题　目　以整数规划为主的数据隐藏模型

摘　要

针对问题一，我们根据已经给出的数据集，对已有数据进行分类，将每一种数据都归为一类。由于每组数据之间都具有差异性，我们将数据之间的差异度作为二元关系，将这些二元关系组合成效益矩阵。通过效益矩阵建立 0-1 规划模型。利用匈牙利法分别求解，得到对应的结果，将两组结果对比。

针对问题二，问题 2 针对的是是多元数据表，对于同一个位置当中的数字可以有更多种情况。先经过初始化，将相同的数据合在一起，产生一个矩阵，矩阵中的行列是第 i 行关于第 j 列合并所需消耗的 * 的个数，可以进行动态规划，对每一列的最少的 * 消耗进行合并，若最少消耗大于 3 则先不分配，全部数据进行过一次处理后，再对没有分配过的数据进行分配，直至所有数据分配完玩，并将上述模型，通过 lingo 进行求解计算，得出多元 1 的结果为 610*，多元 2 解为 7844*。

针对问题三，对于多重保护的问题，我们采用贪心思想获取近似解。贪心的主体算法思想为：依次遍历所有的隐匿维度的组合数，共 2^n 种情况，从只隐匿 1 维—n 维的顺序遍历。当隐匿某种组合的维度时，导致某些数据相同且达到保护限制，则将其移至已配对区，直到所有的数据配对成功。算法的最后可能遗留一些少于最低保护数目的数据，这时仅需要对其进行最优分配即可。最终的结果是：2 元 2 的 3 重保护 687 隐藏量。2 元 2 的 5 重保护 1061 隐藏量。多元 1 的 3 重保护 893 的隐藏量。多元 1 的 5 重保护 1089 的隐藏量。

针对问题四，这是一个有输出模式限定的 2 重保护问题，对于每一行数据，只能隐一个或者是全隐，目标是隐掉的个数最少，在约束条件上，添加每一行的数据最多只能隐掉一个。在这种情况的前提下，可以建立 0-1 规划模型，可以通过列举取得较优的解，但是由于数据量较大，会产生许多的情况。为了简化这个规划问题求解，采用算出比较优的解进行代替，并用数学方法证明出该结果与最优解之间的差距。得到的结果是，由于条件的难以达成性，二元 1 和二元 2 的数据，除了能够配对成功的数据，均需要全隐藏。无单隐情况。

关键词:　0-1 规划模型，贪心算法，近似算法，匈牙利法。

一　问题背景

随着大数据时代的来临，大量用户数据被调查收集或实时抓取并在一定范围内发布共享，这些数据中既包含了用户与平台互动的必要合法信息，但也包含了许多个人隐私信息，而这些个人隐私也会被共享。为了保护每个人的个人隐私，数据的安全问题变得危险起来。为了解决问题，数据的隐藏成为一项重要的工程。在数据隐藏的过程中，既要保证数据能够一定程度上保护个人隐藏，也要保证数据有足够的有效性。

本文旨在通过建立相关的数学模型对已知的数据进行数据的隐藏，找到在不同情形下数据隐藏的相对最优方案。

二　问题分析

已有题目提供的数据如下：

附件：两组二元的数据和两组多元的数据。

问题 1：首先对已有数据进行预处理，将每种完全相同的数据统为一类，优化的目标变为如何将单独的数据与成组的数据互相组合，使得当不存在单独成组的数据时，总的隐藏数量最少。利用二元数据间的差异度，建立效益矩阵，矩阵中的元素值与对应的两条数据的差异度有关，将隐藏数据的数量作为目标函数，保证任何一组数据通过隐藏，存在至少一组其它的数据能与它完全一致，将这个条件作为约束变量，可以得到利用改进匈牙利方法求解 0-1 规划模型，对这个规划模型进行求解，就可以得到相对最优的方案。

问题 2：相对于二元数据表，多元数据表对于同一个位置当中的数字可以有更多种情况。因此对于多元数据而言，两两配对并不一定得到最优解。使用同样的方法对数据进行预分组，并产生分组后的效益矩阵。同样使用问题 1 的模型进行求解。在此基础上尝试优化算法，对原始数据的排序做轻微扰动，逐渐获取隐藏量的最小值

问题 3：考虑多重保护的问题时，模型 1 失效。因为模型 1 只能做到两两配对，不能保证任何一组的保护程度均达到要求。本题中，使用贪心原则，从隐藏维度数小-大依次拟隐藏，将满足 k 匿名的组别剔除，如此循环，最终得到结果矩阵

问题 4：分析题目要求，数据将被分为 3 类：类 1：所有维度数字均相同或已凑对的一组数据。类 2：只有一个维度与其他维度不同的数据且单独成组的数据。类 3：两个或以上维度与其他维度不同的数据且单独成组的数据。分好类后，题目意图简单明了：依题意，类 3 的数据只能全部隐藏。类 2 的数据是需要配对单隐或全隐的。类 1 的数据要么配合类 2 进行单隐，要么由于已配对，无需隐藏。通过贪心原则可以获得分配的满意解。

三　模型假设

1：假设一组数据的所有变量都是敏感数据。

2：假设给出的数据都具有通用性。

四　符号说明

表 1：符号说明

符号	说明
d_{ij}	第 i 组数据与第 j 组数据之间的差异度
x_{ij}	是否将第 i 组数据与第 j 组数据合并
y_{ij}	是否隐藏第 i 组数据的第 j 个数据

