结果如下：

DWInstancePhase2Step2RMPFinal

```

1 new_columns = [[240, 20, 0, 1]
2     ]
3 for i in range(1):
4     rmp_col = Column(new_columns[i][1:], rmp_cons)
5     rmp_vars.append(RMP.addVar(lb=0.0
6         , ub=GRB.INFINITY
7         , obj=new_columns[i][0] # obj=new_obj_coef
8         , vtype=GRB.CONTINUOUS
9         , name='y'+ str(i+7) #name='y_ ' + str(len(rmp_vars))
10        , column=rmp_col
11    ))
12 # RMP.remove(rmp_vars[0])
13 # del rmp_vars[0]
14 RMP.update()
15 RMP.write('DW.lp')
16 RMP.optimize()
17 rmp_dual = RMP.getAttr("Pi", RMP.getConstrs())
18 print('rmp_dual = ', rmp_dual)
19 print('Objective = \t', RMP.ObjVal)
20 for var in RMP.getVars():
21     print(var.varName, '= \t', var.x)
22
23 [Out]:
24 rmp_dual = [12.0, 80.0, 0.0]
25 Objective = 1040.0
26 s_1 = 0.0
27 y1 = 0.0
28 y2 = 0.0
29 y5 = 1.0
30 y6 = 0.0
31 y7 = 1.0

```

因此，得到最优值为 1040。其中 y_5 对应的极点为 $[0, 10]$ ，是属于子问题 1 产生的极点，因此 $[x_1, x_2] = y_5 \cdot [0, 10] = [0, 10]$ 。 y_7 对应的极点为 $[0, 4]$ ，是属于子问题 2 产生的极点，因此 $[x_3, x_4] = y_7 \cdot [0, 4] = [0, 4]$ 。因此原问题的最优解为 $[x_1, x_2, x_3, x_4] = [0, 10, 0, 4]$ 。至此，用列生成的方法结合 Gurobi 的 Python 接口，实现了 Dantzig-Wolfe 分解算法的详细迭代过程。在实际大规模问题中，我们可以将上述过程写成循环，以实现自动化运行整个算法的目的。

在后面的章节中将利用上述思路，用 Dantzig-Wolfe 分解算法求解多商品网络流问题 (MCNF) 以及带时间窗的车辆路径规划问题 (Vehicle Routing Problem with Time Windows, VRPTW)。

16.5 Python 调用 Gurobi 实现 Dantzig-Wolfe 分解求解多商品流运输问题

16.5.1 多商品网络流模型的区块结构

我们以多商品网络流 (Multicommodity Network Flow, MCNF) 问题为例，阐述用 DW 分解算法求解该问题的整体思路。

首先参照第 2 章相关内容，给出 MCNF 的模型 (点-弧模型, Node-Arc Model) 如下：

$$\min \sum_{k \in K} \sum_k c_{ij}^k x_{ij}^k \quad (16.57)$$

$$\sum_{j \in V} x_{ij}^k - \sum_{i \in V} x_{ji}^k = \begin{cases} d_k, & i = s_k, \forall i \in V \\ -d_k, & i = t_k, \forall i \in V \\ 0, & \text{其他, } \forall i \in V \end{cases} \quad (16.58)$$

$$\sum_{k \in K} x_{ij}^k \leq u_{ij}, \quad \forall (i, j) \in A \quad (16.59)$$

$$x_{ij}^k \geq 0, \quad \forall (i, j) \in A, k \in K \quad (16.60)$$

为了方便分解，我们将模型改写为

$$\min \sum_{k \in K} \sum_{(i,j) \in A} c_{ij}^k x_{ij}^k \quad (16.61)$$

$$\sum_{j \in V} x_{ij}^k - \sum_{i \in V} x_{ji}^k = b_i^k, \quad \forall i \in V, \forall k \in K \quad (16.62)$$

$$\sum_k x_{ij}^k \leq u_{ij}, \quad \forall (i, j) \in A \quad (16.63)$$

$$x_{ij}^k \geq 0, \quad \forall (i, j) \in A, \forall k \in K \quad (16.64)$$

其中，当 $i = s_k$ 时， $b_i^k = d_k$ ；当 $i = t_k$ 时， $b_i^k = -d_k$ ；其余情况 $b_i^k = 0$ 。

