经典运筹学问题与模型

(Models for Classical Operations Research Problems)

主讲人：朱晗

东北财经大学 管理科学与工程学院

hanzhu@dufe.edu.cn

感谢东北大学系统工程研究所课程组

第八章 集覆盖问题与模型

集覆盖问题(Set Covering Problem-SCP)

- 集覆盖问题实例与特征
- 集覆盖问题的数学模型
- 集覆盖问题的特例
- SCP问题的求解算法
- SCP问题的应用
- 航线机组成员的调度问题

集覆盖问题

- 问题描述:
  - 对于一个 *m* 行 *n* 列的 **0-1** 矩阵，选择不同的列，会产生不同的费用，同时也会覆盖不同的行
  - 问题是：如何选择一些矩阵的列的组合，能使其覆盖所有的行，而且选择费用最小。

$$C = [10 \quad 15 \quad 11 \quad 10 \quad 8 \quad 2]$$

$$A = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

模型描述

- 数学表示为：

$$\min\ f(\boldsymbol{x}) = \sum_{j=1}^{n} c_j x_j$$

$$\text{s.t.}\quad g_i(x) = \sum_{j=1}^{n} a_{ij} x_j \geq 1, \quad i = 1,2,\ldots,m$$

$$x_j \in \{0,1\}, \qquad j = 1,2,\ldots,n$$

$$C = [10\ \ 15\ \ 11\ \ 10\ \ 8\ \ 2]$$

$$A = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

where

$$x_j = \begin{cases} 1, & \text{if column j (with a cost } c_j > 0) \text{ is in the solution} \\ 0, & \text{otherwise} \end{cases}$$

- 约束式(1)保证每行至少被一列覆盖
- 约束式(2)是完整性约束。

模型特征

- NP难问题
- 0-1 整数规划问题

$$\min\ f(\boldsymbol{x}) = \sum_{j=1}^{n} c_j x_j$$

$$\text{s.t.}\quad g_i(x) = \sum_{j=1}^{n} a_{ij} x_j \geq 1, \quad i = 1,2,\ldots,m$$

$$x_j \in \{0,1\}, \qquad j = 1,2,\ldots,n$$

集覆盖问题的特例

- 集覆盖问题的变种
  - 单一费用问题（unicost setcovering problem）
    - 如果所有费用 c_j 都相同
    - 问题变为：如何选择最少的列，包含所有的行。
  - 集划分问题（set partitioning problem）
    - 如果为等式约束
    - 一行只能被一列所覆盖，把列看作是集合，相当于：把行划分到相应的集合（列）中。

应用实例

- 许多网络问题可以建模为带有特定 $A=(a_{ij})$ 的集覆盖问题
  - 例如：为了监测网络性能和安全性，需要在网络上设置监测器,对网络链路进行监测。网络上的边（链路）→行；待设置的监测器→列，设置在不同位置的监视器所能监视的链路是不同的，构成0-1矩阵，问题是：如何设置最少的监测器能监测所有的链路。
- 故障检测中的测试点选取问题
  - 例如：为检测一个系统当中的所有故障，需要设置很多个测试点，所有的故障→行，备选测试点→列，备选测试点所能测试的故障点是不同的，构成0-1矩阵，测试时间也不同构成费用向量。问题是：如何选取一些测试点, 使得这些测试点可以检测所有故障, 而所需时间最少。
- 物流中心选址问题
  - 物流中心需要服务的需求点→行；

可以提供服务的“候选”物流中心→列，物流中心设置的位置不同，所能服务的需求点也不同，构成0-1矩阵。物流中心的建设成本构成费用向量。问题是：在什么位置建立物流中心，既能满足所有的需求，又能使建立物流中心的成本最低。

$$C=[10\ 15\ 11\ 10\ 8\ 2]$$

$$A=\begin{bmatrix}1&1&0&1&0&0\\0&0&1&1&0&0\\1&1&1&0&0&0\\0&0&0&0&1&1\\0&0&1&0&0&1\\0&0&0&1&1&0\end{bmatrix}$$

应用

- 枢纽站选址问题
  - 运输流→行；“候选”枢纽站→列，如何选择枢纽站，能使运输成本最低
- 交通消防设施点布局问题
  - 可以把交通事故常发地点作为“应急点”→行；将“备选的交通消防设施点”→列，由于应急队伍必须在规定的时间之内到达事发地点，所以一个备选点所能到达的应急点是不同的，构成了01矩阵。如何选择最少的设施点，能覆盖所有的应急点。
- 软件测试用例选择问题
  - 测试的问题→行；测试用例→列，如何选择最少的测试用例来完成所有的测试。
- 宽带码分多址(WCDMA) 基站布局问题
  - 待服务的移动台→行；备选的基站→列，备选基站的建设成本→费用向量，在哪里建设基站能使建设成本最低。

$$C = [10 \ 15 \ 11 \ 10 \ 8 \ 2]$$

$$A = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

- 节点覆盖问题(node-covering problem)
  - Salkin和Saha提出
- 匹配问题(matching problem)
  - Balinski提出
- 最大流问题(maximum flow problem)
  - Sallkin和Mathur提出