五　模型建立

5.1　模型一

5.1.1　模型一的建立

问题一方案的选取，既要保证每个个体的信息都能得到保护，也要使得隐藏数据量最少。由于数据量的选取采用的是二元数据，我们首先引入数据的差异度和效益矩阵两个概念。

定义 1（效益矩阵）：

为表示每一类二元数据之间的关系，我们将第 i 组数据和第 j 组数据之间不同数据的（维度数 * 组成员数）d_{ij} 作为两组数据的差异度指标。将这些差异度组合成一个矩阵，用来表示整个数据集的差异度，并组合成一个矩阵，记为效益矩阵，记为 C

$$C = \begin{pmatrix} d_{11} & d_{12}/2 & \cdots & d_{1n}/2 \\ d_{21}/2 & d_{22} & \cdots & d_{2n}/2 \\ \vdots & \vdots & \ddots & \vdots \\ d_{n1}/2 & d_{n2}/2 & \cdots & d_{nn} \end{pmatrix}$$

例如 C_{12} 表示第一组与第二组不同的数据总量的一半。矩阵的数据表明，$C_{i,j} + C_{j,i} = d_{ij} = d_{ji}$

定义二（隐藏矩阵）利用 0-1 规划的思想，将第 i 组数据与第 j 组数据是否合并作为函数的变量，记为 x_{ij}，并组合成矩阵，作为判别数据是否隐藏的效益矩阵，记为 X, 其中

$$x_{ij} = \begin{cases} 0, & \text{第 i 组数据与第 j 组数据不合并} \\ 1, & \text{第 i 组数据与第 j 组数据合并} \end{cases} \tag{1}$$

$$X = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nn} \end{pmatrix}$$

对于二元 1 的数据，对完全相同的组别进行合并后，变为 44 组，对于二元 2 的数据，对于完全相同的组别进行合并后，剩余 195 组。

将隐藏数据的总量作为目标函数，各组之间的是否两两配对的情况作为约束变量，从而得到求解分配问题的 0-1 规划模型，通过求取最优解，可以找到一种使得不同类型的数据之间差异度最小的方案，从而找到数据隐藏的最优方案。实际上，这将建立一个分配问题的模型：每一组能且仅能与另一组配对。符合匹配问题的“每一行 1 个 1，每一列 1 个 1”的要求，模型建立如下

$$\begin{aligned} \min \quad & = \quad CX \\ s.t. \quad & \begin{cases} \sum_{i=1}^{n} x_{ij} = 1, \quad j = 1,2,3,\ldots,n \\ \sum_{j=1}^{n} x_{ij} = 1, \quad i = 1,2,3,\ldots,n \\ x_{ij} = x_{ji}, \qquad\qquad\qquad\qquad i,j = 1,2,3,\ldots,n \\ x_{ij} \in \{0,1\} \end{cases} \end{aligned} \tag{2}$$

其中，第三组约束条件表明，解矩阵应是一个对称矩阵，这是显然的，因为若 A 组仅与 B 组配对，按照每组仅与 1 组配对的要求，B 组能且仅能与 A 组配对。故 $x_ij = x_ji$

