中国研究生创新实践系列大赛

“华为杯”第二十届中国研究生数学建模竞赛

学　　校	杭州电子科技大学
参赛队号	23103360083
队员姓名	1.张辉 2.赵青蕾 3.孙一帆

中国研究生创新实践系列大赛

“华为杯”第二十届中国研究生数学建模竞赛

题　　目　大规模创新类竞赛评审方案研究

摘　　要：

在创新类竞赛的评审中，专家打分往往因没有标准答案以至于评分结果参次不齐，当面临更大规模的赛事时，该问题往往也被逐渐放大，不同专家对同一份作品评分可能出现较大差异。基于此，为寻求解决方案以确保评审的准确性和公正性，本文深入剖析数据关系，由表及里，通过建立数学优化模型并设计有效的启发式算法对问题进行逐步求解。

针对问题一，在作品集与专家组确定的情况下，采用交叉评审的方式往往能够增加作品成绩的可比性，而一个作品的可比性又往往依赖于交叉评阅它的专家数量。因此，保证每名专家拥有一个合适的作品交集是一个增加作品成绩可比性的方案。为了保证每一个专家拥有合适的作品交集，我们主要从三个方面来保证专家之间作品交集的均衡，即**专家评阅作品的总数目、专家与其他专家相交的总人数和专家与其他专家相交的总作品数**，通过维持上述三个目标的均衡来保证作品交集的均衡。具体来说，在模型建立方面，我们依据上述目标，构建**基于混合整数规划的多目标优化模型**，通过**线性加权**的方法，将其转换为单目标模型。在模型求解方面，我们提出了**基于辅助矩阵和滑动窗口的启发式算法**，该算法能够快速且高效产生分配方案。最终我们取得的分配方案中，每位专家评审作品数**稳定在 120 个左右**，每个专家与他有评审作品交集的专家总数**稳定在 122 个左右**，且每个专家与其他专家作品集的交集作品总数**稳定在 600 个左右**。启发式算法时间复杂度为 **O(M+Nlog(N))**。

针对问题二，针对现有评审方案的局限性需要讨论方案优劣以及改进标准分计算模型，首先分析原始数据可知专家评分存在主观性：评分区间具有主观

性、评分分布存在差异性以及评分存在极差大问题，其次分别剖析方案一、二、四的优缺点，考虑使用方案四**两阶段、标准分**的思路缓解专家评分区间及分布差异大的问题，引入**分值偏差**和**排序偏差**的概念计算专家评分可信度，同时使用**层次聚类**划分专家评分权重，建立融合**专家评分可信度**的标准分模型，最后利用附件 2.1，以**重合度最大**为目标寻优得到最合适的专家权重以及标准分计算公式的最优参数 γ=0.9, 得到更加贴合复议方案的作品排序。取复议成绩、方案四成绩与新方案的一等奖作品进行分析，方案四与复议成绩的差异度为 **84**，新方案为 **74**，较方案四有明显提升。算法时间复杂度为 **O(N*log(N))**。

针对问题三，重点在于如何总结规律使“极差”模型能够调整分类后的大极差作品，经过一系列数据分析，确定调整标准分的极差标准和调整标准分的一些规律，构建可以解决**在什么情况下需要调整标准分、怎样调整标准分问题**的“极差”模型。对于模型建立，建立**基于混合整数规划的多目标优化模型**，约束中主要是针对调整标准分位置以及调整数值大小的限制，目标函数设计为最小化极差均值与标准分方差均值。对于模型求解，采用基于极差的启发式方案，设定有效性分数衡量调整方案效果，带入附件进行求解，在数据 2.1 一阶段中，有复议、无复议及设计方案的不一致性分别为 **0、112、66**，在二阶段中分别为 **0、44、14**，分析对比可发现“极差”模型**可以有效降低原始分数的极差**。在程序化第一评审阶段时，关键步骤如下：**低评分作品的过滤、大极差作品的处理、调用极差模型以及结果输出**。启发式算法时间复杂度为 **O(N*log(N))**。

针对问题四要求构建出一个契合大规模创新类竞赛的完整评审模型。对于模型建立，主要从作品分发、第一阶段规则和第二阶段规则三个方面阐述评审模型流程, 应用前三问中**作品分发、新标准分计算公式和大极差调整**的优化模型建立整体评审模型。在模型求解方面，将我们设计的方案应用于数据 2.1 和 2.2，与无复议总分评审方案相比，我们的方案能够将大极差作品的数量从 **28** 降到 **5**，与有复议标准排名差异数值从 **1720** 降到 **676**，表明我们方案的有效性和合理性。启发式算法时间复杂度为 **O(N*log(N))**。

针对问题二和问题三中的超参数进行**灵敏度分析**，可以得出下面的规律和结果。一是问题二的原标准分占比参数变化规律，**超参数越大与最优排名差异度越小**，且当参数值大于 **0.55** 之后就超过了原始方案 4 的表现；二是问题三中的优先级指标的超参数变化规律，**超参数越大则与最优排名之间的差距越大**，当参数值为 **0.3** 时，差异度最小且表现超过原始方案 4。

关键字：混合整数规划模型　贪心算法　专家可信度模型　层次聚类

目录

1. 问题重述

1.1 引言

当前大规模创新类竞赛很多，创新类竞赛的特点是没有标准答案，需要评审专家根据命题人 (组) 提出的评审框架 (建议) 独立评审。但是不同专家评分可能会有自己的主观性，所以当竞赛规模较大、评审专家人数众多时，极差大 (一个参赛作品最高分与最低分的差值) 的问题更为突出。因此，探讨大规模创新类竞赛评审方案的公正性、公平性和科学性具有深远意义。

目前创新类竞赛的评审方案大致包括以下四种：

（1）对每位评审专家的评分进行标准化 (公式见附件 1)，按作品将标准分相加得每件作品总分，依照总分进行排序；

（2）去掉同一份作品得分中的最高分、最低分，再将剩余评分相加，最后依总分排序；

（3）同一份作品如果专家的评分差异 (极差) 较大，组织相关专家协商调整，将调整后得分相加，再依总分排序；

（4）当竞赛规模很大时，首先利用上述方案（1）或（2）或（3）对作品进行初选，再对初选入围的作品组织专家评审（第二阶段评审）或经过答辩等环节确定获奖名单。

这些方案都存在一定的合理性，但也有一定缺点，特别是针对大规模创新类竞赛评审，现有方案偏简单。

为了探索大规模创新类竞赛评审的好方法，附件给出了模拟大规模创新类竞赛的数据。数据包含两阶段评审:

第一阶段：

每个作品有五位专家进行打分，按照附件中标准分计算公式对每个专家的评分进行标准化，求得五个专家标准分加和后的均值，按照均值对所有作品进行排序，筛选出排名靠前的部分作品参加第二阶段评审。

第二阶段：

由三位专家对作品评审，分别取标准分，并对少数极差大作品的标准分进行必要的调整后，再将第一阶段五位专家评审标准分的均值，第二阶段三位专家的评审标准分共 4 份成绩求和，得出每个作品的总成绩。

未参与第二阶段评审的作品总成绩为五位专家标准分的均值，排名始终在进入第二阶段的作品之后。按照上述两阶段计算每个作品得分，得出最终的总排序。请利用这批数据建立数学模型，探讨建立更为合理，公平的评审方案。

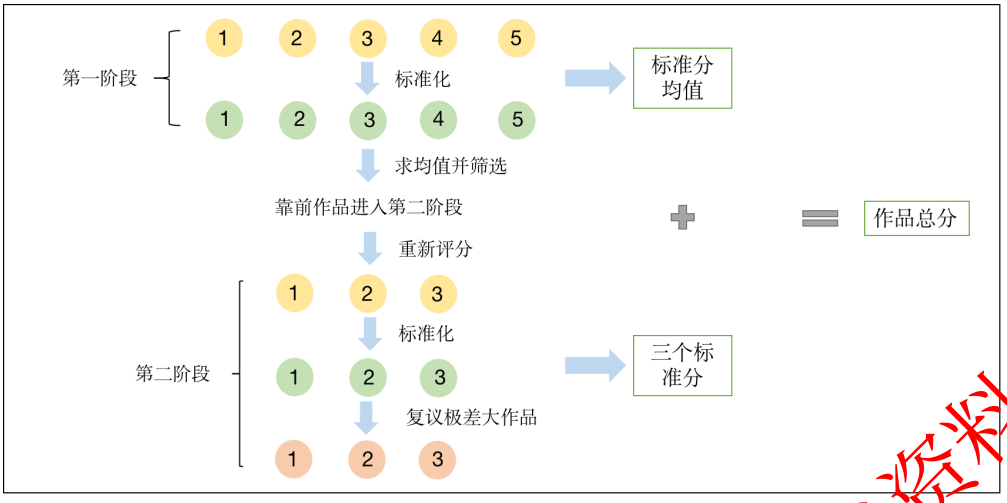

图 1　评审流程图

1.2 问题的提出

1.2.1 问题一

在每个评审阶段，作品通常都是随机分发的，每份作品需要多位评委独立评审。为了增加不同评审专家所给成绩之间的可比性，不同专家评审的作品集合之间应该有一些交集。交集的大小影响成绩之间的可比性。请针对 3000 支参赛队和 125 位评审专家，每份作品由 5 位专家评审的情况，建立数学模型确定最优的“交叉分发”方案，并讨论此方案的有关指标 (自己定义) 和实施细节。

1.2.2 问题二

请选择两种或两种以上现有或自己设计的评审方案和题目附件数据，分析每位专家、每份作品原始成绩、调整之后 (如取标准分) 成绩的分布特点，按不同方案进行排序，并设法比较这些方案的优劣。进而针对大规模创新类竞赛的评审，设计新的标准分计算模型。

一般认为经多专家协商一致的获奖论文具有最大的可信度，附件 2 提供的数据 1，其第二评审阶段评选出的一等奖作品排序是经专家协商取得一致的，请利用这批数据，改进标准分计算模型。

1.2.3 问题三

请根据题目所给的模拟数据 2.1 和 2.2，讨论两阶段的成绩整体的变化和两阶段极差整体的变化，分析两阶段评审方案相比不分阶段评审方案的优劣。注意极差大和创新性强两大特点之间会有一定关系，为了发掘创新论文，请建立“极差”模型（含分析、分类、调整等），并针对所给数据，尝试给出第一评审阶段程序化（不需要人工干预）处理非高且非低分段作品的“大极差”的办法。

1.2.4 问题四

对创新类竞赛，给出一个完整的评审模型（提示：例如优化模型），并针对所给的数据研究如何求解？也可对现行的评审方案给出改进的具体建议（包括未来还要收集哪些数据）。

2. 模型的假设

- 假设问题一中 3000 支参赛队伍和 125 位评审专家随机给出，没有明显差别，这里仅考虑参赛队伍与评审专家的分配问题；
- 假设评审专家原始评审结果是独立且不受其他专家评分的影响；
- 假设评审专家之间的评分差异可以用统计分析方法来衡量；
- 假设最高分段和最低分段的作品不受打分方案影响，仅中间段的作品成绩会因方案不同产生波动；

3. 符号说明

符号	意义
$X_{i,j}$	参数队伍 i 是否分配给专家 j
C_j	专家 j 的评审作品集合
$Y_{j,k}$	专家 j 与专家 k 重复作品集合
$y_{j,k}$	专家 j 与专家 k 重复作品数
$Z_{j,k}$	专家 j 与专家 k 之间是否存在重复作品
$Q_{i,j}$	作品 i 在专家 j 处是否需要调整
$S_{i,j}$	作品 i 在专家 j 处调整的数值
l_j	专家 j 的评审作品总数
$\bar{l}$	所有专家评审作品数的均值
σ_l	所有专家评审作品数的均值
m_j	与专家 j 有重复作品的专家个数
$\bar{m}$	所有专家的重复专家数的均值
σ_m	所有专家的重复专家数的方差
n_j	专家 j 与其他专家重复作品总数
$\bar{n}$	所有专家重复作品总数的均值
σ_n	所有专家重复作品总数的方差
$D_{i,j}$	参赛作品 i 在专家 j 处的得分
$\delta_{0,j}$	专家 j 与专家群体评审意见之间的分值偏差
E_j	专家 j 内部作品评分排序集合
g_i	专家 j 内部参赛作品 i 的总得分
H_j	按照 g_i 进行排序得到专家 j 内包含作品的总分排序集合
$\delta_{1,j}$	专家 j 与专家群体评审意见之间的排序偏差

符号	意义
$P_{0,j}$	专家 j 的分值可信度
$P_{1,j}$	专家 j 的排序可信度
$\chi_{u,v}$	专家 u 和专家 v 的相似系数
W_j	专家 j 的可信度权重
x_k	原始分为 a_k 的评分修改后的标准分值
s_i	修改标准分进行排序后，第 i 个顺序处的作品编号
t_i	标准排名，第 i 个顺序处的作品编号
T	需要复议的作品集合
η_i	第 i 个作品的极差
$P_{i,j}$	作品 i 在专家 j 处的标准分
K	原始标准分均值曲线斜率
K'_j	修改专家 j 标准分后，专家 j 处分数的斜率
V_i	调整标准分后每个作品 i 的标准分方差

4. 问题分析

4.1 问题一分析

题目要求将 3000 支参赛队伍分配给 125 位专家，每个队伍由五位专家评审，给出最优的“交叉分发方案”。由于无法清晰量化方案好坏的标准，因此我们需要先对附件 1 数据进行分析，得出一些好的分发方案的标准指标。

从模型建立方面看，分析附件 1 数据后得出了一些评判分发方案好坏的指标，因此可以用这些指标作为目标函数建立优化模型。使用 0-1 变量表示作品与专家之间的分配关系，需要约束每个作品只有 5 个专家进行评审。

从模型求解方面看，本题的约束条件和目标函数非常清晰，重点就是设计有效的贪心策略在较大的解空间中搜索满足约束条件的可行解，并找到与目标函数最接近的较优解。

本题的**重点和难点**在于需要设计方法缩小求解空间，同时需要设计方法保证产生可行解的过程中一直满足约束条件，这是我们在求解过程中需要重点关注的两个点。

4.2 问题二分析

问题二要求针对现有方案或自己提出方案分析专家、原始成绩和修改后成绩的分布特点，设计新的标准分计算公式，并根据标准数据改进计算模型。

首先需要进行数据的分析，找出当前方案存在的不足从而提出新的标准分计算方案。可以从题目中给出的专家、原始成绩和修改后成绩三点出发进行分析，专家方面，可以分析同一作品不同专家评分差异和不同专家评分分布等特点；原始数据方面，可以分析极差问题和专家评分主观性问题；修改后的数据，可以针对修改方法分析可能存在的缺点并通过数据分析判断是否存在上述缺点。

从模型建立方面看，根据数据分析可以得知当前方案存在的一些缺点，因此需要考虑能够解决上述方案缺点的新标准分计算模型，建立新标准分方案中需要用到的数据的数学求解模型并构建新标准分计算公式。

从模型求解方面看设计新的标准分计算公式后，很可能面临超参数调优问题，如何减少求解时间，在有效时间内找到与标准数据重合度较高的标准分计算模型参数，是需要思考的问题。

本题的**重点和难点**在于新标准分计算公式中新成分的确定以及超参数的调优。需要结合数据分析出当前方案存在问题，找出能够解决此问题的新成分，与原始标准分计算公式做融合。设计新的标准分计算模型后，可能设计调参问题，在有效时间内找到较优的参数也是求解的重点。

4.3 问题三分析

问题三需要先分析附件数据中存在的规律，主要从三个方面进行分析，一是两阶段成绩整体的变化，二是两阶段极差整体的变化，三是两阶段评审方案与不分阶段评审方案的优劣。每个维度都可以从多个角度进行分析，为了建立极差模型，对极差的变化以及其他规应该分析的更为详细，大致可以从以下几个角度进行分析：**复议前后极差变化、复议作品极差范围、复议前后标准分方差变化以及复议改变标准分的规律**等。

