《运筹学》案例3，9，10，11，12参考求解

## 案例3：红牌罐头公司的制造方案（第一章：线性规划；难度：中等）

**案例背景：**

红牌罐头(Red Brand Canners）公司的副总裁米切尔·戈登要求管理员、销售经理和生产经理跟他开会讨论本季节番茄产品的装罐数量问题。该厂已经在种植园购买了番茄，即将运抵工厂，装罐操作将在下周一开始。红牌罐头制造商是一个中型公司，在西部各州，它以自己的商标生产和销售各种水果和蔬菜罐头。

管理员威廉姆斯·古柏和销售部经理查尔斯·梅耶尔是首先到达戈登办公室的。生产经理丹·塔克几分钟之后进来说他已经获得关于收购进来的番茄质量的最新评估。依照报告，罐头公司在今年以每磅6美分收购来的300万磅番茄中，大约20%的质量等级为A，剩下的部分质量等级为B。

戈登询问梅耶尔来年对番茄产品的需求。梅耶尔说他们能够销售完他们生产的所有整番茄罐头。另外，对番茄汁和番茄酱的期望需求是有限的。销售经理分发了最新的需求预测，参见表1。他指出已经依照公司长期的营销战略设定了出售价格，而且对于相应价格的潜在销售量也进行了预测。

表1 需求预测

|  |  |  |
| --- | --- | --- |
| 产品 | 销售价格（美元） | 需求预期 |
| 整番茄罐头 | 4 | 800000 |
| 精选的桃块 | 5.4 | 10000 |
| 桃子饮料 | 4.6 | 5000 |
| 番茄汁 | 4.5 | 50000 |
| 烹饪苹果 | 4.9 | 15000 |
| 番茄酱 | 3.8 | 80000 |

古柏看过梅耶尔的需求预测之后说，看上去公司“应该好好利用今年的番茄”。随着新的统计系统的建立，他已经能够计算每一种产品的毛利，而且根据他的分析，整番茄罐头的利润增幅大于其他任何番茄产品。古柏计算了番茄商品的毛利：

表2 产品收益率

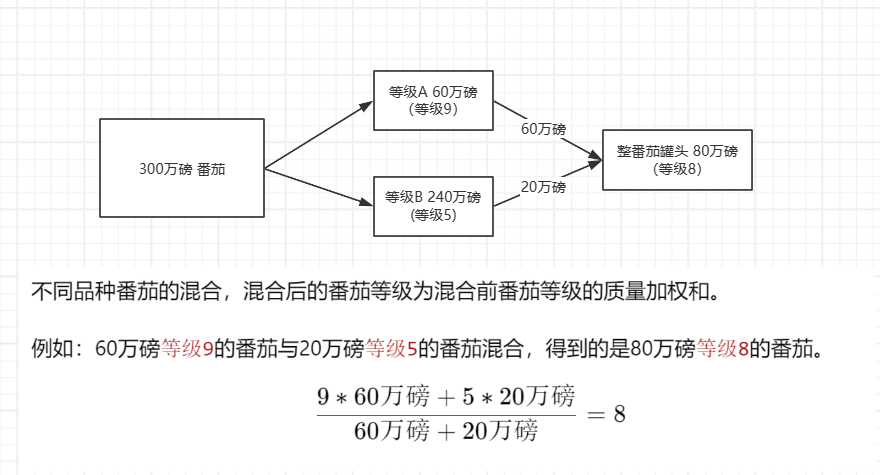
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 产品 | 整番茄罐头 | 精选的桃块 | 桃子饮料 | 番茄汁 | 烹饪苹果 | 番茄酱 |
| 销售价格 | 4 | 5.4 | 4.6 | 4.5 | 4.9 | 3.8 |
| 可变成本中的直接人工成本 | 1.18 | 1.4 | 1.27 | 1.32 | 0.7 | 0.54 |
| 可变日常费用 | 0.24 | 0.32 | 0.23 | 0.36 | 0.22 | 0.26 |
| 可变销售费用 | 0.4 | 0.3 | 0.4 | 0.85 | 0.28 | 0.38 |
| 包装材料 | 0.7 | 0.56 | 0.6 | 0.65 | 0.7 | 0.77 |
| 水果 | 1.08 | 1.8 | 1.7 | 1.2 | 0.9 | 1.5 |
| **可变成本总计(美元)** | **3.6** | **4.38** | **4.2** | **4.38** | **2.8** | **3.45** |
| 毛利 | 0.4 | 1.02 | 0.4 | 0.12 | 1.1 | 0.35 |
| 减去分担成本 | 0.28 | 0.7 | 0.52 | 0.21 | 0.75 | 0.23 |
| **净利润(美元)** | **0.12** | **0.32** | **-0.12** | **-0.09** | **0.35** | **0.12** |

表3 各产品的使用量

|  |  |
| --- | --- |
| 产品 | 番茄使用量（磅） |
| 整番茄罐头 | 18 |
| 精选的桃块 | 18 |
| 桃子饮料 | 17 |
| 番茄汁 | 20 |
| 烹饪苹果 | 27 |
| 番茄酱 | 25 |

丹·塔克让古柏注意虽然有着充分的生产能力，但是因为质量等级为A的番茄原料比例太小不可能将所有的番茄都加工成整番茄罐头。红牌公司用了一个数字等级记录产品和备用产品的质量。有0-10个等级，较高的数字表示较好的质量。依照这个等级，质量为A的番茄平均每磅9美分，等级为B的番茄平均每磅5美分。塔克指出加工成罐装的整番茄最小的输入质量为每磅8美分，加工成汁的每磅为6美分。而等级为B的番茄都可以加工成酱。这就意肤着整番茄罐头被限制在800 000磅\*。

\*说明：



戈登认为这个不是真正的限制。最近有人愿意以每磅8.5美分的价格出售给他80000磅质量为A的番茄，他当时就拒绝了。然而，他认为这样的番茄依然是可以获得的。梅耶尔做了一些计算，他虽然同意公司“应该好好利用今年的番茄”，但这并不等价于生产整番茄罐头。在他看来番茄的成本应该以番茄的质量和数量为基础来定，而不是像古柏那样仅仅通过质量。

1. 按照古柏的分析（即不考虑番茄的等级），请写出这个问题的线性规划模型，并计算加工这批番茄所能获得的最大利润与最优生产方案。
2. 按照丹·塔克的分析，如果考虑番茄的等级，不同等级的番茄能够充分混合，该工厂需要设计一种番茄的加工混合方案，使得生产**整番茄罐头、番茄汁、番茄酱**这三种产品所获得的利润最大。请写出该问题的线性规划模型，并求出最大利润及生产方案。
3. 为了进一步提高利润，该工厂是否应该以每磅8.5美分的价格购入80000磅质量为A的番茄？

**案例解答：**

1. 该问题的变量、目标函数和约束条件为：

x1,x2,x3,x4,x5,x6分别为整番茄罐头、精选的桃块、桃子饮料、番茄汁、烹饪苹果、番茄酱的生产数量（单位：罐）。

目标函数（最大化利润）为：

max z=0.12x1+0.32x2-0.12x3-0.09x4+0.35x5+0.12x6

约束条件有：

番茄使用量限制：生产各产品的番茄使用总量应不超过番茄收购量。即：

18x1+18x2+17x3+20x4+27x5+25x6<=3000000

需求预期限制：整番茄罐头、精选的桃块、桃子饮料、番茄汁、烹饪苹果、番茄酱的生产数量应不超过其需求预期。即：

x1<=800000

x2<=10000

x3<=5000

x4<=50000

x5<=15000

x6<=80000

非负限制：变量为正整数

xi>=0,i=1,2,3,4,5,6且为整数

综上，该问题的数学模型为：

max z=0.12x1+0.32x2-0.12x3-0.09x4+0.35x5+0.12x6

s.t.

18x1+18x2+17x3+20x4+27x5+25x6<=3000000

x1<=800000

x2<=10000

x3<=5000

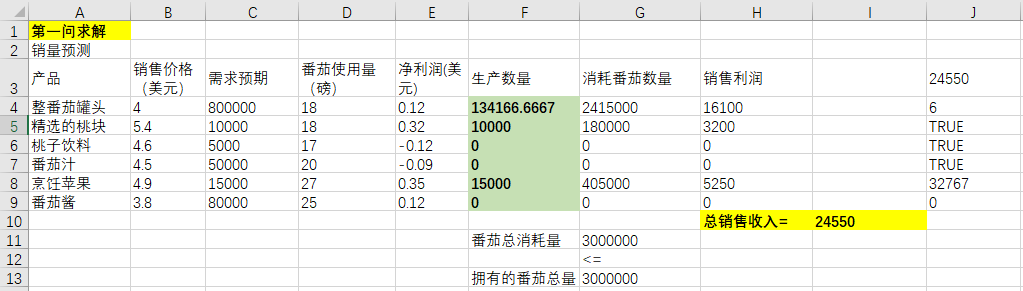
x4<=50000

x5<=15000

x6<=80000

xi>=0,i=1,2,3,4,5,6且为整数

使用Excel求解，得到结果为：



最优加工方案为：分配2415000磅番茄生产整番茄罐头、分配180000磅番茄生产精选的桃块，分配405000磅番茄生产烹饪苹果，最大利润为24550美元

（2）建立线性规划模型：

决策变量：设生产整番茄、番茄汁和番茄酱中所用A级番茄为X1，X2，X3磅，B级番茄为X4，X5，X6磅

目标函数（最大化利润）：max z=0.12(x1+x4)/18-0.09(x2+x5)/20+0.12(x3+x6)/25

约束条件有：

①　数量限制可得：A级番茄600000磅，B级番茄2400000磅

x1+x2+x3≤600000

x4+x5+x6≤2400000

②　需求限制可得：

整番茄罐头需求上限为800000\*18=14400000磅:x1+x4≤14400000

番茄汁需求上限为50000\*20=1000000磅 :x2+x5≤1000000

番茄酱需求上限为80000\*25=2000000磅:x3+x6≤2000000

③　质量限制可得：整番茄罐头等级最低为8最高为10；番茄汁质量最低为6最高为10；番茄酱质量最低为5最高为8

10>=(9\*x1+5\*x4)/(x1+x4)>=8;

10>=(9\*x2+5\*x5) /(x2+x5)>=6;

(9\*x3+5\*x6) /(x3+x6)>=5;

化简得到：

x1-3x4>=0

x1+5x4>=0

3x2-x5>=0

X2+x5>=0

非负约束：

xi≥0，i=1,2,3,4,5,6且为整数

综上，整合数学模型

max z=0.12(x1+x4)/18-0.09(x2+x5)/20+0.12(x3+x6)/25

x1+x2+x3≤600000

x4+x5+x6≤2400000

x1+x4≤14400000

x2+x5≤1000000

x3+x6≤2000000

x1-3x4>=0

x1+5x4>=0

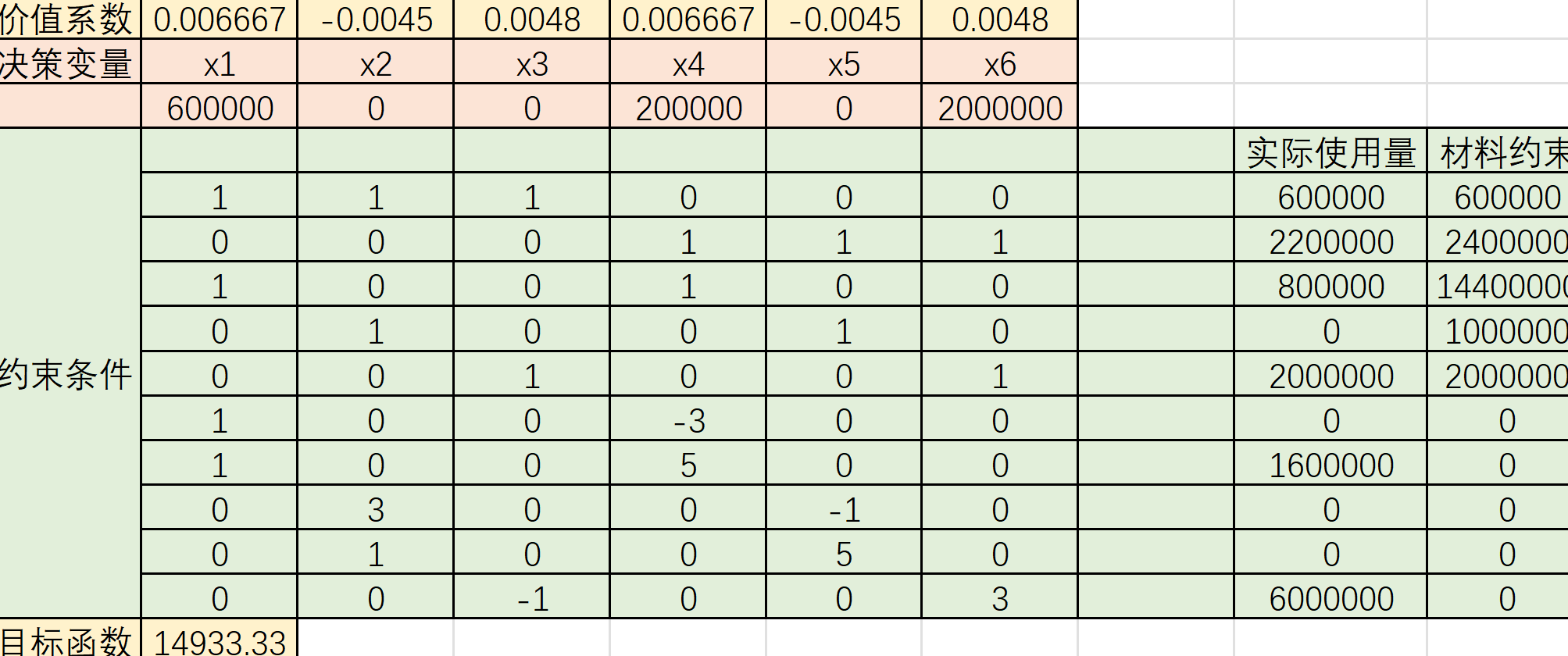
3x2-x5>=0

X2+x5>=0

-x3+3x6>=0

xi≥0，i=1,2,3,4,5,6且为整数

excel求解过程和结果如下：

****

最佳生产方案为：用于生产整番茄罐头的A级番茄为600000、B级番茄为0；用于生产整番茄汁的A级番茄为0、B级番茄为200000；用于生产整番茄酱的A级番茄为0、B级番茄为2000000，最大利润为14933美元。

（3）线性规划模型和问题二类似，购买新增成本为0.085＊80000＝6800美元，总利润应减去新增成本，且第一个约束条件变为X1＋X2＋X3＜=680000，其余约束条件不变。

得到新的线性规划模型如下：

max z=0.12(x1+x4)/18-0.09(x2+x5)/20+0.12(x3+x6)/25

s.t.

x1+x2+x3≤680000

x4+x5+x6≤2400000

x1+x4≤14400000

x2+x5≤1000000

x3+x6≤2000000

x1-3x4>=0

x1+5x4>=0

3x2-x5>=0

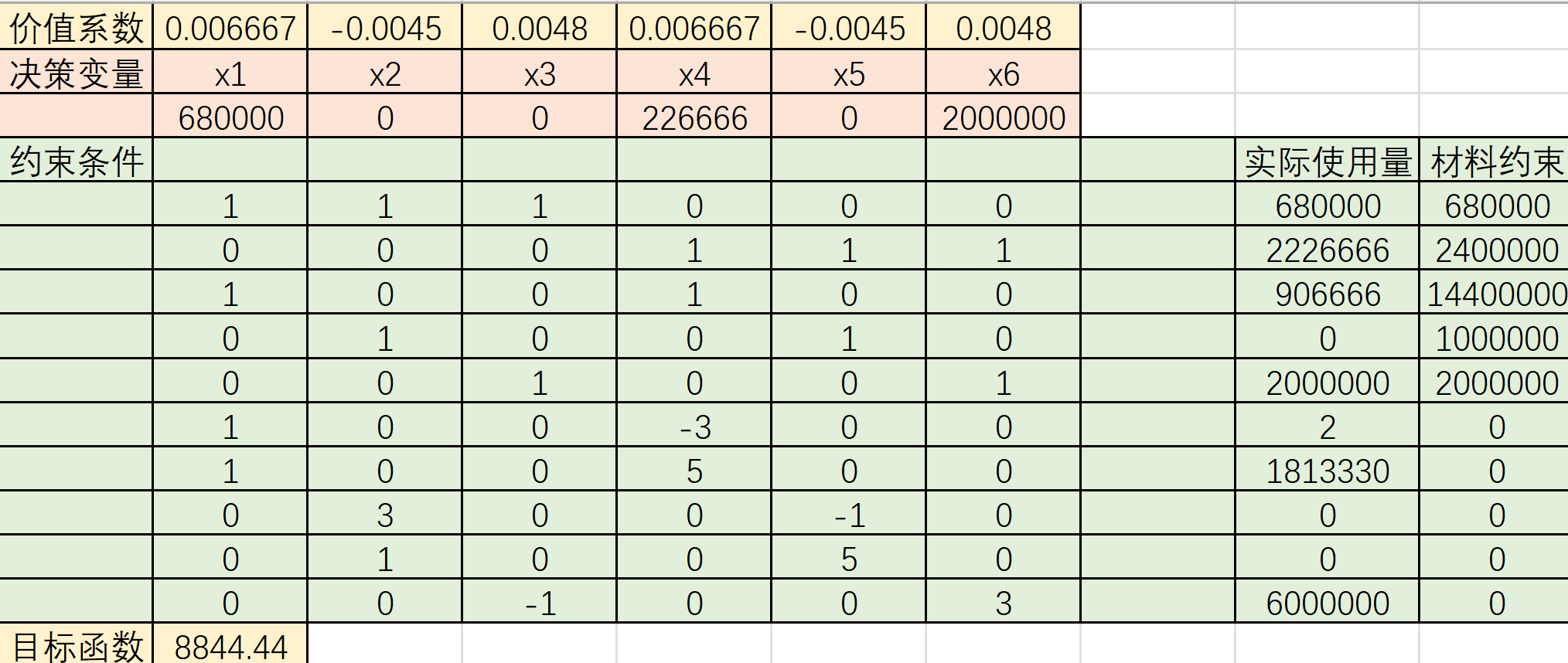
X2+x5>=0

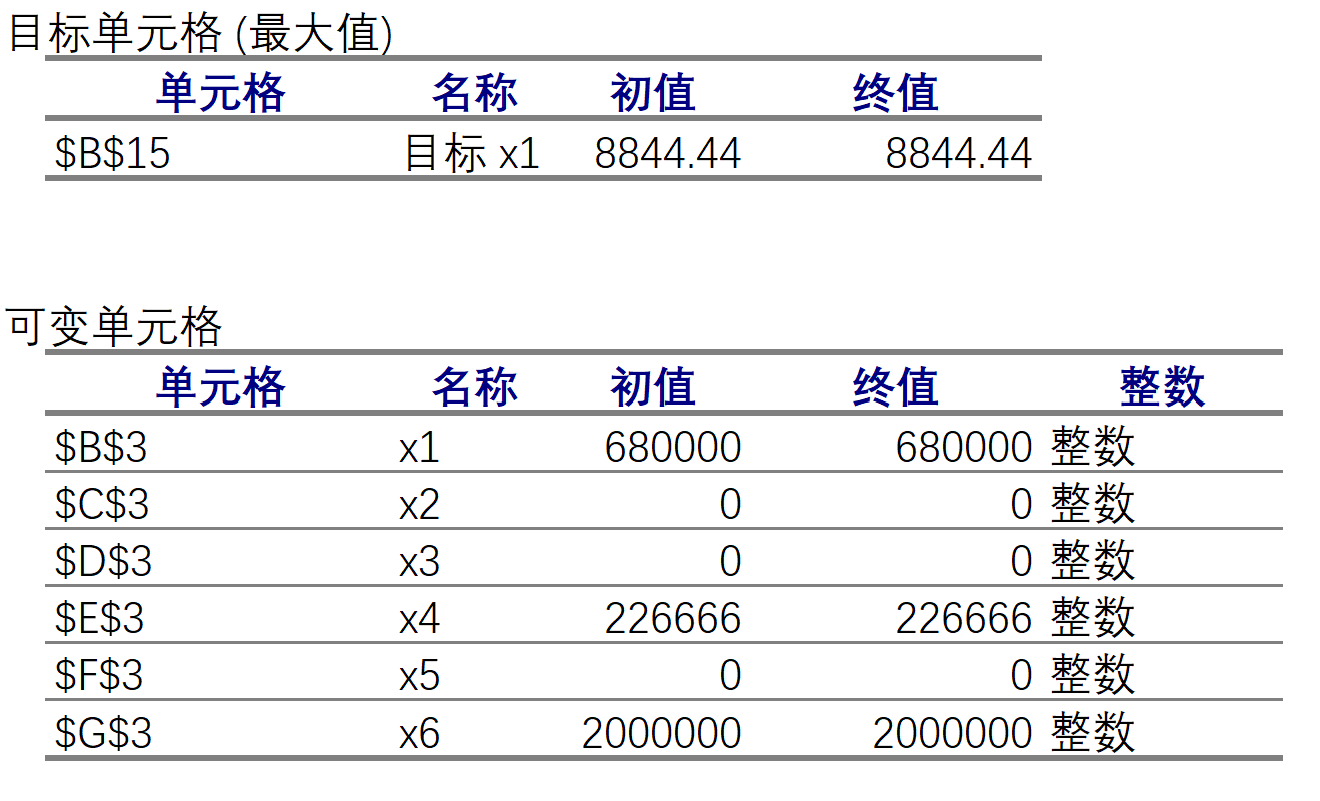
-x3+3x6>=0

xi≥0，i=1,2,3,4,5,6且为整数

通过excel求解有三种解题思路，过程和结果如下:

方法一：直接分析求解：



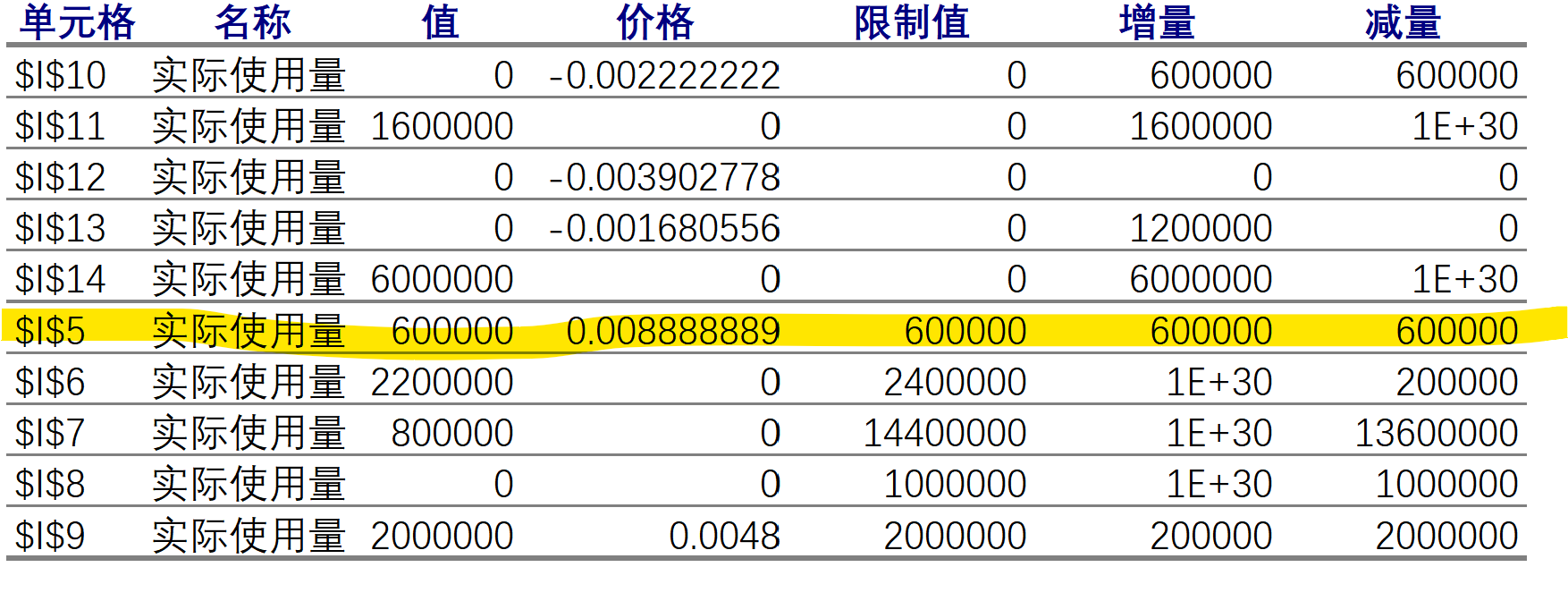




目标函数即利润较问题二的最优解降低，故不应该加购。

方法二：灵敏度分析

购入80000磅A级番茄会在问题二建立的模型的基础上改变第一个约束条件的右端项，故可以通过对问题二的规划模型进行灵敏度分析求解问题三



故不应该加购。

方法三：影子价格

可以通过比较影子价格和购入价格决定是否应该加购



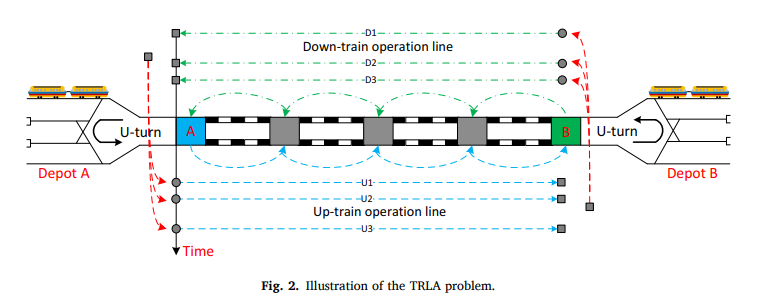
由图，A级番茄的影子价格为0.008888889美元即0.89美分，故不应该以8.5美分的价格购入A级番茄。

## 案例9：城际列车的运行时刻安排（第四章：整数规划；难度：难）

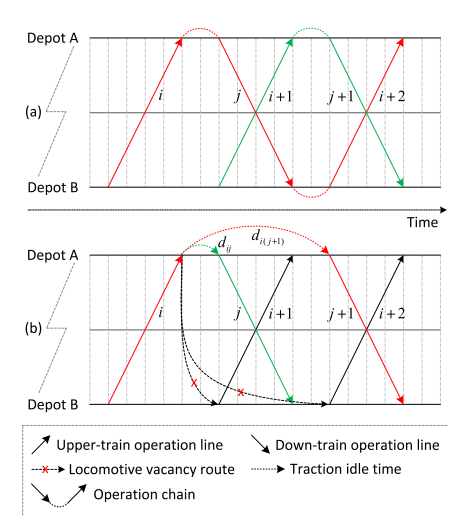
**案例介绍：**

在现代城市化进程中，城际列车系统的建设和运营对于促进区域经济发展、改善交通效率和提升居民生活质量都具有重要意义。有效的线路排图方案能够最大程度地满足乘客的出行需求，提高列车系统的运营效率和服务水平。

考虑这样一条城际铁路线，这条铁路线有A、B两个车站，车站后有一个供列车折返和停放的车辆段Depot A和Depot B， 执行完某一车次任务的列车，可以进入车辆段等待下一次列车的任务。这里定义从A站发往B站为下行、从B站发往A站为上行。



列车的运行图可以用下图表示。图中横轴代表时间，纵轴代表列车的位置。假设列车1执行线路i从B发往A，在A站折返后，执行由A发往B的线路j，在车站B折返后，执行由B发往A的线路（i+2）。列车1的运行图可以表示为下图红色曲线。同样的，如果列车2执行线路（i+1）从B发往A，在A折返后执行线路（j+1）返回B，列车2的运行图就可以表示为如图绿色曲线。



每条运营线路i、j都由1辆列车提供服务。每辆列车能够承载的旅客数量也不同。每辆列车最多承载的客流如下图所示：

|  |  |
| --- | --- |
| 列车编号 | 最大承载客流（单位：五百人） |
| Train 1 | 1 |
| Train 2 | 2 |
| Train 3 | 1 |
| Train 4 | 2 |
| Train 5 | 2 |
| Train 6 | 2 |
| Train 7 | 2 |
| Train 8 | 2 |
| Train 9 | 2 |

当列车执行完某条线路后，它将在车站等待下一线路的任务，这期间的等待时间被称为为**牵引力怠速时间**。在等待过程中，每辆列车需要一定的时间来进行车辆清洁、设备检查等准备工作，这个准备时间称为**固定技术准备时间**，通常情况下，上行和下行线路的固定技术准备时间分别为30 min和20 min）。因此，列车的牵引力怠速时间包括了列车的固定技术准备时间。

列车时刻表上标注了线路的发车时间与到达时间。考虑到调度的实际情况，线路的最早和最晚发车/到达可以在时刻表的 分钟内进行调整。为了防止同向的两辆列车追尾，同向线路的最小行车间隔为 12 分钟。列车将在当天运行结束后返回车库，等待下一日的运行。

1. 该条铁路的运营方希望用最少的列车实现时刻表上所有线路的运营。请你建立上述问题的数学模型，使得全天运营**所需的列车数量最少**。
2. 铁路的运营方希望减少列车的牵引力怠速时间，以提高列车的运营效率。请你建立上述问题的数学模型，使得所有列车**总牵引力怠速时间最短**。
3. 下表给出了该城际铁路的列车时刻表。铁路运营方希望以最小的列车数量运营该条线路，并且使得运营的总牵引力怠速时间最短。请基于列车的时刻表，先求出需要最少的机车数量，再将其作为约束条件，求出最短的总牵引力怠速时间，并给出列车的运营方案。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 车次 | 发车站 | 发车时间 | 到达站 | 到达时间 | 上下行 | 预估客流（单位：500人） |
| C1 | A | 6:15 | B | 6:55 | 下行 | 1 |
| C3 | A | 8:33 | B | 9:03 | 下行 | 2 |
| C5 | A | 9:31 | B | 10:01 | 下行 | 1 |
| C7 | A | 10:13 | B | 10:50 | 下行 | 2 |
| C9 | A | 11:08 | B | 11:38 | 下行 | 1 |
| C11 | A | 12:12 | B | 12:42 | 下行 | 1 |
| C13 | A | 12:51 | B | 13:21 | 下行 | 2 |
| C15 | A | 14:51 | B | 15:21 | 下行 | 1 |
| C17 | A | 15:08 | B | 15:38 | 下行 | 1 |
| C19 | A | 17:22 | B | 17:52 | 下行 | 2 |
| C21 | A | 18:39 | B | 19:12 | 下行 | 2 |
| C23 | A | 20:57 | B | 21:30 | 下行 | 1 |
| C2 | B | 6:19 | A | 6:49 | 上行 | 1 |
| C4 | B | 6:33 | A | 7:06 | 上行 | 2 |
| C6 | B | 7:26 | A | 7:56 | 上行 | 2 |
| C8 | B | 9:17 | A | 9:54 | 上行 | 2 |
| C10 | B | 10:09 | A | 10:39 | 上行 | 2 |
| C12 | B | 11:17 | A | 11:47 | 上行 | 1 |
| C14 | B | 12:01 | A | 12:31 | 上行 | 2 |
| C16 | B | 13:01 | A | 13:31 | 上行 | 2 |
| C18 | B | 13:57 | A | 14:27 | 上行 | 2 |
| C20 | B | 15:41 | A | 16:11 | 上行 | 2 |
| C22 | B | 16:14 | A | 16:44 | 上行 | 1 |
| C24 | B | 18:13 | A | 18:46 | 上行 | 2 |

1. 在上述案例中，列车最早（晚）的出发（到达）时间与公布的时间∆t不能超过 2 分钟， 假设上述时间发生变化，对于所需列车的最少数量、列车的平均怠速时间有何种影响？请对 2 中的模型进行敏感性分析。

**案例解答：**

（1） 建立该问题的数学模型

a）参数设置：

M 上行列车数量

N ： 下行线路数量

:可用的列车数量

:上行线路的客流量

下行线路的客流量

上行线路最早/最晚的出发时间

上行线路最早/最晚的到达时间

下行线路最早/最晚的出发时间

下行线路最早/最晚的到达时间

技术等待时间

最小列车间隔12分钟:

b）模型变量

(binary)列车执行完上行线路后是否立即执行下行线路

(binary)列车执行完下行线路后是否立即执行上行线路

(binary)是否使用列车

上行线路的实际出发时间 上行线路的实际到达时间 下行线路的实际出发时间 下行线路的实际到达时间

c）目标函数:

使用的机车数量最少

d) 约束条件

对于每辆列车k，除了最后的入库操作外，每个上行线路之前只有一个下行线路，每个下行线路前只有一个上行线路。

：进入出库线路 :进入回库线路

列车的运行线路是连续的,

对于回库线路(),该线路之前可能对于多条下行线/多条上行线

如果列车上线运行，则必须经过出库线(从A出库i=0，还是从B出库j=0)和入库线(A入库j=N+1;B入库i=M+1)。

如果列车不上线运行，则任意

运行时间约束

牵引重量限制

（2） 本问题相较于问题（1），目标函数变为

机车总怠速时间最少(不计出库与入库怠速时间)

上述约束一个非线性约束，需要转化为线性约束,例如：

可以转化为：

s.t.

参数及变量设置、其它约束均与第（1）问一致。

（3）由于问题变量较多，因此使用Python Pulp求解器进行求解，求解过程为：

*# 输入参数*

**import** pandas **as** pd

**import** numpy **as** np

train\_data **=** pd**.**read\_excel("case\_1\_data.xlsx")

*# 将时间转化为分钟为单位*

train\_data['发车时间'] **=** pd**.**to\_datetime(train\_data['发车时间']**.**astype(str), format**=**'%H:%M:%S')**.**dt**.**hour **\*** 60 **+** pd**.**to\_datetime(train\_data['发车时间']**.**astype(str), format**=**'%H:%M:%S')**.**dt**.**minute

train\_data['到达时间'] **=** pd**.**to\_datetime(train\_data['到达时间']**.**astype(str), format**=**'%H:%M:%S')**.**dt**.**hour **\*** 60 **+** pd**.**to\_datetime(train\_data['到达时间']**.**astype(str), format**=**'%H:%M:%S')**.**dt**.**minute

uptrain **=** train\_data[train\_data['上下行']**==**'上行']**.**head(12)

downtrain **=** train\_data[train\_data['上下行']**==**'下行']**.**head(12)

print("UpTrains:")

display(uptrain)

print("DownTrains:")

display(downtrain)

*# 上下行线路数量*

M **=** len(uptrain['发车时间'])

N **=** len(downtrain['发车时间'])

print(f"上行线路{M}条，下行线路{N}条")

*# 机车的牵引重量*

weight **=** [1,2,1,2,2,2,2,2,2]

MOTIVE\_NUM **=** len(weight)

*# 上下行线路所需的重量*

weight\_trainup **=** list(uptrain['列车重量'])

weight\_traindown **=** list(downtrain['列车重量'])

*#上行线路i实际的出发时间t\_i\_smin t\_i\_smax ，到达时间t\_i\_emin t\_i\_emax在公布时刻表的T\_DELTA = 2分钟之内调整*

T\_DELTA **=** 2

ti\_smin **=** [t**-**T\_DELTA **for** t **in** list(uptrain['发车时间'])]

ti\_smax **=** [t**+**T\_DELTA **for** t **in** list(uptrain['发车时间'])]

ti\_emin **=** [t**-**T\_DELTA **for** t **in** list(uptrain['到达时间'])]

ti\_emax **=** [t**+**T\_DELTA **for** t **in** list(uptrain['到达时间'])]

*# 下行列车同理：*

tj\_smin **=** [t**-**T\_DELTA **for** t **in** list(downtrain['发车时间'])]

tj\_smax **=** [t**+**T\_DELTA **for** t **in** list(downtrain['发车时间'])]

tj\_emin **=** [t**-**T\_DELTA **for** t **in** list(downtrain['到达时间'])]

tj\_emax **=** [t**+**T\_DELTA **for** t **in** list(downtrain['到达时间'])]

*# 技术等待时间*

theta\_i **=** 20

theta\_j **=** 30

*# 最小列车间隔12分钟*

H\_UP\_MIN **=** 12

H\_DOWN\_MIN **=** 12

*# 将浮点数转换为时间*

**def** min2time(minutes):

minutes **=** int(minutes)

hours **=** minutes **//** 60

minutes **=** minutes **%** 60

**return** f"{hours}:{minutes:02d}"

*# 以360分钟为例*

minutes **=** 360.0

time **=** min2time(minutes)

print(time)

*# 求解问题-最小车辆数量*

**from** pulp **import** **\***

prob **=** LpProblem("train", LpMinimize)

*# 创建变量*

*# x\_up[ijk] 机车k执行上行线路i后执行下行线路j（包括出库（i=0->j）选择回库（i=?->j=N+1））*

x\_up **=** LpVariable**.**dicts('x\_up', [(i,j,k) **for** i **in** range(0, M**+**1) **for** j **in** range(1, N**+**2) **for** k **in** range(1,MOTIVE\_NUM**+**1) ], cat**=**'Binary')

*# x\_down[jik] 机车k执行下行线路j后执行上行线路i（包括出库（j=0->i=?）选择回库（j=?->i=M+1））*

x\_down **=** LpVariable**.**dicts('x\_down', [(j,i,k) **for** j **in** range(0, N**+**1) **for** i **in** range(1, M**+**2) **for** k **in** range(1,MOTIVE\_NUM**+**1) ], cat**=**'Binary')

*#xe[k] 机车k是否执行任务*

xe **=** LpVariable**.**dicts('xe',[k **for** k **in** range(1,MOTIVE\_NUM**+**1)],cat**=**'Binary')

*# 上行线路i的实际发车时间ti\_s，到达时间ti\_e*

ti\_s **=** LpVariable**.**dicts("ti\_s", range(1, M**+**1), cat**=**'Continuous')

ti\_e **=** LpVariable**.**dicts("ti\_e", range(1, M**+**1), cat**=**'Continuous')

*# 下行线路j的实际发车时间tj\_s，到达时间tj\_e*

tj\_s **=** LpVariable**.**dicts("tj\_s", range(1, N**+**1), cat**=**'Continuous')

tj\_e **=** LpVariable**.**dicts("tj\_e", range(1, N**+**1), cat**=**'Continuous')

*#x\_up[i,j,k] :x\_up[i,j,k]的的辅助变量AAA*

y\_up **=** LpVariable**.**dicts("y\_up",[(i,j,k) **for** i **in** range(1,M**+**1) **for** j **in** range(1,N**+**1) **for** k **in** range(1,MOTIVE\_NUM**+**1)],cat**=**'Continuous')

*# y\_down[j,i,k]:x\_down[j,i,k]的辅助变量AAA*

y\_down **=** LpVariable**.**dicts("y\_down",[(j,i,k) **for** j **in** range(1,N**+**1) **for** i **in** range(1,M**+**1) **for** k **in** range(1,MOTIVE\_NUM**+**1)],cat**=**'Continuous')

TotalMotive **=** LpVariable("TotalMotive", lowBound**=**0, cat**=**'Continuous')

TotalWait **=** LpVariable("TotalWait",lowBound **=** 0,cat**=**'Continuous')

*# 定义问题*

prob **+=** TotalMotive

*# 定义约束*

prob **+=** TotalMotive **==** lpSum(xe[k] **for** k **in** range(1,MOTIVE\_NUM**+**1) )

prob **+=** TotalMotive **>=** 0

prob **+=** TotalWait **==** lpSum(y\_up[i,j,k] **for** k **in** range(1,MOTIVE\_NUM**+**1) **for** i **in** range(1,M**+**1) **for** j **in** range(1,N**+**1)) **+** lpSum( y\_down[j,i,k] **for** j **in** range(1,N**+**1) **for** i **in** range(1,M**+**1) **for** k **in** range(1,MOTIVE\_NUM**+**1))