应用

作　　者	应　　用
Arabeyre 等(1969)	航线机组成员调度(airline crew scheduling)
Balinski (1965)	开关电路调度(switching circuit scheduling)
Balinski and Quandt (1964)	卡车派遣(trucking dispatching)
Bellmore (1971)	其他例子
Bellmore, Greenberg and Jarvis (1970)	网络攻击与防卫(network attack and defense)
Bellmore (1971)	网络攻击与防卫(network attack and defense)
Busacker and Satty (1965)	图着色(map coloring)
Charnes and Miller (1956)	铁路工作人员调度(railroad crew scheduling)
Cobham, Fridshal and North (1961)	PERT/CPM 分析
	开关电路调度(switching circuit scheduling)
	符号逻辑(symbolic logic)
Day (1965)	信息恢复(information retrieval)
Garey Johnson (1979)	资源分配(resource allocation)
Garfinkel (1968)	政治重新划分选区(political redistricting)
Labordere (1969)	电路设计(circuit design)
Levin (1969)	车辆路径(vehicle routing)
Revelle, Marks and Leibman (1970)	设备定位(facilities location)
Root (1964)	符号逻辑(symbolic logic)
Rubin (1973)	指派问题(assignment problem)
Salverson (1955)	装配线平衡(assembly line balancing)
Steinmann and Schwinn (1969)	装配线平衡(assembly line balancing)
Toregas (1971)	设备定位(facilities location)
Valenta (1969)	资本投资(capital investment)
Walker (1974)	指派问题(assignment problem)

航线机组成员调度问题

- 假设需要为 n 个机组成员安排 m 次航班，这些航班必须在一定时段内完成
- 由于受到时间、地理位置、机组成员能力(比如不是所有飞行员都有驾驶波音 767 飞机的资格)和其他限制的影响，每个机组成员能够服务航线的组合各不一样，费用也不一样，可以列写出来。
- 设 k 表示机组成员的编号，t 表示航线的编号(或航线组合的编号)。决策变量 $x_{k(t)}=1$ 表示安排第 k 个机组成员到第 t 次航线组合上。$a_{ik(t)}=1$ 意味着第 t 个航线组合的第 i 次航程(flight leg)(比如 Flight 347, 从 Cleveland 到 Chicago) 可以由第 k 个机组成员完成。
- 有如下的约束:表明每 1 次航程都至少有 1 个机组成员来完成

$$\sum_{k}\sum_{t} a_{ik(t)}x_{k(t)} \geq 1, \qquad i=1,2,\cdots,m$$

- 设 $c_{k(t)}$ 表示第 k 个机组成员的第 t 次航线组合的费用。
- 问题是：如何安排机组成员的航线组合，在每1航程都至少有1个机组成员完成的情况下，使得总费用最低。

航线机组成员调度问题

- 4 个机组成员和 5 次航程的简单例子。机组成员 1 可以驾驶航程 1, 2, 4 和 5，或 1，3 和 4，或 1 和 3。……
- 在给出第 k 个机组成员的第 t 次航线组合的费用之后，就需要确定最优的航线机组成员调度方案。

机组成员 k	1			2		3		4		
航线组合 t	1	2	3	1	2	1	2	1	2	
变量 $x_{k(t)}$	$x_{1(1)}$	$x_{1(2)}$	$x_{1(3)}$	$x_{2(1)}$	$x_{2(2)}$	$x_{3(1)}$	$x_{3(2)}$	$x_{4(1)}$	$x_{4(2)}$	
费用 $C_{k(t)}$	$C_{1(1)}$	$C_{1(2)}$	$C_{1(3)}$	$C_{2(1)}$	$C_{2(2)}$	$C_{3(1)}$	$C_{3(2)}$	$C_{4(1)}$	$C_{4(2)}$	
航程 i	$a_{i1(1)}$	$a_{i1(2)}$	$a_{i1(3)}$	$a_{i2(1)}$	$a_{i2(2)}$	$a_{i3(1)}$	$a_{i3(2)}$	$a_{i4(1)}$	$a_{i4(2)}$	
1	1	1	1	1	1	0	0	0	0	$\geqslant 1$
2	1	0	0	1	1	1	1	1	1	$\geqslant 1$
3	0	1	1	1	0	0	1	1	0	$\geqslant 1$
4	1	1	0	0	0	1	0	0	0	$\geqslant 1$
5	1	0	0	0	0	0	1	0	0	$\geqslant 1$

本例可能有多少条航线，仅给出了 条航线？

这些航线是如何组成的？航线的费用是与什么有关

一般的求解方法

- 对于集覆盖问题，既有对这类问题的最优解的报道，也有启发式求解的报道。
  - Balas 和 Ho 提出了割平面方法
  - Fisher 和 Kedia 提出了用于解决最多 200 行 2000 列问题的对偶启发式算法
  - Beasley 和 Jornsten 结合了 Lagrangian 启发式方法、可行解排除约束 Gomory f-cuts 和改进的分支策略，解决了最多400行4000列的集覆盖问题。
  - Harche和Thompson开发了一种称作列减少算法(column subtraction algorithm)的直接方法，可以解决集覆盖问题的大规模稀疏矩阵的情况。

智能求解方法

- 与其他组合问题一样，人们最近对于用进化计算方法求解集覆盖问题产生了越来越浓厚的兴趣。
  - Jacobs和Brusco开发了模拟退火算法，并报道在求解最多1000 行10000列问题时取得了相当可观的成功。
  - Sen研究了模拟退火算法和简单遗传算法的性能。
  - Beasley和Chu以及Gonzalez, Hernandez和Corne都提出了用遗传算法求解大规模集覆盖问题的方法。