从模型建立方面看，根据数据分析可以得知人为复议调整标准分的一些规律和指标变化，由此可以建立一个优化极差模型，约束条件根据分析规律得出调整位置限制、调整数值限制和调整范围限制等内容，目标函数可以根据总结的指标设计，比如最小化调整标准分之后的极差均值等。

从模型求解方面看，在确定了”大极差”规律值后，如何将这些规律转换为具体的寻优策略，是值得思考的；具体来说，在设计调整的位置相关策略的时候，一方面要保证搜索的正确性，同时保证能够搜索到所有可能的位置；在调整的数值方面，往往需要考虑时间开销的问题，即如何保证短时间内能够搜索到一个好的解，而这往往依赖于你的搜索目标。因此，如何设计合理的目标，保证调整方案的有效性，同样是一个值得思考的问题。

本题的**重点和难点**在于调整极差规律的分析和应用，具体来说，在模型建立的过程中，将已有的极差规律转换为一个优化模型是一个重难点，以及在模型求解的过程中，如何依据规律去设置有效的调整策略和搜索策略也是一个重难点。

4.4 问题四分析

问题四要求针对创新类竞赛给出一个完整的评审模型，并针对所给数据研究如何求解。

从模型建立方面分析，结合前面三问，我们可以得出一整个**优化分发作品方案、优化标准分计算模型和优化极差修改方案**的评审流程。因此可以集合前面三问的优化模型和优化方法，建立一个完整的评审优化模型。

从模型求解方面分析，需要按照建立的完整评审模型对原有评分数据进行处理，得出最后的作品排序和获奖名次。

本题的**重点和难点**在于评审模型的完整化。需要将整个评审过程（从一开始拿到所有参赛作品到最后获得每个作品的最终成绩）的处理方法完整阐述并结合前面三问的成果建立优化模型。

5. 模型的建立和求解

5.1 问题一

问题一要求针对 3000 支参赛队和 125 位评审专家，每份作品由 5 位专家评审的情况，建立数学模型确定最优的“交叉分发”方案，讨论此方案的有关指标（自定义）和实施细节。假设 3000 支参赛队伍和 125 位评审专家随机给出，没有明显差别，这里仅考虑参赛队伍与评审专家的分配问题。

5.1.1 数据分析

题目中要求需要在交叉分配的过程中，保证专家之间拥有一个合适的作品交集，以此来增加作品分数的可比性。基于此，我们主要从以下三个方面对附件 1 分发数据进行分析，以此研究如何保证专家之间拥有合适的额作品交集：

1. 每位专家评审的作品总数。

在正常的评审过程中，每位专家应该被分配数量相近的作品数，这样对每位专家都比较公平。因此统计附件 1 中 97 位专家每位专家评审的作品总数，画图查看他们之间的差距。

2. 与任一位专家有交集的专家总数。

为了增加不同评审专家所给成绩之间的可比性，与某一位专家作品集有交集的专家个数应该越多越好，因此统计与每位专家作品集有交集的专家个数并画图查看数值差距。

3. 每位专家与其他专家交集作品总数。

与第二点同理，每位专家作品集与其他专家作品集相交的作品个数应该越多越好，且每个专家的总交集作品数应该保持均衡。因此统计每位专家与其他专家作品交集总数之和，画图查看数值差距。

将上述三个数据的折线图统计在一个图中，具体趋势如下图所示，从图中可以看出每一个指标都处于一种均衡状态。其中每个专家作品集与其他专家作品集的交集作品总数**均维持在 520 左右**，专家测评作品数**维持在 104 左右**，每个专家与其他专家有交集的交集专家**总数维持在 20 左右**。

依据上述的分析，这三个指标有很明显的规律性，因此在设计交叉分配方案时，可从上述三个角度去建立模型并进行求解。

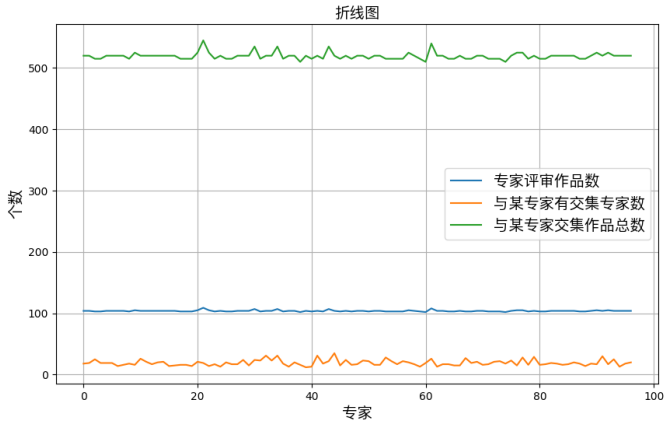

图 2　数据对比图

5.1.2 模型建立

1. 决策变量

(1) 参赛队分配变量 $X_{i,j}$

设队伍集合为 A=1,2,……,3000，专家集合为 B=1,2,……,125。引入 0-1 变量 $X_{i,j}$ 表示参赛队 i 是否被分配给专家 j，若队伍 i 被分配给专家 j，则 $X_{i,j}$ 为 1，若队伍 i 未被分配给专家 j，则 $X_{i,j}$ 为 0，可以表述成下式：

$$X_{i,j}=\begin{cases}0, \text{队伍 i 未被分配给专家 j}\\ 1, \text{队伍 i 被分配给专家 j}\end{cases},\forall i\in A,\forall j\in B \tag{1}$$

则根据变量 $X_{i,j}$ 的值构成的 3000×125 大小的矩阵最终呈现的形式如下：

专家 / 作品	1	2	3	…	125	总计
1	1	0	1	…	1	5
2	0	0	0	…	1	5
3	0	1	1	…	0	5
…	…	…	…	…	…	5
3000	1	0	1	…	0	5

图 3　矩阵示意图

(2) 专家之间重复作品数变量 $y_{j,k}$

根据 $X_{i,j}$ 的值可以确定每个专家评审的作品集合，用 C_j 来表示，具体构成方式如下：

$$C_j = \{X_{i,j} | \forall i \in A, j = j\} \tag{2}$$

引入集合 $Y_{j,k}$ 表示专家 j 评审作品集合 C_j 与专家 k 评审作品集合 C_k 之间的重复作品集合，用 $y_{j,k}$ 表示对应的重复作品数。则 $Y_{j,k}$ 的构成判定方式如下：

$$\begin{cases} i \in Y_{j,k}, C_{j,i} + C_{k,i} = 2 \\ i \notin Y_{j,k}, C_{j,i} + C_{k,i} \neq 2 \end{cases}, \forall i \in A, \forall j \in B, \forall k \in B \tag{3}$$

统计每个 $Y_{j,k}$ 集合内的个数即可得到对应的 $y_{j,k}$ 的值，最终 $y_{j,k}$ 构成的 125×125 矩阵形式如下图所示：

专家 / 专家	1	2	3	…	125
1	120	23	34	…	17
2	23	119	7	…	41
3	34	7	121	…	9
…	…	…	…	…	…
125	17	41	9	…	120

图 4　矩阵示意图

(3) 专家之间是否存在重复作品变量 $Z_{j,k}$

引入一个 0-1 变量 $Z_{j,k}$ 表示专家 j 和专家 k 之间是否存在重复作品，若存在则为 1，不存在则为 0。则 $Z_{j,k}$ 按如下条件取值：

$$Z_{j,k} = \begin{cases} 1, \text{如果} y_{j,k} \neq 0 \\ 0, \text{如果} y_{j,k} = 0 \end{cases}, \forall j \in B, \forall k \in B \tag{4}$$

2. 约束条件

(1) 固定评审参赛作品的专家个数为 5。

每份作品只能由 5 位专家评审，分配的 5 位专家随着分发方案的不同可能会产生差异。在求解分发方案时需要约束作品只能被分发给 5 位专家。

$$\sum_{j \in B} X_{i,j} = 5, \forall i \in A \tag{5}$$

(2) 每位专家评审的作品总数需要保持均衡状态。

专家 j 评审的作品总数 l_j 按照如下方式计算：

$$l_j = \sum_{i \in A} X_{i,j}, \forall j \in B \tag{6}$$

所有专家评审作品数的均值和方差按如下方式计算：

$$\bar{l} = \frac{1}{125} \times \sum_{j \in B} l_j \tag{7}$$

$$\sigma_l = \sqrt{\frac{1}{124} \times \sum_{j \in B} (l_j - \bar{l})^2} \tag{8}$$

设定一个约束值 σ_1 限制专家评审作品总数的方差范围，此值可以通过分析附件 1 数据获得，总体目标是约束每位专家评审作品总数大致相等，可以表示成下式：

$$0 < \sigma_l \leq \sigma_1 \tag{9}$$

(3) 与任一位专家有重复作品的专家个数需要保持均衡状态。

对任一专家而言，与他有重复作品的专家个数 m_j 可以按下式计算：

$$m_j = \sum_{k \in B} \Sigma_{j,k}, \forall j \in B \tag{10}$$

所有专家的重复专家数的均值与方差为：

$$\bar{m} = \frac{1}{125} \times \sum_{j \in B} m_j \tag{11}$$

$$\sigma_m = \sqrt{\frac{1}{124} \times \sum_{j \in B} (m_j - \bar{m})^2} \tag{12}$$

与上一个约束同理，设定一个约束值 σ_2 限制重复专家数的方差范围，使每个专家的重复专家数在一定范围内保持均衡，约束表述为：

$$0 < \sigma_m \leq \sigma_2 \tag{13}$$

3. 目标函数

我们需要设计目标函数以衡量分发方案的优劣，这里考虑两点，涉及每位专家与其他专家重复作品总数的数量和分布。

(1) 专家重复作品总数需要尽可能大。

考虑到题目中要求增加不同评审专家所给成绩之间的可比性，因此专家与专家之间重复作品数量应该越大越好，因此目标一为最大化所有专家重复作品数总和。用 n_j 表示专家 j 与其他专家重复作品总数，计算方式为：

$$n_j = \sum_{k \in B} y_{j,k}, \forall j \in B \tag{14}$$

则目标函数一为：

$$\max \sum_{j \in B} n_j \tag{15}$$

(2) 专家重复作品总数需要尽可能均衡。

考虑到若仅最大化所有专家重复作品数总和，可能会出现有一些专家与其他专家重复作品数很少，而有一些专家与其他专家重复作品数非常大的情况，因此还需要保证每位专家重复作品总数尽可能均衡，计算所有专家重复作品数的均值和方差：

$$\bar{n} = \frac{1}{125} \sum_{j \in B} n_j \tag{16}$$

$$\sigma_n = \sqrt{\frac{1}{124} \times \sum_{j \in B} (n_j - \bar{n})^2} \tag{17}$$

则目标函数二为：

$$\min \sigma_n \tag{18}$$

引入两个超参数 α 和 β 结合两个目标，用于衡量目标函数中两个子目标的侧重点，最终的目标函数为：

$$\max \alpha \times \sum_{j \in B} n_j - \beta \times \sigma_n \tag{19}$$

综上，问题一的**最终模型**为：

$$\max \alpha \times \sum_{j\in B} n_j - \beta \times \sigma_n$$

$$\text{s.t.} \begin{cases} X_{i,j} = \begin{cases} 0, \text{队伍 i 未被分配给专家 j} \\ 1, \text{队伍 i 被分配给专家 j} \end{cases}, \forall i \in A, \forall j \in B \\ Z_{j,k} = \begin{cases} 1, \text{如果} y_{j,k} \neq 0 \\ 0, \text{如果} y_{j,k} = 0 \end{cases}, \forall j \in B, \forall k \in B \\ C_j = \{X_{i,j} | \forall i \in A, j = j\} \\ \begin{cases} i \in Y_{j,k}, C_{j,i} + C_{k,i} = 2 \\ i \notin Y_{j,k}, C_{j,i} + C_{k,i} \neq 2 \end{cases}, \forall i \in A, \forall j \in B, \forall k \in B \\ \sum_{j\in B} X_{i,j} = 5, \forall i \in A \\ l_j = \sum_{i\in A} X_{i,j}, \forall j \in B \\ \bar{l} = \frac{1}{125} \times \sum_{j\in B} l_j \\ \sigma_l = \sqrt{\frac{1}{124} \times \sum_{j\in B} (l_j - \bar{l})^2} \\ 0 < \sigma_l \leq \sigma_1 \\ m_j = \sum_{k\in B} Z_{j,k}, \forall j \in B \\ \bar{m} = \frac{1}{125} \times \sum_{j\in B} m_j \\ \sigma_m = \sqrt{\frac{1}{124} \times \sum_{j\in B} (m_j - \bar{m})^2} \\ 0 < \sigma_m \leq \sigma_2 \\ n_j = \sum_{k\in B} y_{j,k}, \forall j \in B \\ \bar{n} = \frac{1}{125} \sum_{j\in B} n_j \\ \sigma_n = \sqrt{\frac{1}{124} \times \sum_{j\in B} (n_j - \bar{n})^2} \end{cases}$$

5.1.3 模型求解

1. 算法关键思路

本节算法核心是：基于指定目标和题干约束，设计新颖的启发式策略规则。

启发算法通常可以在较快的时间内获得较优解，但容易陷入局部最优，因此，针对该问题，设计一个高效的贪心策略是求解该问题的关键。

该算法大致思路是，首先设计一个辅助矩阵来表示每个作品的所需专家数量，并对专家分阶段进行分配，第一阶段依赖于随机数生成器生成专家对应的分配方案，具体来说，在大量的候选分配方案中，依据候选方案与辅助矩阵的相似性，来筛选部分更加有效的方案。第二阶段的专家评审方案的生成依赖于辅助矩阵的分布特性，设计基于滑动窗口策略的生成方案，其能够优先处理所需专家更多的作品。然后，依据当前分配方案中的专家的评审作品的总数、与其它专家相交的专家总数以及相交的作品总数来设定有效性分数，以此来评判各个候选方案的有效性。最后，依次保留有效分数最高的候选方案作为该专家的分配方案。

表 1　基于辅助矩阵与滑动窗口的启发式算法主要思路

基于辅助矩阵与滑动窗口的启发式算法
（1）**准备阶段：**利用随机数生成器，生成第一个专家的分配方案，若矩阵第 1 个位置的元素为 1 即代表第 1 位专家选择评审第 1 个作品。同时引入辅助矩阵，用于衡量每个作品所需专家评审的数目，即用于保证每个作品有且仅需要五个专家评审。 （2）**两阶段候选方案的生成：** a. **基于余弦相似的候选方案生成策略：**该阶段依赖于随机数生成器生成专家对应的分配方案，具体来说，在大量的候选分配方案中，依据候选方案与辅助矩阵的相似性，来筛选部分更加有效的方案。 b. **基于滑动窗口的候选方案生成策略：**在第 $t-1$ 次遍历之后，依据辅助矩阵，生成候选方案，其目标是保证作品能够相对公平的获得专家的评审。 （3）**候选方案的修正：**在每次遍历中，如果当前最优方案导致作品评审数量约束，则将该方案的对应项修正为 0。 （4）**有效性分数的设定：**依据当前分配方案中的专家的评审作品的总数、与其它专家相交的专家总数以及相交的作品总数来设定有效性分数，具体来说，我们利用线性加权的方法，综合考量多个目标，来设计有效性分数，即分数越大，对应候选方案的越有效。 （5）**交叉分发评审方案输出：**若满足所有约束条件和规则后，输出最终的专家评审方案；若不满足，放宽评审方案限制评审作品数量 C 的限制，进入下一轮迭代。

