20. 匹配及人员分配运筹学问题的 1stOpt 实现

线性规划问题是运筹学重要组成部分，在实际生活中有广泛的应用场景，结合两个经典案例，展示 1stOpt 在处理这类问题上的便捷和强大。

20.1 匹配问题

8 位专家准备搬入新办公楼 4 间办公室，两人一间，根据以往经验，该 8 位专家相互间相处的融洽度从 1 到 9 划分为 9 个等级，如下表示，等级越高表示相处的越好，如何给专家安排办公室使得总体融洽度最高？或工作环境最和谐？

表 20.1 专家融洽度表

专家	1	2	3	4	5	6	7	8
1	-1	9	3	4	2	1	5	6
2	9	-1	1	7	3	5	2	1
3	3	1	-1	4	4	2	9	2
4	4	7	4	-1	1	5	5	2
5	2	3	4	1	-1	8	7	6
6	1	5	2	5	8	-1	2	3
7	5	2	9	5	7	2	-1	4
8	6	1	2	2	6	3	4	-1

如果用 R(i,j)表示融洽度，0-1 类型求解变量 M(i,j)表示专家 i 和 j 是否在一个办公室，1 表示是，0 表示否。可得如下目标函数，因为是对称矩阵，每一组专家都被计算了两次，因此目标函数中有乘 0.5 一项。另外注意专家自己不能和自己组合，因此当 i=j 时的融洽度 R(i,j)，可用一具体数值表示，该数值小于融洽度中的最小值即可，在此取-1，其作用相当于惩罚值.

$$Max. \quad \sum_{i=1}^{8}\left(\sum_{j=1}^{8}(M_{i,j}\cdot R_{i,j})\right)\cdot 0.5 \tag{20-1}$$

约束条件：每行每列的变量 M 之和必需等于 1，表示每一组专家只能组合一次。

$$S.T. \quad \begin{cases} \sum_{j=1}^{8}(m_{i,j}) \quad for \quad i = 1..8 \\ \sum_{i=1}^{8}(m_{i,j}) \quad for \quad j = 1..8 \end{cases} \tag{20-2}$$

根据上述模型描述，求解代码如下：

```
Constant n=8,D=-1;
Constant R(n,n)=[D,9,3,4,2,1,5,6,
                 9,D,1,7,3,5,2,1,
                 3,1,D,4,4,2,9,2,
                 4,7,4,D,1,5,5,2,
                 2,3,4,1,D,8,7,6,
                 1,5,2,5,8,D,2,3,
                 5,2,9,5,7,2,D,4,
                 6,1,2,2,6,3,4,D];
BinParameter M(n,n);
Algorithm = LP;
MaxFunction Sum(i=1:n)(Sum(j=1:n)(M[i,j]*R[i,j]))*0.5;
        For(i=1:n)(Sum(j=1:n)(M[i,j])=1);
        For(j=1:n)(Sum(i=1:n)(M[i,j])=1);
```

结果：

Objective Function(Max.): 30	m[2,7]: 0	m[4,8]: 0	m[7,1]: 0
	m[2,8]: 0	m[5,1]: 0	m[7,2]: 0
Best Estimated Parameters:	m[3,1]: 0	m[5,2]: 0	m[7,3]: 1
m[1,1]: 0	m[3,2]: 0	m[5,3]: 0	m[7,4]: 0
m[1,2]: 0	m[3,3]: 0	m[5,4]: 0	m[7,5]: 0
m[1,3]: 0	m[3,4]: 0	m[5,5]: 0	m[7,6]: 0
m[1,4]: 0	m[3,5]: 0	m[5,6]: 1	m[7,7]: 0
m[1,5]: 0	m[3,6]: 0	m[5,7]: 0	m[7,8]: 0
m[1,6]: 0	m[3,7]: 1	m[5,8]: 0	m[8,1]: 1
m[1,7]: 0	m[3,8]: 0	m[6,1]: 0	m[8,2]: 0
m[1,8]: 1	m[4,1]: 0	m[6,2]: 0	m[8,3]: 0
m[2,1]: 0	m[4,2]: 1	m[6,3]: 0	m[8,4]: 0
m[2,2]: 0	m[4,3]: 0	m[6,4]: 0	m[8,5]: 0
m[2,3]: 0	m[4,4]: 0	m[6,5]: 1	m[8,6]: 0
m[2,4]: 1	m[4,5]: 0	m[6,6]: 0	m[8,7]: 0
m[2,5]: 0	m[4,6]: 0	m[6,7]: 0	m[8,8]: 0
m[2,6]: 0	m[4,7]: 0	m[6,8]: 0	