GA求解

- 编码
  - 基于列的表示(column-based representation)
  - 基于行的表示(row-based representation)
- 基于列的表示
  - 表示了集覆盖问题本质上的 0-1 变量，是显而易见的表示方式
  - 问题:初始化或交叉变异后的染色体，未必能保证是可行的。

$$x = \begin{array}{c} i \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccccc} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 \end{array} \\ \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

$$[a_{ij}] = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

$$C = \begin{bmatrix} 10 & 15 & 11 & 10 & 8 & 2 \end{bmatrix}$$

染色体，以 列为基因

x_1	x_2	x_3	x_4	x_5	x_6
1	0	1	0	1	0

$x_1 = x_3 = x_5 = 1$

$g_1(x) = 1 \geq 1,\ g_2(x) = 1 \geq 1,\ g_3(x) = 2 \geq 1$

$g_4(x) = 1 \geq 1,\ g_5(x) = 1 \geq 1,\ g_6(x) = 1 \geq 1$

$f(x) = 10 + 11 + 8 = 29$

GA求解

- 启发式算子
  - Beasley和Chu倾向于修补策略，设计了一种附加启发式算子，将不可行解转换为可行解。
  - 算子包含两个主要成分：
    - 修补给定个体的不可行性；
    - 执行附加的局部搜索以试图提高遗传算法的效率。
  - 算子的基本想法：
    - 确定所有没被覆盖的行
    - 添加必要的列使得所有行都被覆盖。
    - 一旦添加了列，使得个体成为可行解，就应用局部搜索过程来消除个体中冗余的列。

GA求解

- 启发式算子的步骤
  - 不失一般性，假设列按照费用升序排列。具有相同费用的列按照他们覆盖行的数量进行降序排列。

I：所有行的集合

J：所有列的集合

α_i：覆盖行 $i(i\in I)$的列的集合

β_j：被列 $j(j\in J)$覆盖的行的集合

S：解中列的集合

w_i：S 中覆盖行 $i(i\in I)$的列的数量

U：未被覆盖的行的集合

第 1 步：初始化 $w_i:=|S\cap a_i|,\forall i\in I$

第 2 步：初始化 $U:=\{i|w_i=0,\forall i\in I\}$

第 3 步：对于 U 中每一行 i（按照 i 的升序）

(3.1) 在 α_i 中寻找第一个最小化 $c_j/|U\cap\beta_j|$ 的列（按照 j 的升序）

(3.2) 将第 j 列添加到 S，令 $w_i:=w_i+1,\forall i\in\beta_j$，令 $U:=U-\beta_j$

第 4 步：对 S 中的每一列 j（按照 j 的降序），如果 $w_i\geqslant 2,\forall i\in\beta_j$，则令 $S:=S-j$ 以及 $w_i:=w_i-1,\forall i\in\beta_j$

第 5 步：S 现在就是没有冗余列的可行解。

注：$|U\cap\beta_j|$表示被第 j 列覆盖但目前尚未被当前解覆盖的行的数量。最小化 $c_j/|U\cap\beta_j|$表示希望第 j 列费用低，而且在当前解中加入第 j 列后覆盖尚未被覆盖的行数多。

计算说明

I：所有行的集合 [1,2,3,4,5,6]

J：所有列的集合 [1,2,3,4,5,6]

α_i：覆盖行 $i(i\in I)$的列的集合

β_j：被列 $j(j\in J)$覆盖的行的集合

S：解中列的集合 [1,3]

w_i：S 中覆盖行 $i(i\in I)$的列的数量

U：未被覆盖的行的集合 [4,6]

$$C=\begin{matrix}[x_1 & x_2 & x_3 & x_4 & x_5 & x_6]\\ [2 & 8 & 10 & 11 & 11 & 15]\end{matrix}$$

$$A=\begin{matrix}\\ \begin{bmatrix}1&1&0&1&0&0\\0&0&1&1&0&0\\1&1&1&0&0&0\\0&0&0&0&1&1\\0&0&1&0&0&1\\0&0&0&1&1&0\end{bmatrix}\end{matrix}$$

w_i	α_i
1	[1,2,4]
1	[3,4]
2	[1,2,3]
0	[5,6]
1	[3,6]
0	[4,5]

β_j	$\{1,3\}$	$\{1,3\}$	$\{2,3,5\}$	$\{1,2,6\}$	$\{4,6\}$	$\{4,5\}$
$\lvert U\cap\beta_j\rvert$	0	0	0	1	2	1
$c_j/\lvert U\cap\beta_j\rvert$					5.5	15

第 1 步：初始化 $w_i:=|S\cap a_i|,\forall i\in I$

第 2 步：初始化 $U:=\{i\,|\,w_i=0,\forall i\in I\}$

第 3 步：对于 U 中每一行 i（按照 i 的升序）

(3.1) 在 α_i 中寻找第一个最小化 $c_j/|U\cap\beta_j|$ 的列（按照 j 的升序）

(3.2) 将第 j 列添加到 S，令 $w_i:=w_i+1,\forall i\in\beta_j$，令 $U:=U-\beta_j$

第 4 步：对 S 中的每一列 j（按照 j 的降序），如果 $w_i\geqslant 2,\forall i\in\beta_j$，则令 $S:=S-j$ 以及 $w_i:=w_i-1,\forall i\in\beta_j$