2. 算法关键步骤

step1: 初始阶段：随机生成一个 1×3000 的一维矩阵作为初始方案，若矩阵第 1 个位置的元素为 1 即代表第 1 位专家选择评审第 1 个作品，指定候选方案中元素为 1 的个数为 C，代表限定选择评审的作品总数为 C。因每个作品需由 5 位专家评审，创建一个同规格且各位置元素为 5 的辅助矩阵，候选矩阵与辅助矩阵做差可表示每个作业的可选专家数量状态。

step2: 候选方案的生成阶段：

a. 一阶段高相似度方案优先：给定一个阈值 t, 在第 1 次到第 t 次遍历中，随机生成 M 个候选方案作为搜索空间，将辅助矩阵和候选矩阵阵抽象为向量，同时求出每个向量与辅助向量的余弦相似度，选取前 N 个高相似度矩阵作为优先候选方案。

b. 二阶段滑动窗口改进方案优先：在第 $t+1$ 次到第 125 次遍历中，按升序对辅助矩阵进行排序，返回长度为 3000 的排序索引，对顺序索引进行切片，取出一块起点为 0、终点为 C 的窗口，从极大候选方案空间随机取出一个候选方案，将矩阵中对应窗口上的顺序索引位置的元素值改为 1，代表专家评审该索引位置的作品。随机选择候选方案空间的另一一维矩阵，从排序索引取出起点、终点位置累加 100 的新窗口，对矩阵进行改进，如此往复直至无法进行切片。按照上述操作改进后的若干个方案作为优先候选方案。

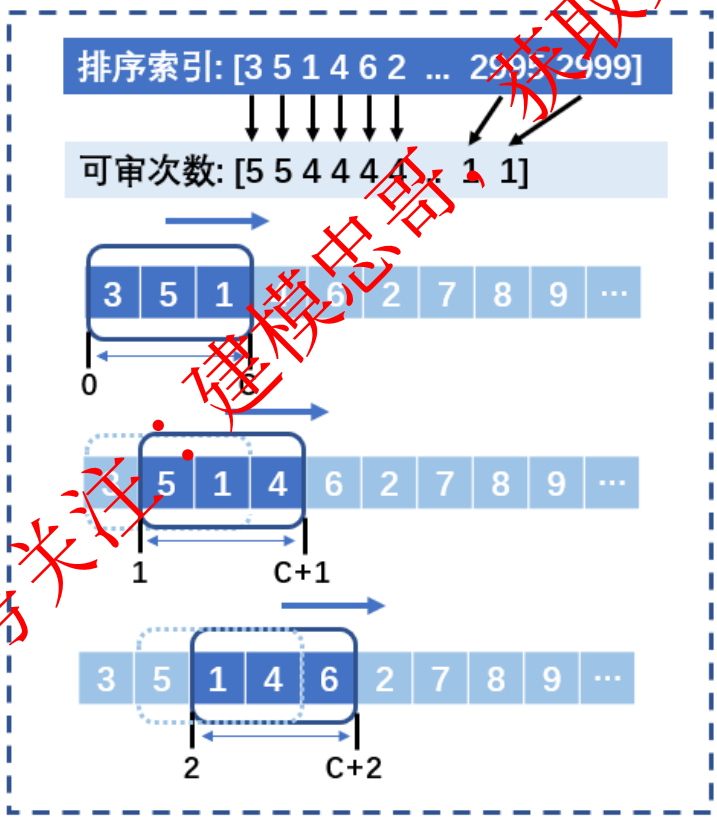

图 5　滑动窗口示意图

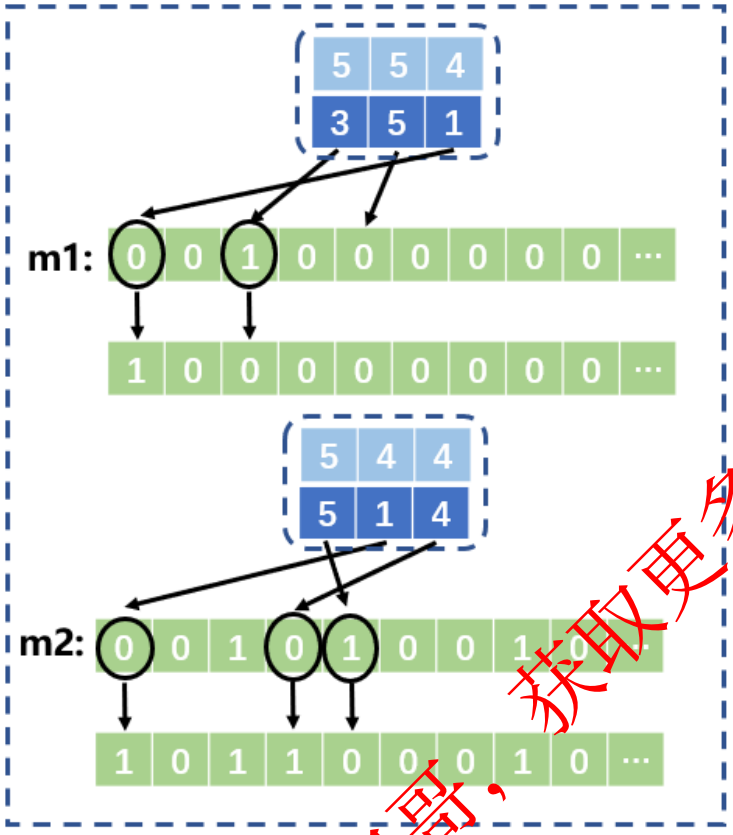

图 6　索引切片插入示意图

step3: 评审专家数量超额修正：在每次遍历中，如果优先候选方案与固定方案的矩阵加和后出现超过 5 的元素，即表示违背 1 个作品只能由 5 位专家评审的约束，需将候选方案的对应索引位置的值修正为 0。

step4: 保留高指标方案：在每次遍历中，最后需计算所有候选方案中中每个方案的指标值，将分数最大的方案保留，作为此次遍历得到某位专家的评审方案。

step5: 交叉分发评审方案输出：若满足所有约束条件和规则后，输出最终的专家评审方案；若为满足，放宽评审方案限制评审作品数量 C 的限制，进入下一轮迭代。

5.1.4 求解结果与分析

1. 求解结果

针对问题一所指定的评审专家与作品规模，结合建立的二次规划模型，首先使用了贪心算法对该问题进行求解，但由于解空间过于庞大，不带有特殊策略的贪心算法收敛速度过于缓慢，求解一次较优方案花费时间长，因此针对问题特性，在贪心算法中加入辅助矩阵和滑动窗口匹配的策略进一步提高算法寻优的

效率，改进求解的部分较优“交叉分发”方案展示如下表，以第一行为例，表示作品 1 由专家 26、专家 53、专家 63、专家 79 和专家 100 五人共同评审：

表 2　改进求解后的“交叉分发”方案

作品	专家一	专家二	专家三	专家四	专家五
1	26	53	63	79	100
2	1	34	78	99	122
3	52	62	78	99	122
⋮	⋮	⋮	⋮	⋮	⋮
3000	36	54	64	80	103

2. 结果分析

从三项指标分析求解获得的“交叉分发”方案，绘制出折线对比图进行对比分析：

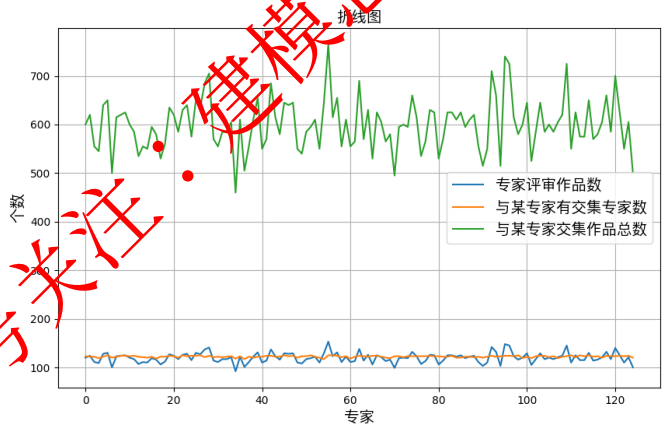

图 7　指标对比图

由上图可知，带策略的贪心算法求解的方案的三项指标均较为平衡，其中每位专家评审作品数稳定在 120 个左右，每个专家与有评审作品交集的专家总计约 122 个，且每个专家与其他专家作品集的交集作品总数稳定在 600 左右。这三项指标的表现说明算法求解的结果中专家评审交集作品数多，专家间的数值差距小，交集作品数均衡，因此可以很好地证明专家们所给的成绩之间具有较强的可比性以及算法的高效性。

5.2 问题二

5.2.1 数据与方案分析

1. 专家评分原始数据分析

针对附件 2.1 中的原始数据进行分析，可以得出专家评分存在主观性的缺点，主要体现在以下三个方面：

(1) 专家评分区间具有主观性。

对于同样的作品集合，不同专家评分可能存在明显差异，有些专家评分普遍偏高，有些专家评分则普遍较低。以 P282 专家和 P672 专家为例，统计两位专家对作品交集中的作品的评分，绘制图像进行对比，可以明显看出对于同样的作品，专家 P282 评分普遍偏高，即时他对作品好坏的评判与专家 P672 是一致的。

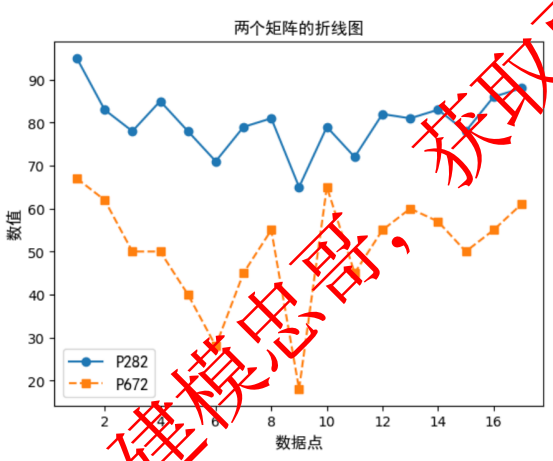

图 8　作品交集评分对比图

(2) 专家评分分布存在差异性。

不同专家对自身评审作品集的评分分布存在差异性，主要体现在有些专家评分区间呈现正态分布，每个分段的评分都存在，但有的专家评分只集中在高分分段，有的专家评分只存在在低分分段，且没有正态分布的规律。

这与专家评审作品集学术水平分布和专家评分习惯均有关系，即便作品水平分布一致，专家评分分布可能也会产生不同，如下图所示。

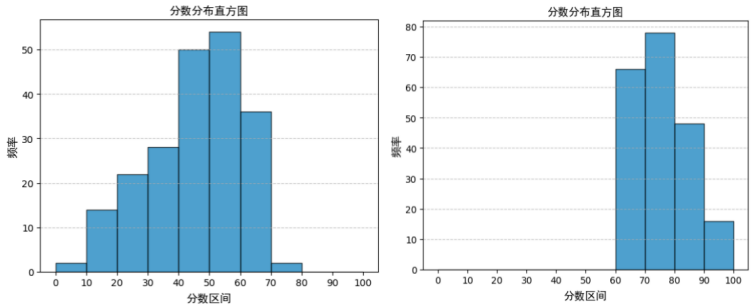

图 9　专家评分分布对比图

(3) 专家评分存在极差大问题。

创新类竞赛的特点是没有标准答案，专家只能按照一定的评审框架独立评审。由于专家评分与专家个人有关，因此同一份作品的几位专家给出的成绩可能会有较大的差异。分析附件 2.1 中第一阶段专家原始评分的极差，发现极差大的作品比较多。

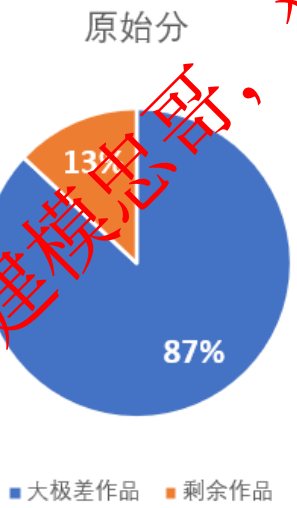

图 10　第一阶段大极差作品占比

2. 方案一分析

方案一的规则是对每位评审专家的评分进行标准化 (公式见附件 1)，按作品将标准分相加得每件作品总分，依照总分进行排序。

(1) 优点

使用标准分对专家评分进行修改，限定专家评分在固定范围内，这样会使具有不同评分区间的专家评分限制在同一范围内，解决了专家评分区间和评分分布的问题。

下图展示了部分专家调整分数为标准分后的分数分布，可以看到每个专家的评分分布基本都呈正态分布，在限定范围的区间内。

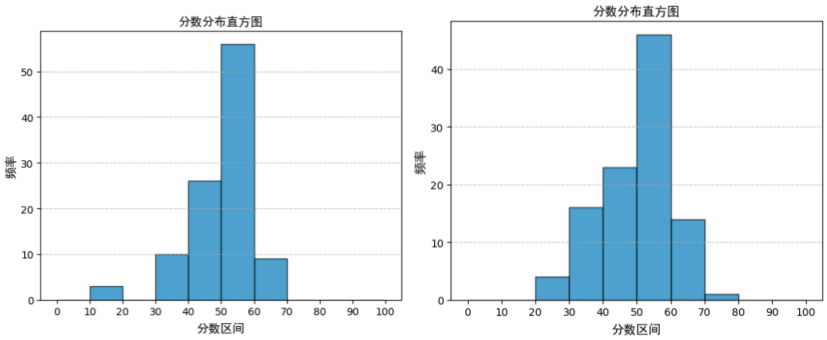

图 11　专家标准分分布对比图

(2) 缺点

第一点：此方法需要假设每个专家作品集学术水平分布保持一致。

若每个专家作品集学术水平分布保持一致，则即使专家评分习惯不同，也能使用标准分归一化到一致区间。但是在大规模创新类竞赛中，不能保证每个专家作品集的学术水平是一致的，专家分到的作品数也不能保证一致。

第二点：没有解决极差大的问题。

使用标准分之后，较多作品评分极差大的问题并没有彻底解决。极差大意味着同一作品多位专家评分存在较大差异，作品的优异性存疑。对比使用标准分前后的极差变化，可以看到极差大的作品占比明显减少，但是大极差作品仍然存在10%，因此极差问题仍然没有被彻底解决。

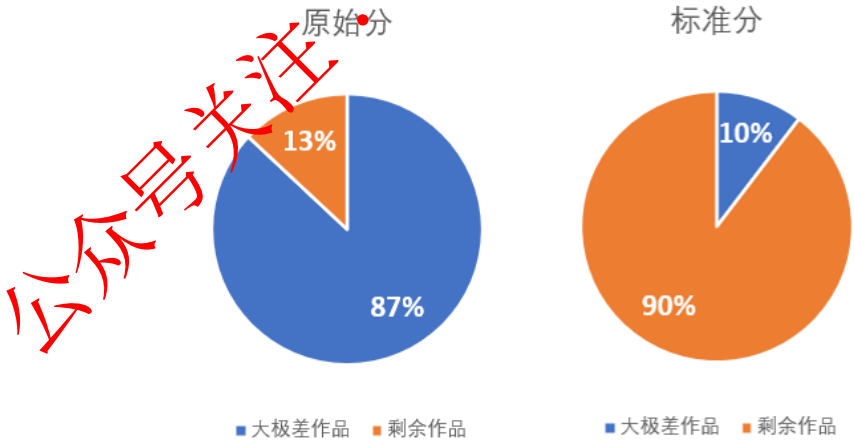

图 12　使用标准分前后极差对比

3. 方案二分析

方案二的规则是去掉同一份作品得分中的最高分、最低分，再将剩余评分相加，最后依总分排序。

(1) 优点

缓解了极差大的问题，去掉最高分和最低分后计算总分，可以减小此作品的极差值，使得评分专家对作品的优劣更加确定统一。下图中对比了附件 2.1 中第一阶段数据按方案二计算总分并排序，前 50 名的前后极差对比结果，极差大的作品明显减少。