BIGM **=** 999999999

**for** i **in** range(1,M**+**1):

**for** j **in** range(1,N**+**1):

**for** k **in** range(1,MOTIVE\_NUM**+**1):

prob **+=** y\_up[i,j,k] **<=** BIGM**\***x\_up[i,j,k]

prob **+=** y\_up[i,j,k]**-**(tj\_s[j]**-**ti\_e[i]) **<=** BIGM **\*** (1**-**x\_up[i,j,k])

prob **+=** y\_up[i,j,k]**-**(tj\_s[j]**-**ti\_e[i]) **>=** BIGM **\*** (x\_up[i,j,k]**-**1)

prob **+=** y\_up[i,j,k] **>=** 0

prob **+=** y\_down[j,i,k] **<=** BIGM**\***x\_down[j,i,k]

prob **+=** y\_down[j,i,k]**-**(ti\_s[i]**-**tj\_e[j]) **<=** BIGM **\*** (1**-**x\_down[j,i,k])

prob **+=** y\_down[j,i,k]**-**(ti\_s[i]**-**tj\_e[j]) **>=** BIGM **\*** (x\_down[j,i,k]**-**1)

prob **+=** y\_down[j,i,k] **>=** 0

*#执行上行线路i前必须且仅有一个下行线路j（包括从上行线出库的情况）*

**for** i **in** range(1,M**+**1):

prob **+=** lpSum(x\_down[j,i,k] **for** k **in** range(1,MOTIVE\_NUM**+**1) **for** j **in** range(0,N**+**1)) **==** 1

*# 执行下行线路j前必须且仅有一个上行线路i（包括从下行线出库的情况）*

**for** j **in** range(1,N**+**1):

prob **+=** lpSum(x\_up[i,j,k] **for** k **in** range(1,MOTIVE\_NUM**+**1) **for** i **in** range(0,M**+**1)) **==** 1

*# 列车的运行线路是连续的*

**for** k **in** range(1,MOTIVE\_NUM**+**1):

**for** j **in** range(1,N**+**1):

prob **+=** lpSum(x\_up[i,j,k] **for** i **in** range(0,M**+**1)) **<=** lpSum(x\_down[j,i,k] **for** i **in** range(1,M**+**2))

**for** i **in** range(1,M**+**1):

prob **+=** lpSum(x\_down[j,i,k] **for** j **in** range(0,N**+**1)) **<=** lpSum(x\_up[i,j,k] **for** j **in** range(1,N**+**2))

*# 出库的车辆等于入库的车辆*

**for** k **in** range(1,MOTIVE\_NUM**+**1):

prob **+=** lpSum(x\_up[0,j,k] **for** j **in** range(1,N**+**2))**+**lpSum(x\_down[0,i,k] **for** i **in** range(1,M**+**2)) **==** xe[k]

prob **+=** lpSum(x\_up[i,N**+**1,k] **for** i **in** range(0,M**+**1))**+**lpSum(x\_down[j,M**+**1,k] **for** j **in** range(0,N**+**1)) **==** xe[k]

*# 如果列车k不上线运行，则任意x\_up[i,j,k],x\_down[j,i,k]=0*

**for** k **in** range(1,MOTIVE\_NUM **+** 1):

**for** i **in** range(0,M**+**1):

**for** j **in** range(1,N**+**2):

prob **+=** x\_up[i,j,k] **<=** xe[k]

**for** j **in** range(0,N**+**1):

**for** i **in** range(1,M**+**2):

prob **+=** x\_down[j,i,k] **<=** xe[k]

*#时光不能倒流，执行下一次列车的最晚出发时间不能晚于上一次列车的最早到达时间+准备时间*

**for** i **in** range(1,M**+**1):

**for** j **in** range(1,N**+**1):

**if** ti\_emin[i**-**1] **+** theta\_j **>** tj\_smax[j**-**1] :

*# print(f"上行{i}最早到达时间{ti\_emin[i-1]},下行{j}最晚出发时间{tj\_smax[j-1]} ")*

**for** k **in** range(1,MOTIVE\_NUM**+**1):

prob **+=** x\_up[i,j,k] **==** 0

**if** tj\_emin[j**-**1] **+** theta\_i **>** ti\_smax[i**-**1]:

*# print(f"下行{j}最早到达时间{tj\_emin[j-1]},上行{i}最晚出发时间{ti\_smax[i-1]}")*

**for** k **in** range(1,MOTIVE\_NUM**+**1):

prob **+=** x\_down[j,i,k] **==** 0

*#每辆车执行的任务数量<=8*

**for** k **in** range (1,MOTIVE\_NUM**+**1):

prob **+=** lpSum(x\_up[i,j,k] **for** i **in** range(0,M**+**1) **for** j **in** range(1,N**+**2)) **+** lpSum(x\_down[j,i,k] **for** i **in** range(1,M**+**2) **for** j **in** range(0,N**+**1)) **<=** 8

*# 列车发车与到达的时间限制*

**for** i **in** range(1,M**+**1):

prob **+=** ti\_s[i] **<=** ti\_smax[i**-**1]

prob **+=** ti\_smin[i**-**1] **<=** ti\_s[i]

prob **+=** ti\_e[i] **<=** ti\_emax[i**-**1]

prob **+=** ti\_emin[i**-**1] **<=** ti\_e[i]

**for** j **in** range(1,N**+**1):

prob **+=** tj\_s[j] **<=** tj\_smax[j**-**1]

prob **+=** tj\_smin[j**-**1] **<=** tj\_s[j]

prob **+=** tj\_e[j] **<=** tj\_emax[j**-**1]

prob **+=** tj\_emin[j**-**1] **<=** tj\_e[j]

*# 最小时间间隔限制*

**for** i **in** range(1,M):

prob **+=** ti\_s[i**+**1] **-** ti\_s[i] **>=** H\_UP\_MIN

prob **+=** ti\_e[i**+**1] **-** ti\_e[i] **>=** H\_UP\_MIN

**for** j **in** range(1,N):

prob **+=** tj\_s[j**+**1] **-** tj\_s[j] **>=** H\_DOWN\_MIN

prob **+=** tj\_e[j**+**1] **-** tj\_e[j] **>=** H\_DOWN\_MIN

*# 牵引重量限制*

**for** i **in** range(1,M**+**1):

prob **+=** lpSum(weight[k**-**1]**\***x\_up[i,j,k] **for** k **in** range(1,MOTIVE\_NUM**+**1) **for** j **in** range(1,N**+**2)) **>=** weight\_trainup[i**-**1]

*# 牵引重量限制*

**for** j **in** range(1,N**+**1):

prob **+=** lpSum(weight[k**-**1]**\***x\_down[j,i,k] **for** k **in** range(1,MOTIVE\_NUM**+**1) **for** i **in** range(1,M**+**2)) **>=** weight\_traindown[j**-**1]

*# 求解问题*

prob**.**solve()

**if** LpStatus[prob**.**status] **==** "Optimal":

print("找到最优解")

**else**:

print("No solution")

*# 打印结果：最小车辆数量*

print(f"Best Motive number:{value(TotalMotive)}")

print(f"Shorest wait time:{value(TotalWait)}")

**import** matplotlib.pyplot **as** plt

**if** LpStatus[prob**.**status] **==** "Optimal":

**for** k **in** range(1,MOTIVE\_NUM**+**1):

print(f"Train {k}:")

plt**.**figure(figsize **=** (8,2))

**for** i **in** range(0,M**+**1):

**for** j **in** range(1, N**+**1):

**if** value(x\_up[i,j,k]) **==** 1:

*# 绘制带箭头的线段*

dx **=** value(tj\_e[j]) **-** value(tj\_s[j])

dy **=** **-**2

plt**.**arrow(value(tj\_s[j]), 2, dx, dy, head\_width**=**0.1, head\_length**=**0.5, fc**=**'red', ec**=**'red')

*# 添加文字标签*

plt**.**text(value(tj\_s[j])**+**dx**/**2, 1, f"{downtrain**.**iloc[j**-**1]['车次']}", ha**=**'center', va**=**'center')

print(f"执行线路{downtrain**.**iloc[j**-**1]['车次']}（down j={j}）,离开A时间:{min2time(value(tj\_s[j]))} 到达B时间{min2time(value(tj\_e[j]))} ")

**for** j **in** range(0,N**+**1):

**for** i **in** range(1, M**+**1):

**if** value(x\_down[j,i,k]) **==** 1:

*# 绘制带箭头的线段*

dx **=** value(ti\_e[i]) **-** value(ti\_s[i])

dy **=** 2

plt**.**arrow(value(ti\_s[i]), 0, dx, dy, head\_width**=**0.1, head\_length**=**0.5, fc**=**'green', ec**=**'green')

plt**.**text(value(ti\_s[i])**+**dx**/**2, 1, f"{uptrain**.**iloc[i**-**1]['车次']}", ha**=**'center', va**=**'center')

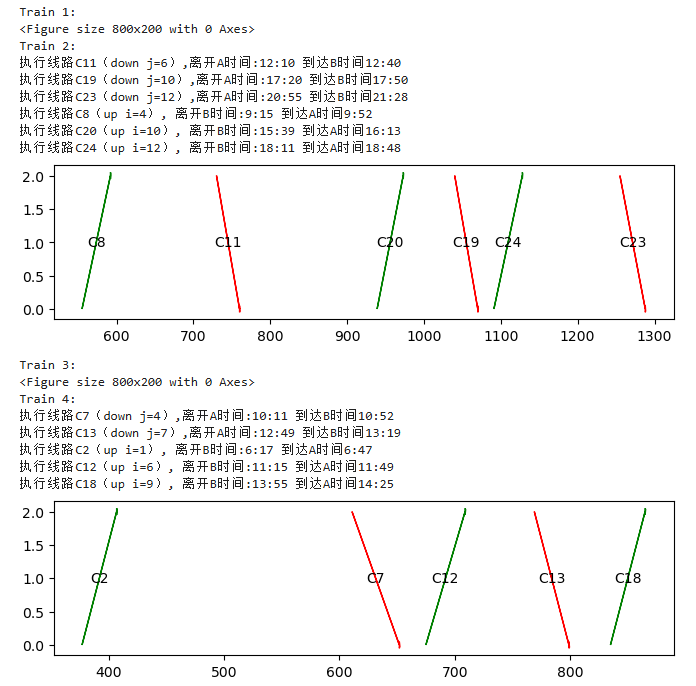
print(f"执行线路{uptrain**.**iloc[i**-**1]['车次']}（up i={i}）, 离开B时间:{min2time(value(ti\_s[i]))} 到达A时间{min2time(value(ti\_e[i]))} ")

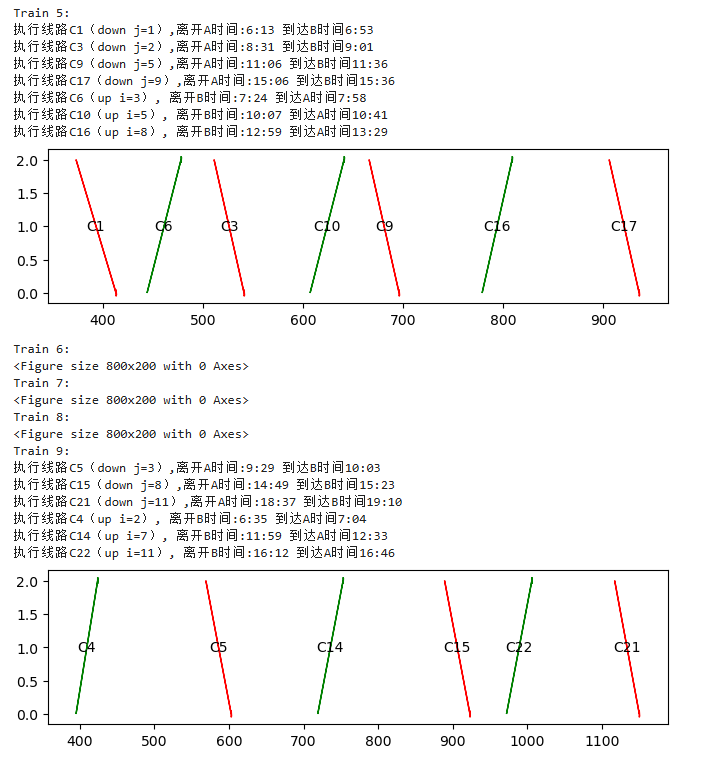
plt**.**show()

**else**:

print("No solution")

**求得最优结果为：最小车辆数4，运行方案如下所示：**





在知道最小车辆数=4的情况下，求解最短怠速时间。代码如下：

*# 求解问题-最小怠速时间*

**from** pulp **import** **\***

prob **=** LpProblem("train", LpMinimize)

*# 创建变量*

*# x\_up[ijk] 机车k执行上行线路i后执行下行线路j（包括出库（i=0->j）选择回库（i=?->j=N+1））*

x\_up **=** LpVariable**.**dicts('x\_up', [(i,j,k) **for** i **in** range(0, M**+**1) **for** j **in** range(1, N**+**2) **for** k **in** range(1,MOTIVE\_NUM**+**1) ], cat**=**'Binary')

*# x\_down[jik] 机车k执行下行线路j后执行上行线路i（包括出库（j=0->i=?）选择回库（j=?->i=M+1））*

x\_down **=** LpVariable**.**dicts('x\_down', [(j,i,k) **for** j **in** range(0, N**+**1) **for** i **in** range(1, M**+**2) **for** k **in** range(1,MOTIVE\_NUM**+**1) ], cat**=**'Binary')

*#xe[k] 机车k是否执行任务*

xe **=** LpVariable**.**dicts('xe',[k **for** k **in** range(1,MOTIVE\_NUM**+**1)],cat**=**'Binary')

*# 上行线路i的实际发车时间ti\_s，到达时间ti\_e*

ti\_s **=** LpVariable**.**dicts("ti\_s", range(1, M**+**1), cat**=**'Continuous')

ti\_e **=** LpVariable**.**dicts("ti\_e", range(1, M**+**1), cat**=**'Continuous')

*# 下行线路j的实际发车时间tj\_s，到达时间tj\_e*

tj\_s **=** LpVariable**.**dicts("tj\_s", range(1, N**+**1), cat**=**'Continuous')

tj\_e **=** LpVariable**.**dicts("tj\_e", range(1, N**+**1), cat**=**'Continuous')

*#x\_up[i,j,k] :x\_up[i,j,k]的的辅助变量AAA*

y\_up **=** LpVariable**.**dicts("y\_up",[(i,j,k) **for** i **in** range(1,M**+**1) **for** j **in** range(1,N**+**1) **for** k **in** range(1,MOTIVE\_NUM**+**1)],cat**=**'Continuous')

*# y\_down[j,i,k]:x\_down[j,i,k]的辅助变量AAA*

y\_down **=** LpVariable**.**dicts("y\_down",[(j,i,k) **for** j **in** range(1,N**+**1) **for** i **in** range(1,M**+**1) **for** k **in** range(1,MOTIVE\_NUM**+**1)],cat**=**'Continuous')

TotalMotive **=** LpVariable("TotalMotive", lowBound**=**0, cat**=**'Continuous')

TotalWait **=** LpVariable("TotalWait",lowBound **=** 0,cat**=**'Continuous')

*# 定义问题*

prob **+=** TotalWait

*# 定义约束*

prob **+=** TotalMotive **==** lpSum(xe[k] **for** k **in** range(1,MOTIVE\_NUM**+**1) )

prob **+=** TotalMotive **<=** 4

prob **+=** TotalWait **==** lpSum(y\_up[i,j,k] **for** k **in** range(1,MOTIVE\_NUM**+**1) **for** i **in** range(1,M**+**1) **for** j **in** range(1,N**+**1)) **+** lpSum( y\_down[j,i,k] **for** j **in** range(1,N**+**1) **for** i **in** range(1,M**+**1) **for** k **in** range(1,MOTIVE\_NUM**+**1))

BIGM **=** 999999999

**for** i **in** range(1,M**+**1):

**for** j **in** range(1,N**+**1):

**for** k **in** range(1,MOTIVE\_NUM**+**1):

prob **+=** y\_up[i,j,k] **<=** BIGM**\***x\_up[i,j,k]

prob **+=** y\_up[i,j,k]**-**(tj\_s[j]**-**ti\_e[i]) **<=** BIGM **\*** (1**-**x\_up[i,j,k])

prob **+=** y\_up[i,j,k]**-**(tj\_s[j]**-**ti\_e[i]) **>=** BIGM **\*** (x\_up[i,j,k]**-**1)

prob **+=** y\_up[i,j,k] **>=** 0

prob **+=** y\_down[j,i,k] **<=** BIGM**\***x\_down[j,i,k]

prob **+=** y\_down[j,i,k]**-**(ti\_s[i]**-**tj\_e[j]) **<=** BIGM **\*** (1**-**x\_down[j,i,k])

prob **+=** y\_down[j,i,k]**-**(ti\_s[i]**-**tj\_e[j]) **>=** BIGM **\*** (x\_down[j,i,k]**-**1)

prob **+=** y\_down[j,i,k] **>=** 0

*#执行上行线路i前必须且仅有一个下行线路j（包括从上行线出库的情况）*

**for** i **in** range(1,M**+**1):

prob **+=** lpSum(x\_down[j,i,k] **for** k **in** range(1,MOTIVE\_NUM**+**1) **for** j **in** range(0,N**+**1)) **==** 1

*# 执行下行线路j前必须且仅有一个上行线路i（包括从下行线出库的情况）*

**for** j **in** range(1,N**+**1):

prob **+=** lpSum(x\_up[i,j,k] **for** k **in** range(1,MOTIVE\_NUM**+**1) **for** i **in** range(0,M**+**1)) **==** 1

*# 列车的运行线路是连续的*

**for** k **in** range(1,MOTIVE\_NUM**+**1):

**for** j **in** range(1,N**+**1):

prob **+=** lpSum(x\_up[i,j,k] **for** i **in** range(0,M**+**1)) **<=** lpSum(x\_down[j,i,k] **for** i **in** range(1,M**+**2))

**for** i **in** range(1,M**+**1):

prob **+=** lpSum(x\_down[j,i,k] **for** j **in** range(0,N**+**1)) **<=** lpSum(x\_up[i,j,k] **for** j **in** range(1,N**+**2))

*# 出库的车辆等于入库的车辆*

**for** k **in** range(1,MOTIVE\_NUM**+**1):

prob **+=** lpSum(x\_up[0,j,k] **for** j **in** range(1,N**+**2))**+**lpSum(x\_down[0,i,k] **for** i **in** range(1,M**+**2)) **==** xe[k]

prob **+=** lpSum(x\_up[i,N**+**1,k] **for** i **in** range(0,M**+**1))**+**lpSum(x\_down[j,M**+**1,k] **for** j **in** range(0,N**+**1)) **==** xe[k]

*# 如果列车k不上线运行，则任意x\_up[i,j,k],x\_down[j,i,k]=0*

**for** k **in** range(1,MOTIVE\_NUM **+** 1):

**for** i **in** range(0,M**+**1):

**for** j **in** range(1,N**+**2):

prob **+=** x\_up[i,j,k] **<=** xe[k]

**for** j **in** range(0,N**+**1):

**for** i **in** range(1,M**+**2):

prob **+=** x\_down[j,i,k] **<=** xe[k]