第 5 步：S 现在就是没有冗余列的可行解。

GA求解

- 基于行的表示
  - 染色体的长度与给定问题行的数量相同。
  - 这种染色体，基因的序号表示行的信息，基因的值表示覆盖该行的列的信息。
  - 采用这种表示，在交叉和变异过程中始终保持可行性。
  - 问题：同一解可以表示成不同的形式。

$$x = \begin{array}{c} i \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccccc} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 \end{array} \\ \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

$$[a_{ij}] = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

$$C = [10\ 15\ 11\ 10\ 8\ 2]$$

染色体，以行为基因

行	1	2	3	4	5	6
列	2	3	3	6	3	4

$$x_2 = x_3 = x_4 = x_6 = 1$$

$$g_1(x) = 2 \geq 1,\ g_2(x) = 2 \geq 1,\ g_3(x) = 2 \geq 1$$

$$g_4(x) = 1 \geq 1,\ g_5(x) = 2 \geq 1,\ g_6(x) = 1 \geq 1$$

$$f(x) = 15 + 11 + 10 + 2 = 38$$

GA求解

- 基于行的表示
  - 问题：同一解可以表示成不同的形式。
  - 染色体评价：
    - 由于同一列可能在超过一个基因位置中出现，需抽取惟一的列集合(重复的列仅考虑一次)并进行适应值评价。

$$x = \begin{array}{c} \\ i\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccccc} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 \end{array} \\ \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

$$[a_{ij}] =$$

$$C = [\,10\ 15\ 11\ 10\ 8\ 2\,]$$

行表示

行	1	2	3	4	5	6
列	2	3	3	6	3	4

行	1	2	3	4	5	6
列	4	3	2	6	3	4

同一解，不同形式

列表示

0	1	1	1	0	1

GA求解

- 熔合算子 (fusion operator)
  - Beasley和Chu提出了一种通用的基于适应值的交叉算子，称作熔合算子
  - 这种算子既考虑父代解的结构，又考虑他们的适应值。
  - 与传统杂交算子通常生成两个后代不同，而熔合算子仅生成一个后代。
  - 计算过程：