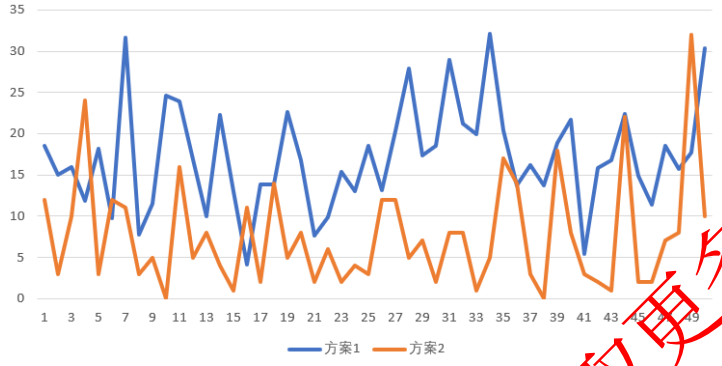

图 13　方案一、二的大极差作品数量对比

(2) 缺点

第一点：作品分数分布变得更加均匀，区分度与原方案相比更小。

去掉最高分和最低分后，导致构成作品总分的主要部分被去掉，作品总分之间的差异性就会变小。因此有可能会出现总分比较大，排名比较高的作品，去掉最高分和最低分后反而排名降低的情况。下图展示了部分原始分的变化曲线和对应的作品去掉最高最低分之后的曲线对比，说明存在作品分数分布更加均匀的问题。

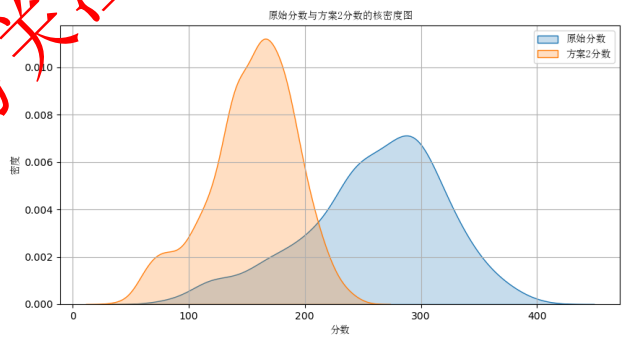

图 14　核密度图

第二点：专家评分分布仍然不统一，依赖于专家评分的主观性。

这里仍然没有解决有些专家评分习惯较高，有些专家评分习惯较低的情况。对于同一作品，习惯评高分的专家会给出较高的分数，但是这并不一定是此作品的最高分数从而被去掉。因此专家评分主观性的评分区间问题没有被解决。

4. 方案四分析

方案四的规则为当竞赛规模很大时，首先利用上述方案（1）或（2）或（3）对作品进行初选，再对初选入围的作品组织专家评审（第二阶段评审）或经过答辩等环节确定获奖名单。

这里假设方案四使用前面的标准分模型计算专家标准分。方案四分为两阶段，第一阶段使用前面方案进行初步筛选，第二阶段采用第二次专家评审或答辩形式获得分数，最后综合两部分分数得到总分进行排名，未进入第二阶段的作品排名一定在进入第二阶段作品排名之后。**这里的方案四不考虑复议的情况。**

(1) 优点

第一点：解决了专家评分区间有差异的问题。

同样使用标准分的模型计算标准分，因此如果专家作品集水平分布一致，可以将不同评分区间的专家分数统一到一个标准区间，解决了专家评分差异的问题。

第二点：分两阶段测评使得结果可靠性更强。

第一阶段使用标准分模型计算总分并排序筛选出前面表现较好的队伍作品，针对这些作品进行第二次评审，第一阶段评审分数仅占很小的一部分，多次不同专家测评增加了评判作品水平的可靠性。

(2) 缺点

两阶段评审过程中仍然存在较大问题没有解决。因为对于一个作品而言，不同专家会有不同看法，不可避免出现评分高低的问题。虽然可以使用标准分将专家作品评分限制在一定范围，但是若某专家认为一作品质量高，其他专家认为此作品质量低，则相应分值会落在标准范围的较大部分和较小部分，从而产生了大极差，这与专家自身有关，依靠标准分等方案无法解决。

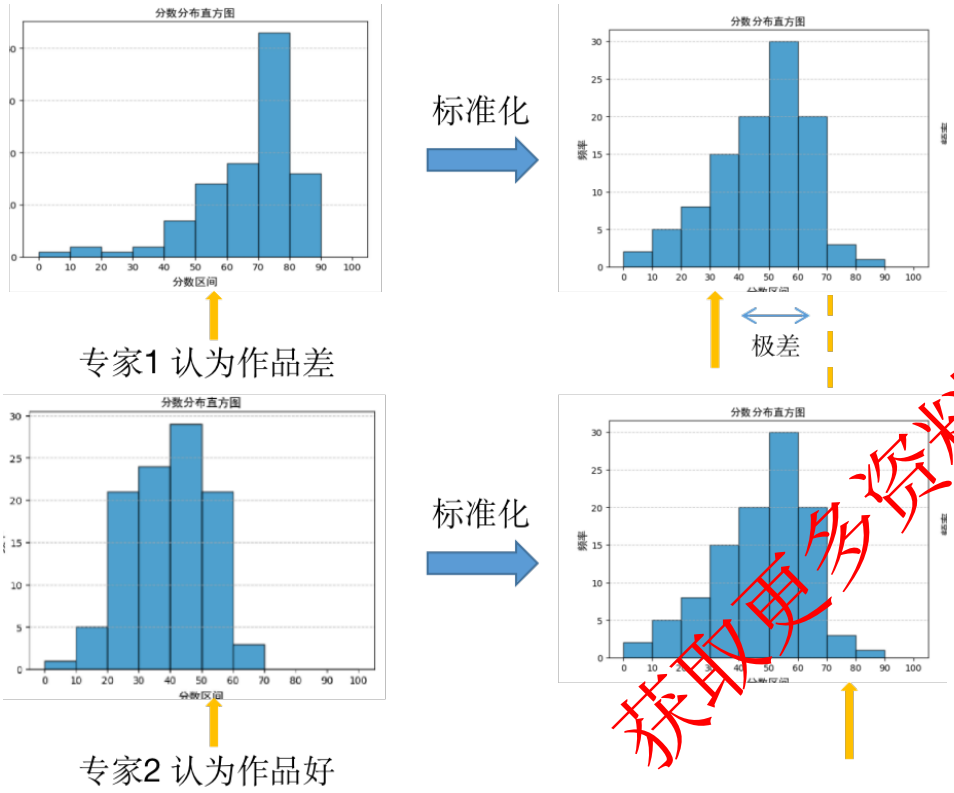

图 15　极差产生原因图

5. 总结

综合上述分析，可以得知原始数据存在专家评分具有主观性的缺点，主要体现在**专家评分区间、专家评分分布和专家评分极差**三个方面。不同的评审方案存在不同的优劣点，**方案一**解决了专家评分区间和评分分布差异大的问题，但是调整后的标准分仍然存在极差大的问题；**方案二**缓解了极差大的问题，但专家自身评分区间和评分分布差异仍然存在；**方案四**可以解决专家评分差异的问题且结果更加可靠，但是仍然存在极差问题。

从上面的结论来看，可以使用方案四中**两阶段、标准分**的思路缓解专家评分区间及分布差异大问题，考虑专家评分主观性的特点，加入**专家评分可信度**，缓解极差问题。意味着如果专家评分可信度较低，那么即使专家认为作品较好给了较高的分数，此分数在最终的分数构成中也会降低占比，这样做就可以从一定程度上缓解极差问题。

因此问题二**求解的重点**在于如何科学合理的计算专家评分可信度，给专家评分赋予权重，并结合之前的标准分计算公式设计出新的，能够解决原始数据存在问题的标准分计算模型。

5.2.2 模型建立

根据上面的分析和对比，专家与专家之间存在差异性，不同专家的评分习惯和评分分布都有不同，因此我们需要思考设计融合了**专家自身评分可信度**的模型。主要分为三步进行整体模型的建立，分别为**计算专家评分可信度、计算专家评分权重和设计新的标准分计算模型**。

1. 计算专家评分可信度。

专家评分可信度可以从多个角度衡量，这里考虑通过专家个体与专家群体评审意见的一致性程度来衡量专家个体评分的可信度。主要从两个方面综合考虑专家评分的可信度，分别为**分值偏差**和**排序偏差**。

(1) 计算分值偏差

由附件数据可知每个参赛作品的五位评审专家组成及其评分，引入一个变量 $D_{i,j}$ 表示参赛作品 i 在专家 j 处的得分，如果参赛作品 i 未被分配给专家 j，则 $D_{i,j}$ 为 0，其中 $i \in A, j \in B$，这里的 A 仍表示参赛作品集合，但是不再是问题一中的 3000 个参赛作品，而是针对附件数据的作品集合，设共有 t 个参赛作品，则 A 的范围为 (1,t)。B 同理，也是针对附件数据的专家集合，设共有 s 个专家，则 B 的范围为 (1,s)。对任意参赛作品 i, 它的总评分为 $\sum_{j \in B} D_{i,j}$。

引入集合 F_j 表示专家 j 评审的参赛作品集合，其中 $j \in B$, 用 r_j 表示集合 F_j 内包含的作品个数。集合构成方式为：

$$\begin{cases} i \in F_j, \text{如果} D_{i,j} \neq 0 \\ i \notin F_j, \text{如果} D_{i,j} = 0 \end{cases}, \forall i \in A \tag{20}$$

专家 j 与专家群体评审意见之间的分值偏差 $\delta_{0,j}$ 为：

$$\delta_{0,j} = \frac{1}{r_j} \times \sum_{i \in F_j} \frac{D_{i,j} - \sum_{j \in B} D_{i,j}}{\sum_{j \in B} D_{i,j}}, \forall j \in B \tag{21}$$

(2) 计算排序偏差

对专家 j 评审的所有参赛作品评分从大到小进行排序，即对 $\forall i \in F_j$，比较 $D_{i,j}$ 的大小，得出专家 j 内部作品评分排序集合 E_j。则参赛队伍 i 在专家 j 中的评分排序为 $E_{j,i}$。

需要计算任一专家 j 内任一参赛作品 i 的总得分 g_i，公式为：

$$g_i = \sum_{j \in B} D_{i,j}, \forall i \in F_j \tag{22}$$

对于 $\forall i \in F_j$，按照 g_i 进行排序得到专家 j 内包含作品的总分排序集合 H_j，则作品 i 在集合 H_j 中的排序为 $H_{j,i}$。

则专家 j 与与专家群体评审意见之间的排序偏差 $\delta_{1,j}$ 的计算公式为：

$$\delta_{1,j}=\frac{1}{r_j}\times\sum_{i\in F_j}\frac{|E_{j,i}-H_{j,i}|}{r_j-1},\forall j\in B \tag{23}$$

2. 计算专家评分权重。

上述步骤可以得到每个专家 j 的分值偏差和排序偏差，偏差越小越好，意味着专家的可信度更高，在计算作品总分时应该赋予此专家更高的权重。

可以利用层次聚类的分析方法，根据 δ_{0j} 和 δ_{1j} 的值对专家 j 进行分类，具体步骤分为**确定专家可信度矩阵、确定专家相似矩阵和层次聚类得到专家类别并赋予相应权重**三步。

(1) 确定专家可信度矩阵。

引入新的变量 $P_{0,j}$ 和 $P_{1,j}$ 表示专家 j 的可信度值，其中 $j\in B$，具体计算公式为：

$$P_{0,j}=1-\delta_{0,j} \tag{24}$$

$$P_{1,j}=1-\delta_{1,j} \tag{25}$$

则专家的可信度矩阵为：

$$\begin{bmatrix}P_{0,1} & P_{1,1}\\ P_{0,2} & P_{1,2}\\ \vdots & \vdots\\ P_{0,j} & P_{1,j}\end{bmatrix} \tag{26}$$

(2) 确定专家相似矩阵。

按数量积法确定相似系数，任意专家 u 和专家 v 的相似度计算方式如下，式中 M 的计算方式也已经给出：

$$\chi_{u,v}=\begin{cases}\frac{P_{0,u}\times P_{0,v}+P_{1,u}\times P_{1,v}}{M}, u\neq v\\ 1, u=v\end{cases},\forall u\in B,\forall v\in B \tag{27}$$

$$M=\max\{P_{0,u}\times P_{0,v}+P_{1,u}\times P_{1,v}\},\forall u\in B,\forall v\in B \tag{28}$$

最终得出由 $\chi_{u,v}$ 构成的相似矩阵 $R(\chi_{u,v})_{s\times s}$，此相似矩阵 R 是一个对称矩阵。

(3) 层次聚类得到专家类别并赋予相应权重。

得到相似矩阵 R 后，可以利用层次聚类算法对专家进行分类，设分为 n 类，每一类中包含专家集合为 B_n，用 b 表示每一类内包含的专家个数，计算每一类内部包含专家对应的可信度的均值 $\bar{n}$，公式为：

$$\bar{n}=\frac{\sum_{j\in B_n}(P_{0,j}+P_{1,j})}{b} \tag{29}$$

按照 $\bar{n}$ 的相对关系对每一类分类赋一个权重，属于同一类别的专家权重相同，最终得出每位专家的可信度权重 W_j。

3. 设计新的标准分计算模型。

根据前面的分析可以得知方案 1,2 和 4 都没有考虑到专家评分差异的问题，因此我们设计了计算专家可信度从而给专家赋权重的方案，结合前面方案的一些优点，得出了能够解决**专家评分差异、专家评分分布差异和极差较大问题**的新标准分计算模型，

首先计算每个专家评审作品评分的均值和方差,假设专家 j 评分为 $a_1, a_2, \ldots\ldots, a_k$（不同专家矩阵不同），设专家共评审 n 个作品，则均值和方差的计算公式为：

$$\bar{a} = \frac{1}{n}\sum_{k=1}^{n} a_k \tag{30}$$

$$s = \sqrt{\frac{1}{n-1}\sum_{k=1}^{n}(a_k - \bar{a})^2} \tag{31}$$

新标准分计算模型的第一部分 $x_{k,1}$ 按照原始标准分计算方式进行计算。第二部分应该考虑专家可信度的问题，即如果专家的可信度低，与专家群体评分差距较大，则此专家的评分不可信，应该降低此专家评分在最终得分中的占比。则第一部分 $x_{k,1}$ 与第二部分 $x_{k,2}$ 计算公式为：

$$x_{k,1} = 50 + 10 \times \frac{a_k - \bar{a}}{s} \tag{32}$$

$$x_{k,2} = W_j \times a_k \tag{33}$$

结合上述两个部分，用参数 γ 区分对两个部分的侧重点，因此最终的标准分计算公式为：

$$x_k = \gamma \times x_{k,1} + (1-\gamma) \times x_{k,2} \tag{34}$$

使用不同的参数计算标准分会得到不同的排序结果，因此我们需要设计一个目标从而找到较优的参数值。根据新的标准分公式计算标准分并按照规则统计总分进行排序后，设原一等奖对应的个数为 n，则设使用新标准分计算模型之后前 n 个位置上的对应作品编号为 s_i，设标准排名前 n 个位置上的对应作品编号为 t_i，$i = 1, 2, \ldots\ldots, n$。则最终的目标是减少改变前后我们的方案与标准排名之间的差异度：

$$\min \sqrt{\sum_{i=1}^{n}(s_i - t_i)^2} \tag{35}$$

改变参数计算标准分得到不同的排序结果，计算与标准排名的差异度，差异度最小的就是参数的最优解。、

综上，问题二的**最终模型**为：

$$\min \sqrt{\sum_{i=1}^{n}(s_i - t_i)^2}$$

$$\text{s.t.} \begin{cases} \begin{cases} i \in F_j, \text{如果} D_{i,j} \neq 0 \\ i \notin F_j, \text{如果} D_{i,j} = 0 \end{cases}, \forall i \in A \\ \delta_{0,j} = \dfrac{1}{r_j} \times \displaystyle\sum_{i \in F_j} \frac{D_{i,j} - \sum_{j \in B} D_{i,j}}{\sum_{j \in B} D_{i,j}}, \forall j \in B \\ g_i = \displaystyle\sum_{j \in B} D_{i,j}, \forall i \in F_j \\ \delta_{1,j} = \dfrac{1}{r_j} \times \displaystyle\sum_{i \in F_j} \frac{|E_{j,i} - H_{j,i}|}{r_j - 1}, \forall j \in B \\ P_{0,j} = 1 - \delta_{0,j} \\ P_{1,j} = 1 - \delta_{1,j} \\ \chi_{u,v} = \begin{cases} \frac{P_{0,u} \times P_{0,v} + P_{1,u} \times P_{1,v}}{M}, u \neq v \\ 1, u = v \end{cases}, \forall u \in B, \forall v \in B \\ M = \max\{P_{0,u} \times P_{0,v} + P_{1,u} \times P_{1,v}\}, \forall u \in B, \forall v \in B \\ \bar{n} = \dfrac{\sum_{j \in B_n}(P_{0,j} + P_{1,j})}{b} \\ \bar{a} = \dfrac{1}{n} \displaystyle\sum_{k=1}^{n} a_k \\ s = \sqrt{\dfrac{1}{n-1} \displaystyle\sum_{k=1}^{n} (a_k - \bar{a})^2} \\ x_{k,1} = 50 + 10 \times \dfrac{a_k - \bar{a}}{s} \\ x_{k,2} = W_j \times a_k \\ x_k = \gamma \times x_{k,1} + (1 - \gamma) \times x_{k,2} \end{cases}$$