上述结果表明专家 1 和 8，2 和 4，3 和 7，5 和 6 是最佳组合，融洽度最高达 30。

相反，假设如果想使得工作环境最差，也即和谐度最低，该如何安排？只需将上述代码中的“MaxFunction”改成“MinFunction”，同时将“D=-1”改成“D=100”

即可，最高融洽度仅为 6，组合为 1 和 6，2 和 7，3 和 8 及 4 和 5.

20.2 人员分配问题

某工厂一周每天需要员工数及每天工资费用如下表，要求每个员工每周连续工作 5 天再休息 2 天，这 2 天休息必须是连续的，如何安排员工的工作，即满足工作需要，又使总工资费用最小？

表 20.2 员工需求数据

时间	所需员工数 （x）	工资费用/位 （P）
星期一	27	220
星期二	34	230
星期三	20	230
星期四	25	220
星期五	24	220
星期六	30	250
星期日	23	280

每个员工都需连续工作 5 天，所需总人数并非每天安排人数之和。设 x1, x2,…, x7 代表一周内每天安排的员工数，目标函数很容易表示如下：

$$Min. \quad \sum_{i=1}^{7}(x_i \cdot p_i) \tag{20-3}$$

对于周一而言，除了周二和周三开始工作的之外，其余都会在周一工作，有约束条件：

$$x_1 + x_2 + x_3 + x_6 + x_7 \geq a_1 \tag{20-4}$$

同理对周二至周日有：

$$\begin{cases} x_1 + x_2 + x_5 + x_6 + x_7 \geq a_2 \\ x_1 + x_2 + x_3 + x_6 + x_7 \geq a_3 \\ x_1 + x_2 + x_3 + x_4 + x_7 \geq a_4 \\ x_1 + x_2 + x_3 + x_4 + x_5 \geq a_5 \\ x_2 + x_3 + x_4 + x_5 + x_6 \geq a_6 \\ x_3 + x_4 + x_5 + x_6 + x_7 \geq a_7 \end{cases} \tag{20-5}$$

由上分析，求解代码如下：

```
Constant n=7;                          //天数
Constant a=[27,34,20,25,24,30,23];     //人数
```

```
Constant p=[220,230,230,220,220,250,280];  //费用
IntParameter x(n);
Algorithm = LP;
MinFunction Sum(i=1:n,x,p)(x*p);
x1+x4+x5+x6+x7>=a1;
x1+x2+x5+x6+x7>=a2;
x1+x2+x3+x6+x7>=a3;
x1+x2+x3+x4+x7>=a4;
x1+x2+x3+x4+x5>=a5;
x2+x3+x4+x5+x6>=a6;
x3+x4+x5+x6+x7>=a7;
```

上述求解代码计算没任何问题，但是否还存在可优化改进之处？比如对应连休两日的 7 个约束是有一定规律的：周一的约束缺 x2 和 x3，周二的约束缺 x3 和 x4，。。。，周六的约束缺 x7 和 x1，周日的约束缺 x1 和 x2，周一至周五的约束较好处理，只要判断是否比该日期数大于 1 和 2 即可，但周六与周日的话大于 1 或 2 会出现 8 和 9 的日期数，实际对应周一和周二，也即大于 7 后会重新从头开始，具体数是该日期数减 7，1stOpt 中有专门的对应函数 Wrap(k,n)，当 k 小于等于 n 时返回值为 n，反之，返回值为 k-n，如 Wrap(3,7)=7, Wrap(8,7)=1，由此，结合“for”语句，如果循环天数 n=7, i 表示第 i 日的约束，j 表示第 j 天的员工数，通过增加一个判断语句 j 是否不等于 Wrap(i+1,n) 和 Wrap(i+2,n) ，以确定是否用 x[j]的值，具体如“if((j<>Wrap(i+1,n))and(j<>Wrap(i+2,n)),x[j]))”，由此前述代码中的 7 个约束可用一句表达：

```
For(i=1:n)(Sum(j=1:n)(if((j<>Wrap(i+1,n))and(j<>Wrap(i+2,n)),x[j]))>=a[i]);
```