*#时光不能倒流*

**for** i **in** range(1,M**+**1):

**for** j **in** range(1,N**+**1):

**if** ti\_emin[i**-**1] **+** theta\_j **>** tj\_smax[j**-**1] :

*# print(f"上行{i}最早到达时间{ti\_emin[i-1]},下行{j}最晚出发时间{tj\_smax[j-1]} ")*

**for** k **in** range(1,MOTIVE\_NUM**+**1):

prob **+=** x\_up[i,j,k] **==** 0

**if** tj\_emin[j**-**1] **+** theta\_i **>** ti\_smax[i**-**1]:

*# print(f"下行{j}最早到达时间{tj\_emin[j-1]},上行{i}最晚出发时间{ti\_smax[i-1]}")*

**for** k **in** range(1,MOTIVE\_NUM**+**1):

prob **+=** x\_down[j,i,k] **==** 0

*#每辆车执行的任务数量<=8*

**for** k **in** range (1,MOTIVE\_NUM**+**1):

prob **+=** lpSum(x\_up[i,j,k] **for** i **in** range(0,M**+**1) **for** j **in** range(1,N**+**2)) **+** lpSum(x\_down[j,i,k] **for** i **in** range(1,M**+**2) **for** j **in** range(0,N**+**1)) **<=** 8

*# 列车发车与到达的时间限制*

**for** i **in** range(1,M**+**1):

prob **+=** ti\_s[i] **<=** ti\_smax[i**-**1]

prob **+=** ti\_smin[i**-**1] **<=** ti\_s[i]

prob **+=** ti\_e[i] **<=** ti\_emax[i**-**1]

prob **+=** ti\_emin[i**-**1] **<=** ti\_e[i]

**for** j **in** range(1,N**+**1):

prob **+=** tj\_s[j] **<=** tj\_smax[j**-**1]

prob **+=** tj\_smin[j**-**1] **<=** tj\_s[j]

prob **+=** tj\_e[j] **<=** tj\_emax[j**-**1]

prob **+=** tj\_emin[j**-**1] **<=** tj\_e[j]

*# 最小时间间隔限制*

**for** i **in** range(1,M):

prob **+=** ti\_s[i**+**1] **-** ti\_s[i] **>=** H\_UP\_MIN

prob **+=** ti\_e[i**+**1] **-** ti\_e[i] **>=** H\_UP\_MIN

**for** j **in** range(1,N):

prob **+=** tj\_s[j**+**1] **-** tj\_s[j] **>=** H\_DOWN\_MIN

prob **+=** tj\_e[j**+**1] **-** tj\_e[j] **>=** H\_DOWN\_MIN

*# 牵引重量限制*

**for** i **in** range(1,M**+**1):

prob **+=** lpSum(weight[k**-**1]**\***x\_up[i,j,k] **for** k **in** range(1,MOTIVE\_NUM**+**1) **for** j **in** range(1,N**+**2)) **>=** weight\_trainup[i**-**1]

*# 牵引重量限制*

**for** j **in** range(1,N**+**1):

prob **+=** lpSum(weight[k**-**1]**\***x\_down[j,i,k] **for** k **in** range(1,MOTIVE\_NUM**+**1) **for** i **in** range(1,M**+**2)) **>=** weight\_traindown[j**-**1]

*# 求解问题*

prob**.**solve()

**if** LpStatus[prob**.**status] **==** "Optimal":

print("找到最优解")

**else**:

print("No solution")

*# 满足车辆数量<=4的情况下，总空闲最短时间（求解时间约15分钟）*

print(f"Best Motive number:{value(TotalMotive)}")

print(f"Shorest wait time:{value(TotalWait)}")

**import** matplotlib.pyplot **as** plt

**if** LpStatus[prob**.**status] **==** "Optimal":

**for** k **in** range(1,MOTIVE\_NUM**+**1):

print(f"Train {k}:")

plt**.**figure(figsize **=** (8,2))

**for** i **in** range(0,M**+**1):

**for** j **in** range(1, N**+**1):

**if** value(x\_up[i,j,k]) **==** 1:

*# 绘制带箭头的线段*

dx **=** value(tj\_e[j]) **-** value(tj\_s[j])

dy **=** **-**2

plt**.**arrow(value(tj\_s[j]), 2, dx, dy, head\_width**=**0.1, head\_length**=**0.5, fc**=**'red', ec**=**'red')

*# 添加文字标签*

plt**.**text(value(tj\_s[j])**+**dx**/**2, 1, f"{downtrain**.**iloc[j**-**1]['车次']}", ha**=**'center', va**=**'center')

print(f"执行线路{downtrain**.**iloc[j**-**1]['车次']}（down j={j}）,离开A时间:{min2time(value(tj\_s[j]))} 到达B时间{min2time(value(tj\_e[j]))} ")

**for** j **in** range(0,N**+**1):

**for** i **in** range(1, M**+**1):

**if** value(x\_down[j,i,k]) **==** 1:

*# 绘制带箭头的线段*

dx **=** value(ti\_e[i]) **-** value(ti\_s[i])

dy **=** 2

plt**.**arrow(value(ti\_s[i]), 0, dx, dy, head\_width**=**0.1, head\_length**=**0.5, fc**=**'green', ec**=**'green')

plt**.**text(value(ti\_s[i])**+**dx**/**2, 1, f"{uptrain**.**iloc[i**-**1]['车次']}", ha**=**'center', va**=**'center')

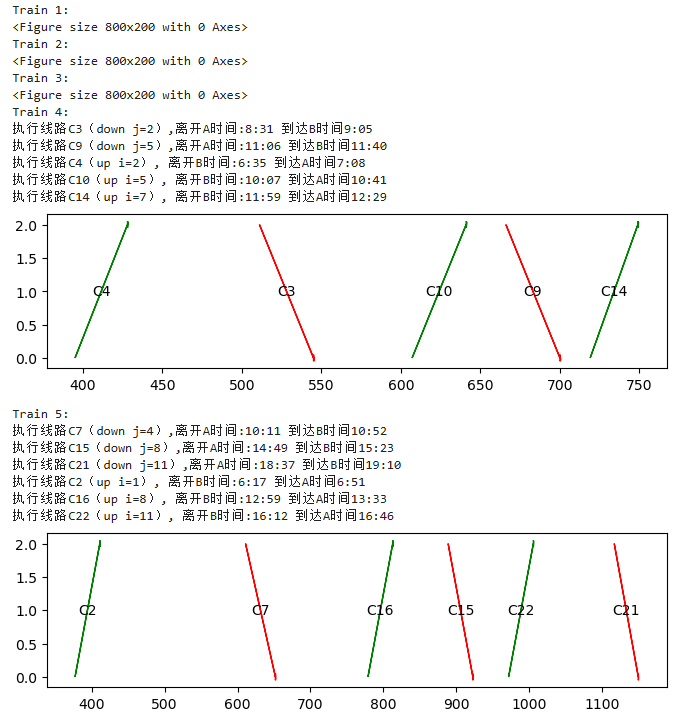
print(f"执行线路{uptrain**.**iloc[i**-**1]['车次']}（up i={i}）, 离开B时间:{min2time(value(ti\_s[i]))} 到达A时间{min2time(value(ti\_e[i]))} ")

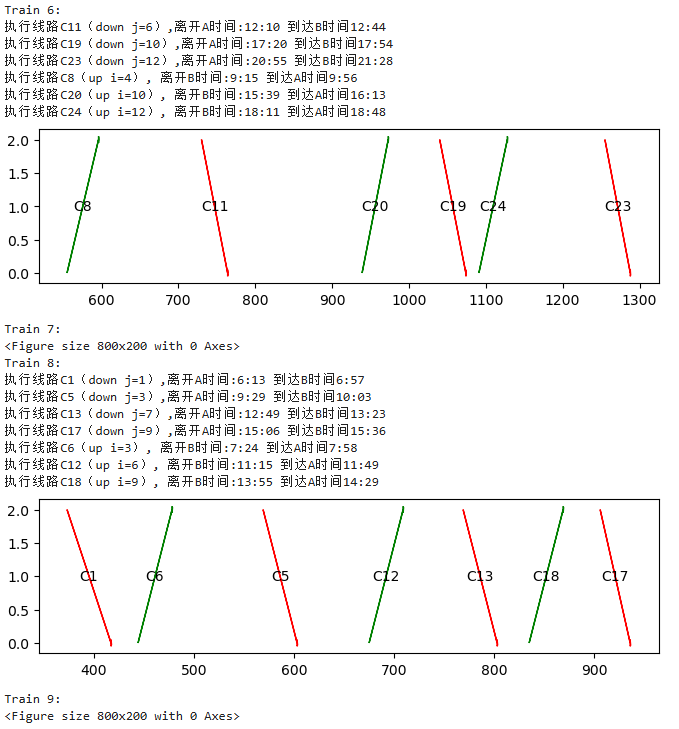
plt**.**show()

**else**:

print("No solution")

**求解结果为：最短怠速时间是1591.0分钟，对应的调度方案为：**





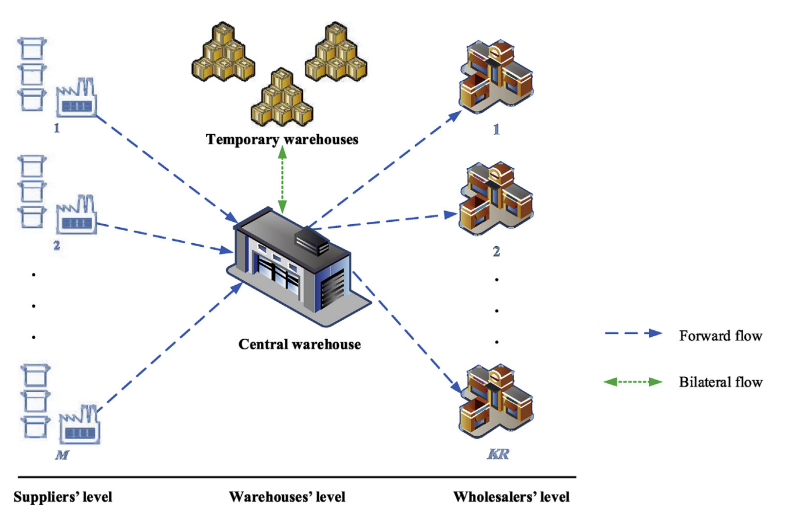
（5）因为是整数规划问题，无法使用Excel的灵敏度分析工具，可以进行手动灵敏度分析

## 案例10：持续可靠的供应链设计（第四章：整数规划；难度：难）

**案例背景：**

供应链管理在现代商业环境中具有重要意义。通过优化物流、降低成本、提高效率，企业可以在激烈的市场竞争中脱颖而出。良好的供应链管理不仅可以降低风险，提升服务质量，还能促进合作伙伴关系，提升企业竞争力。在追求可持续发展的今天，供应链管理也越来越受到关注，帮助企业实现经济效益的同时，也注重环境和社会责任。

在这里，考虑一个只有三层的供应链，该供应链由供应商、中央仓库仓库和批发商组成。整条供应链共有7家供应商（Supplier 1- Supplier 7），16家批发商（Wholesaler 1- Wholesaler 16），供应产品有3种（Product 1-Product 3），供应链供应4个时期（Period 1-Period 4）。供应链的示意图如下图所示：



中央仓库从供应商处得到商品，中央仓库可以自由选择是否从某家供应商处订购某商品。同时，中央仓库将这些商品发送给下游的批发商。由于批发商的需求在现实条件下会波动，当中央仓库存储空间不足时，中央仓库也会租用临时仓库。

供应商（Supplier）层面，供应商在每个时期对每种产品的供应能力各不相同，订购每种产品的价格、每次订购的固定成本也不相同。供应商i的每批次产品存在也一定的缺陷率。不同供应商由于其在产品成本、供应准时性上的表现不同，所被赋予的权重也不同。在上述方面表现较好的供应商、权重也相对较高，下表展示了各供应商的权重：

|  |  |
| --- | --- |
| Supplier | 供应商权重 |
| 1 | 0.169 |
| 2 | 0.129 |
| 3 | 0.117 |
| 4 | 0.105 |
| 5 | 0.099 |
| 6 | 0.098 |
| 7 | 0.082 |

仓库（Warehouse）层面，中央仓库在每个时期都有一个最大容量，同时，每个时期中央仓库最多租用5个临时仓库。无论是中央仓库还是临时仓库，一个仓库都可以同时存储多种产品，产品之间的存储互相不干扰。中央仓库和临时仓库在一个时期内持有单位数量的产品j的持有成本均为。仓库中产品j的次品率不得高于。在每个时期，中心仓库从供应商处订购的产品总数不得超过其产能；同时，中央仓库的发送给供应商的订单通常不会立即满足，供应商需要一定的时间生产和准备商品，这个时间称为“产品设置时间”。通常而言，在一个时期内，对于某种商品，各供货商的产品设置时间之和不超过该产品最大交付周期限制。

批发商（Wholesaler）层面，批发商在每个时期对不同产品有一定的需求量需求，同时，产品到从仓库运输到批发商需要支付一定的运输费用。当中央仓库的不能满足批发商的需求时，会产生销售损失成本。

同时，必须考虑到，有时供应商和中央仓库的活动可能会出现延迟、中断或突然变化，从而影响供应过程的可靠性。在周期 的第 i 个供应商活动中发生故障所需的时间应该是一个指数分布后的变量，参数为 。这里，表示第 i 个供应商在第 t 个周期的故障率。因此，第 i 个供应商的可靠性如下:

其中是未发生故障的最短时间。同样，为第t个周期的中央仓库故障定义了参数，其可靠性表达式也定义得类似：

相关的参数如下表所示：

**Table 14:供应商i在时期t的故障率参数λit:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| λit | Period 1 | Period 2 | Period 3 | Period 4 |
| Supplier 1 | 0.4232 | 0.2544 | 0.3696 | 0.334 |
| Supplier 2 | 0.4652 | 0.1592 | 0.3648 | 0.1316 |
| Supplier 3 | 0.1608 | 0.12 | 0.1736 | 0.4196 |
| Supplier 4 | 0.2984 | 0.11 | 0.3512 | 0.1204 |
| Supplier 5 | 0.3748 | 0.3344 | 0.166 | 0.1148 |
| Supplier 6 | 0.3688 | 0.35 | 0.4184 | 0.2128 |
| Supplier 7 | 0.2124 | 0.2256 | 0.1868 | 0.4836 |

**Table 15:第t个周期中央仓库故障率的指数λ' t**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Period 1 | Period 2 | Period 3 | Period 4 |
| λ' t | 0.8176 | 0.8734 | 0.8798 | 0.8842 |

需要注意的是，这里与和不同。故障率和衡量了供应的稳定性。而和则衡量了供应一批产品的缺陷率（次品率）。两者不是一个概念。

为了发挥供应链的最大性能，供应链的运营方希望实现下面的3个目标：

**目标1：整个供应链的成本最低。**包括产品采购成本、产品订购成本、持有产品成本、中央仓库销售给批发商的损失成本、产品总运输成本和临时仓库建立成本的供应链总成本。

**目标2：优先考虑高优先级供应商的订单。**将从供应商i购买的货物数量乘以该供应商的权重作为目标，并希望最大化该目标，实现对供应商优先级与供应数量的综合考量。

**目标3：供应链的可靠性较好。**与反映了供应商和仓库的可信度，当，降低时，供应商和仓库的可信度与升高。与目标2类似，当希望在从Period 1到Period 4期间，能够最大化整个系统（供应商和仓库）的可信度，优先选择可信度较高的供应商和时期订购和存储货物。与目标2类似，可以将Period t从Supplier i订购的货物数量乘以该时期的可信度、中央仓库在第 t 个期间发往批发商 k 的第 j 个产品数量乘以该时期中央仓库的可信度，上述两部分所有时期相加作为整个系统从Period 1到Period 4的总体可信度。

1. 请建立供应链成本的线性规划模型，使得整条供应链的成本最低。
2. 请建立优先考虑高优先级供应商的订单的模型，使得供应商的权重与从供应商订购的商品数量加权和最大。
3. 请建立供应链的可靠性线性规划模型，使得供应链整体的可靠性最大。
4. 供应链在运行过程中，希望（1）-（3）中的目标同时实现，但在实际情况下，上述三个目标可能不能同时取到最优。在实际情况中，设三个目标的权重分别为0.5，0.3与0.2。请结合（1）-（3）中的结果，建立线性规划模型，并利用所给的数据，求出最佳的供应链运行方案。

|  |  |
| --- | --- |
| 目标 | 权重 |
| 目标1（供应链的成本最低） | 0.5 |
| 目标2（优先考虑高优先级供应商的订单） | 0.3 |
| 目标3（供应链的可靠性较好） | 0.2 |

如果你的目标函数是求最小值，你可以考虑：min( 0.5\*目标1 -0.3\*目标2 -0.2 \* 目标3)

1. 在现实生产中，供应商i产品j的单价采购价格、批发商k对产品j在第t个周期的需求的变化通常会对供应链造成较大的影响。请结合（2）中给出的模型，讨论上述两个变量在-10%至10%之间的波动对三个目标的影响。

**案例解答：**

本案例问题可大致描述为供应商、仓库、批发商的三方购货运货问题，同时整个过程又分为四个时期

对于供应商而言，每个时期对每种产品的供应能力各不相同，订购每种产品的价格、每次订购的固定成本也不相同。供应商的每批次产品也同时存在一定的缺陷率。并且不同供应商由于其在产品成本、供应准时性上的表现不同，所被赋予的权重也不同

而对于仓库而言，总体分为中央仓库和临时仓库两部分。中央仓库在每个时期都有一个最大容量，同时，每个时期临时仓库有租用数量上限。两种仓库皆可同时存储多种产品并且各自存储互相不干扰。每种产品的单位储存成本固定，且认为将某时期的期末库存作为该时期的平均库存水平。此外仓库中产品的次品率不得高于仓库能接收的最大次品率，并且任一时期，中心仓库从供应商处订购的产品总数不得超过其产能。该案例还引入“产品设置时间”概念，代表供应商需要生产和准备商品的时间。该时间不能超过产品最大交付周期限制。

对于批发商而言，在每个时期对不同产品有一定的需求量需求，如果当仓库的不能满足批发商的需求时，会产生销售损失成本。其中，批发商需要支付产品到从仓库运输到的运输费用。案例又引入了 “可靠性”概念，用具体表达式衡量供应的稳定性。

经分析，该案例是一个稍微复杂的整数规划型的生产运货问题，同时由于对任意供应商的可选择性，更具体可以归为0-1型整数规划问题。首先根据题干信息定义出所需要的所有变量 ：

|  |  |  |
| --- | --- | --- |
| I | 供应商,i = 1, 2, …, 7 | |
| J | 产品,j = 1, 2, 3 | |
| K | 批发商,k = 1, 2, …, 16 | |
| T | 时间段,t = 1, 2, 3，4 | |
| Xijt | 第t 期间向第 i个供应商订购的第 j个产品的数量 | |
| hj | 第j个产品的单位持有成本 | |
| Ijt | 第 t期第 j个产品的库存量 | |
| Pij | 第 i个供应商订购的第 j个产品的采购单价 | |
| Oij | 第 i个供应商第 j个产品的订购成本 | |
| Yijt | 当第t个周期内选择第i个供应商供应第j个产品时，二元变量取值为1；否则，它等于0 | |
| Bkjt | 当第k个批发商对第t 期第 j个产品的需求得到满足时，二元变量取值为 1；否则等于0 | |
| LO’jt | 中央仓库第t期末第 j个产品的销售损失数量 | |
| LWj | 中央仓库第j个产品的单位销售成本损失 | |
| DOk | 产品到批发商的单位运输成本k | |
| Xkjt | t期间中央仓库向批发商k发送的第 j个产品的数量 | |
| OXt | 第t期建立的临时仓库数量 | |
| PO | 每个临时仓库的建立成本 |
| Wi | 第i个供应商的权重 |
| λit | 第t期供应商i故障率的指数分布参数 |
| λ’t | 第t个周期中央仓库故障率的指数 |
| τ | 在供应链没有失效的某一段时间 |
| Ri,t =e-τλit | 第 i 个供应商的可靠性Ri |
| Rt=e-τλ’t | t期间中央仓库的可信度Rt |
| Djt | t期间中央仓库对j种产品的需求 |
| DKkjt | 批发商k在第t 时期内对第 j个产品的需求 |
| qai | 第i个供应商的平均缺陷率 |
| Qaj | 第j个产品的最大可接受缺陷率 |
| Cijt | 第 i个供应商在第 t时期供应第 j个产品的能力 |
| stij | 第 i个供应商的第 j个产品的准备时间，包括生产时间和交货时间 |
| LTjt | t期间内第 j个产品的最大交货提前期 |
| SCjt | 中央仓库在第 t时期供应第 j个产品的最大容量 |
| SOj | 每个临时仓库供应第 j个产品的最大容量 |
| PO | 每个临时仓库的建立成本 |
| OXt | 第t期建立的临时仓库数量 |
| M | 一个极大数 |