5.2.3 模型求解

1. 标准分模型设计思路

表 3　算法求解主要思路

算法
（1）方案优劣分析：由原始分数据，可知隐含存在的问题主要有四个：极差问题、专家打分区间偏好问题、专家区间内打分偏好问题及专家自身的可信度问题。以附件数据依次验证方案，匹配各方案暴露的问题。 **（2）专家可信度计算与赋权：**专家之间水平不一，其自身因素会影响成绩认定的公平性，利用原始分计算专家可信度，确定专家可信度矩阵与相似矩阵，利用层次聚类对专家进行分类并赋权，旨在削弱专家差异对评分带来的影响。 **（3）融合标准分计算模型设计：**针对方案设计出融合标准分计算模型，标准分计算公式保留原始标准分计算方式，并融合每个专家可信度权重后的原始分，降低低可信度专家分数效力，提升高可信度专家分数效力，平衡专家差异性，最后对这两部分分数赋予权重参数加权求和计算最终的新标准分，达到进一步削弱原始分极差影响的目的。 **（4）改进融合标准分计算模型：**使用融合标准分计算模型求解附件数据得到作品排序后，与一等奖排序差值的绝对值最小为目标，通过不断调整权重参数寻找最佳匹配的最优参数，确定最终的新标准分计算模型。

5.2.4 求解结果与分析

1. 求解结果

首先通过层次聚类获得各位专家所属大类，42 位专家分成了 **2** 大类，具体分类情况如下图所示：

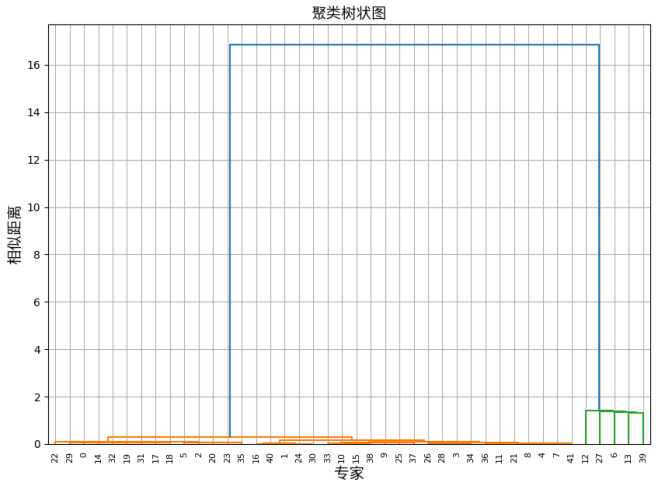

图 16　聚类结果

确定专家所属分类后，根据大类内专家平均可信度，计算出大类的初始可信度权重，以重合度最大为目标迭代寻优得到大类 1 的权重取值 **1.02**，大类 2 的权重取值 **0.82**，参数 γ 取值为 **0.9**。

（1）最终标准分计算公式

结合寻优后得到的最优参数，最终的标准分计算公式表示为：

$$x_k = 0.9 \times (50 + 10 \times \frac{a_k - \overline{a}}{s}) + 0.1 \times W_j \times a_k \tag{36}$$

（2）新公式计算后的排名

使用新标准分计算公式带入附件 2.1 数据，可以得到所有作品的成绩排序，以下是截取其中的一等奖名单：

表 4　新标准分排序后的一等奖名单

名次	1	2	3	4	5	6	7	8	9	10	11	12
方案四	1	2	3	4	5	7	10	12	8	20	16	19
新方案	1	2	7	4	3	5	20	8	11	12	10	23
名次	13	14	15	16	17	18	19	20	21	22	23	-
方案四	11	14	13	23	18	22	15	9	17	21	6	-
新方案	14	13	15	17	16	18	21	19	22	9	6	-

2. 求解结果分析

取复议成绩、方案四成绩与新方案的一等奖作品进行分析，可以得到不同方案下专家评分分布图：

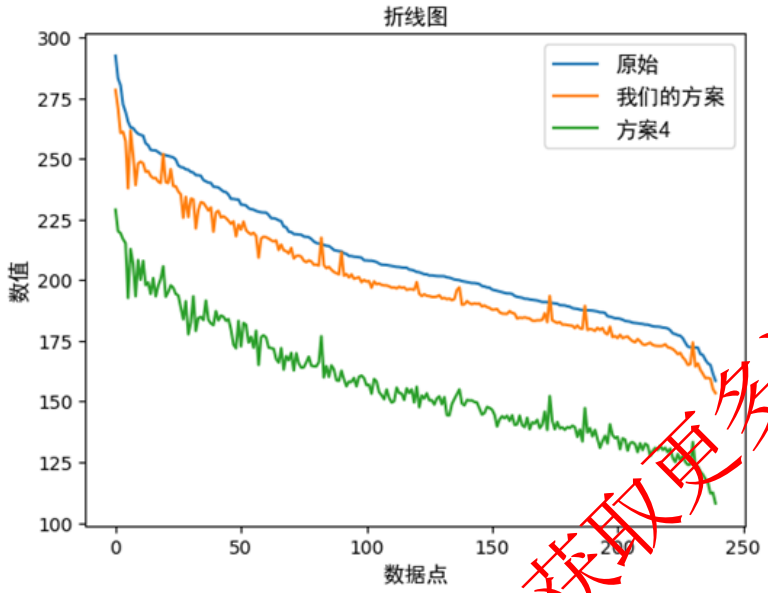

图 17　专家评分分布对比图

由图可知，新设计的标准分计算模型相对于两阶段不复议方案的评分更接近于复议方案，并且分数折线相较于不复议方案更加平稳，波动较少，更加拟合最优的复议方案。分别计算所有作品的复议方案分数与设计方案、方案四评审分数的差值平方和，开根号后可以得到不同方案与复议方案之间的的方差，即**差异度**，方差越小代表越贴合复议方案，一等奖的排序更加可靠。计算后得到方案四的差异度为 **84**，设计方案的差异度为 **74**, 较方案四有明显提升。

5.3 问题三

5.3.1 数据分析

1. 两阶段成绩整体变化

(1) 成绩均值

以附件 2.1 中能进入两阶段的作品为测试数据，计算一阶段和两阶段成绩的均值进行对比：

由下图可知，**成绩均值分布层次化**：一阶段评审方案的成绩均值分布区间较小，区间内波动多且幅度大，说明**一阶段方案难以对作品水平划分层次**，同时 5 位专家的打分有一定差异，导致均值波动；而**二阶段方案**对前 5 位专家的 1 份标准平均分和后 3 位专家的 3 份标准平均分求和后，成绩均值分布区间变大，其均值折线也更加平稳，**可以有效划分出作品的水平高低**。

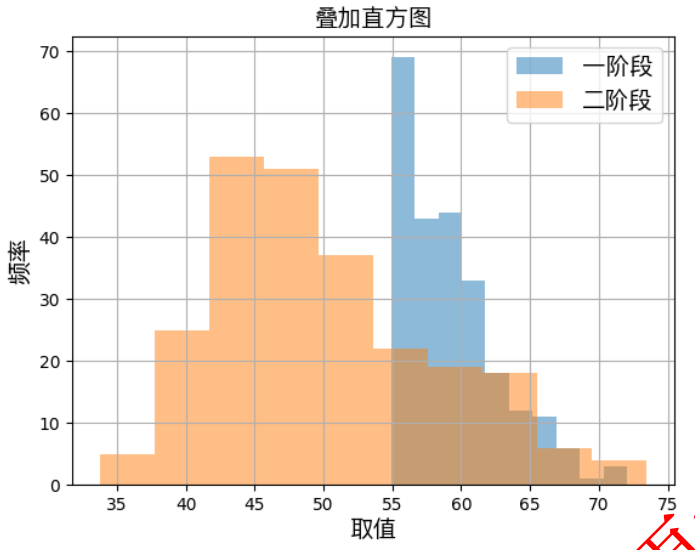

图 18　一阶段与两阶段成绩均值分布

(2) 区域分布

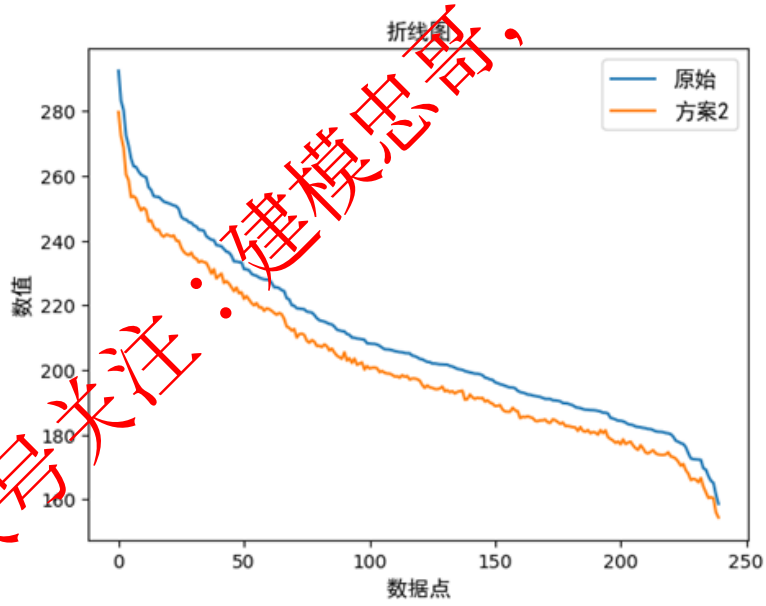

图 19　一阶段与两阶段成绩均值分布

对比多个方案的排序结果，发现专家们对于最高分和最低分的作品意见基本一致，差异非常小，一般专家会在中间段的作品中产生不同意见。以方案四和复议后的方案四为例，选择第二阶段的分数均值，其中选取最高的分数以及最低的分数，计算该中间阶段的分布曲线（这里分数选择尽量选择在其他方案中依旧能够保证排名的分数），可以看出两条曲线头部和尾部较为陡峭，斜率更大，而中间部分可以近似看成有固定斜率的一条直线，因此在具体的调整过程中，调整

后的标准分尽量依旧能够保证类似的分布。

根据以上数据分析并结合附件数据，总结出以下**从成绩分析角度的规律：**

a. 两阶段评审相较于一阶段评审，能更有效地区分出作品层次。

b. 复议调整时，针对中间部分的大极差作品，调整后标准分均值曲线的中间部分最好能够近似成一条直线。

2. 两阶段极差整体变化

下图中一阶段和二阶段的小提琴图所展示的数据分布形状比较相似，从极差整体变化角度分析，可认为一阶段和两阶段的极差区间与波动相似，究其原因首先可能是两个阶段内的专家都是孤立的，不像成绩分布变化中最终成绩会参考一阶段的专家标准平均分，其次，两个阶段专家人数不同，一阶段专家人数多差异性大，极差大的概率也因此相对较大。接下来考虑从复议前后两个阶段对极差变化进行分析:

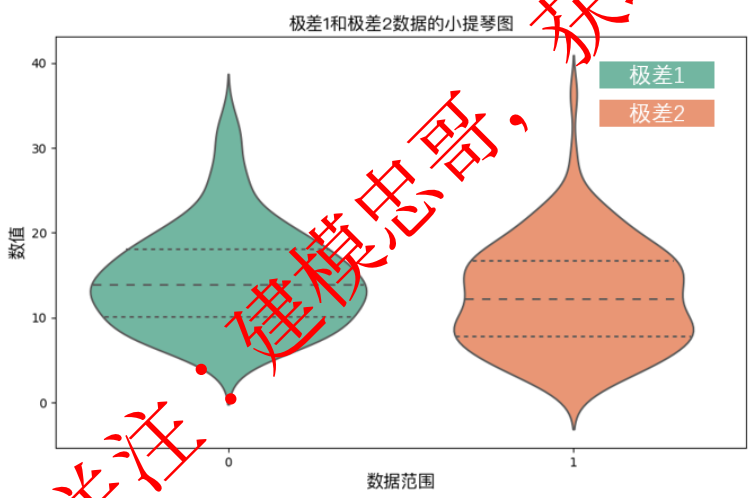

图 20　一阶段与两阶段极差对比

(1) 复议阈值

通过分析进入复议的作品数据，可以明显看出该方案以**二阶段极差值是否大于等于 20** 作为可以进入复议评审的评判标准，如果是，则需要专家复议调整标准分。

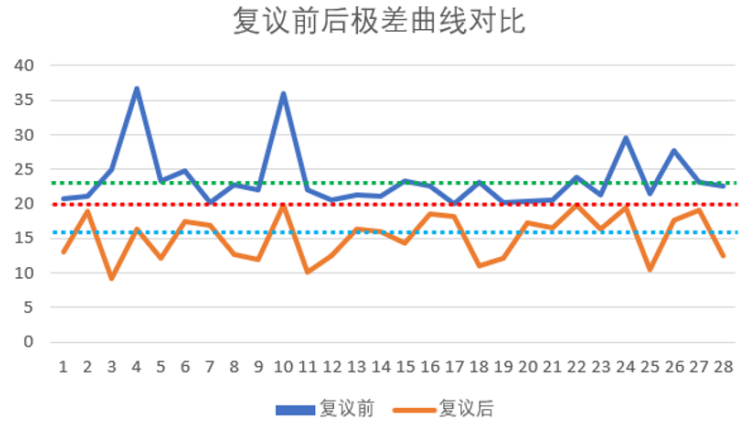

图 21　复议前后极差对比

(2) 极差均值

进一步分析上图复议前后的极差均值。复议后的极差曲线均在两阶段的极差曲线下方，如蓝色、绿色虚线所示，复议前的极差均值约为 **23.47**，复议后的极差缩小至 **15.26**，说明专家复议调整标准分后，可以有效控制**极差均值变小**。

(3) 标准分方差

同样地，对于复议前后标准分方差的变化如图中所示，复议后标准分均比复议前小，并且曲线上的波动小了很多，各专家复议前的标准分方差在 150 上下浮动，标准分复议后的方差在 70 上下浮动，说明专家复议调整后各位专家的**标准分趋近**。