代码简化为：

```
Constant n=7;                                  //天数
Constant a=[27,34,20,25,24,30,23];             //人数
Constant p=[220,230,230,220,220,250,280];  //费用
IntParameter x(n);
Algorithm = LP;
MinFunction Sum(i=1:n,x,p)(x*p);
        For(i=1:n)(Sum(j=1:n)(if((j<>Wrap(i+1,n))and(j<>Wrap(i+2,n)),x[j]))>=a[i]);
```

该段代码简单明了通用，尤其对约束语句非常多却又有规律的情况，可以极大减少人为出错概率。上面两段代码都可得到下面相同的结果。

结果：

Objective Function(Min.): 8930	2: x1+x2+x5+x6+x7-(34) = 0
	3: x1+x2+x3+x6+x7-(20) = 0
Best Estimated Parameters:	4: x1+x2+x3+x4+x7-(25) = 0
x1: 5	5: x1+x2+x3+x4+x5-(24) = 11

x2: 11 x3: 0 x4: 5 x5: 14 x6: 0 x7: 4 Constrained Functions: 1: x1+x4+x5+x6+x7-(27) = 1	6: x2+x3+x4+x5+x6-(30) = 0 7: x3+x4+x5+x6+x7-(23) = 0 8: x1-(0) = 5 9: x2-(0) = 11 10: x3-(0) = 0 11: x4-(0) = 5 12: x5-(0) = 14 13: x6-(0) = 0 14: x7-(0) = 4

如果上述问题从 7 天改为 15 天为一个周期，每个员工连续工资 10 天再连续休息 5 天，数据如下表，如何安排员工的工作？

表 20.3 员工需求数据

时间	所需员工数 （x）	工资费用/位 （P）
1	25	208
2	16	212
3	24	213
4	17	217
5	29	206
6	25	203
7	22	212
8	28	214
9	21	200
10	17	204
11	28	208
12	14	209
13	12	219
14	25	216
15	11	214

15 天就有 15 个约束，如果一个个写不仅麻烦而且容易出错，参考前面简化代码，可以很容易得到如下代码，注意因为要连休 5 天，判断语句用了 5 次 Wrap 函数：

(j<>Wrap(i+1,n))and(j<>Wrap(i+2,n))and(j<>Wrap(i+3,n))and(j<>Wrap(i+4,n))and(j<>Wrap(i+5,n))

稍显麻烦，下一个版本中会有一个新的函数“SumAnd”，使用方式如下，这样即使连休 10 天，只需把“k=1:5”改成“k=1:10”即可。

SumAnd(k=1:5)(j<>Wrap(i+k,n))

求解代码：

```
Constant n=15;                                          //天数
Constant a=[25,16,24,17,29,25,22,28,21,17,28,14,12,25,11];   //人数
```

```
Constant p=[208,212,213,217,206,203,212,214,200,204,208,209,219,216,214]; //费用
IntParameter x(n);
Algorithm = LP;
MinFunction Sum(i=1:n,x)(x*p[i]);
For(i=1:n)(Sum(j=1:n)(if((j<>Wrap(i+1,n))and(j<>Wrap(i+2,n))and(j<>Wrap(i+3,n))and(j<>Wrap(i+4,n))and(j<>Wrap(i+5,n)),x[j]))>=a[i]);
```

结果：

Objective Function(Min.): 8052	Constrained Functions:
	1: x1+x7+x8+x9+x10+x11+x12+x13+x14+x15-(25) = 0
Best Estimated Parameters:	2: x1+x2+x8+x9+x10+x11+x12+x13+x14+x15-(16) = 8
x1: 11	3: x1+x2+x3+x9+x10+x11+x12+x13+x14+x15-(24) = 0
x2: 2	4: x1+x2+x3+x4+x10+x11+x12+x13+x14+x15-(17) = 0
x3: 0	5: x1+x2+x3+x4+x5+x11+x12+x13+x14+x15-(29) = 0
x4: 0	6: x1+x2+x3+x4+x5+x6+x12+x13+x14+x15-(25) = 0
x5: 12	7: x1+x2+x3+x4+x5+x6+x7+x13+x14+x15-(22) = 6
x6: 0	8: x1+x2+x3+x4+x5+x6+x7+x8+x14+x15-(28) = 0
x7: 3	9: x1+x2+x3+x4+x5+x6+x7+x8+x9+x15-(21) = 14
x8: 0	10: x1+x2+x3+x4+x5+x6+x7+x8+x9+x10-(17) = 18
x9: 7	11: x2+x3+x4+x5+x6+x7+x8+x9+x10+x11-(28) = 0
x10: 0	12: x3+x4+x5+x6+x7+x8+x9+x10+x11+x12-(14) = 12
x11: 4	13: x4+x5+x6+x7+x8+x9+x10+x11+x12+x13-(12) = 14
x12: 0	14: x5+x6+x7+x8+x9+x10+x11+x12+x13+x14-(25) = 1
x13: 0	15: x6+x7+x8+x9+x10+x11+x12+x13+x14+x15-(11) = 3
x14: 0	
x15: 0	