$eval(P_1)=20$ P_1 | 0 | 1 | 1 | 0 | 0 | 1 |

$eval(P_2)=10$ P_2 | 1 | 1 | 0 | 0 | 1 | 0 |

设：P_1, P_2 — 父代染色体，C — 子代染色体
$eval(P_1), eval(P_2)$ — 两个父代的适应值

第 1 步：$i=1$

第 2 步：如果 $P_1[i]=P_2[i]$，则 $C[i]:=P_1[i]:=P_2[i]$

C | * | 1 | * | 0 | * | * |

第 3 步：如果 $P_1[i]\neq P_2[i]$，则

(1) 以概率 $p=eval(P1)/(eval(P1)+eval(P2))$ 让 $C[i]:=P_1[i]$

(2) 以概率 $1-p$ 让 $C[i]:=P_2[i]$

第 4 步：如果 $i=n$，停止；否则 $i=i+1$，返回第 1 步。

GA求解

- **GA求解过程**
  - **Beasley和Chu采用的求解过程**

第 1 步：随机产生 N 个解构成初始种群。令 $t:=0$。

第 2 步：采用熔合杂交算子生成新解 C。

第 3 步：变异 C 中 k 个随机选出的列。

第 4 步：采用启发式算子确保 C 可行并去除冗余列。

第 5 步：如果 C 与种群中任意解相同，返回第 2 步。否则令 $t:=t+1$。

第 6 步：从种群中随机选出一个低于平均适应值的个体并用 C 替代(稳态复制方法)。

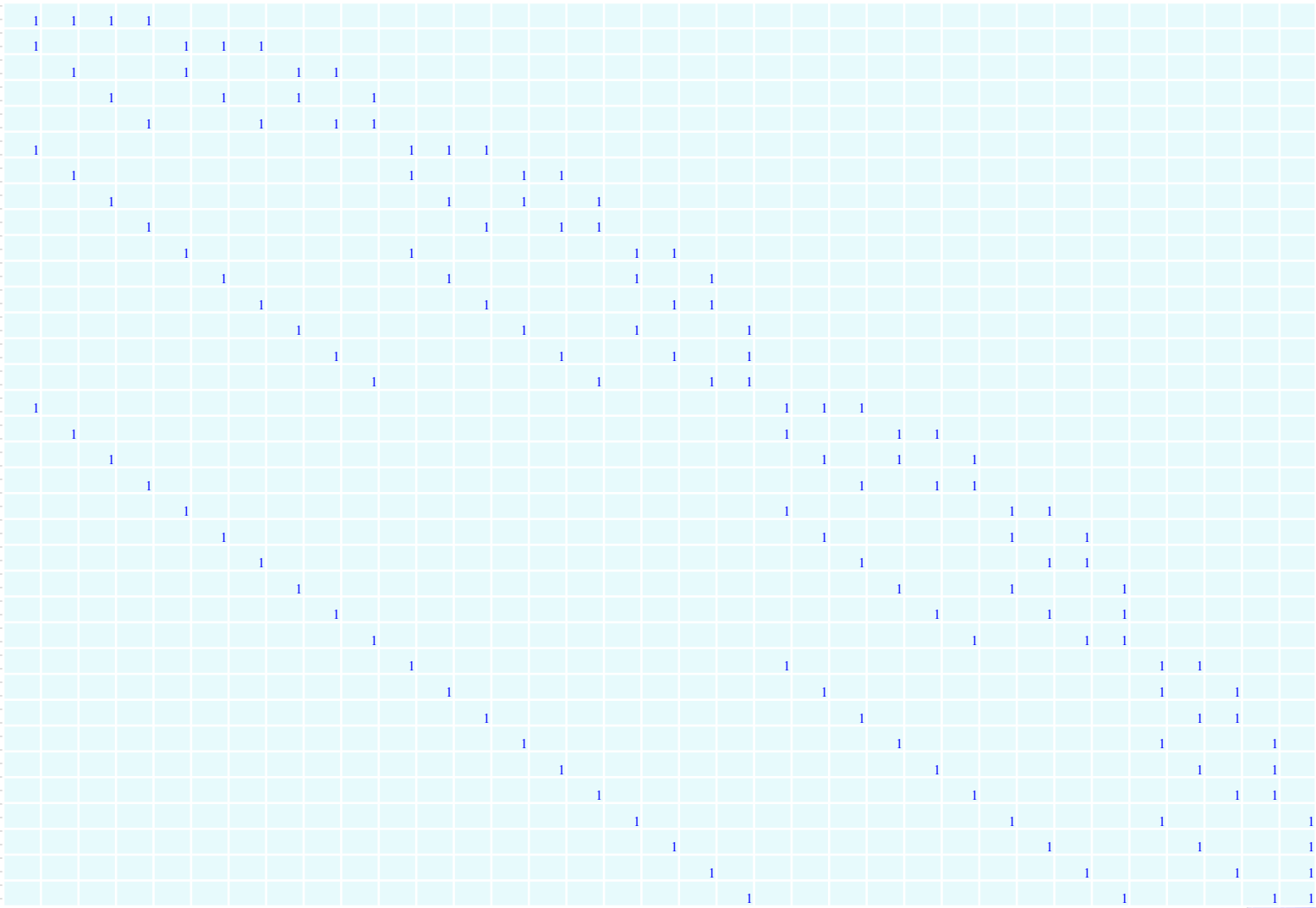
第 7 步：重复第 2 到第 6 步直到满足终止条件，输出最优解。

针对特殊问题构造输入矩阵方法

- s1={1,2,3}， s2={1,2,4}， s3={1,2,5}， s4= {1,3,4}， s5= {1,3,5}，s6={1,4,5}， s7={2,3,4}， s8={2,3,5}， s9= {2,4,5}， s10={3,4,5}
- x1={1,2}， x2={1,3}， x3={1,4}， x4={1,5}， x5={2,3}， x6={2,4}，x7={2,5}，x8={3,4}，x9={3,5}，x10={4,5}
- 若 $x_i \subseteq s_j$，即 s_j包含 x_i，则令 $a_{ij}=1$，否则 $a_{ij}=0$

1	1	1	0	0	0	0	0	0	0
1	0	0	1	1	0	0	0	0	0
0	1	0	1	0	1	0	0	0	0
0	0	1	0	1	1	0	0	0	0
1	0	0	0	0	0	1	1	0	0
0	1	0	0	0	0	1	0	1	0
0	0	1	0	0	0	0	1	1	0
0	0	0	1	0	0	1	0	0	1
0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	1	0	0	1	1

K=3时输入矩阵构造



构造矩阵递推关系式

$(a_{k+1})_{ij}$第 1 行为 k+2 个 1 连续排列

$(a_{k+1})_{ij}$第 2 行到第（k+3）行为一个（k+2）行（k+2）列的单位矩阵跟着$(a_k)_{ij}$的前 k+2 行

$(a_{k+1})_{ij}$第 $(1+C_{k+2}^1+1)$ 行到第 $(1+C_{k+2}^1+C_{k+3}^2)$ 行为 C_{k+3}^2 阶单位矩阵跟着 $(a_k)_{ij}$的前 C_{k+3}^2 行

• • •

第 $(1+C_{k+2}^1+C_{k+3}^2+\ldots+C_{2k-1}^k+1)$ 行到第 $(1+C_{k+2}^1+C_{k+3}^2+\ldots+C_{2k}^{k-1}+C_{2k}^{k-1})$ 行为 C_{2k}^{k-1} 阶单位矩阵跟着 $(a_k)_{ij}$的前 C_{2k}^{k-1} 行

第 $(1+C_{k+2}^1+C_{k+3}^2+\ldots+C_{2k}^{k-1}+1)$ 行到第 $(1+C_{k+2}^1+C_{k+3}^2+\ldots+C_{2k}^{k-1}+C_{2k+1}^k)$ 行为 C_{2k+1}^k 阶单位矩阵跟着 $(a_k)_{ij}$的全部

其他部分均可由 $a_{ij}=a_{(n-j+1)(n-i+1)}$ 原则得出。

CPLEX精确解/贪婪算法/遗传算法的比较

精确解——利用CPLEX工具求取

- **CPLEX默认求解算法——分支定界**
- **当k=3，k=4时，在一定时间内可以求出精确解**
- **K=5时由于分支定界节点过多，内存溢出，无法求出精确解**