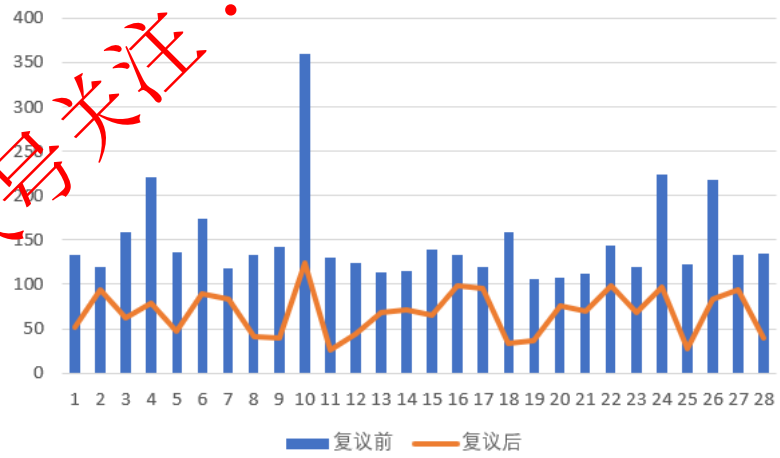

图 22　复议前后标准分方差对比

结合以上数据分析并结合附件数据，总结出以下关于**大极差作品标准分复议调整的规律**：

a. 第二次评审标准分极差不小于 20 的作品被认定位大极差，需要由专家复议，重新调整标准分。

b. 复议调整时，专家标准分与标准分均值差值越大，优先调整该专家的标准分。

c. 复议调整时，如果存在两个标准分大于标准分均值的情况或者两个标准分都小于标准分均值的情况，一定会调整降低或者调高其中一位专家的标准分。

3. 两阶段方案优劣分析

综上所述，两阶段方案相较于一阶段方案，可以突出不同水平的作品，**区分度高**，第一阶段使用标准分模型计算出总体表现比较好的队伍作品，第二阶段会有更多专家结合第一阶段的结果进行评分得到更可靠的作品排序方案，但是更多专家差异性评阅也会**带来更不确定的极差**，因此两阶段仍然会存在因专家主观因素影响带来的劣势。

5.3.2 调整模型建立

已知附件 2.1 和 2.2 数据中第一和第二阶段专家评定标准分、第二阶段复议的作品、复议作品前后的标准分、第二阶段复议前后极差变化等数据内容。经过了一系列数据分析之后，已经确定了需要调整标准分的极差标准和调整标准分的一些规律，接下来会按照上述分析结果建立“极差模型”，即**解决在什么情况下需要调整标准分、怎样调整标准分的问题**。假设在建模过程中使用附件 2.1 中第二阶段的数据做示例，其他同样类型的数据同样适用此建模。

1. 决策变量

通过分析附件 2.1 数据可知，当专家评议标准分**极差大于 20** 时，就需要组织专家进行复议，调整标准分。考虑到其他数据规模或专家评分可能有差异，在建模过程中我们使用符号 ϵ 表示极差界限，当极差大于此界限时，表示需要复议。此数值可以根据具体数据分析得到，一般取极差的前 10% 或前 20% 确定。在附件 2.1 和 2.2 数据中，$\epsilon = 20$

引入符号 η_i 表示作品 i 的极差，其中 $i \in A$，A 表示附件数据中的作品全集。如果作品 i 的极差超过 ϵ，那么作品 i 属于需要调整标准分的集合 T，否则不属于此集合，表述为：

$$\begin{cases} i \in T, \eta_i \geq \epsilon \\ i \notin T, \eta_i < \epsilon \end{cases}, \forall i \in A \tag{37}$$

引入常量 $P_{i,j}$ 表示参赛作品 i 在专家 j 处的标准分，其中 $i \in T, j \in B'$，此处的参赛作品是需要调整标准分的作品集合内的元素，专家是统称为专家 1、专家 2 和专家 3 的集合 B' 内的元素，即 $B' = 1, 2, 3$。即我们可以得到任一参赛作

品在专家 1、专家 2 和专家 3 处的标准分，这里每个作品的评审专家仍然是不同的，仅用专家名称区分位置。

(1) 调整位置变量

引入 0-1 变量 $Q_{i,j}$ 表示作品 i 在专家 j 处是否需要调整，如果需要调整，则 $Q_{i,j}$ 为 1，否则为 0，表述为：

$$Q_{i,j}=\begin{cases}1, \text{如果第 i 个作品在专家 j 处的标准分需要调整;}\\0, \text{如果第 i 个作品在专家 j 处的标准分不需要调整;}\end{cases},\forall i\in T,\forall j\in B' \tag{38}$$

(2) 调整数值变量

引入变量 $S_{i,j}$ 表示参赛作品 i 在专家 j 处调整的数值，若此值大于 0，表明相较原始标准分向上调整，若小于 0，则表明相较原始标准分向下调整。

2. 约束条件

(1) 调整位置需要限制。

为了缩小求解空间，可以根据前面的数据分析中的规律限制调整位置的选择。由前面的规律可知，调整位置大部分可能出现在与标准平均分差距方向一致的两个位置上，且两个值中与平均分差距越大的越容易被调整。因此我们在求解过程中限制每个作品中的这两个位置或其中与平均值差距最大的位置才能被选中调整。

因此 $Q_{i,j}$ 中的所有作品 i 的调整位置只存在两种情况：

情况 1：

调整两个标准分与平均分差距方向一致对应的位置，意味着对于任任一作品任一专家的标准分，如果它与平均分的差值与另外两个与平均分的差值的方向均存在差异，则此位置不需要调整。

$$\begin{cases}Q_{i,j}=0, \text{如果} P_{i,j}-\bar{P}_i \text{与} P_{i,k}-\bar{P}_i \text{方向不一致,}\\\qquad \text{且} P_{i,j}-\bar{P}_i \text{与} P_{i,m}-\bar{P}_m \text{方向不一致}\\Q_{i,j}=1, \text{否则}\end{cases} \tag{39}$$

上式对 $\forall j,k,m\in B',\ j\neq k\neq m$ 都成立。

情况 2：

存在两个标准分与平均分差距一致对应位置，调整与平均分差距最大的标准分对应的位置。表述为条件：如果 $P_{i,j}-\bar{P}_i$ 与 $P_{i,k}-\bar{P}_i$ 方向一致, 或 $P_{i,j}-\bar{P}_i$ 与 $P_{i,m}-\bar{P}_m$ 方向一致，且在 j 和与 j 平均值差异方向相同的另一个值中，$P_{i,j}$ 与

均值差距绝对值更大。

$$Q_{i,j}=\begin{cases}1,\text{如果满足条件 1;}\\0,\text{否则}\end{cases},\forall i\in T,\forall j\in B' \tag{40}$$

(2) 对于任一作品与专家确定的标准分，只有此位置需要调整时才能有调整数值的大小。

在由某个作品和某个专家确定的标准分位置上，如果此位置需要调整，则需要确定此位置调整的数值大小，如果此位置不需要调整，则不需要调整此位置的标准分，表述为下式：

$$\begin{cases}S_{i,j}\neq 0,\text{如果}Q_{i,j}=1\\S_{i,j}=0,\text{如果}Q_{i,j}=0\end{cases},\forall i\in T,\forall j\in B' \tag{41}$$

(3) 若需要调整数值，则调整数值 $S_{i,j}$ 的范围是限定的。

$S_{i,j}$ 变量有调整范围，取决于标准值和标准平均分的差值，具体的，如果某位置标准分需要调整，且标准分低于平均分，则 $S_{i,j}$ 的取值范围为 0 到平均分与标准分差值，若标准分高于平均分，则 $S_{i,j}$ 的取值范围为平均分与标准分差值到 0。表述为：

$$\bar{P}_i=\frac{\sum_{j\in B'}P_{i,j}}{3},\forall i\in T \tag{42}$$

$$\begin{cases}0<S_{i,j}\leq \bar{P}_i-P_{i,j},\text{如果}Q_{i,j}=1\text{且}P_{i,j}<\bar{P}_i\\\bar{P}_i-P_{i,j}\leq S_{i,j}<0,\text{如果}Q_{i,j}=1\text{且}P_{i,j}>\bar{P}_i\end{cases},\forall i\in T,\forall j\in B' \tag{43}$$

(4) 调整极差后保证标准分均值仍然在固定范围内。

经过数据分析可知，最高分段和最低分段的作品基本固定，使用多种方案进行排序，此部分也不会产生较大变化。因此产生极差较大且需要调整的都是中间部分的作品，中间部分的标准分均值变化曲线是斜率几乎固定的曲线，因此可以设定调整某一作品的标准分从而得出新的标准分后的曲线斜率保持在一定范围内。

设调整后的标准分变量为 $P'_{i,j}$，i 与 j 的范围与 $P_{i,j}$ 相同，修改后的作品标准分均值计算方式为：

$$\bar{P}'_i=\frac{\sum_{j\in B'}P'_{i,j}}{3},\forall i\in T \tag{44}$$

原始标准分均值曲线斜率和修改标准分之后每一个作品标准分均值与最大均值曲线斜率的计算方式为：

$$K=\frac{\max_{i\in T}\bar{P}_i-\min_{j\in T}\bar{P}_j}{i-j} \tag{45}$$

$$K'_j = \frac{\max_{i \in T} \bar{P}'_i - \bar{P}'_j}{i - j} \tag{46}$$

设定一个差值界限 θ 表示修改标准分前后斜率的差值上限，则对于修改后的任意作品斜率与原始斜率之间的差值应该满足下式：

$$|K'_j - K| \leq \theta, \forall j \in T \tag{47}$$

3. 目标函数

经过数据分析可知，复议之后作品的极差均值更小，整体标准分方差更小，因此可以使用这两个指标作为目标函数寻找最优的极差调整方案。

(1) 最小化调整标准分之后的极差均值。

设调整标准分之后每个作品的极差为 η'，设集合 T 中的作品个数为 t，则调整后的极差均值计算方式与目标 1 为：

$$\bar{\eta}' = \frac{\sum_{i \in T} \eta'_i}{t} \tag{48}$$

$$\min \bar{\eta}' \tag{49}$$

(2) 最小化调整标准分之后标准分的方差均值。

调整标准分后的方差计算公式与目标 2 为：

$$V_i = \frac{1}{t-1} \times \sum_{j \in B'} (P'_{i,j} - \bar{P'}_i)^2 \tag{50}$$

$$\bar{V} = \frac{\sum_{i \in T} V_i}{t} \tag{51}$$

$$\min \bar{V} \tag{52}$$

综上，**最终目标为：**

$$\min \alpha \times \bar{\eta}' + \beta \times \bar{V} \tag{53}$$

最终的模型为：

$$\min \alpha \times \bar{\eta'} + \beta \times \bar{V}$$

$$\text{s.t.} \begin{cases}
Q_{i,j} = \begin{cases} 1, \text{如果第 i 个作品在专家 j 处的标准分需要调整；} \\ 0, \text{如果第 i 个作品在专家 j 处的标准分不需要调整；} \end{cases} \\
\forall i \in T, \forall j \in B' \\
\begin{cases} i \in T, \eta_i \geq \epsilon \\ i \notin T, \eta_i < \epsilon \end{cases}, \forall i \in A \\
\begin{cases} Q_{i,j} = 0, \text{如果} P_{i,j} - \bar{P_i} \text{与} P_{i,k} - \bar{P_i} \text{方向不一致,} \\ \qquad \text{且} P_{i,j} - \bar{P_i} \text{与} P_{i,m} - \bar{P_m} \text{方向不一致} \\ Q_{i,j} = 1, \text{否则} \end{cases} \\
Q_{i,j} = \begin{cases} 1, \text{如果满足条件 1;} \\ 0, \text{否则} \end{cases}, \forall i \in T, \forall j \in B' \\
\begin{cases} S_{i,j} \neq 0, \text{如果} Q_{i,j} = 1 \\ S_{i,j} = 0, \text{如果} Q_{i,j} = 0 \end{cases}, \forall i \in T, \forall j \in B' \\
\bar{P_i} = \frac{\sum_{j \in B'} P_{i,j}}{3}, \forall i \in T \\
\begin{cases} 0 < S_{i,j} \leq \bar{P_i} - P_{i,j}, \text{如果} Q_{i,j} = 1 \text{且} P_{i,j} < \bar{P_i} \\ \bar{P_i} - P_{i,j} \leq S_{i,j} < 0, \text{如果} Q_{i,j} = 1 \text{且} P_{i,j} > \bar{P_i} \end{cases}, \forall i \in T, \forall j \in B' \\
\bar{P'_i} = \frac{\sum_{j \in B'} P'_{i,j}}{3}, \forall i \in T \\
K = \frac{\max_{i \in T} \bar{P_i} - \min_{j \in T} \bar{P_j}}{i - j} \\
K'_j = \frac{\max_{i \in T} \bar{P'_i} - \bar{P'_j}}{i - j} \\
|K'_j - K| \leq \theta, \forall j \in T \\
\bar{\eta'} = \frac{\sum_{i \in T} \eta'_i}{t} \\
V_i = \frac{1}{t-1} \times \sum_{j \in B'} (P'_{i,j} - \bar{P'}_i)^2 \\
\bar{V} = \frac{\sum_{i \in T} V_i}{t}
\end{cases}$$

5.3.3 模型求解

1. 算法关键思路

本节算法核心是：基于指定目标和题干约束，同时，依据专家调整“大极差”的规律，设计新颖的启发式策略。启发式算法通常可以在较快的时间内获得较优解，但容易陷入局部最优，难以达到全局最优，因此算法的关键在于如何设计一个高效的贪心策略。

本节采用的启发式策略是将复议后标准分的改变量转换为“调整矩阵”，具体来说该变量量是由需要调整分数的专家的位置和相应专家调整分数的额度决定。因此，我们从确定调整分数的位置和调整分数的额度的角度来设计方案。针对调整分数的位置，我们依赖于附件中的极差调整位置规律，即通过比较标准分与标准分均值构建差值矩阵和状态矩阵，依据这两个矩阵的重叠状况，确定具体的调整位置。针对调整分数的额度，依赖于附件中的极差调整额度规律，即调整后的分数更加倾向于标准分的均值，以此确定调整的大概范围；同时，我们利用随机数在该范围内产生大量候选调整方案，依据模型的目标函数，设计了确定调整方案有效性分数，以此在大量的候选方案选取分数最大的作为对应作品调整后的分数。

(1) 有效性分数的设定

依据模型中的优化目标，即减小调整后极差均值、减小调整后分数方差以及保证调整后总分的分布水平，构建评估调整矩阵有效性的分数，具体公式见下式。

$$Score_{i,j} = \beta * (\alpha * \frac{1}{\bar{\eta}'} + (1-\alpha) * \frac{1}{\bar{P}'}) + (1-\beta) * (-(\frac{(\bar{P}'_{i,j} - \bar{P}'_0)}{(i-0)} - \frac{(\bar{P}_n - \bar{P}_0)}{(n-0)})^2) \quad (54)$$

上式中 $Score_{i,j}$ 表示第 i 作品的第 j 个候选方案的有效性分数，$\bar{\eta}'$ 表示方案 j 加入分数矩阵的极差的均值，$\bar{P}'$ 表示方案 j 加入分数矩阵后标准分的均值；$\bar{P}'_{i,j}$ 表示第 i 个作品的第 j 个候选方案的均值，P'_0 表示调整后第 0 个作品的均值；$\bar{P}_n$ 表示原始方案中标准分均值最小的，$\bar{P}_0$ 表示原始方案中标准分均值最大的。

