19. 最小生成树、基站选址和营养调配问题的 1stOpt 实现

以最小生成树、基站选址和营养调配三个经典运筹学问题为案例，演示如何用 1stOpt 快捷方便解决这些规划问题。

19.1 最小生成树问题

最小生成树或最小权重生成树，有时也叫做最小支撑树，是一副连通加权无向图中一棵权值最小的生成树。如图 19.1 示，A、B、C、D、E、F、G、H、I 和 J 共 10 个结点及结点间连接权重值，红线连接的网络既是一个权值数总和最小的连通路线，也可称之为最小生成树；在保证连接权重值最小的前提下，最小生成树有两个基本的约束要求：一是必须通过每个结点，二是不能有任何连通回路，即不能有任何闭环圈。

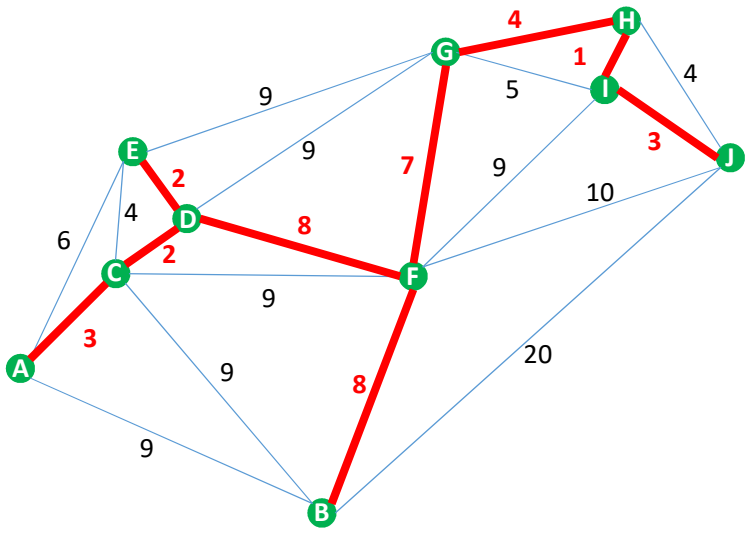

图 19.1 最小生成树示意图

王继强在《科学技术与工程》2021 年第 12 期发表了一篇论文“基于 LINGO 的最小支撑树问题的模型与解法”，该文给出了 Lingo 的相应求解代码，在此即以此论文中的光纤网络铺设问题为案例，用 1stOpt 进行建模求解，并与论文中的 Lingo 进行对比。