5.1.2 问题一模型的求解

将已经处理的数据利用 lingo 对应两组二元数据进行求解，二元 1 的结果是 20 个隐藏数据，二元 2 的结果是 346 个隐藏数据

index	old	new	index	old	new
5	(1, 0, 0, 0, 0, 1)	(*, *, 0, 0, 0, 1)	71	(0, 0, 0, 0, 1, 1)	(0, 0, 0, 0, 1, 1)
30	(0, 1, 0, 0, 0, 1)		72	(0, 0, 0, 0, 1, 1)	
29	(0, 1, 0, 1, 1, 0)	(0, *, 0, 1, 1, 0)	68	(0, 0, 1, 0, 0, 1)	(0, 0, 1, 0, 0, 1)
35	(0, 0, 0, 1, 1, 0)		50	(0, 0, 1, 0, 1, 1)	(0, 0, 1, 0, 1, 1)
0	(0, 0, 0, 1, 0, 1)	(0, 0, 0, 1, 0, 1)	41	(0, 0, 1, 1, 0, 0)	(0, 0, 1, 1, 0, 0)
10	(0, 0, 0, 1, 0, 1)		74	(0, 0, 1, 1, 0, 0)	
32	(0, 0, 0, 1, 0, 1)	(0, 0, 0, 1, 0, 1)	42	(0, 0, 1, 1, 1, 0)	(0, 0, 1, 1, 1, 0)
8	(0, 0, 1, 0, 0, 0)	(0, 0, 1, 0, 0, 0)	57	(0, 0, 1, 1, 1, 0)	
39	(0, 0, 1, 0, 0, 0)		45	(0, 0, 1, 1, 1, 1)	(0, 0, 1, 1, 1, 1)
11	(0, 0, 1, 0, 0, 1)	(0, 0, 1, 0, 0, 1)	54	(0, 0, 1, 1, 1, 1)	
27	(0, 0, 1, 0, 1, 1)	(0, 0, 1, 0, 1, 1)	75	(0, 1, 0, 1, 1, 1)	(0, 1, 0, 1, 1, 1)
15	(0, 0, 1, 1, 1, 0)	(0, 0, 1, 1, 1, 0)	52	(0, 1, 1, 0, 0, 1)	(0, 1, 1, 0, 0, 1)
18	(0, 0, 1, 1, 1, 0)		61	(0, 1, 1, 0, 0, 1)	
17	(0, 1, 0, 1, 1, 1)	(0, 1, 0, 1, 1, 1)	60	(0, 1, 1, 0, 1, 0)	(0, 1, 1, 0, 1, 0)
9	(0, 1, 1, 0, 1, 1)	(0, 1, 1, 0, 1, 1)	76	(0, 1, 1, 0, 1, 0)	
19	(0, 1, 1, 0, 1, 1)		40	(0, 1, 1, 1, 0, 0)	(0, 1, 1, 1, 0, 0)
24	(0, 1, 1, 1, 0, 1)	(0, 1, 1, 1, 0, 1)	69	(0, 1, 1, 1, 0, 0)	
26	(0, 1, 1, 1, 0, 1)		46	(0, 1, 1, 1, 1, 0)	(0, 1, 1, 1, 1, 0)
37	(0, 1, 1, 1, 1, 0)	(0, 1, 1, 1, 1, 0)	66	(0, 1, 1, 1, 1, 1)	(0, 1, 1, 1, 1, 1)
28	(0, 1, 1, 1, 1, 1)	(0, 1, 1, 1, 1, 1)	79	(1, 0, 0, 0, 0, 0)	(1, 0, 0, 0, *, 0)
13	(1, 0, 0, 1, 1, 1)	(1, 0, 0, *, 1, 1)	59	(1, 0, 0, 1, 0, 1)	(1, 0, 0, 1, 0, 1)
23	(1, 0, 0, 0, 1, 1)		65	(1, 0, 0, 1, 0, 1)	(1, 0, 0, 1, 0, 1)
34	(1, 0, 0, 0, 1, 0)	(1, 0, 0, 0, *, 0)	73	(1, 0, 1, 0, 0, 1)	(1, 0, 1, 0, 0, *)
2	(1, 0, 0, 1, 0, 0)	(1, 0, 0, 1, *, 0)	53	(1, 0, 1, 0, 1, 0)	(1, 0, 1, 0, 1, 0)
7	(1, 0, 0, 1, 1, 0)		43	(1, 0, 1, 1, 0, 1)	(1, 0, 1, 1, 0, 1)
6	(1, 0, 1, 0, 0, 0)	(1, 0, 1, 0, 0, *)	51	(1, 0, 1, 1, 0, 1)	
38	(1, 0, 1, 0, 1, 0)	(1, 0, 1, 0, 1, 0)	56	(1, 0, 1, 1, 0, 1)	(1, 0, 1, 1, 0, 1)
3	(1, 0, 1, 1, 1, 0)	(1, 0, 1, 1, 1, 0)	58	(1, 0, 1, 1, 1, 0)	(1, 0, 1, 1, 1, 0)
12	(1, 0, 1, 1, 1, 0)		44	(1, 1, 0, 0, 0, 0)	(1, 1, 0, 0, *, 0)
20	(1, 1, 0, 0, 1, 0)	(1, 1, 0, 0, *, 0)	49	(1, 1, 0, 0, 0, 1)	(1, 1, 0, 0, 0, 1)
21	(1, 1, 0, 0, 1, 1)	(1, 1, 0, 0, 1, 1)	78	(1, 1, 0, 0, 0, 1)	
22	(1, 1, 0, 0, 1, 1)		47	(1, 1, 0, 1, 0, 0)	(1, 1, 0, 1, 0, 0)
25	(1, 1, 0, 0, 1, 1)	(1, 1, 0, 0, 1, 1)	55	(1, 1, 0, 1, 0, 1)	(1, 1, 0, 1, 0, 1)
36	(1, 1, 0, 1, 0, 0)	(1, 1, 0, 1, 0, 0)	48	(1, 1, 1, 0, 0, 0)	(1, 1, 1, 0, 0, 0)
16	(1, 1, 0, 1, 0, 1)	(1, 1, 0, 1, 0, 1)	70	(1, 1, 1, 0, 0, 0)	
1	(1, 1, 1, 0, 1, 0)	(1, 1, 1, 0, 1, 0)	63	(1, 1, 1, 0, 1, 0)	(1, 1, 1, 0, 1, 0)
4	(1, 1, 1, 0, 1, 1)	(1, 1, 1, 0, 1, 1)	64	(1, 1, 1, 0, 1, 1)	(1, 1, 1, 0, 1, 1)
31	(1, 1, 1, 0, 1, 1)		62	(1, 1, 1, 1, 1, 0)	(1, 1, 1, 1, *, *)
33	(1, 1, 1, 1, 0, 1)	(1, 1, 1, 1, *, *)	67	(1, 1, 1, 1, 0, 0)	(1, 1, 1, 1, 0, 0)
14	(1, 1, 1, 1, 0, 0)	(1, 1, 1, 1, 0, 0)	77	(1, 1, 1, 1, 0, 0)	