观察得知约束 (16.62) 中是针对 $\forall k \in K$ 的，而约束 (16.63) 却只针对 $\forall (i, j) \in A$ ，因此可以判断，该模型是符合块角矩阵形式的。其中，约束 (16.63) 是每一条弧 (i, j) 上所有商品流 k 的加和。而约束 (16.62) 则可以按照商品流序号 k 分成 $|K|$ 个块。正好可以使用 DW 分解算法来求解。

也许上述描述对于初学者来讲并不是很清楚，那么我们继续以文献 (Cappanera and Scaparra, 2011) 中的例子来讲解。示例网络如图 16.4 所示。

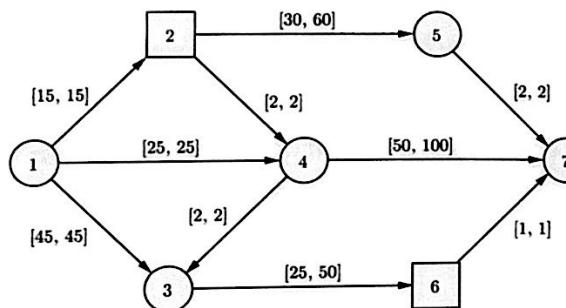


图 16.4 多商品网络流：示例网络

假设我们考虑有 2 个商品流，分别表示如下：

- (1) commodity 1: [1, 7, 25];
- (2) commodity 2: [2, 6, 2];

其中，[1, 7, 25] 表示起点为 1、终点为 7、需求是 25 单位。我们继续用 Excel 将其具体模型写出来。整理可以得出如图 16.5 所示的表格。

根据图 16.5 中的信息，我们就很清楚地看到 MCNF 的区块结构。

16.5.2 多商品流运输问题：Dantzig-Wolfe 分解求解

本章的实践中我们不做 MCNF 的具体实现，我们仅做多商品流运输问题 (Multicommodity Transportation Problem, MCTP) 的实现。回顾第 2 章的内容，我们知道 MCNF 和 MCTP 的区别在于，MCTP 中货物并没有网络中间点的转运，而是直接从起点运输到终点。我们首先在此给出 MCTP 的数学模型如下：

$$\min \sum_{k \in K} \sum_{(i,j) \in A} c_{ij}^k x_{ij}^k \quad (16.65)$$

$$\sum_{j \in C} x_{ij}^k = s_i^k \quad \forall i \in S, \forall k \in K \quad (16.66)$$

$$\sum_{i \in S} x_{ij}^k = d_j^k, \quad \forall j \in C, \forall k \in K \quad (16.67)$$

$$\sum_{k \in K} x_{ij}^k \leq u_{ij}, \quad \forall (i, j) \in A \quad (16.68)$$

$$\mathbf{B}_{ij}^0 \mathbf{y}_0 \leq u_{ij}, \quad \forall (i, j) \in A \quad (16.78)$$

$$\text{Null} = \text{Null} \quad \forall k \in K \rightarrow \text{凸组合约束} \quad (16.79)$$

$$\mathbf{y}_0 \geq \mathbf{0} \quad (16.80)$$

当产生了一系列 λ 以后，我们的主问题中的凸组合约束 (convex combination constraints) 就不再为空。我们用 \bar{x} 表示子问题的解（或者子问题的极点），因此主问题就更新为

$$\min_{\lambda} \mathbf{c}_0^T \mathbf{y}_0 + \sum_{k \in K} \sum_{v=1}^{SP_k} (\mathbf{c}_k^T \bar{x}_k^{(v)}) \lambda_{k,v} \quad (16.81)$$

$$\text{s.t. } \mathbf{B}_{ij}^0 \mathbf{y}_0 + \sum_{k \in K} \sum_{v=1}^{SP_k} (\mathbf{B}_k \bar{x}_k^{(v)}) \lambda_{k,v} \leq u_{ij}, \quad \forall (i, j) \in A \quad (16.82)$$

$$\sum_{v=1}^{SP_k} \delta_{k,v} \lambda_{k,v} = 1, \quad \forall k \in K \quad (16.83)$$

$$\mathbf{y}_0 \geq \mathbf{0} \quad (16.84)$$

$$\lambda_{k,v} \geq 0, \quad \forall k \in K, \forall v \in SP_k \quad (16.85)$$