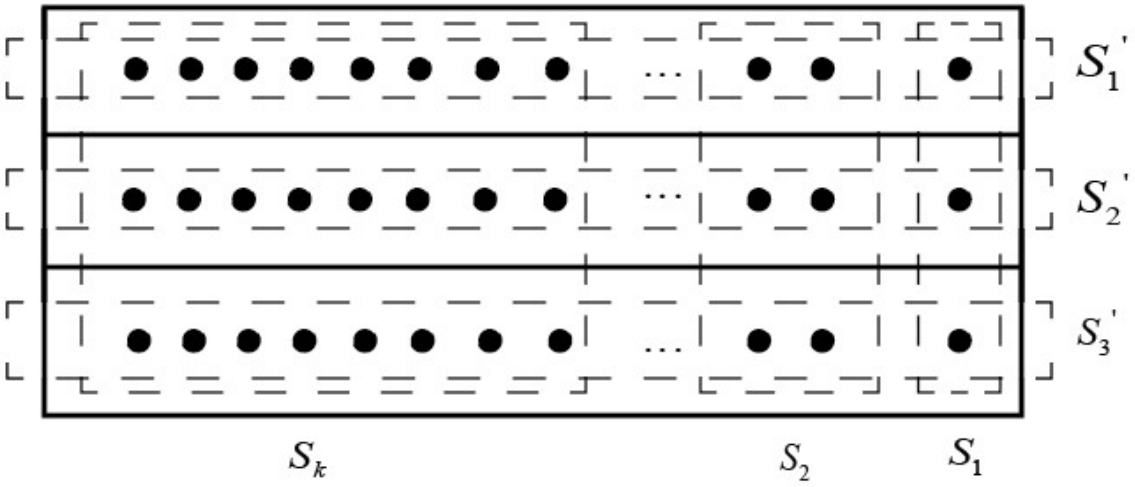
贪婪算法——算法描述

- **1.** $X \leftarrow A,\ C = \varnothing$ **（C为解集合）**
- **2.重复如下操作直到：** $X = \varnothing$
  - **选择i，使得** S_i **覆盖的元素** X **数目最大，即从矩阵** a_{ij} **中选出包含1最多的那一列，更新** $C = C \cup \{i\}, X = X \setminus S_i$
- **3.最终得到的集合C即为一个可行解**

k取值	目标函数值		计算时间		贪婪算法相对误差
	cplex	贪婪	cplex	贪婪	
k=3	12	14	3.4s	立即	16.6%
k=4	30	32	8min	立即	6.6%
k=5	102*	110	大于48h	5s	大于7.8%

贪婪算法——算法优劣性

- 算法结构简单，运行速度快
  - K=5时依旧可以快速求解
- 无法得出精确解



基于启发修补的遗传算法——算法描述

- step 1：随机产生N个解构成初始种群，令t：=0
- step2：采用二进制竞争选择从种群中选出两个解，.
- step3：采用熔合杂交算子组合 和 生成一个新解C
- step4：变异*C*中*k*个随机选出的列，其中*k*由可变的变异方案确定。
- step5：采用启发式算子确保*C*可行并去除冗余列
- step6：如果*C*与种群中任意解相同，返回第2步，否则令t:=t+1，并跳到step7
- step 7：从种群中随机选出一个高于平均适应值的个体并用C替代
- step 8：重复2~7直到产生t=M个不重复的个体。从种群中选出具有最小适应值的个体作为最优解

基于启发修补的遗传算法——融合算子

- **step 1：i=1**
- **step 2：如果** $P_1[i]=P_2[i]$ **，则** $C[i]:=P_1[i]:=P_2[i]$
- **step 3：如果** $P_1[i]\neq P_2[i]$ **，则**
  - **（3.1）以概率** $p=f_{P2}/(f_{P1}+f_{P2})$ **让** $C[i]:=P_1[i]$
  - **（3.2）以概率1-p让** $C[i]:=P_2[i]$
- **step 4：如果*i=n*，停止；否则*i=i+1*，返回Step 1**

基于启发修补的遗传算法——启发式修补

- **step 1：初始化** $\omega_i := |S \cap a_i|, \forall i \in I$
- **step 2：初始化** $U := \{i \mid \omega_i = 0, \forall i \in I\}$
- **step 3：对于U中每一行i（按照i的升序）**
  - **（3.1）在**α_i **中寻找第一个最小化** $c_j / |U \cap \beta_j|$ **的列（按照j的升序）**
  - **（3.2）将第j列添加到S，令** $\omega_i := \omega_i + 1, \forall i \in \beta_j$ **令** $U := U - \beta_j$
- **step 4：对于S中的每一列j（按照j的降序），如果** $\omega_i \geq 2, \forall i \in \beta_j$ **，则令** $S := S - j$**，以及** $\omega_i := \omega_i - 1, \forall i \in \beta_j$
- **step 5：S现在就是没有冗余列的可行解**