20.3 最小生成树的补充

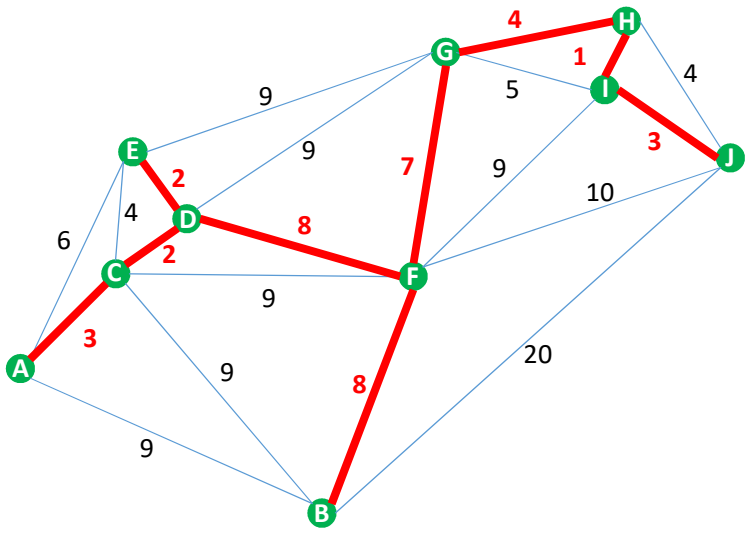


图 20.1 最小生成树示意图

前篇公众号演示了最小生成树问题，其中给出了如上图所示的示意图却没有

给出相应的求解代码，在此补充给出，同时演示“MDataSet”关键字的使用。

“MDataSet”是设定网络对称节点数据的简化格式，也可称之为结点格式，效果等同于矩阵格式。结点格式仅需给出点与点间有连接的数据及相应的结点号即可，如下表 20.4；矩阵形式如表 20.5，给出全部结点俩俩连接对应的数据，无连接的用一相对较大的数据（如 1000）代替

表 20.4 结点形式数据

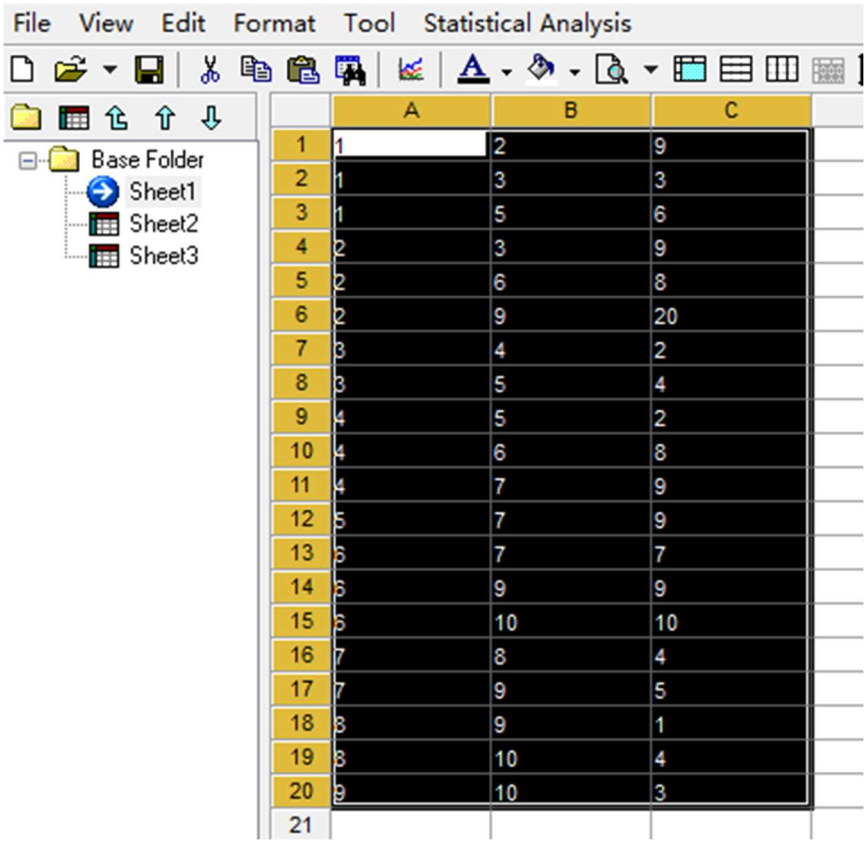
结点	结点	距离
1	2	9
1	3	3
1	5	6
2	3	9
2	6	8
2	9	20
3	4	2
3	5	4
4	5	2
4	6	8
4	7	9
5	7	9
6	7	7
6	9	9
6	10	10
7	8	4
7	9	5
8	9	1
8	10	4
9	10	3