其中， SP_k 表示由第 k 个子问题产生的极点的个数或者极射线的集合的大小。 $\mathbf{c}_k^T \bar{x}_k^{(v)}$ 表示第 k 个子问题产生的第 v 个极点或者极射线对应的解产生的总成本（一个极点就对应子问题的一个基可行解，其实质是子问题中 x_{ij} 的取值 \bar{x}_{ij} ，这里为了简略直接将子问题 k 的解写成了 \bar{x}_k ，但是其实 \bar{x}_k 是一个 $|A|$ 维的向量）。也就是这个极点 j （或者解）如果被选择，会对主问题的目标函数贡献多少。而这个极点对应的主问题的决策变量（或者主问题的列）就是 $\lambda_{k,v}$ ：并且有

$$\mathbf{c}_k^T \bar{x}_k^{(v)} = \sum_{(i,j) \in A} c_{ij} \bar{x}_{ij}$$

这里 x_{ij} 是第 k 个子问题中的决策变量， \bar{x}_{ij} 就是子问题的一个解（或者极点）。

$\lambda_{k,v}$ 在中心约束中的约束系数就是对应第 k 个子问题对弧段 (i, j) 贡献的总流量，也就是 $\mathbf{B}_k \bar{x}_k^{(v)}$ ，且在主问题第 (i, j) 个中心约束中， $\lambda_{k,v}$ 的约束系数为

$$\lambda_{k,v} \text{ 在中央约束中的约束系数} = \mathbf{B}_k \bar{x}_k^{(v)} = \bar{x}_{ij}$$

因为只有 x_{ij} 才是弧 (i, j) 上的流量。

另外，参数 $\delta_{k,v}$ 是 $\lambda_{k,v}$ 在凸组合约束中的系数。 $\lambda_{k,v}$ 只有在对应的第 k 个子问题对应的凸组合约束中，系数才为 1，其他情况均为 0。也就是

$$\delta_{k,v} = \begin{cases} 1, & k = k_0 \\ 0, & \text{其他} \end{cases}$$

其中， k_0 是当前子问题的 ID。

$$\sum_{i \in S} x_{ij} = d_j, \quad \forall j \in C \quad (16.96)$$

$$x_{ij} \geq 0, \quad \forall (i, j) \in A \quad (16.97)$$

若子问题目标函数为负，则可以产生如下新列（假设针对子问题 1，此时 $k=1$ ）：

$$\text{new column} = \left[\begin{array}{c} \sum_{(i,j)} c_{ij}^1 \bar{x}_{ij}^{(v)} \\ \bar{x}_{12}^{(v)} \\ \bar{x}_{23}^{(v)} \\ \vdots \\ 1 \\ 0 \\ \vdots \\ 0 \end{array} \right] \cdot \lambda_{1,v} = \left[\begin{array}{c} \text{目标函数中的系数} \\ \text{弧}(1,2) \text{对应中央约束中的约束系数} \\ \text{弧}(2,3) \text{对应中央约束中的约束系数} \\ \vdots \\ \text{子问题 1 对应凸约束中的约束系数} \\ \text{子问题 2 对应凸约束中的约束系数} \\ \vdots \\ \text{子问题 } K \text{ 对应凸约束中的约束系数} \end{array} \right] \cdot \lambda_{1,v}$$