列出题目的约束条件：

1．∑7i=1Xijt+Ijt-1-∑16k=1XKkjt = Ijt ∀j∈J;∀t∈T

该等式定义了库存平衡概念。也就是说，每期末每种产品的库存必须等于从供应商处采购的产品总数加上上一期的库存减去运输给批发商的产品总数

2．Djt = ∑16k=1DKkjt\*Bkjt ∀j∈J;∀t∈T

计算每个时期每种产品的需求

3．∑7i=1(Xijt\*qai)<=Qaj\*Djt ∀j∈J;∀t∈T

表示定性约束，保证订购产品的平均缺陷率不超过一定限度

4．Xijt<= Cijt ∀i∈I;∀j∈J;∀t∈T

向供应商订购的产品总数不得超过其产能

5．∑7i=1stijYijt <= LTjt ∀j∈J;∀t∈T

保证在考虑的交付时间内提交产品

6．∑7i=1Xijt+Ijt-1 <= SCjt+SOj\*OXt ∀j∈J;∀t∈T

根据中央仓库的最大供应能力，确定是否建立临时仓库

7．XKkjt+LO’jt = DKkjt ∀j∈J;∀k∈K;∀t∈T

对于每个时期，确保从中央仓库运输到每个批发商的产品总数加上其销售损失必须等于该产品的批发商的需求

8．OXt <= 5 ∀t∈T

保证建立的临时仓库的数量不应超过允许的临时仓库的最大数量

9．Xijt <= M\*Yijt ∀i∈I;∀j∈J;∀t∈T

表示订单数量和订单分配变量之间的关系

10．Xijt , XKkjt , Ijt , LO’jt , Djt >= 0 ; OXt ∈ Z+ ; Yijt , Bkjt ∈ {0,1} ; ∀i∈I;∀j∈J;∀t∈T;∀k∈K（变量约束）

之后根据问题，建立各个目标的目标函数：

**目标 1：整个供应链的成本最低**

包括产品采购成本、产品订购成本、持有产品成本、中央仓库销售给批发商的损失成本、产品总运输成本和临时仓库建立成本的供应链总成本：

MinZ1= ∑3j=1∑7i=1∑4t=1(Pij\*Xijt)+∑3j=1∑4t=1(hj\*Ijt)+∑7i=1∑3j=1∑4t=1(Oij\*Yijt)+∑3j=1∑4t=1(LWj\*LO’jt)+∑16k=1∑3j=1∑4t=1(DOk\*Xkjt)+∑4t=1(PO\*OXt)

其中：

采购成本：∑3j=1∑7i=1∑4t=1(Pij\*Xijt)

产品订购成本：∑3j=1∑4t=1(hj\*Ijt)

持有产品成本：∑7i=1∑3j=1∑4t=1(Oij\*Yijt)

中央仓库销售给批发商的损失成本：∑3j=1∑4t=1(LWj\*LO’jt)

产品总运输成本：∑16k=1∑3j=1∑4t=1(DOk\*Xkjt)

临时仓库建立成本的供应链总成本：∑4t=1(PO\*OXt)

**目标 2：优先考虑高优先级供应商的订单**

供应商购买的货物数量乘以该供应商的权作为目标，并希望最大化该目标

MaxZ2= ∑7i=1∑3j=1∑4t=1(Wi\*Xijt)

**目标 3：供应链的可靠性较好**

希望在整个时期，最大化整个系统（包括供应商和仓库）的可信度，优先选择可信度较高的供应商和时期订购和存储货物。将订购的货物数量乘以该时期的可信度、仓库发往批发商的产品数量乘以仓库的可信度，上述两部分所有时期相加作为系统整个时期的总体可信度

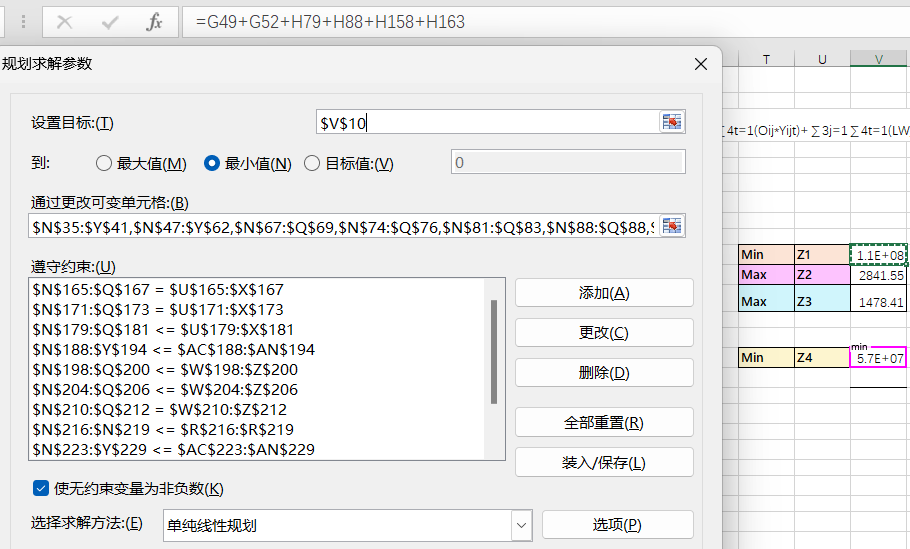
MaxZ3= ∑7i=1∑3j=1∑4t=1(e-τλit\*Xijt)+∑16k=1∑3j=1∑4t=1(e-τλ’t\*XKkjt)

分析完题意后，回答案例问题：

（1）建立的目标函数为：

MinZ1= ∑3j=1∑7i=1∑4t=1(Pij\*Xijt)+∑3j=1∑4t=1(hj\*Ijt)+∑7i=1∑3j=1∑4t=1(Oij\*Yijt)+∑3j=1∑4t=1(LWj\*LO’jt)+∑16k=1∑3j=1∑4t=1(DOk\*Xkjt)+∑4t=1(PO\*OXt)

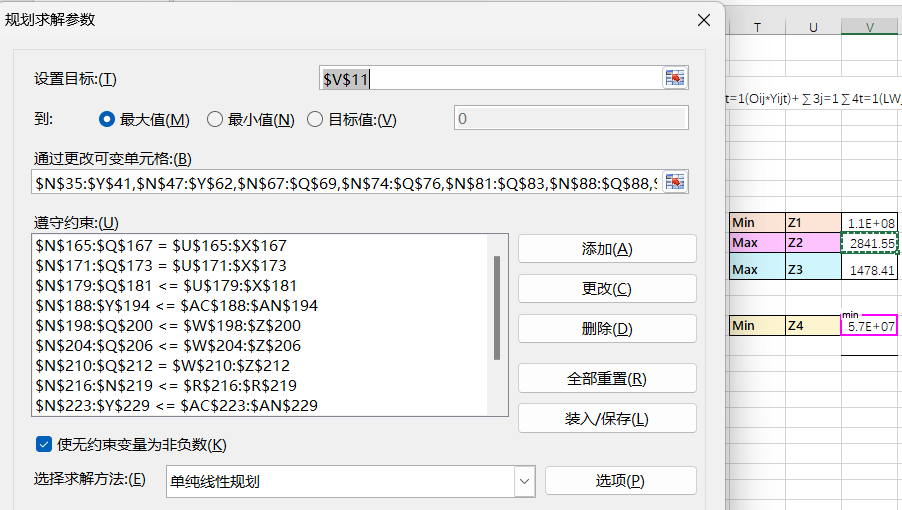
Excel求解：



（2）目标函数为：

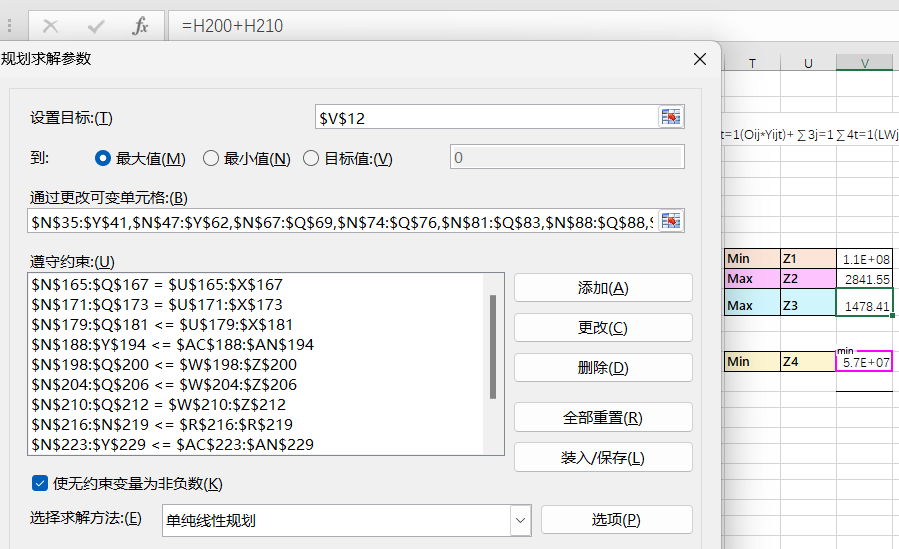
MaxZ2= ∑7i=1∑3j=1∑4t=1(Wi\*Xijt)

Excel求解：

****

**（3）目标函数为：**

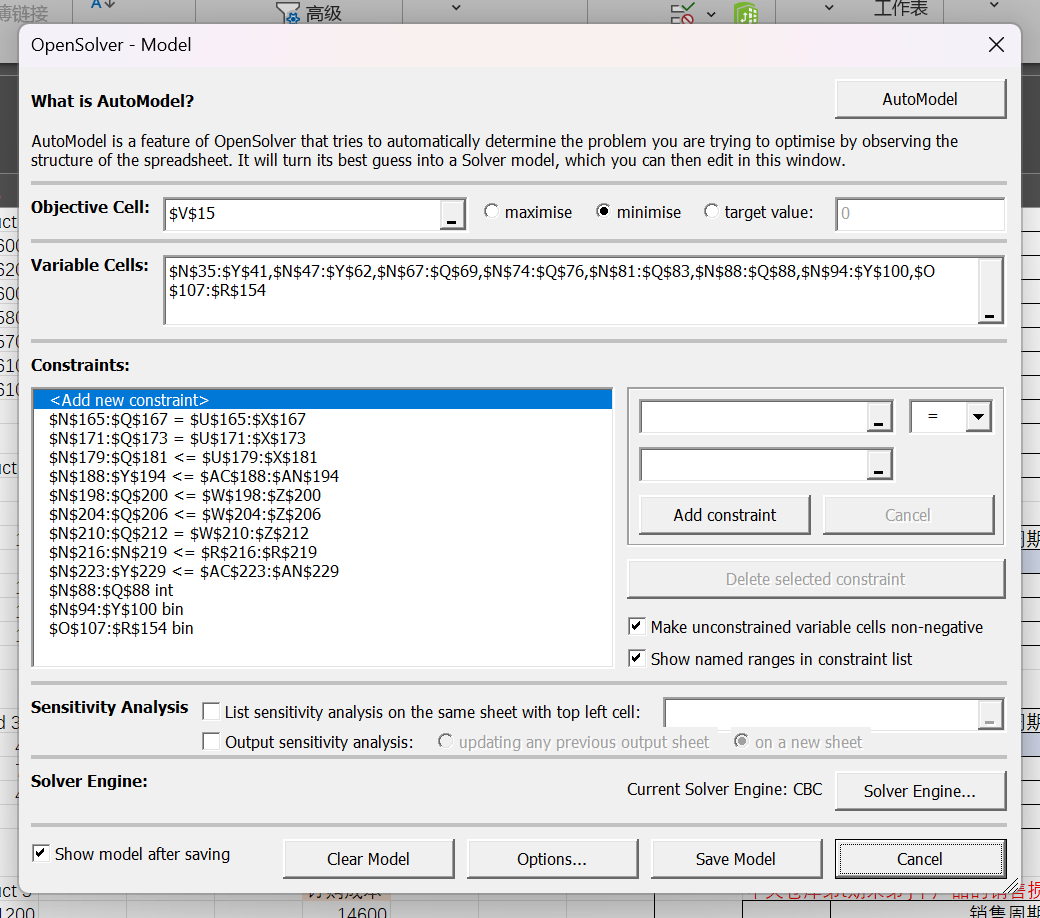
MaxZ3= ∑7i=1∑3j=1∑4t=1(e-τλit\*Xijt)+∑16k=1∑3j=1∑4t=1(e-τλ’t\*XKkjt)

****

**（4）**根据前三题并结合题干做出目标函数：

MinZ4= 0.5\*Z1-0.3\*Z2-0.2\*Z3

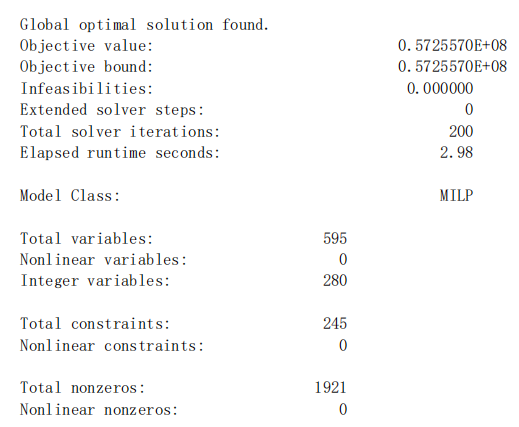
由于可变单元格过多，在excel安装opensolver，并使用此工具进行求解

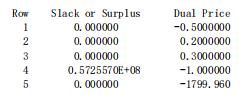


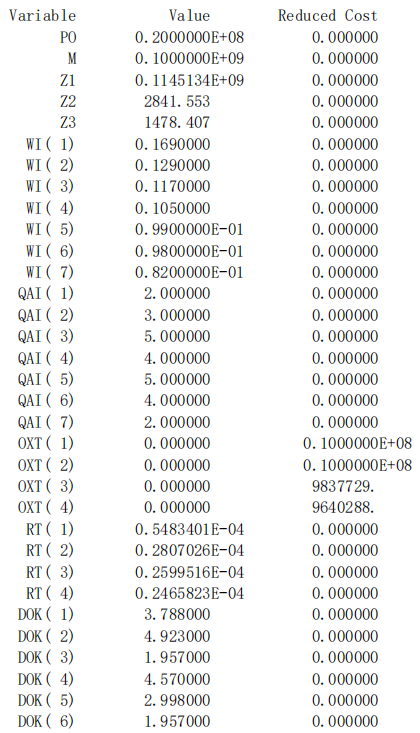
Excel计算取得结果

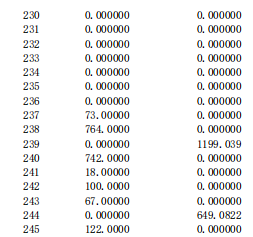


结合lingo建模工具再次进行求解对结果进行验证（图片为部分截图）

****

****

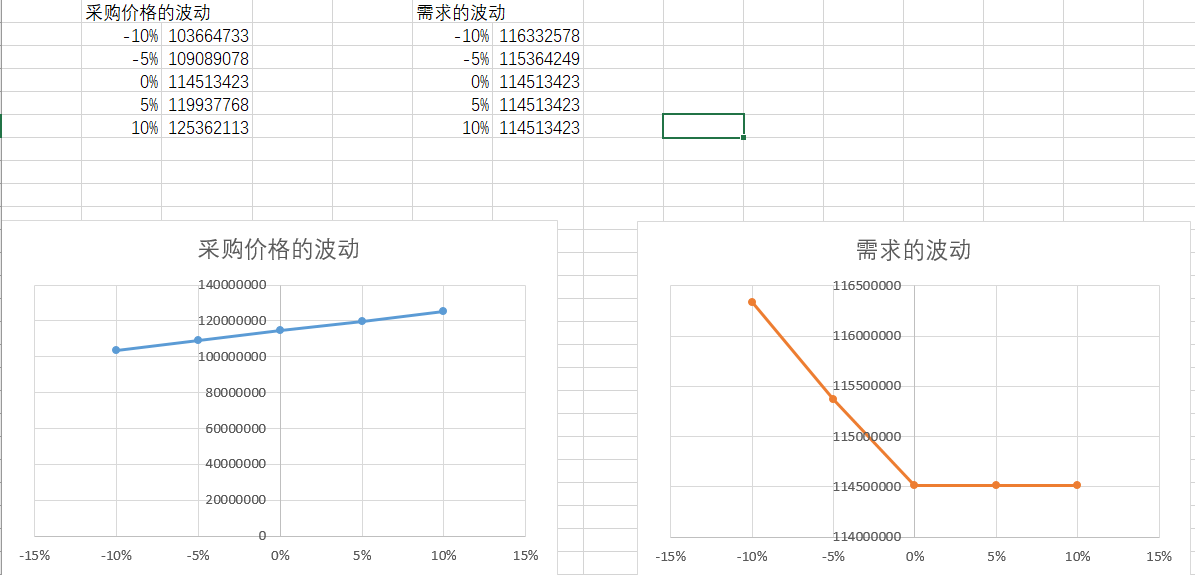
****

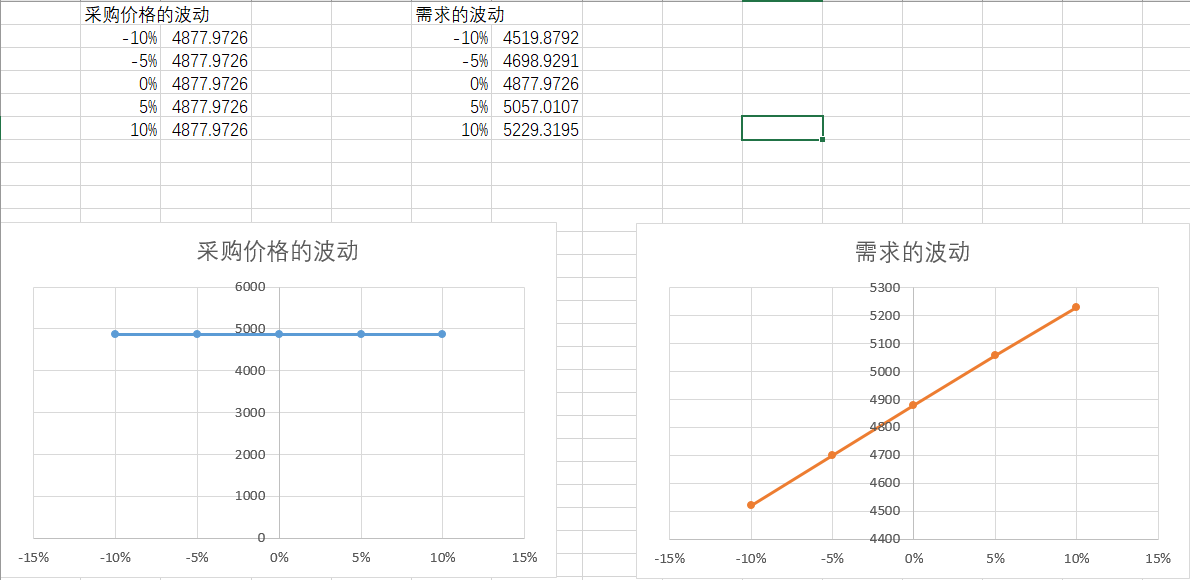
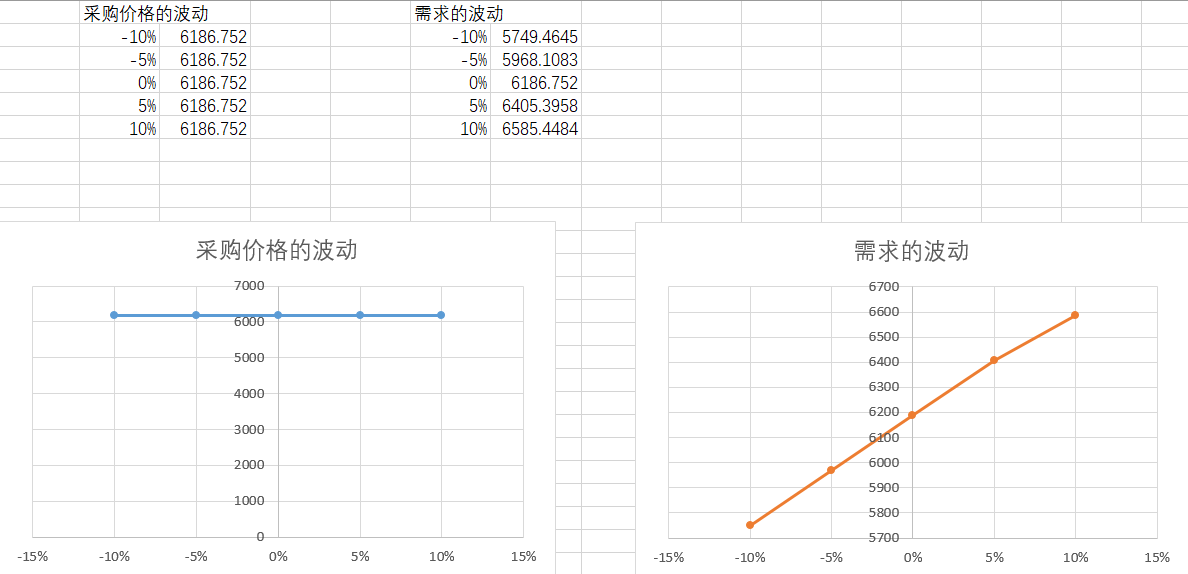
****