光纤网络铺设问题如图 19.2 示，已知某地 P1 至 P8 共 8 个建筑物的位置、相互之间的道路及其长度，通信公司拟为该地铺设高速光纤网络，要求覆盖所有

建筑物，且造价最低，试为该公司设计一个最优的光纤网络铺设方案。该问题实际上就是一最小生成树问题。构建模型如下，有兴趣的可以参照原论文。

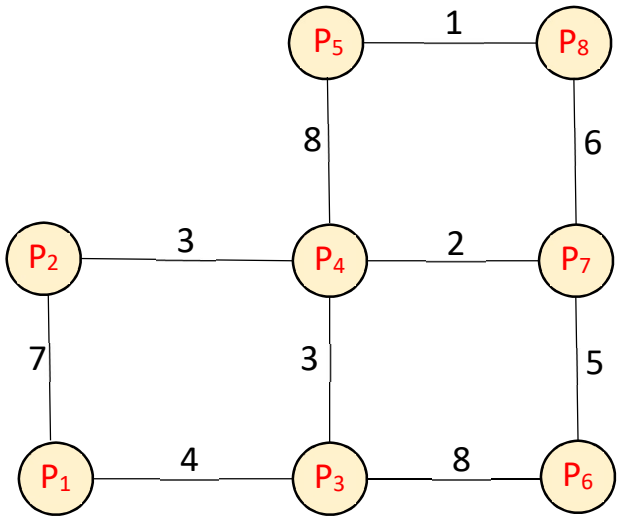

图 19.2 建筑物布局

$$Min. \quad \sum_{i=1}^{n}\sum_{j=1}^{n}(c_{ij}x_{ij}) \tag{19-1}$$

$$S.t. \quad \begin{cases} \sum_{j=2}^{n}(x_{1,j}) \geq 1 \\ \sum_{i=1}^{n}(if(i <> j, x_{i,j})) = 1, j = 2, \ldots, n \\ if(i <> j), u_i - u_j + n \cdot x_{i,j} \leq n-1, j = 1..n, i = 1..n \end{cases} \tag{19-2}$$

其中，n：结点数，为 8；C：结点间连接权重值，如表 19.1，当两结点之间无连接时用一较大的数（远大于所有已知连接权重值中最大者值，比如用 D=100，作用相当于“惩罚”值）表示；x：0-1 决策变量，1 表示两结点相连，反之 0 表示不连接；u：大于 0 的整数辅助变量，用于保证最小生成树的无圈性约束要求。

表 19.1 结点连接权重值表 C

	P1	P2	P3	P4	P5	P6	P7	P8
P1	0	7	4	D	D	D	D	D
P2	7	0	D	3	D	D	D	D
P3	4	D	0	3	D	8	D	D
P4	D	3	3	0	8	D	2	D
P5	D	D	D	8	0	D	D	1
P6	D	D	8	D	D	0	5	D

P7	D	D	D	2	D	5	0	6
P8	D	D	D	D	1	D	6	0

1stOpt 求解代码及原文给出的 Lingo 代码如下：

代码 19-1 1stOpt 求解代码

```
Constant n=8, D=100;
Constant
c(n,n)=[0,7,4,D,D,D,D,D,
        7,0,D,3,D,D,D,D,
        4,D,0,3,D,8,D,D,
        D,3,3,0,8,D,2,D,
        D,D,D,8,0,D,D,1,
        D,D,8,D,D,0,5,D,
        D,D,D,2,D,5,0,6,
        D,D,D,D,1,D,6,0];
BinParameter x(n,n);
Parameter u(n)=[0,];
Algorithm = LP;
MinFunction Sum(i=1:n)(Sum(j=1:n)(c[i,j]*x[i,j]));
            Sum(j=2:n)(x[1,j])>=1;
            For(j=2:n)(Sum(i=1:n)(if(j<>i,x[i,j]))=1);
            For(i=1:n)(For(j=1:n)(if(i<>j,u[i]-u[j]+n*x[i,j]<=n-1)));
```

代码 19-2 Lingo 求解代码

```
model:
sets:
vertex/1..8/: u;
link(vertex, vertex): c,x;
endsets
data:
c = 0 7 4 100 100 100 100 100
7 0 100 3 100 100 100 100
4 100 0 3 100 8 100 100
100 3 3 0 8 100 2 100
100 100 100 8 0 100 100 1
100 100 8 100 100 0 5 100
100 100 100 2 100 5 0 6
100 100 100 100 1 100 6 0;
enddata
min = @sum(link: c*x) ;
@sum(vertex(j)|j#gt#1: x(1,j))>= 1;
@for(vertex(j)|j#gt#1: @sum(vertex(i)|i#ne#j: x(i,j))=1);
n = @size(vertex);
@for(link(i,j)|i#ne#j: u(i)-u(j)+n*x(i,j)<= n - 1);
@for(link: @bin(x));
```