图 1: 二元数据一的最优规划方案

-

5.2 问题二

5.2.1 问题二模型的思路

相比于问题一的二元变量，多元变量使得数据隐藏方式方法变多。多元变量的存在，使得数据无法通过两两配对的方式达到规划模型。为解决这个问题，我们采用迭代的方式来一步一步找到最优解。

由问题一可以知道，二元问题的求解，可以通过求解 0-1 分配问题来解决数据隐藏的问题。那么在解决多元问题时，我们改变了数据组自身的差异度 d_{ii}。通过开始时先设定较小的初始值，再进行从而保证一些差异度小的数据组先行完成隐藏。完成之后，可以保证差异度较小的数据互相完成隐藏。同时可以得到新的效益矩阵，再次进行分配问题求解，将差异较大的数据组相互隐藏。不断进行当前步骤，进行迭代，将差异度由小到大的数据逐

步隐藏。

5.2.2 问题二模型的求解

问题二具体算法：

（1）首先，对数据进行了预处理，将每种相同的数据个归为一类，单独的数据也各分为一类，从而把数据分为互不相同的各类。

（2）然后，我们依旧利用第 i 组数据和第 j 组数据之间的差异度 d_{ij}，组合成效益矩阵，作为用于迭代的矩阵，同时将这第 k 次迭代的效益矩阵记为 $Cm(k)$。，我们将其中角线元素 d_{ii} 即数据自身的差异度进行了改变，将设定初始值改为为 $2+0.5*k$，表示数据刚开始进行两两配对时，可能存在某几组数据与其他组数据的差异度较高，不应该立刻合并起来。

$$Cm(k)=\begin{pmatrix} d_{11} & d_{12}/2 & \cdots & d_{1n}/2 \\ d_{21}/2 & d_{22} & \cdots & d_{2n}/2 \\ \vdots & \vdots & \ddots & \vdots \\ d_{n1}/2 & d_{n2}/2 & \cdots & d_{nn} \end{pmatrix} \tag{3}$$

(3) 通过效益矩阵，将所隐藏的位置的数量作为目标行数，各组之间的是否合并的情况作为约束变量，将问题一中的是否隐藏数据的矩阵 X 引入，得到 0-1 规划模型。求解该模型，可以得到数据组合后的不同的许多组集合。根据第 k 次得到效益矩阵 $Cm(k)$ 建立 0-1 规划模型，得

$$\begin{aligned} \min \quad & = \quad Cm(k)X \\ s.t. \quad & \begin{cases} \sum_{i=1}^{n} x_{ij}=1, & j=1,2,3,\ldots,n \\ \sum_{j=1}^{n} x_{ij}=1, & i=1,2,3,\ldots,n \\ x_{ij}=x_{ji}, & i,j=1,2,3,\ldots,n \\ x_{ij}\in\{0,1\} \end{cases} \end{aligned} \tag{4}$$

这里规划问题的求解，我们使用 lingo 进行求解，得到隐藏矩阵 X 的结果，用于判别的迭代的结束。(4) 计算求解第 k 次迭代时的 0-1 规划模型的解。当对于任意 x_{ii} 都未被选中，即所有数据不再单独存在时，可得到此次规划模型的优化方案。否则，对这些集合再次进行如上操作，回到（2），并令 k+=1。求解结果对多元 1 的分类达到了 610 个隐藏数据，对多元二的结果达到 7844，详细的分类请见附录。

5.3 问题三

5.3.1 K-匿名模型的建立

算法建立：贪心思想的算法将按照下述原则进行选取并匿名：

1. n = 数据维度数。i=0. 所有现存数据即为所有数据。

2. 从 n 个维度中挑出 i 个维度 dim[]，对所有现存数据的 dim[] 维度下的数据设置为-1（代指 *）

3. 在修改后的数据中，将所有相同的数据分为一组，形成若干组别。

4. 对成员数 $\geq$ 匿名强度的数据挑出，放入已配对池，现存数据中删除对应数据代表的行。

5. 重复 2-4，直到遍历了所有的组合情况。

6. i++，如果 i=n+1，则到 7，否则到 2.

7. 结束，输出结果，计算总匿除的数据量。

5.3.2 模型的求解

使用 python 对算法进行实现，主要使用的库有，numpy，pandas 以及 itertools。模拟结果显示，二元 2 的 3 重保护的隐去量为 684. 5 重保护的隐去量为 1061. 多元 1 的三重保护的隐去量为 893，5 重保护的隐去量为 1089. 具体分组情况将在附录中体现。

5.4 问题四

5.4.1 排名驱动的模式向量生成算法

在情形一的问题中，所有数据只有全隐、单隐、不隐三个选项，数据中一旦存在两个及两个以上不同的数字，该数据就会被全隐。那么列出数据之间的合并方式，可以发现一组数据中最多只有一个数据可以被隐藏，不然就会被全隐。将第 i 组第 j 个数据的值设为 a_{ij}，将第 i 组第 j 个数据是否隐藏设为 y_{ij}，则

$$y_{ij} = \begin{cases} 0, & \text{第 i 组第 j 个数据隐藏} \\ 1, & \text{第 i 组第 j 个数据不隐藏} \end{cases} \tag{5}$$