综上所述，得到第4题结果

最佳的供应链运行方案为**57255563**

**（5）**将采购价格的波动与需求的波动设立五个档位，分别改变两个变量，通过opensolver工具进行规划求解，得到三个目标的变化情况，并将其制作为散点图以更加直观的观察两个变量对三个目标的影响

****

****

## 案例11：无人机与货车合作配送（第四章：整数规划；难度：很难）

**案例背景：**

无人机送货已经成为当今快递行业的一大创新。起初，无人机送货的发展历史可以追溯到几年前，随着技术的不断进步，无人机逐渐应用于商业领域。目前，一种新的方式是无人机与卡车合作进行送货，实现更高效的配送服务。在这种合作模式下，无人机负责覆盖较短距离范围内的小包裹送货，而卡车则为需要大件物品或超出无人机飞行范围的顾客提供服务。

针对这种合作模式，有三种不同的调度方式：传统方法是快递卡车按照常规路线依次拜访所有顾客；无人机优先送货则是无人机负责将快递送至无人机飞行范围内的所有符合条件的顾客，而当顾客包裹较大或超出无人机飞行范围时，快递卡车便为其提供送货服务；而顾客分配优化则根据顾客位置及包裹情况，将顾客合理分配给无人机或传统快递卡车，以实现最佳的配送效率。这种新型的合作模式和调度方式为快递行业带来更大的便利和效率，也展现了无人机技术在商业应用中的广阔前景。

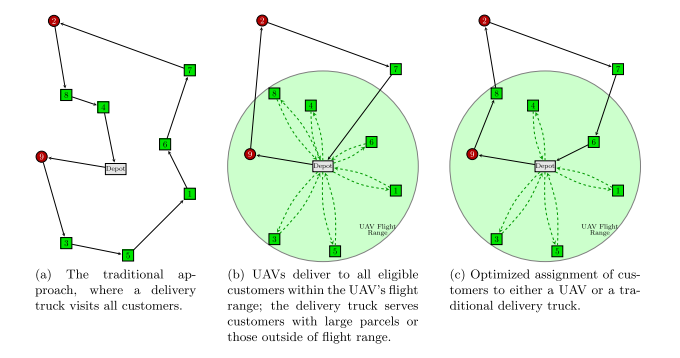


图1： 无人机（UAV）与卡车合作配送的三种模式示例。其中实线代表卡车的路径，虚线代表无人机的路径。绿色圈内的范围代表了以配送中心为原点无人机的最大配送范围。

考虑一种新型的无人机与卡车混合调度方案（如图2所示）。有一组客户集合C = {1, 2, ..., c}，每个客户必须由驾驶员操作的送货卡车或与卡车协调操作的无人驾驶飞机提供一次服务。卡车和无人机必须从配送中心出发并返回。卡车和无人机可以串联或独立出发（或返回）。在串联行驶时，无人机由卡车运输，从而节省电池电量。无人机可以从配送中心（无人机为客户装载包裹的地方）或从客户位置（由卡车司机装载包裹的地方）升空，一旦升空，必须且只能拜访一位客户，并在无人机的飞行续航限制内返回卡车或仓库。在无人机起飞之前，卡车的驾驶员需要为无人机装载包裹，无人机起飞并到达客户位置。无人机离开该客户位置后，无人机必须返回仓库或卡车的位置。如果无人机在卡车上开始或结束任务，驾驶员需要花费一定时间对无人机进行准备（例如装配包裹，更换电池等），设和分别表示卡车准备无人机和回收无人机所需的时间。

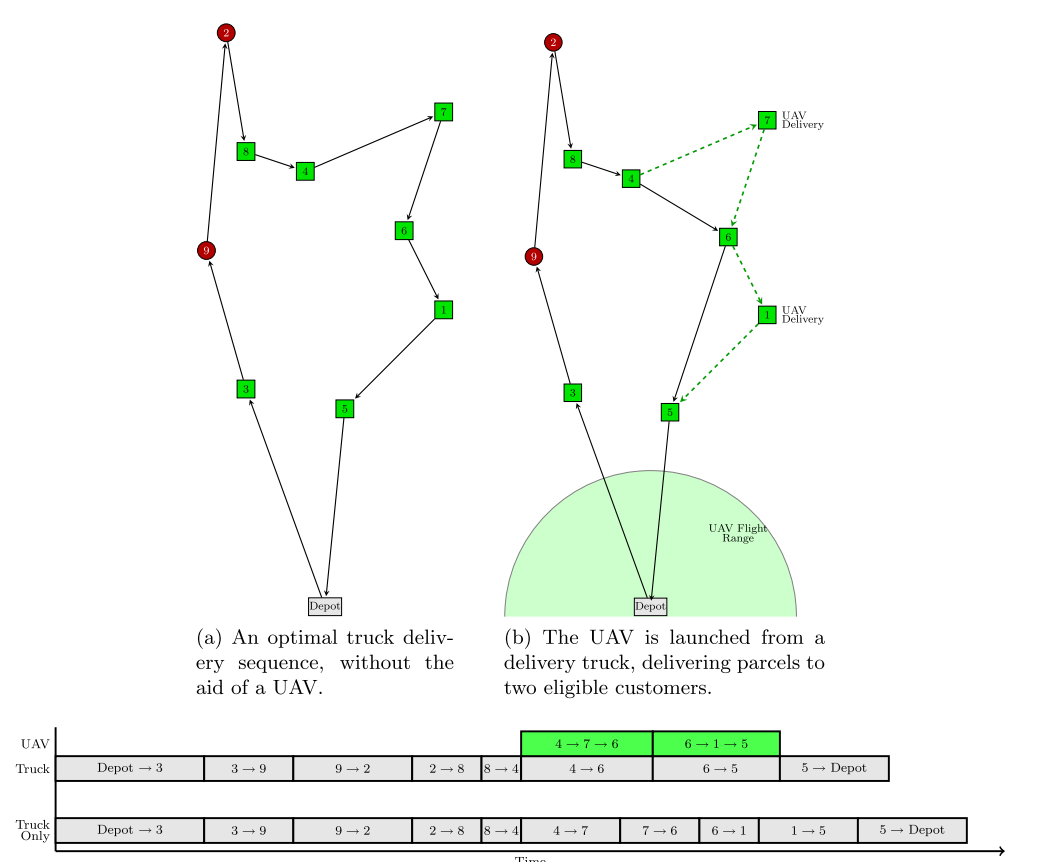


图2：FSTSP的运行示意图。其中实线代表卡车的路径，虚线代表无人机的路径。绿色半圆内的范围代表了以配送中心为原点无人机的最大配送范围。

虽然无人机每次出动可能只访问一个客户，但卡车可以在无人机飞行时访问多个客户。如果无人机先于卡车到达，无人机不能在途中暂时降落。如果无人机在某个客户节点 i 被卡车收集，则无人机可能会从 i 重新发射。但是，如果无人机从 i 发射，它可能不会返回节点 i 处的卡车。 如果无人机配送的最后一段与卡车会合，则必须在卡车服务的客户所在地进行，无人机无法在某个中间位置与卡车重新连接。此外，卡车不得重新访问任何客户节点以取回无人机。

该方案的目标是最大限度地减少为所有客户提供服务并返回仓库所需的时间。为了简化问题，每辆卡车仅配备一架无人机，并且假设所有客户既可以由无人机配送，又可以由卡车进行配送。同时，假设全程无人机的飞行速度与卡车的行驶速度不变，卡车和无人机在客户节点的停留时间不计。

|  |  |
| --- | --- |
| 参数 | 数值 |
| 卡车准备无人机的时间 | 0.1 |
| 卡车回收无人机所需的时间 | 0.1 |
| 无人机单次飞行的最长时间e | 1 |
| 无人机的飞行速度 | 2 |
| 卡车的行驶速度 | 1 |

**思考题：**

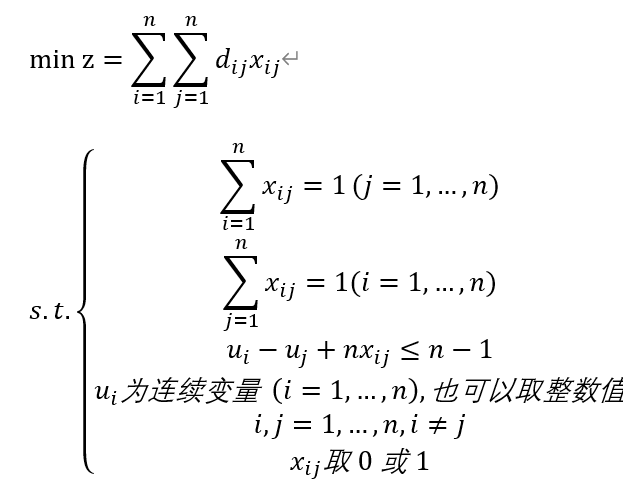
1. 考虑一种简单的情况：不使用无人机配送，而只使用卡车配送所有客户。卡车从配送中心出发，配送完所有客户后，返回配送中心（这个问题通常被成为TSP问题）。请你给出这个问题的优化表达式，使得总的配送时间最小。
2. 请给出该新型的无人机与卡车混合调度方案算法的数学模型，使得总的配送时间最小。
3. 下面给出了配送中心以及5个客户的位置坐标。请你运用（2）中的算法，计算最优的配送方案，并将最短配送时间与（1）中的时间进行比较。

|  |  |  |
| --- | --- | --- |
| 节点名称 | X | Y |
| 配送中心 | 0.668544 | 0.818484 |
| 客户1 | 0.09624 | 0.732234 |
| 客户2 | 0.23265 | 0.888229 |
| 客户3 | 0.497877 | 0.825452 |
| 客户4 | 0.944645 | 0.249912 |
| 客户5 | 0.15626 | 0.148672 |

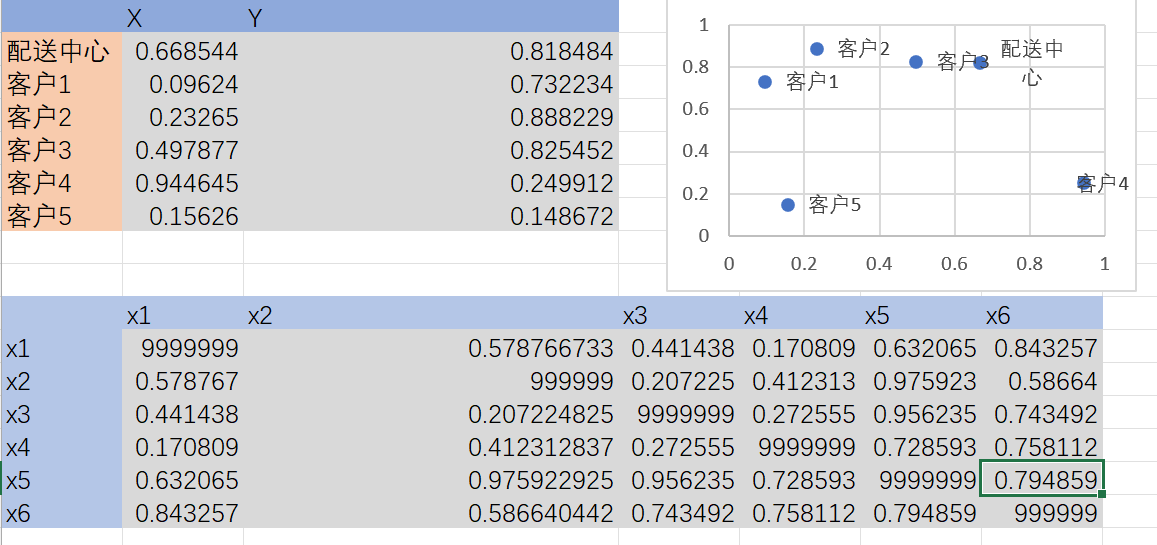
**参考求解：**

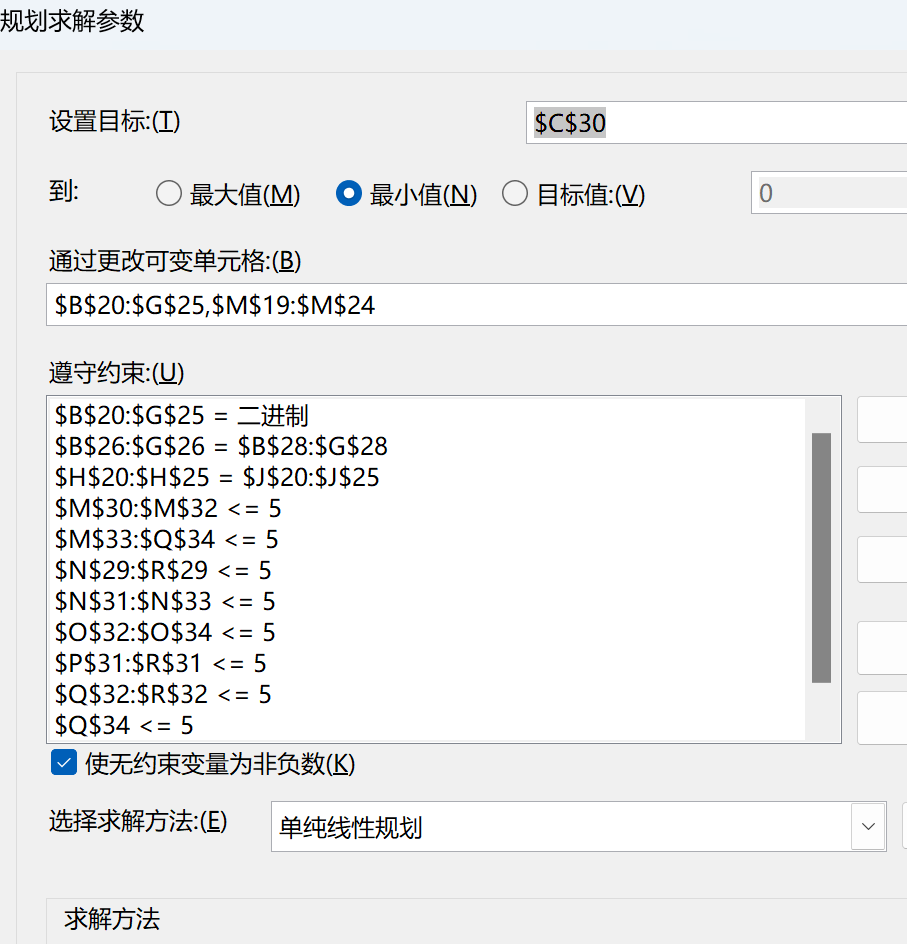
1. **该问题有多种解决方案（如贪心算法、模拟退火算法、遗传算法等），下面展示整数规划的算法**

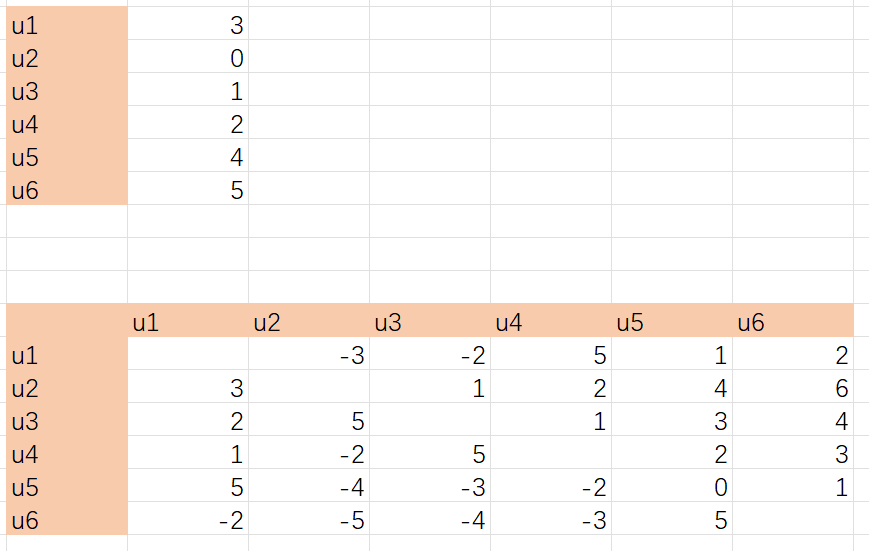
Dij为节点i到j的距离，xij为0-1遍历代表是否从i直接到j，ui为中间参数，让该问题满足无环约束，该问题的数学模型为：

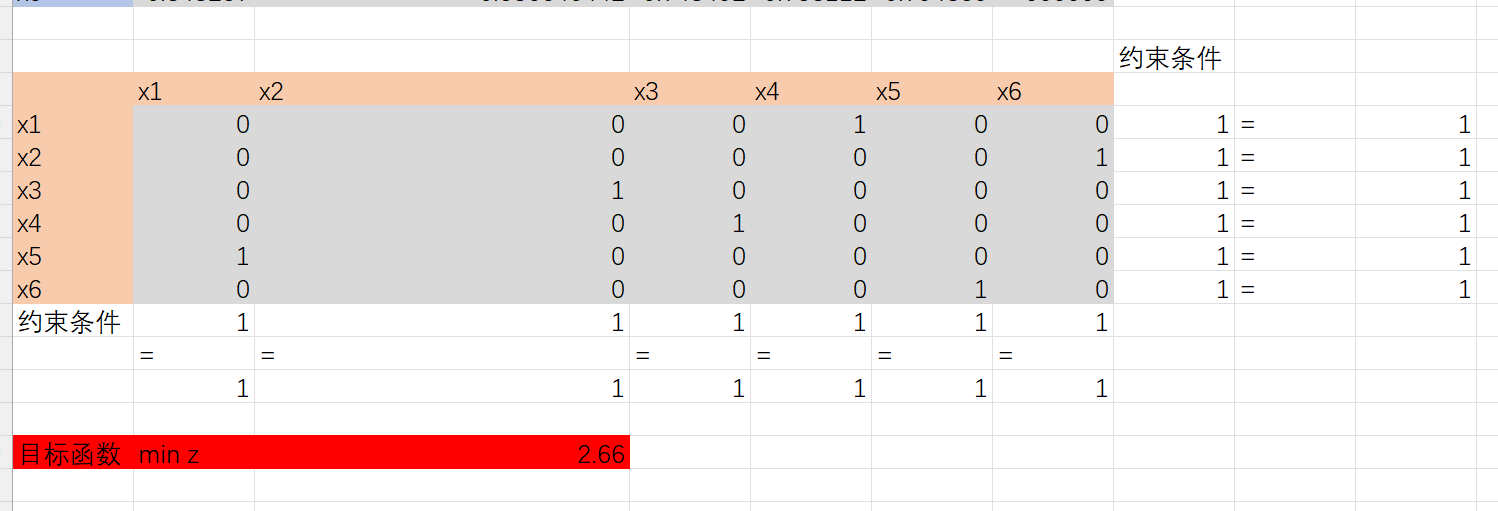


在Excel种进行求解，建模过程与结果如下所示：

****

****

****

****

得到的最短距离为2.66

1. **案例建模过程如下：**



对上图补充条件

1. 
2. 
3. 
4. 将c节点分为0和c+1
5. P表示一个集合，（i,j,k）符合以下条件：

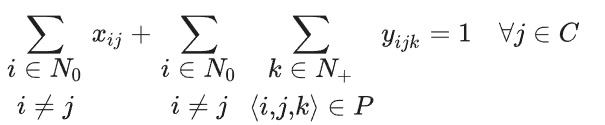
* 发射点i不能是终点仓库节点
* 交付点j必须是无人机可服务的客户，并且不能与发射点i相同
* 交汇点k可以是客户所在地或终点仓库，但不能等于i或j，行驶时间不得超过无人机的续航能力（即，行驶时间 ≤ 无人机续航时间）