结果

Iterations: 62
Elapsed Time (Hr:Min:Sec:Msec): 00:00:00:14
Algorithms: Simplex Linear Program
Objective Function(Min.): 24
Best Estimated Parameters:

x[1,1]: 0	x[2,5]: 0	x[4,1]: 0	x[5,5]: 0	x[7,1]: 0	x[8,5]: 1
x[1,2]: 0	x[2,6]: 0	x[4,2]: 1	x[5,6]: 0	x[7,2]: 0	x[8,6]: 0
x[1,3]: 1	x[2,7]: 0	x[4,3]: 0	x[5,7]: 0	x[7,3]: 0	x[8,7]: 0
x[1,4]: 0	x[2,8]: 0	x[4,4]: 0	x[5,8]: 0	x[7,4]: 0	x[8,8]: 0
x[1,5]: 0	x[3,1]: 0	x[4,5]: 0	x[6,1]: 0	x[7,5]: 0	u1: 0
x[1,6]: 0	x[3,2]: 0	x[4,6]: 0	x[6,2]: 0	x[7,6]: 1	u2: 3
x[1,7]: 0	x[3,3]: 0	x[4,7]: 1	x[6,3]: 0	x[7,7]: 0	u3: 1
x[1,8]: 0	x[3,4]: 1	x[4,8]: 0	x[6,4]: 0	x[7,8]: 1	u4: 2
x[2,1]: 0	x[3,5]: 0	x[5,1]: 0	x[6,5]: 0	x[8,1]: 0	u5: 7
x[2,2]: 0	x[3,6]: 0	x[5,2]: 0	x[6,6]: 0	x[8,2]: 0	u6: 7
x[2,3]: 0	x[3,7]: 0	x[5,3]: 0	x[6,7]: 0	x[8,3]: 0	u7: 5
x[2,4]: 0	x[3,8]: 0	x[5,4]: 0	x[6,8]: 0	x[8,4]: 0	u8: 6

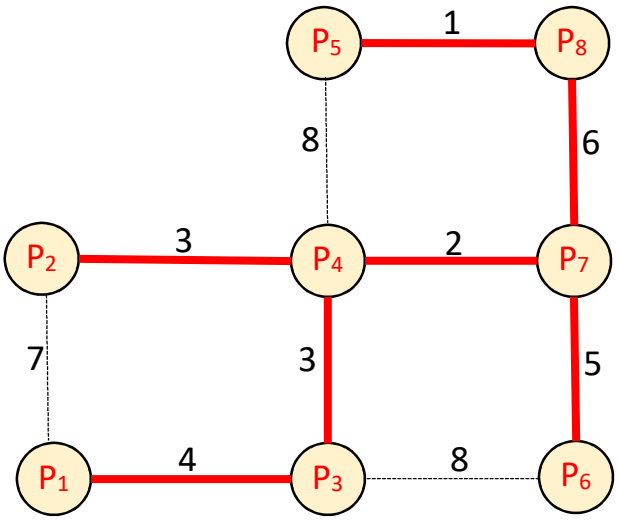

图 19.3 光纤铺设方案

上述 1stOpt 代码运行结果与原论文一致，目标函数值为 24，铺设方案如图 19.3。略有困惑的是上述原文 Lingo 代码执行的结果如下图，目标函数值为 27，不知何故。