16.5.3 Python 调用 Gurobi 实现 Dantzig-Wolfe 分解求解多商品流运输问题

本节尝试用 Python 调用 Gurobi 实现 Dantzig-Wolfe 分解求解 MCTP。具体来讲，就是结合 16.4 节介绍的两阶段法，结合列生成的思想来实现整个 Dantzig-Wolfe 分解算法的框架。最后用一个实际的算例来测试算法，以展示算法的正确性。本章代码很大程度上参考了杉数科技算法工程师伍健^① 共享在 github 上的代码。该代码实现的算法也正是 16.5.2 节介绍的 MCTP 的模型，具体细节参考自文献 (Kalvelagen, 2003)。

16.5.4 完整代码

完整代码如下：

```
MCTP.py
1 from __future__ import division, print_function
2 from gurobipy import *
3
4 class MCTP:
5     def __init__(self):
6         # initialize data
7         self.NumOrg = 0
8         self.NumDes = 0
9         self.NumProduct = 0
```

^① 伍健 E-mail 地址为 wujianjack2@163.com.

```
self.Supply = []
self.Demand = []
self.Capacity = []
self.Cost = []

# initialize variables and constraints in RMP
self.CapacityCons = [] # capacity constraints
self.var_lambda = [] # decision variable of RMP, i.e., columns

# initialize variables in subproblem
self.var_x = []

# initialize parameters
self.Iter = 0
self.dual_convexCons = 1
# 凸组合约束的对偶变量，更新子问题目标函数使用
# 这里赋予一个初值1，为了在第一次迭代的时候，子问题目标函数能小于0。
# 能有新的检验数为负的列产生
self.dual_CapacityCons = []
# 容量约束的对偶变量，更新子问题目标函数使用
self.SP_totalCost = []
# 子问题的解产生的总运输成本，在添加新列的时候，是新变量的目标函数
# 系数

def readData(self, filename):
    # input data
    with open(filename, "r") as data:
        self.NumOrg = int(data.readline())
        self.NumDes = int(data.readline())
        self.NumProduct = int(data.readline())

        for i in range(self.NumOrg):
            col = data.readline().split()
            SupplyData = []
            for k in range(self.NumProduct):
                SupplyData.append(float(col[k]))
            self.Supply.append(SupplyData)

        for j in range(self.NumDes):
            col = data.readline().split()
            DemandData = []
            for k in range(self.NumProduct):
                DemandData.append(float(col[k]))
            self.Demand.append(DemandData)
```

```

47     DemandData.append(float(col[k]))
48     self.Demand.append(DemandData)
49
50     for i in range(self.NumOrg):
51         col = data.readline().split()
52         CapacityData = []
53         for j in range(self.NumDes):
54             CapacityData.append(float(col[j]))
55             self.Capacity.append(CapacityData)
56
57         for i in range(self.NumOrg):
58             CostData = []
59             for j in range(self.NumDes):
60                 col = data.readline().split()
61                 CostData_temp = []
62                 for k in range(self.NumProduct):
63                     CostData_temp.append(float(col[k]))
64                     CostData.append(CostData_temp)
65             self.Cost.append(CostData)
66
67     def initializeModel(self):
68         # initialize master problem and subproblem
69         self.RMP = Model("RMP")
70         self.subProblem = Model("subProblem")
71
72         # close log information
73         self.RMP.setParam("OutputFlag", 0)
74         self.subProblem.setParam("OutputFlag", 0)
75
76         # add initial artificial variable in RMP, in order to start the algorithm
77         self.var_Artificial = self.RMP.addVar(lb = 0.0
78                                         , ub = GRB.INFINITY
79                                         , obj = 0.0
80                                         , vtype = GRB.CONTINUOUS
81                                         , name = 'Artificial'
82                                         )
83
84         # add temp capacity constraints in RMP, in order to start the algorithm
85         for i in range(self.NumOrg):
86             CapCons_temp = []
87             for j in range(self.NumDes):
88                 CapCons_temp.append(self.RMP.addConstr(-self.var_Artificial