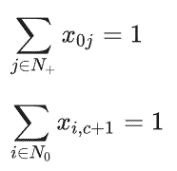
**目标函数：**



**约束条件**

1. 确保每一个节点都被访问





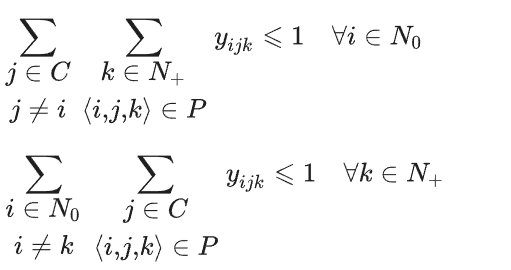
1. 保证访问的次序， ps：如果不进行这一步操作，有可能会造成成环（给出效果图）



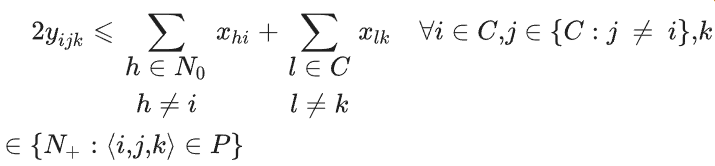
1. 保证到达每个节点的卡车都能从那个节点出去

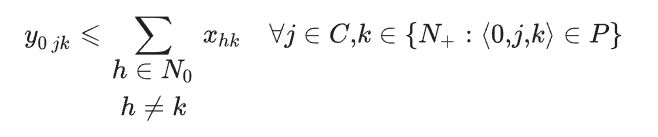


1. 保证到每一个节点只有一个

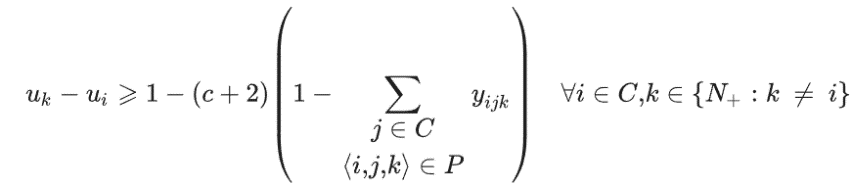


1. 保证无人机的起点和终点卡车都已经到达

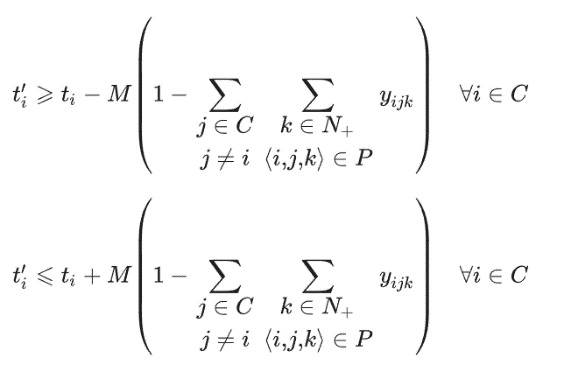


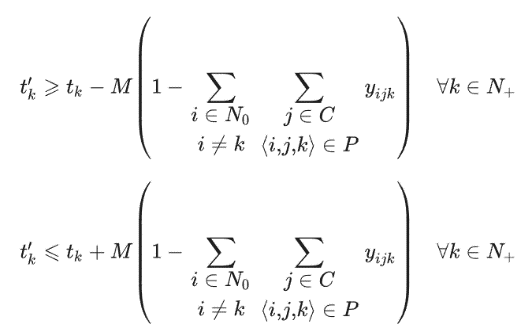


1. 保证放飞无人机的点在回收无人机的点之前

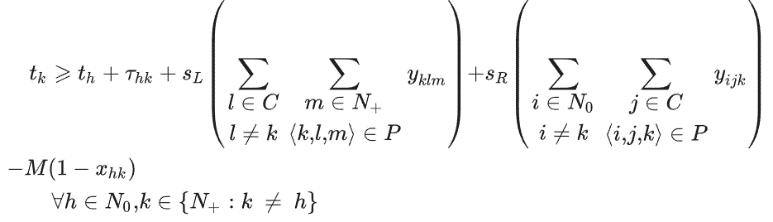


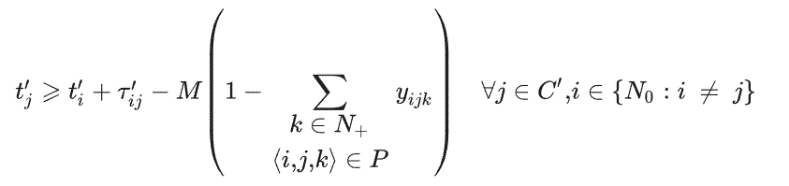
1. 放飞和回收无人机时，无人机的飞到这一点的时间等于卡车到这一点的时间

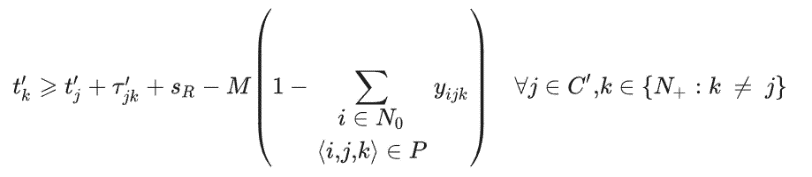




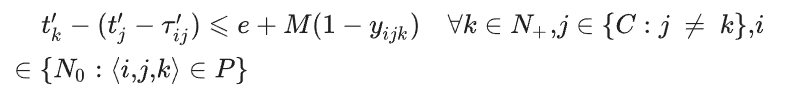
1. 保证无人机和时间约束



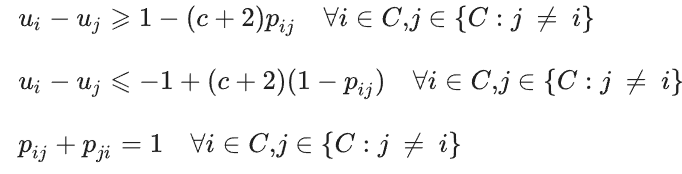


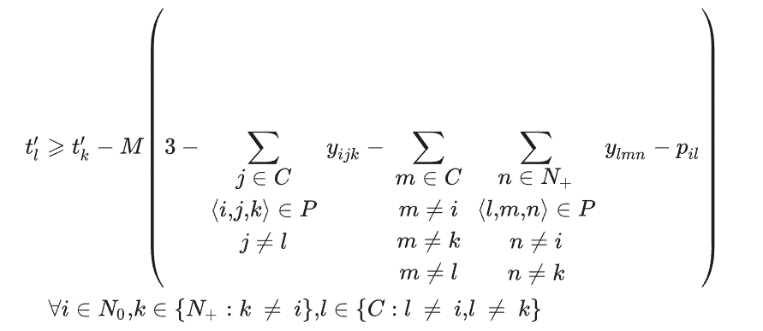


1. 保证续航 e为续航时间

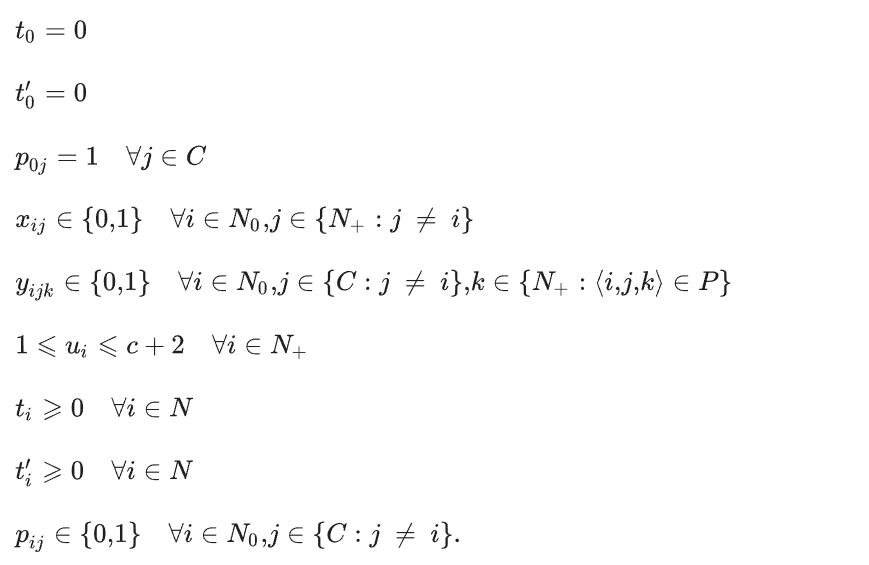


1. 得到一个次序如果两个客户都由卡车访问，并且卡车先访问客户 i 再访问客户 j，那么 p\_ij 就为1，这有助于在后续的优化模型中确保顺序的一致性





1. 其他约束条件



1. 由于上述问题变量较多，使用Python求解较为方便。求解的代码如下：

参考求解代码：

import math

###### Parameters ######

# C = {1,2,...,c} 代表客户

# C' = {1,2,..,c'} 能被无人机服务的客户

# 0--起始点，c+1:终止点

# N0 = {0,1,...,c}出发的节点i

# Nplus = {1,2,...,c+1} 到达的节点j

# i->j 卡车的运行时间t[i,j]

# i->j 无人机的运行时间t1[i,j]

# i = 0,1,...,c; j = 1,2,...,c+1

# t[i,i] ,t1[i,i] 不存在

# t[0,c+1] = 0

# node\_pos: [[0.6685438 0.81848432]

# [0.09623973 0.73223414]

# [0.23265045 0.88822857]

# [0.49787731 0.82545236]

# [0.94464534 0.24991203]

# [0.15625954 0.14867157]

# [0.6685438 0.81848432]]

TRUCK\_SPEED = 1

UAV\_SPEED = 2

times = np.zeros([c+1,c+2])

times1 = np.zeros([c+1,c+2])

for i in range(0,c+1):

for j in range(1,c+2):

if i != j:

d = math.sqrt( (node\_pos[i][0]-node\_pos[j][0])\*\*2 + (node\_pos[i][1]-node\_pos[j][1])\*\*2 )

times[i][j] = d/TRUCK\_SPEED

times1[i][j] = d/UAV\_SPEED

times[0,c+1] = 0

print("times:",times)

print("times1:",times1)

# The prepare time to lauch UAV: sL

sL = 0.1

# The recover time to recycle UAV: sR

sR = 0.1

# The flight endurance time of the UAV : e

e = 1

# <i,j,k> belongs to P when:

# i belongs to N0

# j belongs to C'

# k belongs to Nplus

# i!=j,i!=k,j!=k

# t1[i,j] + t1[j,k] <= e

belongsP = np.zeros([c+2,c+2,c+2])

for i in range(0,c+1):

for j in range(1,c+1):

for k in range(1,c+2):

if (i!= j) and (j != k ) and (i != k) and (times1[i,j] + times1[j,k] <=e):

belongsP[i][j][k] = 1

#print(belongsP)

###### Decision Variables ######

# 定义线性规划问题

from pulp import \*

prob = LpProblem("UAV", LpMinimize)

# x[i,j] (Binary) truck i->j ; i belongs to N0 ,j belongs to Nplus,i!=j

x = LpVariable.dicts('x',[(i,j) for i in range(0,c+1) for j in range(1,c+2) if i != j],cat='Binary')

# y[i,j,k] (Binary) UAV i->j->k .<i,j,k> belongs to P

y = LpVariable.dicts('y',[(i,j,k) for i in range(0,c+1) for j in range(1,c+1) for k in range(1,c+2) if (belongsP[i][j][k] == 1) and (j != i)],cat='Binary')

# t[j] the time truck arrives at j(j belongs to Nplus)

# t1[j] the time UAV arrives at j(j belongs to Nplus)

t = LpVariable.dicts('t',[j for j in range(0,c+2)],lowBound=0,cat='Continuous')

t1 = LpVariable.dicts('t1',[j for j in range(0,c+2)],lowBound=0,cat='Continuous')

######  Auxiliary decision variables ######

# p[i,j] (binary) the truck visit i before j(i != j) i,j belong to C

p = LpVariable.dicts('p',[(i,j)for i in range(0,c+1) for j in range(1,c+2) if i!=j ],cat='Binary')

# 1<=u[i]<=c+2 ,i belongs to Nplus

u = LpVariable.dicts('u',[i for i in range(1,c+2)],cat='Continuous')

prob += t[c+1]

# 约束条件

# （2）

for j in range(1,c+1):

    prob += lpSum(x[i,j] for i in range(0,c+1) if i != j) + lpSum(y[i,j,k] for i in range(0,c+1) for k in range(1,c+2) if belongsP[i][j][k] == 1) ==1

# (3)

prob += lpSum(x[0,j] for j in range(1,c+2)) == 1

# (4)

prob += lpSum(x[i,c+1] for i in range(0,c+1)) == 1

# (5)

for i in range(1,c+1):

    for j in range(1,c+2):

        if i== j :continue

        prob += u[i]-u[j]+1 <= (c+2)\*(1-x[i,j])

# (6)

for j in range(1,c+1):

    prob += lpSum(x[i,j] for i in range(0,c+1) if i != j ) == lpSum(x[j,k] for k in range(1,c+2) if k != j)

#(7)

for i in range(0,c+1):

    prob += lpSum(y[i,j,k] for j in range(1,c+1) for k in range(1,c+2) if (j!=i)and(belongsP[i][j][k] == 1)) <= 1

# (8)

for k in range(1,c+2):

    prob += lpSum(y[i,j,k] for i in range(0,c+1) for j in range(1,c+1) if (i != k)and(belongsP[i][j][k] == 1) ) <= 1

# (9)

for i in range(1,c+1):

    for j in range(1,c+1):

        for k in range(1,c+2):

            if (j!= i) and (belongsP[i][j][k] == 1):

                prob += 2\*y[i,j,k] <= lpSum(x[h,i] for h in range(0,c+1)if h != i) + lpSum(x[l,k] for l in range(1,c+1)if l!= k)

# (10)

for j in range(1,c+1):

    for k in range(1,c+2):

        if belongsP[0][j][k] == 1:

            prob += y[0,j,k] <= lpSum(x[h,k] for h in range(0,c+1) if h != k)

# (11)

for i in range(1,c+1):

    for k in range(1,c+2):

        if k!= i:

            prob += u[k] -u[i] >= 1-(c+2)\*(1-lpSum(y[i,j,k] for j in range(1,c+1) if belongsP[i][j][k] == 1))

#(12)

M = 100000000

for i in range(1,c+1):

    prob += t1[i] >= t[i] - M \* (1-lpSum(y[i,j,k] for j in range(1,c+1) for k in range(1,c+2) if (j!= i)and(belongsP[i][j][k] == 1)))

#(13)

for i in range(1,c+1):

    prob += t1[i] <= t[i] + M \* (1-lpSum(y[i,j,k] for j in range(1,c+1) for k in range(1,c+2) if (j!= i)and(belongsP[i][j][k] == 1)))

# (14)

for k in range(1,c+2):

    prob += t1[k] >= t[k] - M \* (1-lpSum(y[i,j,k] for i in range(0,c+1) for j in range(1,c+1) if (i!= k)and(belongsP[i][j][k] == 1)))

# (15)

for k in range(1,c+2):

    prob += t1[k] <= t[k] + M \* (1-lpSum(y[i,j,k] for i in range(0,c+1) for j in range(1,c+1) if (i!= k)and(belongsP[i][j][k] == 1)))

# (16)

for h in range(0,c+1):

    for k in range(1,c+2):

        if k!=h:

            prob += t[k] >= t[h] + times[h][k] + sL\*(lpSum(y[k,l,m] for l in range(1,c+1) for m in range(1,c+2) if (l != k)and(belongsP[k][l][m] == 1) )) + sR\*lpSum(y[i,j,k] for i in range(0,c+1) for j in range(1,c+1) if (i!=k)and(belongsP[i][k][k] == 1)  ) - M \*(1-x[h,k])

# (17)

for j in range(1,c+1):

    for i in range(0,c+1):

        if i!= j:

            prob += t1[j] >= t1[i] + times1[i][j] - M\*(1-lpSum(y[i,j,k] for k in range(1,c+2) if belongsP[i][j][k] == 1))

# (18)AAA

for j in range(1,c+1):

    for k in range(1,c+2):

        if k!= j:

            prob += t1[k] >= t1[j] + times1[j][k] + sR - M\*(1-lpSum(y[i,j,k] for i in range(0,c+1) if belongsP[i][j][k] == 1))

#(19)

for k in range(1,c+2):

    for j in range(1,c+1):

        for i in range(0,c+1):

            if(j != k)and (belongsP[i][j][k] == 1) :

                prob += t1[k] - ( t1[j] - times1[i][j]) <= e + M\*(1-y[i,j,k])

# (20)(21)(22)

for i in range(1,c+1):

    for j in range(1,c+1):

        if i != j:

            prob += u[i] - u[j] >= 1-(c+2)\* p[i,j]

            prob += u[i] - u[j] <= -1 + (c+2) \* (1-p[i,j])

            prob += p[i,j]+p[j,i] == 1

#(23)

for i in range(0,c+1):

    for k in range(1,c+2):

        for l in range(1,c+1):

            if (k!=i)and(l!=i)and(l!=k):

                prob += t1[l] >= t1[k] - M\*(3-lpSum(y[i,j,k] for j in range(1,c+1) if (belongsP[i][j][k] == 1 )and(j != l ))- lpSum(y[l,m,n] for m in range(1,c+1) for n in range(1,c+2)if (m!=i)and(m!=k)and(m!=l)and(n!=i)and(n!=k)and(belongsP[l][m][n] == 1)) - p[i,l]  )

# （24）(25)

prob += t[0] == 0

prob += t1[0] == 0

#(26)

for j in range(1,c+1):

    prob += p[0,j] == 1

# （29）

for i in range(1,c+2):

    prob += u[i] <= c+2

    prob += u[i] >= 1

prob.solve()

if LpStatus[prob.status] == "Optimal":

    print("找到最优解")

else:

    print("No solution")

# 打印配送方案

print("Time cost:",value(t[c+1]))

print(f"Start:{0},end:{c+1}")

for i in range(0,c+1):

for j in range(1,c+2):

if i!= j and value(x[i,j] ) == 1:

print(f"Truck: {i}（{ value(t[i]):.2f}）->{j}（{value(t[j]):.2f}）")

for i in range(0,c+1):

for j in range(1,c+1):

for k in range(1,c+2):

if (belongsP[i][j][k] == 1) and (j != i) and value(y[i,j,k]) == 1:

print(f"UAV:{i}({value(t1[i]):.2f})->{j}({value(t1[j]):.2f})->{k}({value(t1[k]):.2f})")