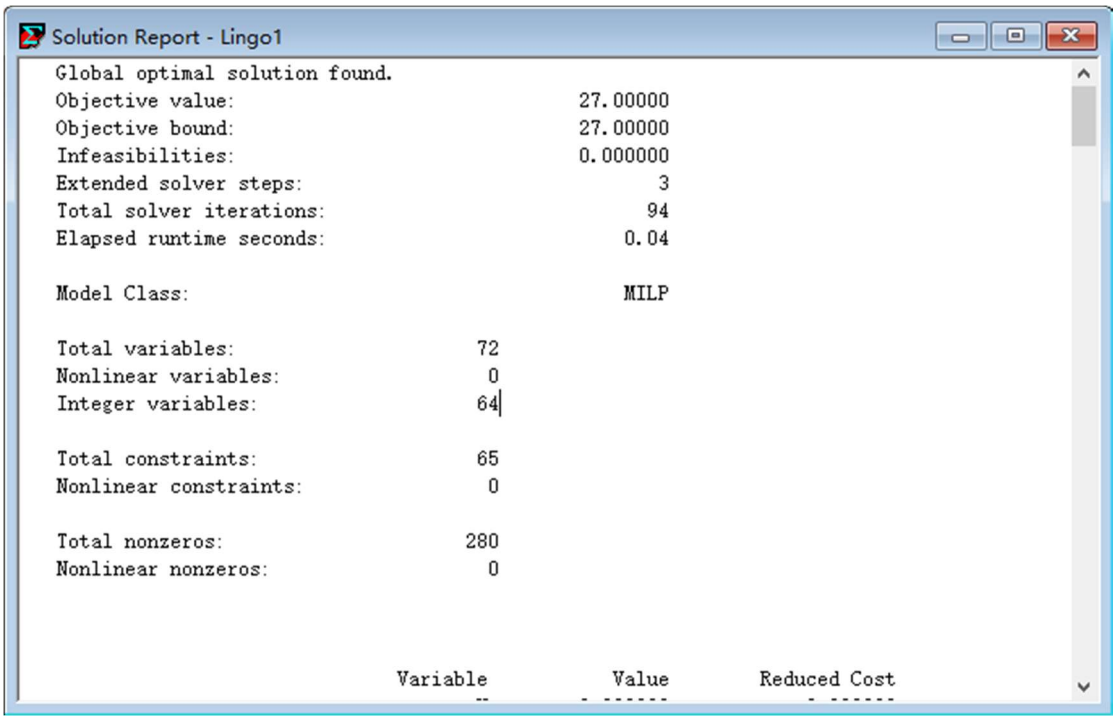

图 19.4 Lingo 计算结果

19.2 基站选址问题

某电话运营商计划在信号尚未覆盖的某区域开展业务，总预算费用为 1000 万。根据调查，该区域共有 15 个社区，7 个位置可以安设基站，每个基站只能覆盖一定数目的社区，具体数据如下，在总预算范围内，如何选择基站位置而使得信号覆盖的人口尽可能多？

表 19.2 每个基站建造费用（百万）和覆盖社区

位置	1	2	3	4	5	6	7
费用	1.8	1.3	4.0	3.5	3.8	2.6	2.1
覆盖社区	1, 2, 4	2, 3, 5	4, 7, 8, 10	5, 6, 8, 9	8, 9, 12	7,10,11,12,15	12,13,14,15

表 19.3 社区居民（千人）

社区	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
人口	2	4	13	6	9	4	8	12	10	11	6	14	9	3	6

$$Max. \quad \sum_{j=1}^{n} p_j y_j \tag{19-3}$$

$$S.t. \quad \begin{cases} \sum_{i=1}^{k} c_i x_i \le M \\ \sum_{i=1}^{k} V_{i,j} x_i \ge y_j, \quad j = 1,2\ldots,k \\ x_i \in 0,1, \quad i = 1,2\ldots,k \\ y_i \in 0,1, \quad i = 1,2\ldots,n \end{cases} \tag{19-4}$$

其中：k=7，可选位置数；n=15，社区数；M=10，总预算费用（百万）

C_i：第 i 个基站建设费用（百万），i=1,2,…k；

P_j: 第 j 个社区人口（千人），j=1,2,…n；

$V_{i,j}$：0-1 变量，取 1 表示第 i 个基站能覆盖第 j 个社区，取 0 则表示不能覆盖；

X_i：0-1 变量，取 1 表示 i 位置件基站，取 0 表示该位置不建，i=1,2,…,k；

y_i：0-1 变量，取 1 表示第 j 个社区能被覆盖，取 0 则表示该社区不覆盖，j=1,2,…,n；

表示第 i 个基站能覆盖第 j 个社区的 V 矩阵数据可由表 19.2 和表 19.3 重新整理获得，如表 19.4

表 19.4 基站覆盖社区数据

社区 j / 基站 i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0
2	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0
3	0	0	0	1	0	0	1	1	0	1	0	0	0	0	0
4	0	0	0	0	1	1	0	1	1	0	0	0	0	0	0
5	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0
6	0	0	0	0	0	0	1	0	0	1	1	1	0	0	1
7	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1