那么将数据隐藏的个数作为目标函数，将当前数据的隐藏要求作为约束变量，假设有

n 组数据，每组数据中有 m 个变量，则可以得到基于全隐和单隐的 0-1 规划模型,

$$\begin{aligned} \min \quad & \sum_{i=1}^{n}\sum_{j=1}^{n} x_{ij} \\ s.t. \quad & \begin{cases} \sum_{j=1}^{m} y_{ij} \le 1, \quad i = 1,2,3,\ldots,n \\ \sum_{k=1}^{n} \frac{1}{1+n\sum_{j=1}^{m}(a_{ij}y_{ij}-a_{kj}y_{kj})^2} \ge 1, \quad i = 1,2,3,\ldots,n \\ y_{ij} \in \{0,1\} \end{cases} \end{aligned} \tag{6}$$

5.4.2 排名驱动的模式向量生成算法的求解

由于在情形一中，所有数据只有全隐、单隐、不隐三个选项，那么对应设置三个集合，cost0，cost1，cost2，数据中存在两个及两个以上不同的数字，且该数据数量只有一个，将放入 cost2 集合中，将只有单个不同，且数量只有一个的数据放入 cost1 中，将所有位置都是相同的数字，无论数量多少都放入 cost0，将相同数量大于 1 的数据放入 cost0。cost2 中的数据全隐，cost1 中的数据，在一个位置数字不同，且另外三个位置数字相同作为一组，隐掉一个位置的数字。如在 cost1 中没有该种数据，查找 cost0 中是否有该种数据，如有，则判断 cost0 中该种数据的数量，若大于 2，就只拿出一个和 cost1 中目标数据作为一组，若数量比 2 小，则拿出全部和目标数组组成一组。cost0 中还是没有该种数据，则找 cost0 中是否有 4 个位置相同的数据，且该数字和目标数据中三个位置数字相同，拿出一个，与其组成一组。没有如上数据，cost1 中的目标数据全隐。

可以证明该算法的算法解，和最优解的近似比是 1.5。证明如下：在最优解，和算法接种，cost2 中的数据都要全隐，所以在 cost2 中隐掉的数量是一样的，记为 Z，在最优解中的 cost1 和 cost2 中隐掉的数量记为 X，在算法解中的 cost1 和 cost2 中隐掉的数量记为 Y，Y>X，显然可以证明 Y+Z/X+Z<Y/X。在 cost0 和 cost1 的差距中，主要误差在 cost0 中挑选帮助 cost1 隐藏数据，最优解 cost0 中拿出 1，而在算法解中 cost0 拿出 2，其他的消耗都是一样，记为 W，4 个位置数字完全相同的种类记为 M，那么 $X/Y < W+4*3*M/W+4*2*M < 3/2$。

这样单隐或全隐的 0-1 规划模型的求解，就可以转化求排名驱动的模式向量生成算法的解，利用近似解来代替求解，降低求解难度。

求解过程：首先将数据进行分类，只需考虑 cost1 中的数据和部分 cost0 中的数据的配对问题。对于二元 1，分类结果显示：仅有 34，63 号数据是 cost1 类别。cost0 无元素。34 号数据是 111101，64 号数据是 111110. 无法互相隐蔽，因此，该两条数据均全隐。

事实上，由于数据的极端性，在求解时并未使用上述算法，仅仅在分类后一眼明晰该数据的去向。经过分类验证，二元 1 和二元 2 的数据，除了能配对的数据外，均需全隐。无单隐的情况出现。

六　模型的评价

6.1　模型的优点

1. 对于二元问题时，使用的改进的匈牙利法，进行指派问题求解，可以得到规划问题的最优解。

2. 动态分配的匈牙利规划模型，在不要求高阶的隐私保护，相比于一般的搜寻算法，具有更高的求解精度和求解速度。

3. 使用模拟退火优化的 K-匿名算法，可以通过局部泛化的方式，优化方案，提供更高的精度。

6.2　模型的缺点

1、对于问题 2 的模型精度仍然不够高。

2、问题 2，问题 3 的模型对于数据量很多的矩阵，运行求解会花费很多时间。

3、0-1 规划模型难对大型数据进行求解计算。

参考文献

[1] 岑婷婷, 韩建民, 王基一, 等. 隐私保护中 K-匿名模型的综述 [J]. 计算机工程与应用,2008(4):130-134.

[2] 王良, 王伟平, 孟丹.FVSk-匿名: 一种基于 k-匿名的隐私保护方法 [J]. 高技术通讯,2015(3):228-238.