# 打印配送方案

print("Time cost:",value(t[c+1]))

print(f"Start:{0},end:{c+1}")

for i in range(0,c+1):

for j in range(1,c+2):

if i!= j and value(x[i,j] ) == 1:

print(f"Truck: {i}（{ value(t[i]):.2f}）->{j}（{value(t[j]):.2f}）")

for i in range(0,c+1):

for j in range(1,c+1):

for k in range(1,c+2):

if (belongsP[i][j][k] == 1) and (j != i) and value(y[i,j,k]) == 1:

print(f"UAV:{i}({value(t1[i]):.2f})->{j}({value(t1[j]):.2f})->{k}({value(t1[k]):.2f})")

**求解结果为：**

Time cost: 1.7875301

Start:0,end:6

Truck: 0（0.00）->2（0.89）

Truck: 1（1.10）->3（1.62）

Truck: 2（0.89）->1（1.10）

Truck: 3（1.62）->6（1.79）

UAV:0(0.00)->5(0.42)->2(0.89)

UAV:2(0.89)->4(1.37)->6(1.79)

**对应的方案为：**

卡车路线为：

0（发点）（时间0.00）->2（时间0.89）->1（时间1.10）->3（时间1.62）->6（终点）（时间1.79）

无人机线路为：

0(放出，时间0.00)->5(送货，时间0.42)->2(收回，时间0.89)

2(放出，时间0.89)->4(送货，时间1.37)->6(收回，时间1.79)

1. 结合（1）的求解结果2.66，（2）提出的模型所需时间1.78更短，因此（2）的方案更具有时间上的优势。

## 案例12：Braneast 航空公司的航线安排（第六章：图论；难度：中等）

**案例背景：**

Braneast 航空公司必须确定应当有多少飞机服务于波士顿一纽约一华盛顿空中走廊，以及要安排哪些航班。表1给出该公司每天可以安排的航班。飞机降落后，至少要在机场停留1小时后才能再次出发。必须确保这个模型包括运营飞机的固定成本。该公司希望能够设计一种运营方案，使该公司日利润最大。

表1 航空公司每天安排的航班

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 离开城市 | 离开时间 | 到达城市 | 到达时间 | 收入/美元 | 航班的可变成本/美元 |
| 纽约 | 上午9点 | 华盛顿 | 上午10点 | 900 | 400 |
| 纽约 | 下午2点 | 华盛顿 | 下午3点 | 600 | 350 |
| 纽约 | 上午10点 | 波士顿 | 上午11点 | 800 | 400 |
| 纽约 | 下午4点 | 波士顿 | 下午5点 | 1200 | 450 |
| 华盛顿 | 上午9点 | 纽约 | 上午10点 | 1100 | 400 |
| 华盛顿 | 下午2点 | 纽约 | 下午3点 | 900 | 350 |
| 华盛顿 | 上午10点 | 波士顿 | 中午12点 | 1500 | 700 |
| 华盛顿 | 下午5点 | 波士顿 | 晚上7点 | 1800 | 900 |
| 波士顿 | 上午 10点 | 纽约 | 上午11点 | 900 | 500 |
| 波士顿 | 下午2点 | 纽约 | 下午3点 | 800 | 450 |
| 波士顿 | 上午11点 | 华盛顿 | 下午1点 | 1100 | 600 |
| 波士顿 | 下午3点 | 华盛顿 | 下午5点 | 1200 | 650 |

1. 该航空公司至少要准备多少架飞机，才能让开行上述所有航线？请建立飞机数量最少的线性规划模型，并求解该模型。（假设所有飞机都正常运行）
2. 假设该航空公司拥有的飞机数量总数为4，飞机抵达某一机场后，需要至少停留在该机场1小时，才能进行下一段航程。请设计一种运营方案，使得该公司日利润最大。
3. 如果飞机停在机场超过1小时，机场将向航空公司收取每架次每小时50美元的机位占用费用。收取时间从上午9:00至晚上7:00（其他时间不收取）。请问航空公司的最大利润与运营方案是否会发生变化？

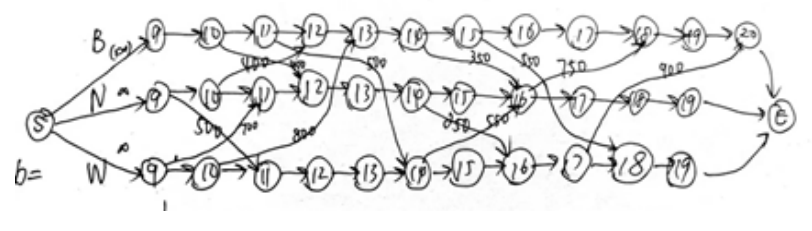
案例解答

1. 建立该问题的数学模型：
2. 参数设置

机场编号（1-波士顿,2-纽约，3-华盛顿）

:出发机场编号 :到达机场编号 出发时间 到达时间

节点:机场，时刻代表的节点。



:虚拟的起始节点S :虚拟的终止节点E

到的最大流量 >即从机场飞往机场，起飞时间为,到达时间为的航线的最大流量。

到的利润 >即从机场飞往机场，起飞时间为,到达时间为的航线的利润

代表了所有可行航线的集合。

代表了包括中的所有航线，以及起始节点S与结束节点E的虚拟航线的集合。

1. 决策变量

:到的流量

1. 目标函数

机队数量最少(最小化总流量)

1. 约束：

航线的流量不超过最大流量限制:

除起始节点S和结束节点E外，其他的每个节点的流入流量=流出流量：

起始节点S的流出流量=结束节点的流入流量：

由于变量较多，因此使用python进行求解，求解的代码为：

*# 定义节点之间的最大流量*

**import** numpy **as** np

flow **=** np**.**zeros([3,3,14,14])

*#i ,j,ts,te = 1,1,8,9*

*#print(flow[i-1][j-1][ts-8][te-8])*

*# 航线的最大流量是1*

flow[0][1][10**-**8][12**-**8] **=** 1

flow[0][1][14**-**8][16**-**8] **=** 1

flow[0][2][11**-**8][14**-**8] **=** 1

flow[0][2][15**-**8][18**-**8] **=** 1

flow[1][0][10**-**8][12**-**8] **=** 1

flow[1][0][16**-**8][18**-**8] **=** 1

flow[1][2][9 **-**8][11**-**8] **=** 1

flow[1][2][14**-**8][16**-**8] **=** 1

flow[2][0][10**-**8][13**-**8] **=** 1

flow[2][0][17**-**8][20**-**8] **=** 1

flow[2][1][9 **-**8][11**-**8] **=** 1

flow[2][1][14**-**8][16**-**8] **=** 1

*# 在机场内部的最大流量没有限制flow[i,i,t,t+1] = BIGM*

BIGM **=** 9999996

**for** i **in** range(0,3):

**for** t **in** range(9,19):

flow[i][i][t**-**8][t**+**1 **-** 8] **=** BIGM

flow[0][0][19**-**8][20**-**8] **=** BIGM

*# 从起始节点S出发去往各个节点的最大流量没有限制*

BIGM **=** 10000000

flow[0][0][8**-**8][9**-**8] **=** BIGM

flow[0][1][8**-**8][9**-**8] **=** BIGM

flow[0][2][8**-**8][9**-**8] **=** BIGM

*# 到达结束节点E的边的最大流量没有限制*

flow[0][0][20**-**8][21**-**8] **=** BIGM

flow[1][0][19**-**8][21**-**8] **=** BIGM

flow[2][0][19**-**8][21**-**8] **=** BIGM

*# 其他边的最大流量为0（即这些边不存在）*

*# 定义各条边的利润/成本*

revenue **=** np**.**zeros([3,3,14,14])

*# 各条航线的利润*

revenue[0][1][10**-**8][12**-**8] **=** 900**-**500

revenue[0][1][14**-**8][16**-**8] **=** 800**-**450

revenue[0][2][11**-**8][14**-**8] **=** 1100**-**600

revenue[0][2][15**-**8][18**-**8] **=** 1200**-**650

revenue[1][0][10**-**8][12**-**8] **=** 800**-**400

revenue[1][0][16**-**8][18**-**8] **=** 1200**-**450

revenue[1][2][9 **-**8][11**-**8] **=** 900**-**400

revenue[1][2][14**-**8][16**-**8] **=** 600**-**350

revenue[2][0][10**-**8][13**-**8] **=** 1500**-**700

revenue[2][0][17**-**8][20**-**8] **=** 1800**-**900

revenue[2][1][9 **-**8][11**-**8] **=** 1100**-**400

revenue[2][1][14**-**8][16**-**8] **=** 900**-**350

*# 定义线性规划问题*

**from** pulp **import** **\***

prob **=** LpProblem("Airlines", LpMinimize)

*# 定义变量*

f **=** LpVariable**.**dicts("f",[(i,j,ts,te) **for** i **in** range(1,3**+**1) **for** j **in** range(1,3**+**1) **for** ts **in** range(8,22) **for** te **in** range(8,22)],lowBound**=**0,cat**=**'Integer')

TotalAirplane **=** LpVariable("TotalAirplane", lowBound**=**0, cat**=**'Integer')

TotalRevenue **=** LpVariable("TotalRevenue",cat**=**'Continuous')

*# 定义目标函数*

prob **+=** TotalAirplane

*# 定义约束条件*

prob **+=** TotalAirplane **==** lpSum(f[1,j,8,9] **for** j **in** range(1,3**+**1))

prob **+=** TotalRevenue **==** lpSum(f[i,j,ts,te] **\*** revenue[i**-**1][j**-**1][ts**-**8][te**-**8] **for** i **in** range(1,3**+**1) **for** j **in** range(1,3**+**1) **for** ts **in** range(8,22) **for** te **in** range(8,22) **if** flow[i**-**1][j**-**1][ts**-**8][te**-**8] **>** 0 )

*# 边<i,j,ts,te>的流量不超过最大流量限制:*

**for** i **in** range(1,3**+**1):

**for** j **in** range(1,3**+**1):

**for** ts **in** range(8,21):

**for** te **in** range(8,21):

prob **+=** f[i,j,ts,te] **<=** flow[i**-**1][j**-**1][ts**-**8][te**-**8]

prob **+=** f[i,j,ts,te] **>=** 0

*# 如果<i,j,ts,te>执行的是航线而不是在机场里面走*

**if** flow[i**-**1][j**-**1][ts**-**8][te**-**8] **==** 1:

prob **+=** f[i,j,ts,te] **==** flow[i**-**1][j**-**1][ts**-**8][te**-**8]

*# 除起始节点S和结束节点E外，其他的每个节点node[i,ts]的流入流量=流出流量*

**for** i **in** range(1,3**+**1):

**for** ti **in** range(9,21):

prob **+=** lpSum(f[h,i,th,ti] **for** h **in** range(1,4) **for** th **in** range(8,21)) **==** lpSum(f[i,j,ti,tj] **for** j **in** range(1,4) **for** tj **in** range(9,22) )

*# 起始节点S的流出流量=结束节点的流入流量*

prob **+=** lpSum(f[1,j,8,9] **for** j **in** range(1,3)) **==** f[1,1,20,21] **+** f[2,1,19,21] **+** f[3,1,19,21]

prob**.**solve()

**if** LpStatus[prob**.**status] **==** "Optimal":

print("找到最优解")

**else**:

print("No solution")

print(value(TotalAirplane))

*# 查看所有航线是否都正常运行*

**for** i **in** range(1,4):

**for** j **in** range(1,4):

**for** ts **in** range(9,20):

**for** te **in** range(9,21):

**if** flow[i**-**1][j**-**1][ts**-**8][te**-**8] **==** 1:

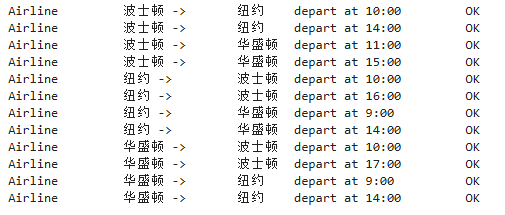
status **=** "OK"

**if** value(f[i,j,ts,te]) **==** 0:

status **=** "Canceled"

print(f"Airline \t{city[i**-**1]} -> \t{city[j**-**1]} \tdepart at {int(ts)}:00 \t{status}" )

求解结果为：**最小飞机数量为6**。输出结果显示，所有的航线都在正常运行



（2）第二问的约束条件与第一问不同，第二位的约束条件为：

利润最高

此外还添加了约束：机队数量为4

使用python进行求解，求解代码为：

*# 定义线性规划问题*

**from** pulp **import** **\***

prob **=** LpProblem("Airlines2", LpMaximize)

*# 定义变量*

f **=** LpVariable**.**dicts("f",[(i,j,ts,te) **for** i **in** range(1,3**+**1) **for** j **in** range(1,3**+**1) **for** ts **in** range(8,22) **for** te **in** range(8,22)],lowBound**=**0,cat**=**'Integer')

TotalAirplane **=** LpVariable("TotalAirplane", lowBound**=**0, cat**=**'Integer')

TotalRevenue **=** LpVariable("TotalRevenue",cat**=**'Continuous')

*# 定义目标函数*

prob **+=** TotalRevenue

*# 定义约束条件*

prob **+=** TotalAirplane **==** lpSum(f[1,j,8,9] **for** j **in** range(1,3**+**1))

prob **+=** TotalRevenue **==** lpSum(f[i,j,ts,te] **\*** revenue[i**-**1][j**-**1][ts**-**8][te**-**8] **for** i **in** range(1,3**+**1) **for** j **in** range(1,3**+**1) **for** ts **in** range(8,22) **for** te **in** range(8,22) **if** flow[i**-**1][j**-**1][ts**-**8][te**-**8] **>** 0 )

prob **+=** TotalAirplane **==** 4

*# 边<i,j,ts,te>的流量不超过最大流量限制:*

**for** i **in** range(1,3**+**1):

**for** j **in** range(1,3**+**1):

**for** ts **in** range(8,21):

**for** te **in** range(8,21):

prob **+=** f[i,j,ts,te] **<=** flow[i**-**1][j**-**1][ts**-**8][te**-**8]

prob **+=** f[i,j,ts,te] **>=** 0

*# 除起始节点S和结束节点E外，其他的每个节点node[i,ts]的流入流量=流出流量*

**for** i **in** range(1,3**+**1):

**for** ti **in** range(9,21):

prob **+=** lpSum(f[h,i,th,ti] **for** h **in** range(1,4) **for** th **in** range(8,21)) **==** lpSum(f[i,j,ti,tj] **for** j **in** range(1,4) **for** tj **in** range(9,22) )

*# 起始节点S的流出流量=结束节点的流入流量*

prob **+=** lpSum(f[1,j,8,9] **for** j **in** range(1,3)) **==** f[1,1,20,21] **+** f[2,1,19,21] **+** f[3,1,19,21]

prob**.**solve()

**if** LpStatus[prob**.**status] **==** "Optimal":

print("找到最优解")

**else**:

print("No solution")

print(value(TotalRevenue))

*# 查看正常运行的航线*

city **=** ["波士顿","纽约","华盛顿"]

**for** i **in** range(1,4):

**for** j **in** range(1,4):

**for** ts **in** range(9,20):

**for** te **in** range(9,21):

**if** flow[i**-**1][j**-**1][ts**-**8][te**-**8] **==** 1:

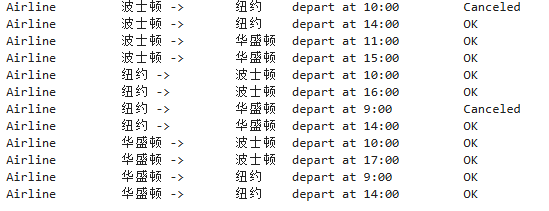
status **=** "OK"

**if** value(f[i,j,ts,te]) **==** 0:

status **=** "Canceled"

print(f"Airline \t{city[i**-**1]} -> \t{city[j**-**1]} \tdepart at {int(ts)}:00 \t{status}" )

计算得到**最大利润为5750.0**，对应的调度方案为：



其中，OK代表该航线正常运营，Canceled代表改航线不运营

（3）若考虑飞机在机场的停机费用，只需调整代表机场内部流动的边的权值。求解代码为：

*# 在机场内部每小时的流动停机成本*

REST\_COST **=** 50

**for** i **in** range(0,3):

**for** t **in** range(9,19):

revenue[i][i][t**-**8][t**+**1 **-** 8] **=** **-**1**\***REST\_COST

revenue[0][0][19**-**8][20**-**8] **=** **-**1**\***REST\_COST

*# 定义线性规划问题*

**from** pulp **import** **\***

prob **=** LpProblem("Airlines2", LpMaximize)

*# 定义变量*

f **=** LpVariable**.**dicts("f",[(i,j,ts,te) **for** i **in** range(1,3**+**1) **for** j **in** range(1,3**+**1) **for** ts **in** range(8,22) **for** te **in** range(8,22)],lowBound**=**0,cat**=**'Integer')

TotalAirplane **=** LpVariable("TotalAirplane", lowBound**=**0, cat**=**'Integer')

TotalRevenue **=** LpVariable("TotalRevenue",cat**=**'Continuous')

*# 定义目标函数*

prob **+=** TotalRevenue

*# 定义约束条件*

prob **+=** TotalAirplane **==** lpSum(f[1,j,8,9] **for** j **in** range(1,3**+**1))

prob **+=** TotalRevenue **==** lpSum(f[i,j,ts,te] **\*** revenue[i**-**1][j**-**1][ts**-**8][te**-**8] **for** i **in** range(1,3**+**1) **for** j **in** range(1,3**+**1) **for** ts **in** range(8,22) **for** te **in** range(8,22) **if** flow[i**-**1][j**-**1][ts**-**8][te**-**8] **>** 0 )

prob **+=** TotalAirplane **==** 4

*# 边<i,j,ts,te>的流量不超过最大流量限制:*

**for** i **in** range(1,3**+**1):

**for** j **in** range(1,3**+**1):

**for** ts **in** range(8,21):

**for** te **in** range(8,21):

prob **+=** f[i,j,ts,te] **<=** flow[i**-**1][j**-**1][ts**-**8][te**-**8]

prob **+=** f[i,j,ts,te] **>=** 0

*# 除起始节点S和结束节点E外，其他的每个节点node[i,ts]的流入流量=流出流量*

**for** i **in** range(1,3**+**1):

**for** ti **in** range(9,21):

prob **+=** lpSum(f[h,i,th,ti] **for** h **in** range(1,4) **for** th **in** range(8,21)) **==** lpSum(f[i,j,ti,tj] **for** j **in** range(1,4) **for** tj **in** range(9,22) )

*# 起始节点S的流出流量=结束节点的流入流量*

prob **+=** lpSum(f[1,j,8,9] **for** j **in** range(1,3)) **==** f[1,1,20,21] **+** f[2,1,19,21] **+** f[3,1,19,21]

prob**.**solve()

**if** LpStatus[prob**.**status] **==** "Optimal":

print("找到最优解")

**else**:

print("No solution")

print(value(TotalRevenue))

*# 查看正常运行的航线*

city **=** ["波士顿","纽约","华盛顿"]

**for** i **in** range(1,4):

**for** j **in** range(1,4):

**for** ts **in** range(9,20):

**for** te **in** range(9,21):

**if** flow[i**-**1][j**-**1][ts**-**8][te**-**8] **==** 1:

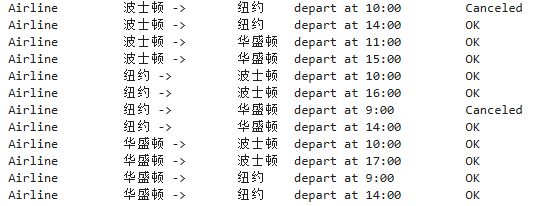
status **=** "OK"

**if** value(f[i,j,ts,te]) **==** 0:

status **=** "Canceled"

print(f"Airline \t{city[i**-**1]} -> \t{city[j**-**1]} \tdepart at {int(ts)}:00 \t{status}" )

运行上述代码，得到**最大利润为5150.0**，对应的运营方案为：



其中，OK代表该航线正常运营，Canceled代表改航线不运营。可以看到,当停机成本上升时，总利润下降了。