1stOpt 求解代码及结果如下。计算结果表明，在 2、4、6、7 位置设置基站，

最多覆盖人口 10.9 万人，总化费 9.5 百万元费用，社区 1 和 4 没有覆盖。

代码 19-4

```
Constant k=7, n=15, M=10;
Constant C = [1.8 1.3 4 3.5 3.8 2.6 2.1],
          P = [2 4 13 6 9 4 8 12 10 11 6 14 9 3 6];
Constant V(k,n) = [1,1,0,1,0,0,0,0,0,0,0,0,0,0,0,
                   0,1,1,0,1,0,0,0,0,0,0,0,0,0,0,
                   0,0,0,1,0,0,1,1,0,1,0,0,0,0,0,
                   0,0,0,0,1,1,0,1,1,0,0,0,0,0,0,
                   0,0,0,0,0,0,0,1,1,0,0,1,0,0,0,
                   0,0,0,0,0,0,1,0,0,1,1,1,0,0,1,
                   0,0,0,0,0,0,0,0,0,0,1,1,1,1,1];
BinParameter x(k),y(n);
Algorithm = LP;
MaxFunction Sum(j=1:n)(P[j]*y[j]);
                Sum(i=1:k)(C[i]*x[i])<=M;
                For(j=1:n)(Sum(i=1:k)(V[i,j]*x[i])>=y[j]);
```

结果：

Objective Function(Max.): 109	Constrained Functions:
	1:
Best Estimated Parameters:	1.8*x1+1.3*x2+4*x3+3.5*x4+3.8*x5+2.6*x6+2.1*x7-
x1: 0	(10) = -0.5
x2: 1	2: 1*x1+0*x2+0*x3+0*x4+0*x5+0*x6+0*x7-y1 = 0
x3: 0	3: 1*x1+1*x2+0*x3+0*x4+0*x5+0*x6+0*x7-y2 = 0
x4: 1	4: 0*x1+1*x2+0*x3+0*x4+0*x5+0*x6+0*x7-y3 = 0
x5: 0	5: 1*x1+0*x2+1*x3+0*x4+0*x5+0*x6+0*x7-y4 = 0
x6: 1	6: 0*x1+1*x2+0*x3+1*x4+0*x5+0*x6+0*x7-y5 = 1
x7: 1	7: 0*x1+0*x2+0*x3+1*x4+0*x5+0*x6+0*x7-y6 = 0
y1: 0	8: 0*x1+0*x2+1*x3+0*x4+0*x5+1*x6+0*x7-y7 = 0
y2: 1	9: 0*x1+0*x2+1*x3+1*x4+1*x5+0*x6+0*x7-y8 = 0
y3: 1	10: 0*x1+0*x2+0*x3+1*x4+1*x5+0*x6+0*x7-y9 = 0
y4: 0	11: 0*x1+0*x2+1*x3+0*x4+0*x5+1*x6+0*x7-y10 = 0
y5: 1	12: 0*x1+0*x2+0*x3+0*x4+0*x5+1*x6+1*x7-y11 = 1
y6: 1	13: 0*x1+0*x2+0*x3+0*x4+1*x5+1*x6+1*x7-y12 = 1
y7: 1	14: 0*x1+0*x2+0*x3+0*x4+0*x5+0*x6+1*x7-y13 = 0
y8: 1	15: 0*x1+0*x2+0*x3+0*x4+0*x5+0*x6+1*x7-y14 = 0
y9: 1	16: 0*x1+0*x2+0*x3+0*x4+0*x5+1*x6+1*x7-y15 = 1
y10: 1	
y11: 1	
y12: 1	
y13: 1	
y14: 1	
y15: 1	