```

```

<= self.Capacity[i][j], name = 'capacity cons'))
self.CapacityCons.append(CapCons_temp)
# initialize the convex combination constraints
self.convexCons = self.RMP.addConstr(1 * self.var_Artificial == 1,
name = 'convex cons')
# add variables to subproblem, x is flow on arcs
for i in range(self.NumOrg):
    var_x_array = []
    for j in range(self.NumDes):
        var_x_temp = []
        for k in range(self.NumProduct):
            var_x_temp.append(self.subProblem.addVar(lb = 0.0
                                         , ub = GRB.INFINITY
                                         , obj = 0.0
                                         , vtype = GRB.CONTINUOUS))
        var_x_array.append(var_x_temp)
    self.var_x.append(var_x_array)
# add constraints supply to subproblem
for i in range(self.NumOrg):
    for k in range(self.NumProduct):
        self.subProblem.addConstr(quicksum(self.var_x[i][j][k] for j
in range(self.NumDes)) \
                                == self.Supply[i][k], name
= 'Supply_' + str(i) + '_' + str(k))
# add constraints demand for subproblem
for j in range(self.NumDes):
    for k in range(self.NumProduct):
        self.subProblem.addConstr(quicksum(self.var_x[i][j][k] for i
in range(self.NumOrg)) \
                                == self.Demand[j][k], name
= 'Demand_' + str(j) + '_' + str(k))
# export the initial RMP with artificial variable
self.RMP.write('initial_RMP.lp')
def optimizePhase_1(self):
    # initialize parameters
    for i in range(self.NumOrg):
        dual_capacity_temp = [0.0] * self.NumDes

```

```

125     self.dual_CapacityCons.append(dual_capacity_temp)
126
127     obj_master_phase_1 = self.var_Artificial
128     obj_sub_phase_1 = -quicksum(self.dual_CapacityCons[i][j] * self.var_
x[i][j][k] \
129         for i in range(self.NumOrg) \
130         for j in range(self.NumDes) \
131         for k in range(self.NumProduct)) - self.
dual_convexCons
132
133     # set objective for RMP of Phase 1
134     self.RMP.setObjective(obj_master_phase_1, GRB.MINIMIZE)
135
136     # set objective for subproblem of Phase 1
137     self.subProblem.setObjective(obj_sub_phase_1, GRB.MINIMIZE)
138
139     # in order to make initial model is feasible, we set initial convex
# constraints to Null
140     # and in later iteration, we set the RHS of convex constraint to 1
141     self.RMP.chgCoeff(self.convexCons, self.var_Artificial, 0.0)
142
143     # Phase 1 of Dantzig-Wolfe decomposition : to ensure the initial
# model is feasible
144     print(" ----- start Phase 1 optimization ----- ")
145
146     while True:
147         print("Iter: ", self.Iter)
148         # export the model and check whether it is correct
149         self.RMP.write('solve_master.lp')
150         # solve subproblem of phase 1
151         self.subProblem.optimize()
152
153         if self.subProblem.objval >= -1e-6:
154             print("No new column will be generated, coz no negative
reduced cost columns")
155             break
156         else:
157             self.Iter = self.Iter + 1
158
159         # compute the total cost of subproblem solution
160         # the total cost is the coefficient of RMP when new column is added
161         totalCost_subProblem = sum(self.Cost[i][j][k] * self.var_x[i]

```

```

] [j] [k] . x \
for i in range(self.NumOrg) \
for j in range(self.NumDes) \
for k in range(self.NumProduct))

self.SP_totalCost.append(totalCost_subProblem)

# update constraints in RMP
col = Column()
for i in range(self.NumOrg):
    for j in range(self.NumDes):
        col.addTerms(sum(self.var_x[i][j][k].x for k in
range(self.NumProduct)), self.CapacityCons[i][j])
        col.addTerms(1.0, self.convexCons)

# add decision variable lambda into RMP, i.e., extreme point
# obtained from subproblems
self.var_lambda.append(self.RMP.addVar(lb = 0.0
, ub = GRB.INFINITY
, obj = 0.0
, vtype = GRB.
CONTINUOUS
, name = "lam_phase1",
, column = col))

# solve RMP in Phase 1
self.RMP.optimize()

# update dual variables
if self.RMP.objval <= 1e-6:
    print(" ---obj of phase 1 reaches 0, phase 1 ends--- ")
    break
else:
    for i in range(self.NumOrg):
        for j in range(self.NumDes):
            self.dual_CapacityCons[i][j] = self.CapacityCons
            [i][j].pi
            self.dual_convexCons = self.convexCons.pi