附录

```
    model:

sets:
N/1..44/;
link(N,N):x,c;

endsets

data:
c=@ole("C:\Users\19800\Desktop\biancheng\数学建模\数据隐藏\res.xls","cost");
@ole("C:\Users\19800\Desktop\biancheng\数据挖掘\result.xlsx", "x")=x;

enddata

min = @sum(link:c*x);
@for(link:@bin(x));
@for(N(i):@sum(N(j):x(i,j))=1);
@for(N(i):@sum(N(j):x(j,i))=1);
@for(N(i):@for(N(j):x(i,j)=x(j,i)));
```

```


import numpy as np
import pandas as pd
def readdata(sheetname):
    df = pd.read_excel("模型1数据.xlsx", header=None,sheet_name=sheetname)
    print("ok")
    s = []
    indexs = [i for i in range(df.shape[0])]
    for index1 in indexs:
        s.append([index1])
        for index2 in indexs:
            if index1 == index2:
                continue
            if df.iloc[index1,:].equals(df.iloc[index2,:]):
                s[-1].append(index2)
        for index2 in s[-1][1:]:
            indexs.remove(index2)
    s.sort(key=len)
    res = np.zeros((len(s),len(s)))
    print("okk")
    for item1 in s:
        for item2 in s:
            a = item1[0]
```

```
            b = item2[0]
            distan = 0
            for i in range(df.shape[1]):
                if df.iloc[a,i] != df.iloc[b,i]:
                    distan += 1
            res[s.index(item1),s.index(item2)] = distan*(len(item1)+len(item2))/2

    for i in range(len(s)):
        if len(s[i]) != 1:
            res[i,i] = 0
        else:
            res[i,i] = 9999
    return res,s


def duqu(filename, s):
    df = pd.read_excel(filename,index_col=0)
    matr = np.asarray(df)
    visited = []
    for i in range(matr.shape[0]):
        for j in range(matr.shape[0]):
            if matr[i, j] == 1 and i!=j and j not in visited and i not in visited:
                visited.append(j)
                s[i].extend(s[j])

    visited.sort(reverse=True)

    for i in visited:
        s.pop(i)

    df2 = pd.read_excel("模型1数据.xlsx", header=None,sheet_name="多元2")
    s.sort(key=len)
    tmp = []
    for i in range(len(s)):
        a = []
        symb = df2.iloc[s[i][0], :]
        for dim in range(df2.shape[1]):
            flag = 1
            for item in s[i]:
                if df2.iloc[item, dim] != symb.iloc[dim]:
                    flag = 0
                    break
            if flag == 0:
                a.append(-1)
            else:
                a.append(symb.iloc[dim])
```

```
        tmp.append(a)

    df3 = pd.DataFrame(columns=["index", "old","new"])
    k = 0
    for item in s:
        for it in item:
            df3.loc[k, 'index'] = it
            old = "("
            new = "("
            for j in range(df2.shape[1]):
                old = old + str(df2.iloc[it, j]) + ","
                new = new + str(tmp[s.index(item)][j]) + ","
            old = old[:-1] + ")"
            new = new[:-1] + ")"
            df3.loc[k, 'old'] = old
            df3.loc[k, 'new'] = new
            k += 1
    df3.to_excel("多元2结果.xlsx",index=False)



if __name__ == '__main__':
    res,s = readdata("多元2")
    #print(s)
    duqu("多元2分配.xlsx", s)
    #df = pd.DataFrame(res)
    #df.to_excel("多元2代价.xlsx")



import numpy as np
import pandas as pd
from random import shuffle
from copy import deepcopy
sheetname="多元2"
df = pd.read_excel("模型1数据.xlsx", header=None,sheet_name=sheetname)


def getMin():
    # 获取极限最小值
    s = []
    for row1 in df.iterrows():
        min_star = df.shape[1]
        for row2 in df.iterrows():
            if row1[0] == row2[0]:
                continue
```

```
            arr1 = np.array(row1[1:])
            arr2 = np.array(row2[1:])
            delta =np.ceil(np.abs(arr1-arr2)/100)
            if np.sum(delta) < min_star:
                min_star = np.sum(delta)
        s.append(min_star)
    return s

def getStars(group1: list, group2: list):
    # 计算两个group合并所需的星星
    arr = np.zeros((len(group1)+len(group2), df.shape[1]))
    arr[0:len(group1),:] = np.asarray(df.loc[group1])
    arr[len(group1):len(group2)+len(group1)+1,:] = np.asarray(df.loc[group2])
    arr = arr - arr[0,:]

    stars = 0
    for i in range(arr.shape[1]):
        if (arr[:,i] == np.zeros((arr.shape[0], 1))).all():
            continue
        stars += 1
    return stars

def firstFenpei():
    #初分配
    indexs = [[i] for i in range(df.shape[0])]
    # 洗混
    shuffle(indexs)
    s = getMin()
    single_pool = deepcopy(indexs)
    win_pool = []

    allstars = 0
    while len(single_pool):
        for index1 in indexs:
            if index1 not in single_pool:
                continue
            for index2 in indexs:
                if index1 == index2:
                    continue
                if index2 not in single_pool:
                    continue
                if getStars(index1, index2) != s[index1[0]]:
                    continue
                if s[index1[0]] != s[index2[0]] and indexs.index(index1) < indexs.index(index2):
                    continue