19.3 营养调配问题

营养调配问题的目标是利用优化模型来设定每日饮食菜单，在满足各类营养需求的同时更能优化总成本.本案例参考自 Jupyter Notebook Viewer (nbviewer.org)。

$$Min. \quad \sum_{j=1}^{n} x_j c_j \tag{19-5}$$

$$S.t. \quad \begin{cases} \sum_{j=1}^{n} (x_j a_{i,j}) \geq L_i, \quad i = 1, 2, \ldots, k \\ \sum_{j=1}^{n} (x_j a_{i,j}) \leq U_i, \quad i = 1, 2, \ldots, k \\ \sum_{j=1}^{n} (x_j v_j) \leq Vmax \end{cases} \tag{19-6}$$

其中

1) 决策变量为 x：芝士汉堡 (CB：Cheeseburger)、火腿三明治 (HS：Ham-sandwich)、汉堡 (H：Hamburger)、鱼肉三明治 (FS：Fish-sandwich)、鸡肉三明治 (CS：Chicken-sandwich)、薯条 (F：Fries)、腊肠饼 (SB：Sausage biscuit)、低脂牛乳 (LM：Low-fat milk)和橙汁 (OJ：Orange juice)；
2) 约束条件为：卡路里 (Cal.)、碳水化合物 (Carbo.)、蛋白质 (Portien)、维生素 A/D (Vit. A/D)、铁 (Iron)和钙质 (Calc.)的每日摄取上/下限制，以及总量 (Volume) 限制；
3) 目标函数则为：总成本的最小化.

表 19.5 每单位食物的成本价和容积量

食物	Cheese burger 芝士汉堡	Ham sandwich 火腿三明治	Ham-burger 汉堡	Fish sandwich 鱼肉三明治	Chicken sandwich 鸡肉三明治	Fries 薯条	Sausage biscuit 腊肠饼	Lowfat milk 低脂牛乳	Orange juice 橙汁
简称	CB	HS	H	FS	CS	F	SB	LM	OJ
成本 C	1.84	2.19	1.84	1.44	2.29	0.77	1.29	0.60	0.72
容量 V	4	7.5	3.5	5	7.3	2.6	4.1	8	12

表 19.6 各营养的单日摄取上限和下限

营养	Cal	Carbo	Protein	VitA	VitC	Calc
下界	2000	355	55	100	100	100
上界	∞	375	∞	∞	∞	∞

表 19.7 各种食物的营养含量

食物	CB	HS	H	FS	CS	F	SB	LM	OJ
Cal	510	370	500	370	400	220	345	110	80
Carbo	34	35	42	38	42	26	27	12	20
Protein	28	24	25	14	31	3	15	9	1
VitA	15	15	6	2	8	0	4	10	2
VitC	6	10	2	0	15	15	0	4	120
Calc	30	20	25	15	15	0	20	30	2
Cal	20	20	20	10	8	2	15	0	2

1stOpt 求解有如下两段代码，该两段代码的不同之处是约束条件上限为∞时的处理方式不同，代码 19-5 是给正无穷上限一个具体的很大的值，比如 M=1E+10，代码 19-6 则给正无穷上限赋值一个标识值如 M=-1，再通过 iff 函数判断，如果 M=-1，则不加该约束；另外决策变量 x 由“IntParameter”定义为正整数，即各食物只能整取而不能分割，如果将代码中的“IntParameter”改为“Parameter”，则可以得到实数解。

代码 19-5

```
Constant k=7, n=9, Vmax=75, M=1E+10;
Constant
C=[1.84,2.19,1.84,1.44,2.29,0.77,1.29,0.6,0.72],
V=[4,7.5,3.5,5,7.3,2.6,4.1,8,12],
L=[2000,350,55,100,100,100,100],
U=[M,375,M,M,M,M,M],
a(k,n)=[510,370,500,370,400,220,345,110,80,
        34,35,42,38,42,26,27,12,20,
        28,24,25,14,31,3,15,9,1,
        15,15,6,2,8,0,4,10,2,
        6,10,2,0,15,15,0,4,120,
        30,20,25,15,15,0,20,30,2,
        20,20,20,10,8,2,15,0,2];
Algorithm = LP;
IntParameter x(n)=[0,];
MinFunction Sum(j=1:n)(x[j]*C[j]);
            For(i=1:k)(Sum(j=1:n)(x[j]*a[i,j])>=L[i]);
            For(i=1:k)(Sum(j=1:n)(x[j]*a[i,j])<=U[i]);
            Sum(j=1:n)(x[j]*V[j])<=Vmax;
```