(2) 具体思路为：

表 5　基于分数分布的启发式算法主要思路

基于极差的启发式算法
（1）**差值矩阵和状态矩阵的构建：**依据作品极差的大小获取需要调整的作品矩阵，依据标准分与均值差值的绝对值，构建差值矩阵；依据标准分与均值的大小情况，构建状态矩阵。 （2）**调整矩阵的生成：**依据（1）的差值矩阵和状态矩阵，来确定调整分数的位置，即将差值越大或存在多个大于（或小于）均值的位置，作为候选调整位置。依赖附件中的极差调整额度规律，即调整后的分数更加倾向于标准分的均值，以此确定调整的大概范围, 同时，根据候选调整位置，利用随机数生成器，生成大量候选调整矩阵。 （3）**有效性分数的设定：**依据模型中的优化目标，即保证调整后极差均值、调整后分数方差以及调整后总分的分布水平，构建评估调整矩阵有效性的分数，具体来说，根据候选方案加入后分数极差的均值和分数方差的大小，来衡量其有效性，即极差均值越小，分数方差越小，越有效；同时，其加入后分数分布越靠近两阶段评审总分的分布，越有效。基于此，我们采取线性加权的方法，保证多个目标的同时成立。 （4）**有效调整方案的确定：**对对单个作品，依赖（2）生成大量的候选调整方案，通过（3），计算其放入最终的调整分数矩阵中的有效分数，选取其中分数最大的候选方案作为当前作品的调整方案。 （5）**调整后标准分矩阵的输出：**遍历每一个的作品，依赖（4），确定每一个作品的有效方案，最终输出调整后的标准分矩阵。

2. 算法关键步骤

step1: 准备阶段：

a. “大极差”分类：计算所有作品标准分的极差 C，依据分类规律，即当极差大于 20 的时候便可以认为需要调整分数。依据规律判定每个作品的极差 C，以此构建对应的分数矩阵 T。

b. 差值矩阵和状态矩阵的构建：首先，计算分数矩阵 TT 中每一个作品标准分与对应分数均值差值的绝对值，以此构建差值矩阵 T1T1。然后，通过比较作品标准分与均值的大小关系，构建状态矩阵 T2T2，其中 00 表示对应位置标准分小于均值，11 表示对应位置标准分大于均值。

step2: 候选调整矩阵的生成：首先，遍历每一个作品，依据差值矩阵和状态矩阵确定需要调整分数的位置集合 ID，具体来说，在差值最大或者对应状态矩阵中大于（或小于）均值数量为 2 的位置处进行调整。然后，在 ID_i 处利用随机数生成器产生对应位置的调整额度 d，对应分数通过增加（或减少）d，来趋近于均值。最终，通过设置产生随机数的多少，产生当前作品的大量候选调整方案。

step3: 有效性分数的设定：依据调整后极差均值的减小、分数方差的减小以及调整后分数分布与两阶段评审总分的分布之间的接近程度，构建有效性分数 $Score$。这里我们使用线性加权方法将这些目标结合起来，以计算每个候选矩阵的有效性分数 $Score_i$。

step4: 候选矩阵的评估：对于每个作品，计算生成的大量候选调整方案的有效性分数，然后选择具有最高有效性分数的候选方案作为当前作品的调整方案。

step5: 新分数矩阵的输出：遍历每个作品，根据第四步确定的有效调整方案，生成调整后的标准分矩阵 T'，最终输出整个矩阵。

5.3.4 求解结果与分析

1. 第一评审阶段程序化

针对第一评审阶段的程序化，首先，我们通过分析现有的数据，**依赖作品的总分以及极差**，过滤“低分段”的作品。其次，针对剩余的作品，我们**选择其中极差较大的作品**，利用我们提出的极差模型，来调整对应作品的分数，具体步骤如下。

(1) 关键步骤

step1: 低评分作品的过滤：依据第一评审阶段打分结果，计算每一个作品对应的总分以及极差，来**滤除低分段**的作品。具体来说，当作品分数均值大于 54，作品便被保留。

step2: 大极差作品的处理：计算第一阶段所有作品标准分的极差，将极差大于 20 的作品保留，然后，**利用极差模型处理**。

step3: 调用极差模型：针对过滤后的大极差作品集，首先计算对应的差错矩阵和状态矩阵，然后直接使用前文提到的极差模型，去调整每一个作品的分数。

step4: 调整后结果的输出：最终，极差模型输出调整后作品的分数集合。

(2) 求解结果

针对数据 2.1，我们通过阈值去过滤第一阶段的作品集合，获得极差大作品。我们对比一阶段原始分数的总分数和一阶段对应作品的复议分数，来展示我们方法有效性。具体如下图所示。

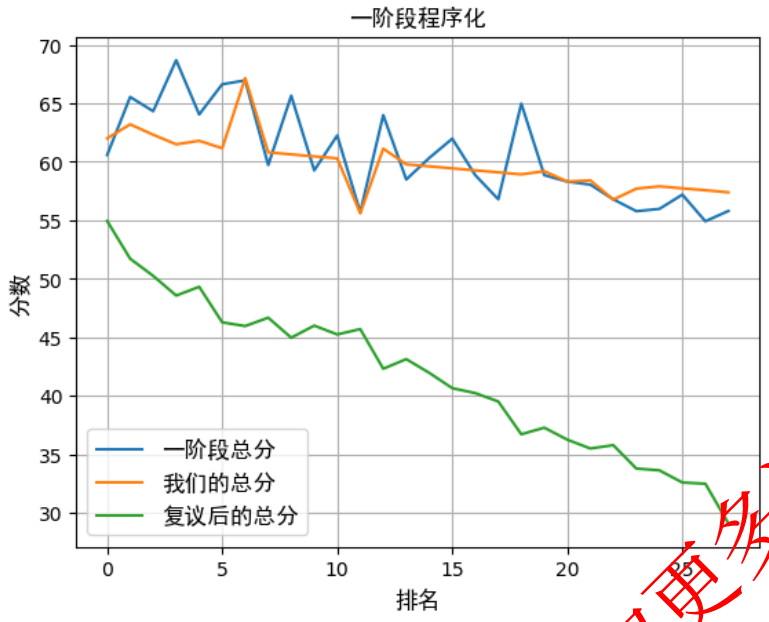

图 23　数据 2.1 的第一阶段调整后分数分布

从图中，可以看出我们的方法很大程度上减少了原有作品中大极差作品的数量。

2. 求解结果

针对问题三，依赖提供的数据，通过分析“大极差”作品处理的规律，发现在处理过程中的分类和调整规律，依据此，我们构建了极差模型，在不进行人工干预的情况下，自动化的调整“大极差”作品，同时能够保证很好的可信度。

针对数据 2.1 第二阶段，计算作品的极差，依据分类阈值，提取出需要调整的作品集合，该作品集合依旧按照复议后的总分进行排序的。然后，使用我们提出的极差模型，在不需要人工协助的情况下，所取得的结果如下图所示。

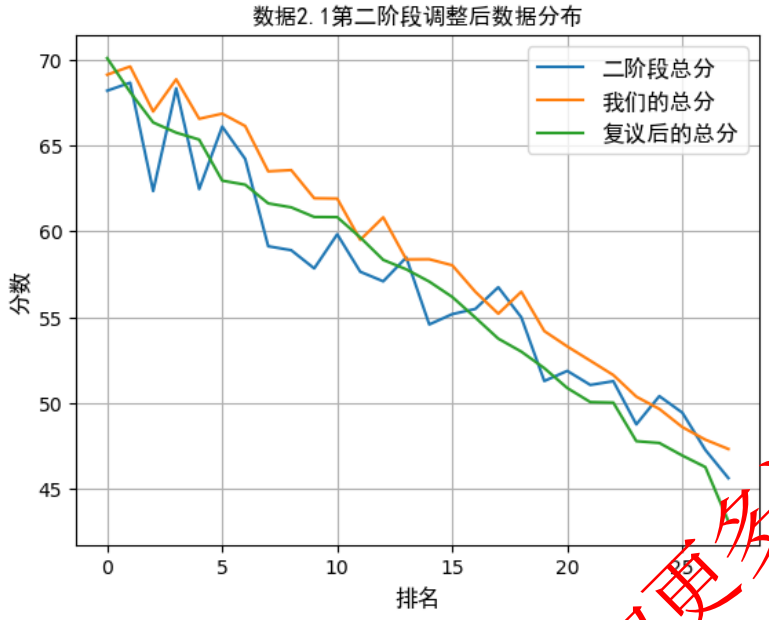

图 24　数据 2.1 的第二阶段调整后分数分布

从图中，在针对第二阶段的标准分进行调整后，原有的大极差作品的数量现状的减小；同时我们调整后后作品的分布更加贴合复议后的分布。

在更大的数据集中，即数据 2.2，针对数据 2.2 的第一阶段和第二阶段，我们同样采用我们的方法，所取得的结果如下图所示。

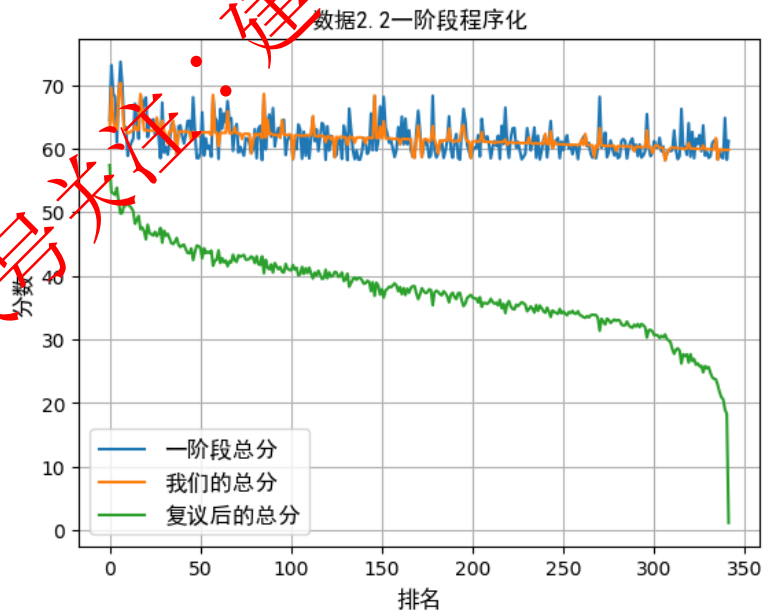

图 25　数据 2.2 的第一阶段调整后分数分布

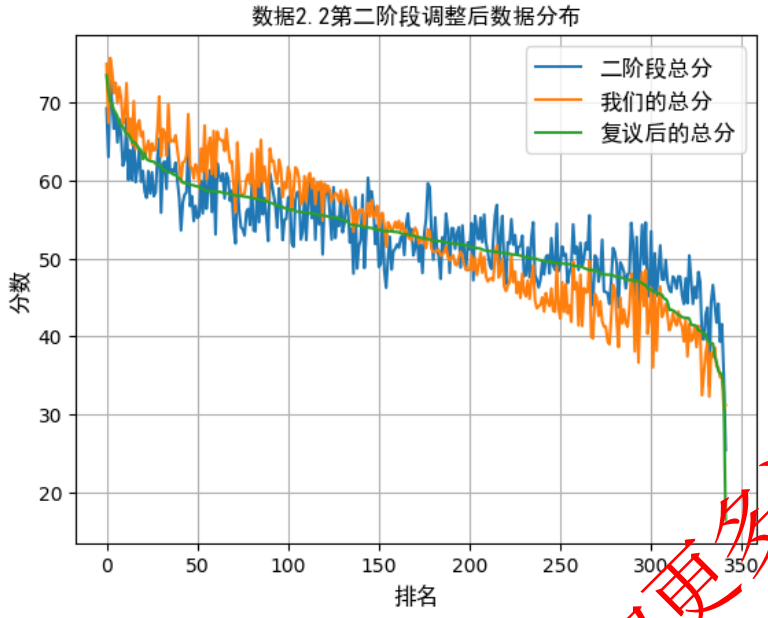

图 26　数据 2.2 的第二阶段调整后分数分布

从图中的结果可以看出，我们的极差模型，能够有效的降低原始分数的极差。同时，也可以发现我们的极差模型，不会依赖原始分数的分布状态，相较于数据 2.1，数据 2.2 的分数数据根据复杂，且相较于复议后的分数，差距很大，这就导致我们最终依据数据 2.2 构建的极差模型，调整得到的分数与复议后分数仍有一定差距。

3. 结果分析

下表中的不一致性，是我们利用具体方案与最终方案的排名做差的绝对值求和得到的。

表 6　第一阶段调整后排名序列的对比

数据 2.1 第一阶段		数据 2.2 第一阶段	
方案	不一致性	方案	不一致性
有复议	0	有复议	0
无复议	112	无复议	30020
我们的方案	66	我们的方案	10958

表 7　第二阶段调整后排名序列的对比

数据 2.1 第二阶段		数据 2.2 第二阶段	
方案	不一致性	方案	不一致性
有复议	0	有复议	0
无复议	44	无复议	14534
我们的方案	14	我们的方案	6308

在上述的表格中，我们发现如下规律，仅依赖一阶段评审，无论评审的规模有多大，其最终取得的分数相较于两阶段更差。同时我们发现，将我们的极差模型无论放入那种方案中，**其至少能够将原方案的不一致性降到原来一半**。

5.4 问题四

问题四要求针对创新类竞赛给出一个完整的评审模型，并针对所给的数据研究如何求解。也可以对现行的评审方案给出改进的具体建议 (包括未来还要收集哪些数据)

结合前面三问的建模和求解，我们可以得出自己设计的一个完整的评审方案，且使用附件数据进行测评后，发现与目前实行的两阶段复议排序结果接近，表明我们设计的评审方案的有效性。

5.4.1 评审模型

这个评审模型分为两个阶段，第一阶段将所有作品交叉分发给 5 位专家进行评审，得到专家评分后按照设定规则筛选出部分作品，剩余作品进入第二阶段评审，将这些作品交叉分发给 3 位专家重新进行评分，得到专家评分后按照设定规则计算作品总成绩并进行排序，得到最终的获奖结果。第一阶段和第二阶段均涉及到了作品分发的问题，且第一阶段和第二阶段设计了不同的规则，下面将从**作品分发、第一阶段规则和第二阶段规则**三个方面进行阐述，两阶段评审结束后即可得出最终的作品排序和获奖名单。

1. 分发作品

竞赛主办方在收集完所有竞赛作品之后，需要将其分发给固定数量的专家进行评分。分发的结果会在一定程度上影响专家评审以及最后的获奖名次。比如若专家工作量不同，可能会影响专家评审效率以及评审主观性，若分配给专家的作品水平差距很大，也会导致最后评分分值的差异。

通过问题一中的数据分析可知，若要增加不同专家所给成绩之间的可比性，

得出比较好的分发方案，应该满足下面**三个约束**：

(1) 每位专家评审的作品总数需要基本保持一致。

(2) 每位专家与其他专家有作品交集的专家个数需要基本保持一致。

(3) 每位专家与其他专家的重复作品总数应该尽可能大且保持一致。

因此我们使用问题一中的优化求解方案将指定个数的参赛作品分配给固定个数的评审专家。求解的主要步骤可以参考问题一的求解部分，其中主要的步骤包含**生成初始解、遵守约束条件生成候选解和遍历寻找最接近目标的最优解**三个部分。在下面求解最终排序的过程中，假设两个阶段的作品分发都是按照我们设计的求解方案进行分发的。

2. 第一阶段规则

第一阶段评审目的在于**筛选出一些明显质量较差的作品**，这些作品的评分表现为所有专家评分都比较低，即分数均值较低且没有较大的极差。

第一阶段评审大致分为以下几步：

(1) 作品分配

将所有作品按照交叉分配的优化方案分配给 5 位专家，得到所有作品的 5 个原始评分。这里使用符号 $X_{i,j}$ 表示参赛作品 i 在专家 j 处的原始评分，这里 $i \in A, j \in B$，A 表示所有参赛作品集合，B 表示专家 1 到专家 5 位置处的代称，不代表实际专家，因此此变量只能表示作品 i 在第几个专家位置处的得分。

(2) 计算标准分

根据问题二的分析可知，原标准分计算公式需要假设每位专家作品集的作品学术分布需要保持一致，这在很多大规模比赛中是没有办法做到的，且原标准分也没有解决极差大的问题，不同专家对同一作品可能产生不同看法。考虑到需要关注专家评分主观性的问题，我们融合了专家可信度权重设计了新的标准分计算公式，具体为：

$$x_k = 0.8 \times (50 + 10 \times \frac{a_k - \bar{a}}{s}) + 0.2 \times W_j \times a_k \tag{55}$$