```

```
                win_pool.append([index1[0], index2[0]])
                single_pool.remove(index1)
                single_pool.remove(index2)
                allstars += s[index1[0]]*2

                break
            if index1 not in single_pool:
                continue
            for group in win_pool:
                delta = getStars(index1, group)*(len(group)+1) - getStars(group, group)*len(group)
                if delta == s[index1[0]]:
                    win_pool[win_pool.index(group)].append(index1[0])
                    single_pool.remove(index1)
                    allstars += delta
                    break


        for index in single_pool:
            s[index[0]] += 1

    print(win_pool, allstars)
    return win_pool, allstars


if __name__ == '__main__':
    firstFenpei()




import numpy as np
import pandas as pd
import pickle
import itertools


def findSame(data):
    s = []
    indexs = list(range(data.shape[0]))
    for index1 in indexs:
        s.append([index1])
        for index2 in indexs:
            if index1 == index2:
                continue
            if (data[index1,1:] == data[index2,1:]).all():
                s[-1].append(index2)
```

```
        for index2 in s[-1][1:]:
            indexs.remove(index2)
    return s



alldata = pickle.load(open("dy1.data", "rb"))
alldata = np.column_stack((np.arange(alldata.shape[0]), alldata))

n = alldata.shape[1]
result = []
min_protect = 3

zuhe = []
for i in range(1, n):
    zuhe.extend(itertools.combinations(range(1,n), i))

thisdata = alldata
delete_rows = []
s = findSame(alldata)
for item in s:
    if len(item) >= min_protect:
        result.append([thisdata[i,0] for i in item])
        delete_rows.extend(item)
thisdata = np.delete(thisdata, delete_rows, axis=0)


allstars = 0
for coms in zuhe:
    delete_rows = []
    # 整体设置星星
    newdata = thisdata.copy()
    newdata[:,coms] = np.ones((newdata.shape[0], 1))*-1
    s = findSame(newdata)
    for item in s:
        if len(item) >= min_protect:
            allstars += len(item)*len(coms)
            result.append([newdata[i,0] for i in item])
            result[-1].append(len(coms))
            delete_rows.extend(item)
    thisdata = np.delete(thisdata, delete_rows, axis=0)
    if thisdata.shape[0] == 0:
        break
```

```python
for i in range(len(result)):
    for j in range(len(result[i])-1):
        result[i][j] = result[i][j] + 2

print(result)
print(allstars)



import numpy as np
import pandas as pd
from chuli import readdata
import copy

df = pd.read_excel("模型1数据.xlsx", "二元1")

def panduan(index):
    arr = np.asarray(df.loc[index])
    arr1 = np.ceil(np.abs(arr - arr[0])/100)
    summ = np.sum(arr1)
    if summ == 0:
        return 0
    if summ == 1:
        return 1
    else:
        return 2

def readdata():
    s = []
    indexs = [i for i in range(df.shape[0])]
    for index1 in indexs:
        s.append([index1])
        for index2 in indexs:
            if index1 == index2:
                continue
            if df.iloc[index1,:].equals(df.iloc[index2,:]):
                s[-1].append(index2)
        for index2 in s[-1][1:]:
            indexs.remove(index2)
    s.sort(key=len)
    return s



all_star = []  # 全隐池
one_star = []  # 单隐池
```

```
no_star = []    # 不隐池
s = readdata()
for item in s:
    if panduan(item[0]) == 0:
        no_star.append(copy.copy(item))
        continue
    if len(item) > 1:
        continue
    if panduan(item[0]) == 1:
        one_star.append(item[0])
    if panduan(item[0]) == 2:
        all_star.append(item[0])

print(one_star)
print(df.loc[32])
print(df.loc[61])
print(no_star)
print(all_star)
```

问题 3 的分组情况展示

每一个中括号中表示的是一组的数据序号。每一个中括号中最后一个元素是该组需要隐蔽的维度。

二元 2 三重

[[24, 73, 100, 1] [4, 50, 57, 1] [28, 153, 157, 1] [29, 38, 104, 2] [37, 106, 116, 2] [117, 159, 181, 2] [84, 85, 125, 2] [36, 168, 183, 2] [81, 93, 134, 2] [58, 99, 193, 2] [17, 79, 114, 3] [2, 19, 169, 3] [47, 92, 197, 3] [34, 65, 133, 3] [12, 87, 194, 3] [31, 94, 166, 3] [23, 40, 147, 3] [10, 167, 179, 3] [46, 60, 176, 3] [83, 171, 198, 3] [42, 61, 66, 3] [96, 131, 187, 3] [91, 139, 196, 3] [16, 70, 170, 3] [80, 130, 144, 3] [6, 44, 78, 3] [48, 75, 113, 3] [54, 118, 164, 3] [49, 121, 190, 3] [82, 152, 172, 3] [59, 62, 109, 3] [53, 55, 140, 3] [67, 161, 189, 3] [51, 146, 155, 3] [115, 122, 180, 3] [18, 20, 77, 3] [56, 165, 192, 3] [89, 182, 185, 3] [21, 22, 41, 3] [39, 43, 142, 3] [101, 110, 137, 3] [3, 162, 175, 4] [64, 107, 163, 4] [98, 136, 150, 4] [124, 149, 158, 4] [123, 127, 173, 4] [72, 90, 143, 4] [52, 120, 126, 4] [45, 102, 156, 4] [14, 135, 195, 4] [9, 88, 97, 4] [35, 71, 188, 4] [69, 86, 111, 4] [95, 160, 200, 4] [76, 141, 184, 5] [30, 68, 199, 5] [27, 74, 103, 5] [8, 132, 178, 5] [13, 138, 186, 5] [15, 33, 191, 5] [112, 154, 174, 5] [25, 26, 148, 5] [5, 145, 177, 6] [7, 129, 151, 6] [11, 105, 119, 6] [32, 63, 128, 9]]