```

```

198     # reset objective for subproblem in 'Phase 1'
199     obj_sub_phase_1 = -quicksum(self.dual_CapacityCons[i][j] *
200         self.var_x[i][j][k] \
201             for i in range(self.NumOrg) \
202                 for j in range(self.NumDes) \
203                     for k in range(self.NumProduct))
204
205     self.subProblem.setObjective(obj_sub_phase_1, GRB.MINIMIZE)
206
207     def updateModelPhase_2(self):
208         # update model of Phase 2
209         print(" ----start update model of Phase 2 ----")
210
211         # set objective of RMP in Phase 2
212         obj_master_phase_2=quicksum(self.SP_totalCost[i]*self.var_lambda[i]
213             for i in range(len(self.SP_totalCost)))
214
215         self.RMP.setObjective(obj_master_phase_2, GRB.MINIMIZE)
216
217         # fix the value of artificial variable to 0
218         self.var_Artificial.lb = 0
219         self.var_Artificial.ub = 0
220
221         # solve RMP in Phase 2
222         self.RMP.optimize()
223
224         # update dual variables
225         for i in range(self.NumOrg):
226             for j in range(self.NumDes):
227                 self.dual_CapacityCons[i][j] = self.CapacityCons[i][j].pi
228
229         self.dual_convexCons = self.convexCons.pi
230
231         # update objective of subproblem by dual variables of RMP
232         obj_sub_phase_2 = quicksum((self.Cost[i][j][k] - self.dual_
233             CapacityCons[i][j]) * self.var_x[i][j][k] \
234                 for i in range(self.NumOrg) \
235                     for j in range(self.NumDes) \
236                         for k in range(self.NumProduct)) - self.
dual_convexCons

```

```

self.subProblem.setObjective(obj_sub_phase_2, GRB.MINIMIZE)
# update iteration info
self.Iter = self.Iter + 1

def optimizePhase_2(self):
# start Phase 2 of dantzig-wolfe decomposition
print(" ----- start Phase 2 optimization ----- ")

while True:
    print("---Iter: ", self.Iter)

    # solve subproblem in Phase 2
    self.subProblem.optimize()

    if self.subProblem.objval >= -1e-6:
        print(" --- obj of subProblem reaches 0, no new columns---")
        break
    else:
        self.Iter = self.Iter + 1

    # compute the total cost of subproblem solution
    # the total cost is the coefficient of RMP when new column is added
    totalCost_subProblem = sum(self.Cost[i][j][k] * self.var_x[i]
[j][k].x \
                    for i in range(self.NumOrg) \
                    for j in range(self.NumDes) \
                    for k in range(self.NumProduct))

    # update RMP : add new column into RMP
    # 1. creat new column
    col = Column()
    for i in range(self.NumOrg):
        for j in range(self.NumDes):
            col.addTerms(sum(self.var_x[i][j][k].x for k in
range(self.NumProduct)), self.CapacityCons[i][j])

    col.addTerms(1.0, self.convexCons)

    # 2. add new columns to RMP
    self.var_lambda.append(self.RMP.addVar(lb = 0.0
, ub = GRB.INFINITY)