I：所有行的集合

J：所有列的集合

α_i：覆盖行$i(i \in I)$的列的集合

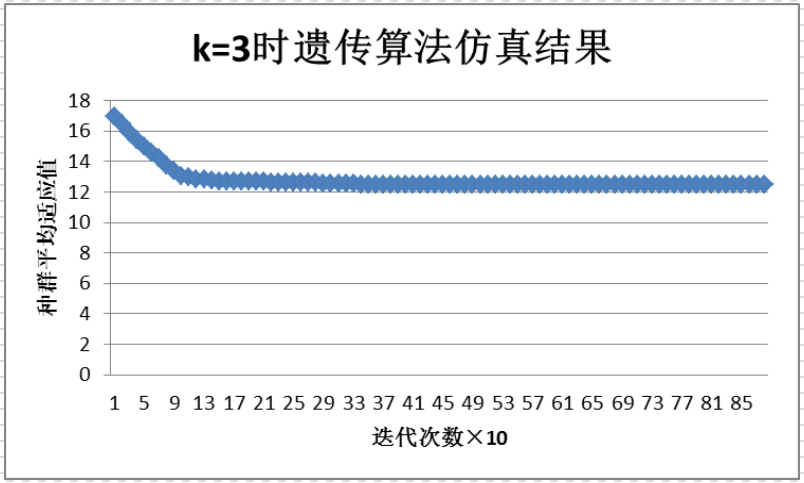
β_i：被列$j(j \in J)$覆盖的行的集合

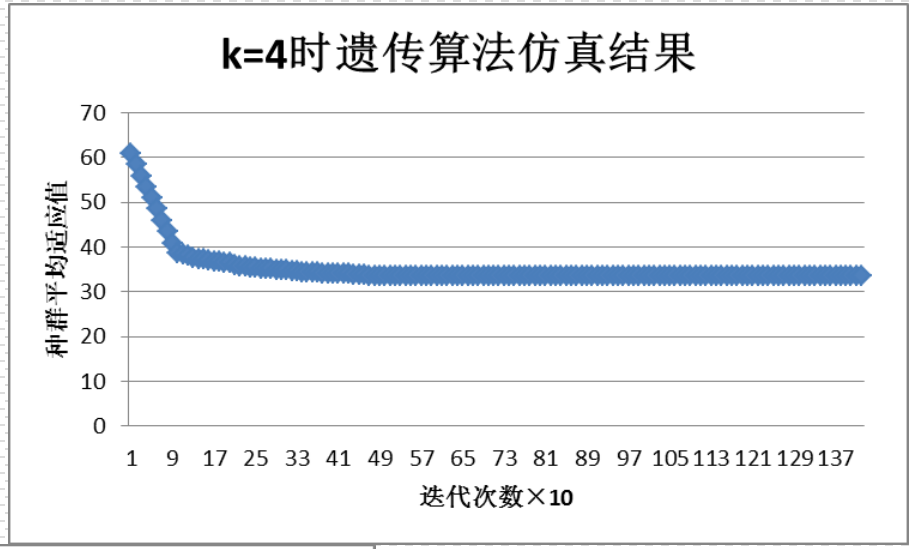
S：解中列的集合

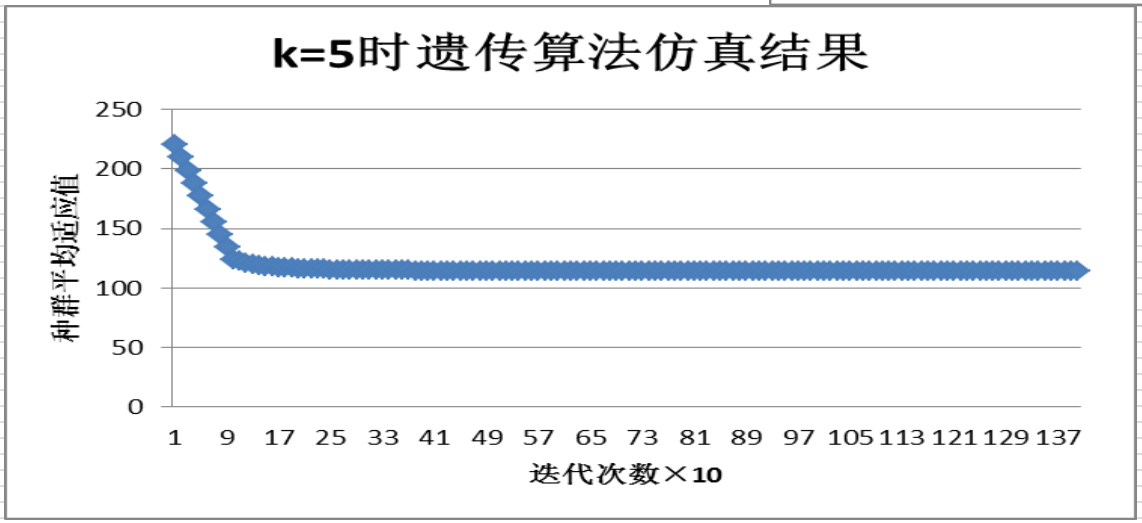
U：未被覆盖的行的集合

ω_i：S 中覆盖行$i(i \in I)$的列的数量

更改算法参数，进行仿真实验







基于启发修补的遗传算法——结果比较

算法名称 / k取值	精确算法	贪婪算法	基于启发修补的遗传算法
k=3	12(3.4s)	14(1s)	12(1min)
k=4	30(8min)	32(1s)	31(1min)
k=5	102(48h)	110(5s)	110(5min)

第八章 集覆盖问题扩展

- 集合覆盖约束（set covering constraint）要求每个子集J中至少有一个元素出现在最优解中，可以表示为：
  - $\sum_{j \in J} x_j \geq 1$
- 集合划分约束（set partitioning constraint）要求每个子集J中有且仅有一个元素出现在最优解中，可以表示为：
  - $\sum_{j \in J} x_j = 1$
- 集合包装约束（set packing constraint）要求每个子集J中最多有一个元素出现在最优解中，可以表示为：
  - $\sum_{j \in J} x_j \leq 1$