表 20.5 矩阵形式数据

结点	1-A	2-B	3-C	4-D	5-E	6-F	7-G	8-H	9-I	10-J
1-A	0	9	3	1000	6	1000	1000	1000	1000	1000
2-B	9	0	9	1000	1000	8	1000	1000	20	1000
3-C	3	9	0	2	4	1000	1000	1000	1000	1000
4-D	1000	1000	2	0	2	8	9	1000	1000	1000
5-E	6	1000	4	2	0	1000	9	1000	1000	1000
6-F	1000	8	1000	8	1000	0	7	1000	9	10
7-G	1000	1000	1000	9	9	7	0	4	5	1000
8-H	1000	1000	1000	1000	1000	1000	4	0	1	4
9-I	1000	20	1000	1000	1000	9	5	1	0	3

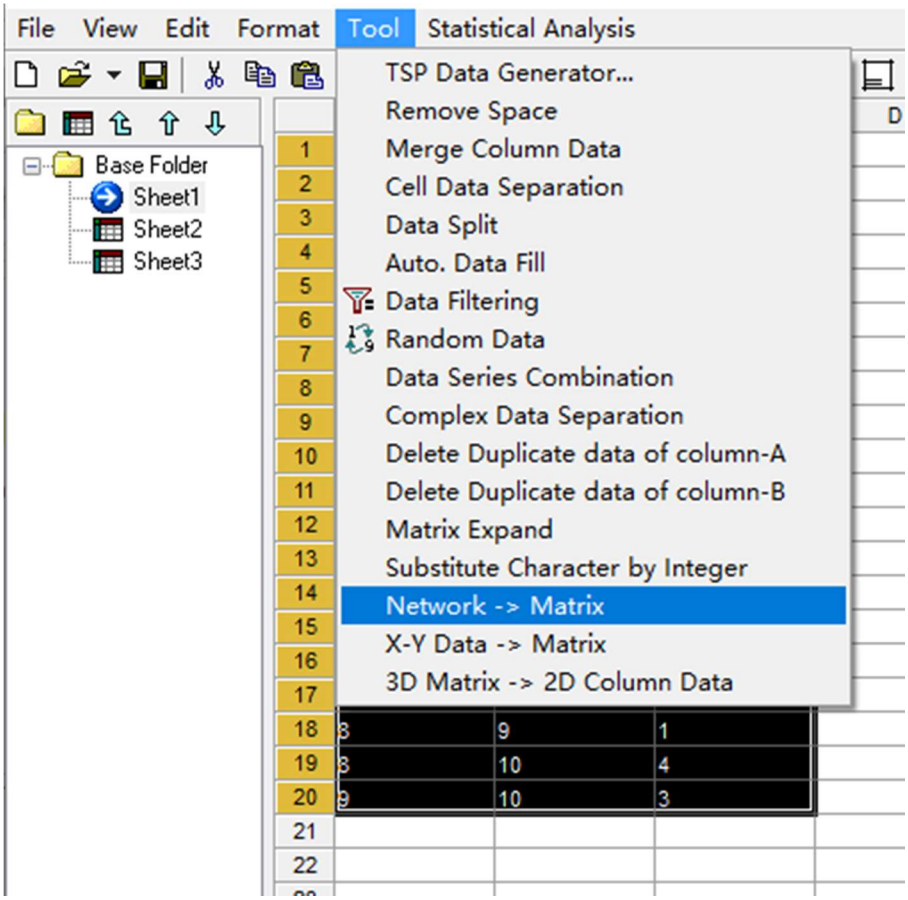
其实在 1stOpt 电子表格中有“网络结点”至“矩阵分布”的可视化命令，操作流程如下：