```

```

275         , obj = totalCost_
276         , vtype = GRB.CONTINUOUS
277         , name = "lam_phase2_"
278         " + str(self.Iter)
279         , column = col))

280     # solve RMP of Phase 2
281     self.RMP.optimize()

282     # update dual variables
283     for i in range(self.NumOrg):
284         for j in range(self.NumDes):
285             self.dual_CapacityCons[i][j] = self.CapacityCons[i][
jl.pi

288     self.dual_convexCons = self.convexCons.pi

289     # update objective of subproblem
290     obj_sub_phase_2 = quicksum((self.Cost[i][j][k] - self.dual_
291 CapacityCons[i][j]) * self.var_x[i][j][k] \
292                         for i in range(self.NumOrg) \
293                         for j in range(self.NumDes) \
294                         for k in range(self.NumProduct))
- self.dual_convexCons

296     self.subProblem.setObjective(obj_sub_phase_2, GRB.MINIMIZE)

297 def optimizeFinalRMP(self):
298     # obtain the initial solution according the RMP solution
299     opt_x = []
300     for i in range(self.NumOrg):
301         opt_x_commodity = [0.0] * self.NumDes
302         opt_x.append(opt_x_commodity)

305     # Dantzig-Wolfe decomposition : solve the final model
306     print(" ----- start optimization final RMP ----- ")
307
308     # update objective for RMP
309     for i in range(self.NumOrg):
310         for j in range(self.NumDes):

```

```

            opt_x[i][j] = self.Capacity[i][j] + self.var_Artificial.x -
self.CapacityCons[i][j].slack

obj_master_final=quicksum(self.Cost[i][j][k] * self.var_x[i][j][k]\
for i in range(self.NumOrg) \
for j in range(self.NumDes) \
for k in range(self.NumProduct))

self.subProblem.setObjective(obj_master_final, GRB.MINIMIZE)

for i in range(self.NumOrg):
    for j in range(self.NumDes):
        self.subProblem.addConstr(quicksum(self.var_x[i][j][k] for k
in range(self.NumProduct)) == opt_x[i][j])

# solve
self.subProblem.setParam("OutputFlag", 1)
self.subProblem.optimize()

def solveMCTP(self):
    # initialize the RMP and subproblem
    self.initializeModel()

    # Dantzig-Wolfe decomposition
    print(" -----")
    print(" Dantzig-Wolfe Decomposition starts ")
    print(" -----")
    self.optimizePhase_1()

    self.updateModelPhase_2()

    self.optimizePhase_2()

    self.optimizeFinalRMP()
    print(" -----")
    print(" Dantzig-Wolfe Decomposition ends ")
    print(" -----")

def reportSolution(self):
    # report the optimal solution
    print(" ----- solution info ----- ")
)

```

```

350     print("Objective: ", self.subProblem.objval)
351     print("Solution: ")
352     for i in range(self.NumOrg):
353         for j in range(self.NumDes):
354             for k in range(self.NumProduct):
355                 if abs(self.var_x[i][j][k].x) > 0:
356                     print(" x[%d, %d, %d] = %10.2f" % (i, j, k, round(
357                         self.var_x[i][j][k].x, 2)))
358
359 if __name__ == "__main__":
360     MCTP_instance = MCTP()
361     MCTP_instance.readData("MCTP.dat")
362     MCTP_instance.solveMCTP()
363     MCTP_instance.reportSolution()

```

16.5.5 算例格式说明

算例格式说明如下：

算例格式 multicommodity

```

1 3      供应商个数
2 7      客户点个数
3 3      商品种类数
4
5 400 800 200
6 700 1600 300    # supply[i,j] = sij, i 是供应商编号, j是产品编号
7 800 1800 300
8
9 300 500 100
10 300 750 100
11 100 400 0
12 75 250 50      # demand[i][j] = dij 需求点i对产品j的需求
13 650 950 200
14 225 850 100
15 250 500 250
16
17 625 625 625 625 625 625
18 625 625 625 625 625 625
19     #容量 capacity[i][j] = uij, i是起始点编号, j是目的地编号
20 625 625 625 625 625 625
21 30 39 41
22 10 14 15

```

	8	11	12
10	14	16	
11	16	17	
71	82	86	
6	8	8	
22	27	29	
7	9	9	#cost[i][j][k] = c _{ij} ^k : 每一行是1个s-t对, 表示3种产品的需求数量
10	12	13	
7	9	9	起点1 终点1 30, 39, 41
21	26	28	起点1 终点2 10, 14, 15
82	95	99	
13	17	18	
19	24	26	
11	14	14	
12	17	17	
10	13	13	
25	28	31	
83	99	104	
15	20	20	