集划分问题

- 问题描述:
  - 对于一个 *m* 行 *n* 列的 0-1 矩阵，选择不同的列，会产生不同的费用，同时也会覆盖不同的行
  - 问题是：如何选择一些矩阵的列的组合，能使其每行中恰好包含一个元素而且选择费用最小。

$$C = [10 \quad 15 \quad 11 \quad 10 \quad 8 \quad 2]$$

$$A = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

模型描述

- 数学表示为：

$$\min\ f(\boldsymbol{x}) = \sum_{j=1}^{n} c_j x_j$$

$$\text{s.t.}\quad g_i(x) = \sum_{j=1}^{n} a_{ij} x_j = 1, \quad i = 1,2,\ldots,m$$

$$x_j \in \{0,1\}, \qquad j = 1,2,\ldots,n$$

$$C = [10\ 15\ 11\ 10\ 8\ 2]$$

$$A = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

where

$$x_j = \begin{cases} 1, & \text{if column j (with a cost } c_j > 0) \text{ is in the solution} \\ 0, & \text{otherwise} \end{cases}$$

- 约束式(1)保证每行恰好被一列覆盖
- 约束式(2)是完整性约束。

集划分问题实例

- *指派问题*
- *指派问题的标准形式是:有n项任务，恰好n个人承担，第i人完成第j项任务的花费为cij，要求确定人和事之间的一一对应指派方案，使总花费最少。*

$$C=(c_{ij})_{n\times n}=\begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix}$$

集划分问题实例

- 定义决策变量如下：

$$x_{ij}=\begin{cases}1, & \text{若指派第}i\text{人做第}j\text{事}\\ 0, & \text{若不指派第}i\text{人做第}j\text{事}\end{cases}(i,j=1,2,\ldots,n)$$

- 构建模型如下

$$\min z=\sum_{i=1}^{n}\sum_{j=1}^{n}c_{ij}x_{ij}$$

$$s.t.\begin{cases}\sum_{i=1}^{n}x_{ij}=1 & (j=1,\cdots,n) \quad (a)\\ \sum_{j=1}^{n}x_{ij}=1 & (i=1,\cdots,n) \quad (b)\\ x_{ij}=0\text{或}1 & (i=1,\cdots,n;\ j=1,\cdots,n)\end{cases}$$

- 约束条件(a)表示每件事必有且只有一个人做。
- 约束条件(b)表示每个人必做且只做一件事

集包装问题

- 问题描述:
  - 对于一个 *m* 行 *n* 列的 **0-1** 矩阵，选择不同的列，会产生不同的费用，同时也会覆盖不同的行
  - 问题是：如何选择一些矩阵的列的组合，能使其每行中至多包含一个元素，而且选择费用最小。

$$C = [10 \quad 15 \quad 11 \quad 10 \quad 8 \quad 2]$$

$$A = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

模型描述

- 数学表示为：

$$\min\ f(\boldsymbol{x}) = \sum_{j=1}^{n} c_j x_j$$

$$\text{s.t.}\quad g_i(x) = \sum_{j=1}^{n} a_{ij} x_j \le 1, \quad i = 1,2,\ldots,m$$

$$x_j \in \{0,1\}, \qquad j = 1,2,\ldots,n$$

$$C = [10\ \ 15\ \ 11\ \ 10\ \ 8\ \ 2]$$

$$A = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

where

$$x_j = \begin{cases} 1, & \text{if column j (with a cost } c_j > 0) \text{ is in the solution} \\ 0, & \text{otherwise} \end{cases}$$

- 约束式(1)保证每行至多被一列覆盖
- 约束式(2)是完整性约束。

集包装问题实例

- 某公司计划在3个地区设置销售点，在不同的地区设置不同数量的销售点每月可得到的利润C_{ij}（i地区设置j个销售点的利润）如下表所示，由于资金有限，该公司多只能设置4个销售点.请问在各个地区应如何设置销售点，才能使每月获得总利润最大?

地区	销售点				
	0	1	2	3	4
A	0	16	25	30	32
B	0	12	17	21	22
C	0	10	14	16	17

集包装问题实例

- *定义决策变量如下：*
  - **对于i=1,2,3,j=1,..，4，定义0-1决策变量x_{ij}:**
    - $x_{ij}=\begin{cases}1, & \text{在地区}i\text{设置}j\text{个销售点}\\ 0, & \text{否则}\end{cases}$
- *构建模型如下：*
- $max\ z\ =\sum_{i=1}^{3}\sum_{j=1}^{4}c_{ij}\ *\ xij$
- $\sum_{i=1}^{3}x_{i1}+2*\sum_{i=1}^{3}x_{i2}+3*\sum_{i=1}^{3}x_{i3}+4*\sum_{i=1}^{3}x_{i4}\leq 4$（1）

  $\sum_{j=1}^{4}x_{ij}\leq 1 \quad \forall i=1,2,3$ （2）

  $xij=\{0,1\} \quad \forall i=1,2,3,j=1,\cdots,4$
- 其中：（1）为背包约束
- （2）保证了每个地区只能最多有一种选择

End