1) 选中结点数据

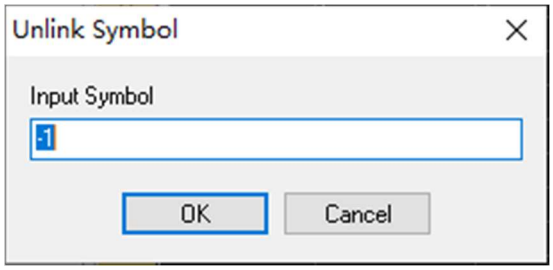


	A	B	C
1	1	2	9
2	1	3	3
3	1	5	6
4	2	3	9
5	2	6	8
6	2	9	20
7	3	4	2
8	3	5	4
9	4	5	2
10	4	6	8
11	4	7	9
12	5	7	9
13	6	7	7
14	6	9	9
15	6	10	10
16	7	8	4
17	7	9	5
18	8	9	1
19	8	10	4
20	9	10	3
21			

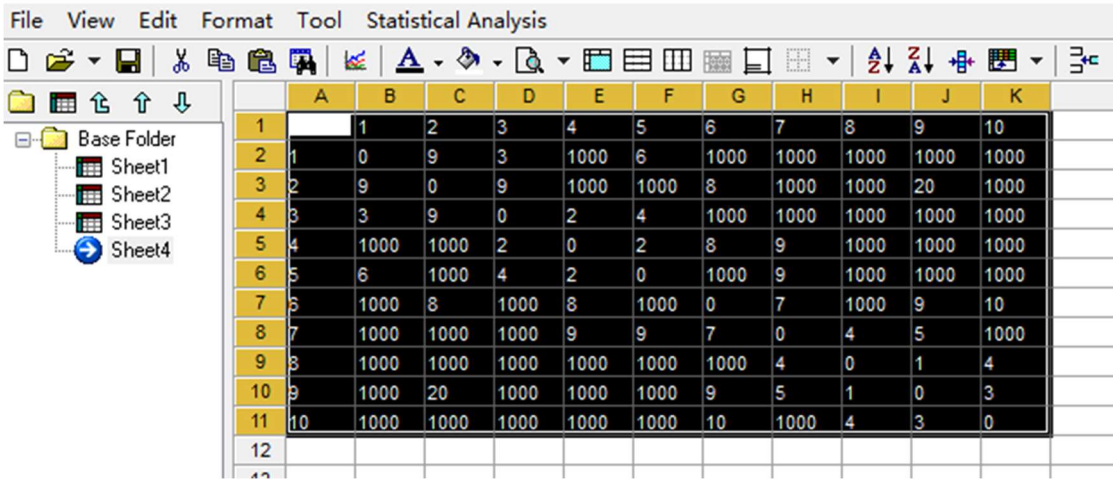
2) 从电子表格主菜单“工具”下拉“网络结点->矩阵分布”



3) 弹出输入窗口中输入无连接结点间的数据，比如 1000



4) 得到矩阵分布数据结果



	A	B	C	D	E	F	G	H	I	J	K
1		1	2	3	4	5	6	7	8	9	10
2	1	0	9	3	1000	6	1000	1000	1000	1000	1000
3	2	9	0	9	1000	1000	8	1000	1000	20	1000
4	3	3	9	0	2	4	1000	1000	1000	1000	1000
5	4	1000	1000	2	0	2	8	9	1000	1000	1000
6	5	6	1000	4	2	0	1000	9	1000	1000	1000
7	6	1000	8	1000	8	1000	0	7	1000	9	10
8	7	1000	1000	1000	9	9	7	0	4	5	1000
9	8	1000	1000	1000	1000	1000	1000	4	0	1	4
10	9	1000	20	1000	1000	1000	9	5	1	0	3
11	10	1000	1000	1000	1000	1000	10	1000	4	3	0
12											