16.5.6 算例运行结果

算例运行结果如下：

Results

```

Dantzig-Wolfe Decomposition starts
-----
----- start Phase 1 optimization -----
Iter: 0
Iter: 1
Iter: 2
Iter: 3
Iter: 4
---obj of phase 1 reaches 0, phase 1 ends---
----- start update model of Phase 2 -----
----- start Phase 2 optimization -----
---Iter: 6
---Iter: 7
---Iter: 8
---Iter: 9
---Iter: 10

```

```

18 ---Iter: 11
19 ---Iter: 12
20 ---Iter: 13
21 ---Iter: 14
22 ---Iter: 15
23 ---Iter: 16
24 ---Iter: 17
25 ---Iter: 18
26 ---Iter: 19
27 ---Iter: 20
28 ---Iter: 21
29 ---Iter: 22
30 ---Iter: 23
31 ---Iter: 24
32 ---Iter: 25
33 ---Iter: 26
34 --- obj of subProblem reaches 0, no new columns---
35 ----- start optimization final RMP -----
36 Changed value of parameter OutputFlag to 1
37   Prev: 0 Min: 0 Max: 1 Default: 1
38 Gurobi Optimizer version 9.0.1 build v9.0.1rc0 (win64)
39 Optimize a model with 51 rows, 63 columns and 189 nonzeros
40 Coefficient statistics:
41   Matrix range [1e+00, 1e+00]
42   Objective range [6e+00, 1e+02]
43   Bounds range [0e+00, 0e+00]
44   RHS range [5e+01, 2e+03]
45 Iteration      Objective      Primal Inf.      Dual Inf.      Time
46          0 -1.4000000e+31  2.400000e+31  1.400000e+01      0s
47         20  1.9950000e+05  0.000000e+00  0.000000e+00      0s
48
49 Solved in 20 iterations and 0.01 seconds
50 Optimal objective  1.9950000000e+05
51 -----
52   Dantzig-Wolfe Decomposition ends
53 -----
54 ----- solution info -----
55 Objective: 199500.0000000003
56 Solution:
57   x[0, 4, 0] =     400.00
58   x[0, 4, 1] =     75.00
59   x[0, 4, 2] =    150.00

```

x[0, 5, 1] =	625.00
x[0, 6, 1] =	100.00
x[0, 6, 2] =	50.00
x[0, 6, 0] =	275.00
x[1, 0, 0] =	525.00
x[1, 1, 1] =	100.00
x[1, 1, 2] =	400.00
x[1, 2, 1] =	234.87
x[1, 3, 1] =	50.00
x[1, 3, 2] =	250.00
x[1, 4, 0] =	250.00
x[1, 4, 1] =	50.00
x[1, 4, 2] =	140.13
x[1, 5, 1] =	100.00
x[1, 5, 2] =	175.00
x[1, 6, 0] =	50.00
x[1, 6, 1] =	25.00
x[2, 0, 0] =	500.00
x[2, 0, 1] =	100.00
x[2, 0, 2] =	300.00
x[2, 1, 0] =	225.00
x[2, 1, 1] =	100.00
x[2, 2, 0] =	75.00
x[2, 3, 0] =	15.13
x[2, 3, 1] =	625.00
x[2, 5, 0] =	225.00
x[2, 5, 1] =	84.87
x[2, 6, 0] =	350.00
x[2, 6, 1] =	200.00

16.6 Dantzig-Wolfe 分解求解 VRPTW

本节介绍 Dantzig-Wolfe 分解算法求解车辆路径规划问题 (Vehicle Routing Problem, VRP), 为了跟之前的章节统一, 我们仍以 VRPTW 为例来讨论, 具体细节参考文献 (Feillet, 2010 和 Desaulniers et al., 2006)。

首先来看 VRPTW 的一般模型, 即

$$\min \sum_{k \in K} \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ijk} \quad (16.98)$$

$$\text{s.t. } \sum_{k \in K} \sum_{j \in V} x_{ijk} = 1, \quad \forall i \in C \quad (16.99)$$