二元 2 五重

[[24, 73, 94, 100, 166, 3], [19, 28, 153, 157, 169, 3], [4, 50, 57, 168, 183, 3], [29, 38, 98, 104, 150, 4], [102, 117, 159, 181, 184, 4], [37, 48, 75, 106, 116, 4], [20, 84, 85, 125, 163, 172, 4], [44, 115, 122, 180, 186, 4], [53, 58, 99, 140, 193, 4], [35, 39, 43, 142, 170, 4], [17, 30, 68, 79, 114, 5], [13, 23, 40, 147, 195, 5], [46, 60, 82, 152, 176, 5], [47, 67, 78, 92, 148, 5], [42, 61, 66, 112, 177, 5], [15, 33, 81, 93, 134, 5], [91, 124, 149, 167, 196, 5], [12, 52, 87, 120, 194, 5], [83, 89, 171, 175, 198, 5], [3, 88, 90, 96, 131, 5], [14, 74, 103, 135, 154, 5], [9, 34, 127, 128, 143, 5], [21, 22, 41, 110, 137, 5], [51, 97, 141, 155, 173, 5], [145, 162, 182, 189, 197, 6], [11, 54,

118, 164, 185, 6], [45, 62, 64, 107, 109, 6], [8, 25, 49, 111, 138, 6], [26, 56, 71, 179, 188, 6], [80, 113, 130, 144, 174, 6], [7, 18, 95, 126, 129, 6], [6, 55, 65, 108, 123, 6], [32, 136, 165, 192, 199, 7], [16, 70, 72, 86, 121, 190, 7], [27, 76, 132, 158, 178, 7], [5, 10, 69, 77, 187, 7], [133, 156, 160, 191, 200, 8], [2, 59, 105, 146, 161, 9], [36, 63, 101, 139, 151, 10]]

多元 1 三重

[[41, 67, 141, 2], [111, 135, 183, 3], [82, 122, 134, 3], [53, 104, 184, 3], [31, 37, 61, 3], [64, 71, 175, 3], [129, 182, 195, 3], [19, 69, 165, 3], [115, 159, 193, 4], [149, 169, 173, 4], [70, 78, 119, 4], [2, 16, 49, 4], [79, 84, 110, 4], [9, 81, 197, 4], [38, 147, 172, 4], [86, 102, 174, 4], [4, 48, 65, 4], [88, 95, 176, 4], [27, 29, 40, 4], [3, 15, 179, 4], [77, 188, 189, 4], [45, 144, 190, 4], [6, 47, 91, 4], [74, 152, 153, 4], [154, 162, 180, 4], [8, 124, 181, 4], [55, 123, 126, 4], [21, 62, 101, 4], [73, 89, 163, 171, 4], [11, 12, 54, 4], [51, 120, 138, 4], [7, 128, 150, 4], [14, 76, 125, 5], [28, 50, 139, 5], [33, 137, 155, 5], [39, 113, 151, 5], [93, 100, 185, 5], [5, 24, 56, 5], [23, 114, 164, 5], [32, 92, 158, 5], [52, 160, 178, 5], [63, 136, 167, 5], [85, 108, 177, 5], [17, 166, 191, 5], [20, 30, 36, 5], [35, 121, 148, 5], [116, 131, 186, 5], [42, 83, 98, 5], [10, 25, 72, 5], [75, 142, 198, 5], [44, 68, 97, 5], [66, 109, 146, 5], [140, 143, 200, 5], [118, 161, 196, 5], [80, 87, 99, 5], [59, 127, 133, 5], [103, 105, 132, 5], [34, 156, 170, 5], [13, 57, 145, 194, 6], [90, 112, 187, 6], [96, 117, 192, 6], [18, 107, 199, 6], [46, 130, 168, 6], [26, 58, 157, 6], [43, 60, 94, 106, 7]]

多元 1 五重

[[41, 67, 128, 141, 172, 4], [3, 115, 138, 143, 159, 193, 5], [23, 37, 48, 114, 164, 5], [52, 53, 104, 160, 178, 184, 5], [9, 81, 105, 150, 197, 5], [32, 64, 92, 121, 169, 5], [12, 82, 96, 122, 134, 5], [79, 129, 188, 195, 196, 5], [42, 65, 83, 84, 98, 5], [14, 88, 95, 152, 176, 5], [2, 16, 28, 49, 57, 5], [6, 44, 47, 91, 181, 5], [56, 154, 162, 180, 189, 5], [5, 19, 38, 46, 94, 147, 5], [86, 102, 109, 135, 174, 5], [51, 77, 110, 183, 186, 5], [11, 17, 58, 78, 131, 5], [69, 139, 142, 173, 198, 5], [33, 40, 140, 177, 192, 5], [10, 55, 59, 126, 133, 5], [20, 21, 62, 101, 149, 5], [73, 89, 163, 171, 199, 5], [8, 27, 29, 137, 155, 6], [18, 76, 85, 108, 125, 6], [24, 36, 45, 97, 130, 6], [93, 100, 156, 161, 185, 6], [34, 35, 127, 166, 179, 191, 6], [39, 113, 123, 151, 168, 6], [66, 118, 136, 158, 194, 6], [7, 68, 72, 106, 117, 144, 6], [13, 31, 60, 61, 145, 153, 170, 6], [25, 43, 103, 111, 165, 6], [26, 71, 74, 157, 175, 6], [4, 15, 70, 119, 182, 7], [22, 107, 120, 124, 146, 7], [30, 54, 90, 112, 116, 148, 187, 7], [50, 63, 75, 80, 87, 99, 132, 167, 7]]