求解代码（结点形式）

```
Constant n=10;
MDataSet[1000];
    i,j,c=
        1,2,9
        1,3,3
        1,5,6
        2,3,9
        2,6,8
        2,9,20
        3,4,2
        3,5,4
        4,5,2
        4,6,8
        4,7,9
        5,7,9
        6,7,7
        6,9,9
        6,10,10
        7,8,4
        7,9,5
        8,9,1
        8,10,4
        9,10,3
EndMDataSet;
BinParameter x(n,n);
IntParameter u(n);
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)));
```

求解代码（矩阵形式）

```
Constant n=10, D=1000;
Constant
c(n,n)=[0,9,3,D,6,D,D,D,D,D,
        9,0,9,D,D,8,D,D,20,D,
        3,9,0,2,4,D,D,D,D,D,
        D,D,2,0,2,8,9,D,D,D,
        6,D,4,2,0,D,9,D,D,D,
        D,8,D,8,D,0,7,D,9,10,
        D,D,D,9,9,7,0,4,5,D,
        D,D,D,D,D,D,4,0,1,4,
        D,20,D,D,D,9,5,1,0,3,
        D,D,D,D,D,10,D,4,3,0];
BinParameter x(n,n);
IntParameter u(n);
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)));
```

上面两段代码均可得到相同的结果

Objective Function(Min.):	x[2,10]: 0	x[5,3]: 0	x[7,6]: 0	x[9,9]: 0
38	x[3,1]: 0	x[5,4]: 0	x[7,7]: 0	x[9,10]: 1
	x[3,2]: 0	x[5,5]: 0	x[7,8]: 1	x[10,1]: 0
Best Estimated Parameters:	x[3,3]: 0	x[5,6]: 0	x[7,9]: 0	x[10,2]: 0
x[1,1]: 0	x[3,4]: 1	x[5,7]: 0	x[7,10]: 0	x[10,3]: 0
x[1,2]: 0	x[3,5]: 0	x[5,8]: 0	x[8,1]: 0	x[10,4]: 0
x[1,3]: 1	x[3,6]: 0	x[5,9]: 0	x[8,2]: 0	x[10,5]: 0
x[1,4]: 0	x[3,7]: 0	x[5,10]: 0	x[8,3]: 0	x[10,6]: 0
x[1,5]: 0	x[3,8]: 0	x[6,1]: 0	x[8,4]: 0	x[10,7]: 0
x[1,6]: 0	x[3,9]: 0	x[6,2]: 1	x[8,5]: 0	x[10,8]: 0
x[1,7]: 0	x[3,10]: 0	x[6,3]: 0	x[8,6]: 0	x[10,9]: 0
x[1,8]: 0	x[4,1]: 0	x[6,4]: 0	x[8,7]: 0	x[10,10]: 0
x[1,9]: 0	x[4,2]: 0	x[6,5]: 0	x[8,8]: 0	u1: 0
x[1,10]: 0	x[4,3]: 0	x[6,6]: 0	x[8,9]: 1	u2: 4
x[2,1]: 0	x[4,4]: 0	x[6,7]: 1	x[8,10]: 0	u3: 1
x[2,2]: 0	x[4,5]: 1	x[6,8]: 0	x[9,1]: 0	u4: 2
x[2,3]: 0	x[4,6]: 1	x[6,9]: 0	x[9,2]: 0	u5: 3
x[2,4]: 0	x[4,7]: 0	x[6,10]: 0	x[9,3]: 0	u6: 3
x[2,5]: 0	x[4,8]: 0	x[7,1]: 0	x[9,4]: 0	u7: 6
x[2,6]: 0	x[4,9]: 0	x[7,2]: 0	x[9,5]: 0	u8: 7
x[2,7]: 0	x[4,10]: 0	x[7,3]: 0	x[9,6]: 0	u9: 8
x[2,8]: 0	x[5,1]: 0	x[7,4]: 0	x[9,7]: 0	u10: 9
x[2,9]: 0	x[5,2]: 0	x[7,5]: 0	x[9,8]: 0	

最小生成树路径为：1-3，3-4，4-5，4-6，6-2，6-7，7-8，8-9，9-10 或 A-C，C-D，D-E，D-F，F-B，F-G，G-H，H-I，I-J

20.4 小结

介绍了两个经典运筹学问题的 1stOpt 实现，对 1stOpt 中两个函数“MDataSet”及“Wrap”函数的使用方式进行了应用演示，对最小生成树问题进行了补充说明。

对线性规划这一类运筹学问题，1stOpt 具有代码简单、易懂和易于实现之特点。