代码 19.6

```
Constant k=7, n=9, Vmax=75, M=-1;
Constant
C=[1.84,2.19,1.84,1.44,2.29,0.77,1.29,0.6,0.72],
V=[4,7.5,3.5,5,7.3,2.6,4.1,8,12],
L=[2000,350,55,100,100,100,100],
```

```
U=[M,375,M,M,M,M,M],
a(k,n)=[510,370,500,370,400,220,345,110,80,
        34,35,42,38,42,26,27,12,20,
        28,24,25,14,31,3,15,9,1,
        15,15,6,2,8,0,4,10,2,
        6,10,2,0,15,15,0,4,120,
        30,20,25,15,15,0,20,30,2,
        20,20,20,10,8,2,15,0,2];
Algorithm = LP;
IntParameter x(n)=[0,];
MinFunction Sum(j=1:n)(x[j]*C[j]);
            For(i=1:k)(iff(L[i]>0,Sum(j=1:n)(x[j]*a[i,j])>=L[i]));
            For(i=1:k)(iff(U[i]>0,Sum(j=1:n)(x[j]*a[i,j])<=U[i]));
            Sum(j=1:n)(x[j]*V[j])<=Vmax;
```

两段代码可以得到相同的结果如下。

结果：

整数解	实数解
Objective Function(Min.): 15.05	Objective Function(Min.): 14.855737704918
Best Estimated Parameters:	Best Estimated Parameters:
x1: 4	x1: 4.38524590163934
x2: 0	x2: 0
x3: 0	x3: 0
x4: 1	x4: 0
x5: 0	x5: 0
x6: 5	x6: 6.14754098360656
x7: 0	x7: 0
x8: 4	x8: 3.42213114754098
x9: 0	x9: 0

此案例阿里巴巴的达摩院也进行了引用以演示他们自行研发的求解器，参考：doc/example_2.md · master · mindopt / mindopt-open-examples · CODE (aliyun.com)，不过有两处低级错误，一是在“各种食物的营养含量与容积量”数据表格里将火腿三明治 (Ham-sandwich)对应的碳水化合物 (Carbo)的数值由 35 误写成了 55（虽然后面代码里又使用了正确的 35），另外一个就是将英文“sandwich”的汉语时而写成“三明治”，时而又是“三民治”，“Cheeseburger”译成了“起司汉堡”而不是通用的“芝士汉堡”，有时难免令人不明所言，虽无伤大雅，但作为顶级科技公司，严谨一些还是必要的。

1stOpt 计算的整数解结果与原案例一致，实数解结果与阿里巴巴相同。

19.4 小结

运用 1stOpt 对三个典型运筹学问题进行了求解，可以看出 1stOpt 的求解代码直观且简单易懂，非常适合非数学及非计算机专业的普通用户使用。

三个案例均为线性规划问题，1stOpt 代码里均使用了“Algorithm = LP;”，也即算法都使用了线性规划算法，那使用全局非线性优化算法是否可行？可以尝试但不推荐，理论上虽然线性问题可视作非线性问题的特例，但具体应用时线性问题最好选用线性算法，不仅效率高，结果也有保证。“杀鸡启用宰牛刀”在此应改为“杀鸡不用宰牛刀”，尤其是对求解规模较大的线性问题。

参考文献：

王继强．基于 LINGO 的最小支撑树问题的模型与解法[J]．科学技术与工程，2021，21(12)：4995-4998.