其中 a_k 表示某一标准分，W_j 为专家可信度权重，专家可信度权重值由专家原始评分经过一定步骤计算得出，主要流程为：计算专家的分值偏差和排序偏差，从而得出专家与专家之间的相似矩阵，根据矩阵进行层次聚类得出固定类别后不同类别内的专家集合，计算不同类别内专家的平均可信度，将此值作为同一类别内专家的权重。此过程可以参考问题二的详细建模，经过聚类分析可以得出每一个专家的权重 W_j。

使用上述的标准分计算公式计算每个专家原始分对应的标准分，用此标准分作为专家的评分。

(3) 筛选进入第二阶段作品

得到 5 位专家标准分后，计算每个作品 5 位专家的标准分平均值记作 $\bar{P}_i$，所有作品标准分均值的平均值记作 $\bar{P}$，所有作品的极差记为 η_i。

$$\bar{P}_i = \sum_{j \in B} X_{i,j}, \forall i \in A \tag{56}$$

$$\bar{P} = \sum_{i \in A} \bar{P}_i \tag{57}$$

筛选出所有**标准分均值小于** $\bar{P}$ **且极差小于 20** 的作品淘汰，剩余作品进入第二阶段继续进行评审。这里制定这两个指标的原因是当标准分均值较小意味着作品总评分较低，极差小于 20 意味着专家意见没有较大偏差，普遍认为此作品质量不佳，因此筛选出这些作品淘汰。

3. 第二阶段规则

第一阶段已经筛选出一些明显质量不佳且专家之间没有歧义的作品，第二阶段的目标是**得到更合理更符合作品水平的最终排名和获奖名次。**

需要重新安排作品进行评审，固定第二阶段作品需要 3 位专家进行评审，这里仍然使用我们设计的交叉分发优化方案对剩余作品进行分配。最后得出的作品及对应专家顺序位置处的原始分用符号 $Z_{i,j}$ 来表示，其中 $i \in A', j \in B', B = 1, 2, 3$

使用新的标准分计算模型计算每个专家原始分对应的标准分，设修改后的标准分变量为 $Z'_{i,j}$，计算此时每一个作品的极差 η_i：

$$\eta_i = \max_{j \in B'} Z'_{i,j} - \min_{k \in B'} Z'_{i,k}, \forall i \in A', j \neq k \tag{58}$$

若极差大于 20，则需要使用我们设计的极差模型调整极差较大的作品的标准分，从而使得修改后的极差均值和标准分方差尽可能小。

基本建模与求解思路和问题三基本一致，具体的，设调整位置变量是 $Q_{i,j}$，调整数据大小变量为 $S_{i,j}$，则这两个变量在求解过程中需要满足的**约束条件**为：

(1) 调整位置需要限制；

(2) 对于任一作品与专家确定的标准分，只有此位置需要调整时才能有调整数值的大小；

(3) 若需要调整数值，则调整数值 $S_{i,j}$ 的范围是限定的；

(4) 调整标准分后保证标准分均值仍然在固定范围内

满足上述四个约束寻找候选解，遍历找到最符合目标函数的解，目标函数设计为使修改后的极差均值和标准分方差均值最小。设修改后的标准分为 $Z''_{i,j}$，修改后的极差为 η'_i，则极差均值和标准分方差均值的计算方式下式，其中 t 表示作品个数：

$$\bar{\eta'} = \sum_{i \in A'} \eta'_i \tag{59}$$

$$V_i = \frac{1}{t-1} \times \sum_{j \in B'} (Z''_{i,j} - \bar{Z''}_i)^2 \tag{60}$$

$$\bar{V} = \frac{\sum_{i \in A'} V_i}{t} \tag{61}$$

则**最终的优化目标**为：

$$\min \alpha \times \bar{\eta'} + \beta \times \bar{V} \tag{62}$$

4. 计算总分并得到最终排序，确定获奖队伍及名次

在上一部分经过优化可以得到最佳的调整极差的方案，使用调整后的标准分替代之前的标准分，若未调整则使用原标准分，计算修改后的标准分加和与第一阶段所有标准分的均值作为此作品的最终成绩，计算公式为：

$$M_i = \sum_{i \in A^{prime}} Z''_{i,j} + \bar{P}_i \tag{63}$$

按照最终成绩对剩余作品进行排序，可以得到每个作品最终的名次。至此整个评审过程全部完成，按照上述模型即可得到最终每个作品的名次以及获奖情况。

5.4.2 求解

根据我们提出的评审模型，我们将其应用于数据 2.1 和数据 2.2，通过将我们的模型方案与方案 4 进行对比，以此来验证我们方案的有效性。具体实施步骤，如下所示：

step1: 计算新标准分数: 首先，根据每个数据中专家对于作品的分数集合，同时利用专家置信度模型，对专家进行分类，同时获得对应专家的打分置信度分数；然后，利用我们在问题二中提出的置信度分数计算公式，计算出新的标准分数，最终依据该标准分计算最终作品的总分。

step2: 大极差作品的调整: 根据作品的新标准分数，计算对应作品的极差以及均值，通过问题三中的极差规律，筛选出大极差的作品集合。然后，调用我们提出的极差模型，对大极差作品集进行调整，最终获得调整后的分数分数。

step3: 对比分析: 通过 step2 之后，我们便可获得一个新的分数矩阵。此时，我们便有三组总分，具体包括我们方案生成的总分，方案 4（无复议）的总分和复议后的总分，我们利用这三组数据，来进行具体的分析，具体分析过程见结果部分。

5.4.3 结果及分析

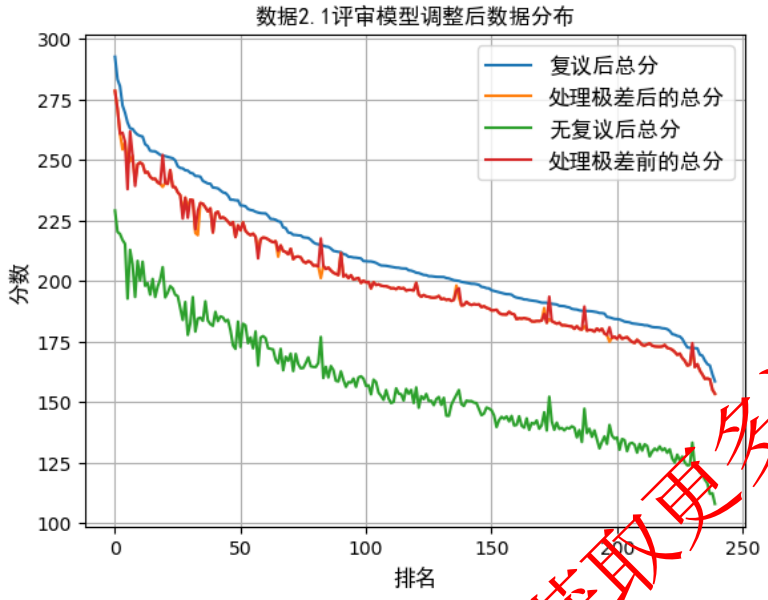

图 27　评审模型调整数据 2.1 的分数对比图

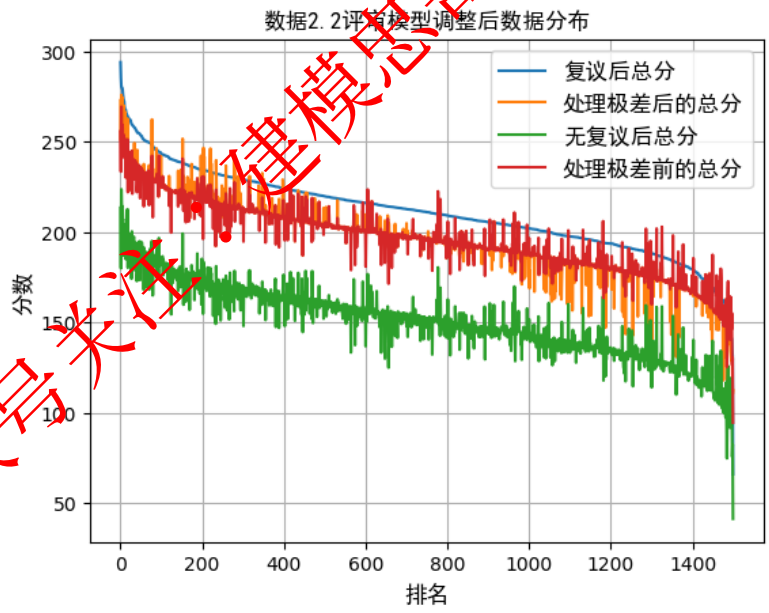

图 28　评审模型调整数据 2.2 的分数对比图

上述两幅分数对比图中，横轴表示当前作品的最终排序，纵轴表示分数。当数据规模较小时，我们的评审方案生成的分数排名，十分靠近于最优排序曲线；同时可以发现，当我们不使用极差模型时，仅使用我们提出的标准分公式，就可以保证分数排名十分靠近最优曲线，同时作品集合中的大极差作品，也在我们的

自动化评审过程中得到了显著下降。当规模增大时，同样能够在一定程度上降低大极差作品的数量，以及更加靠近最优的排序曲线。具体的大极差作品的减少量以及对应分数差异值，见下表。

表 8　数据 2.1 评审方案结果表

数据 2.1 评审结果		
方案	不一致性	大极差作品的数量
有复议总分	0	0
无复议总分	1720	28
新标准分总分	834	19
我们的方案总分	676	5

表 9　数据 2.2 评审方案结果表

数据 2.2 评审结果		
方案	不一致性	大极差作品的数量
有复议总分	0	0
无复议总分	113726	112
新标准分总分	82786	250
我们的方案总分	76508	25

通过以上结果的可视化，可以得到：我们设计的评审模型在复议前可以得到 19 个大极差作品，而两阶段评审方案为 28 个大极差作品，因此相较于两阶段方案能够更好的筛选大极差作品，能够在复议前降低大极差作品集合中的数量。

6. 模型的评价与改进

6.1 算法的有效性和复杂度分析

本文采用启发式算法寻找满足约束条件和优化目标的解，算法中融入了贪心算法的思想，根据具体问题设计不同的贪心目标，以此求解对应问题的可行解。

在问题一中，我们设计了基于规则的启发式算法，具体来说就是辅助矩阵和滑动窗口，辅助矩阵可以更方便得出每个作业的可选专家数量，而滑动窗口则可以根据每个专家的可选专家数量，快速修改作品索引对应位置的方案，这两个规则可以快速生成高匹配率的分配方案，代码运行速度快，结果反馈良好。

在问题二中，我们通过分析了现有四个评审方案，发现它们都没有关注专家在对应专家集合中置信度。因此，我们从专家的角度，结合专家自身的评分习惯与其在专家集合中的置信度，构建专家可信度模型。同时，在分析现有方案的过程中，发现基于标准分的两阶段评审方案，有着良好的特性，基于此，我们采用线性加权的方法，提出新的标准分计算公式。

在问题三中，我们通过分析大极差作品的极差规律，同时结合各个方案中分数曲线的分布特点构建优化模型。具体来说，针对极差规律，我们分析了不同的调整方案，即调整位置与调整额度的规律，这些规律指导我们如何构建决策变量；就目标函数方面，我们通过分析发现，调整后的作品分数，极差更小且标准分的方差更小，另外，参考其它方案的分数曲线，以此构建对应优化模型。

问题一的时间复杂度 **O(M+Nlog(N))**

问题二的时间复杂度 **O(N*log(N))**

问题三的时间复杂度 **O(N*log(N))**

6.2 模型灵敏度分析

下图中展示了问题二和问题三中两个参数随着数值的变化产生的与最优方案差异度的变化折线图。可以分两点进行分析：

1. 问题二参数变化影响

问题二中新标准分的计算方式为原标准分与专家权重原始分数乘积之和，其中有一个参数 α 调整原始标准分在新标准分公式中的占比。图中蓝色曲线表示随着参数数值变化，得出的排序结果与最优排序结果之间的差异度变化，绿色虚线表示原始方案 4 即不进行复议与最优排名之间的差异度。从图中可以看出当参数越来越大时，与最优排名差异度越小，且当参数值大于 0.55 之后就超过了原始方案 4 的表现。

2. 问题三参数变化影响

问题三需要设计优先级指标函数，其中设计到了参数，表示针对不同指标的侧重点，比如对极差均值和标准分方差均值的侧重点。图中红色曲线表示随着参数变化，得到的最终排名与最优排名之间的差异度变化，黄色虚线表示原始方案 4 得出的排名与最优排名之间的差异度。从图中可以看出，随着参数的变大，得到的排名与最优排名之间的差距越来越大，当参数值为 0.3 时，差异度最小且表现超过原始方案 4。

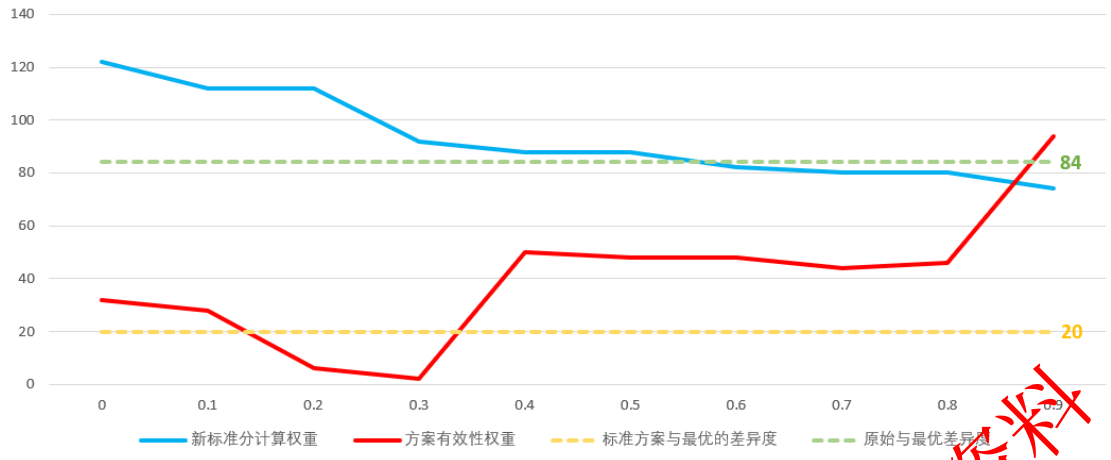

图 29　灵敏度分析

6.3 优点分析

1. 模型的角度

(1) 混合整数规划模型约束和目标非常清晰。

在模型建立的过程中，我们能够通过对给定数据的分析，提取出符合建模约束和目标的规律，从而使得建模过程中的约束条件和目标函数是非常清晰且合理的。

(2) 给定的模型在实际求解过程中给了很大的指导方向建议。

通过建立较为清晰合理的模型，使得后续设计贪心算法，缩小解空间进行优化求解都有了更好的方向。

2. 求解算法的角度

(1) 求解时间短。

在求解过程中，我们通过设计贪心算法，融合一系列缩小解空间范围的优化措施，比如滑动窗口和极差分析等策略，极大的减小了找到较优解的时间，降低了求解的时间复杂度。

(2) 基本符合模型。

求解过程中的基本框架和逻辑是与模型基本一致的，且求解就是以模型建立为导向，提取出更多的规律，设计优化算法进行求解。

6.4 缺点分析

1. 求解算法的角度

由于本题是一个优化问题，我们设计的贪心算法只能找到较为接近自己设计的目标函数的解，并不一定代表能够求解到最优解。

2. 我们设计的极差模型，很大程度依赖于原有方案本身的效果。

6.5 模型改进

1. 极差模型中计算标准斜率点的选取改进。

在目前实现的方法中，基准斜率的计算是使用最高点与最低点之间的斜率，但这样选择点是不太合理的，实际点的选取方法可以改进。选取多方案中位序变化不大或者斜率变化不大的点，从而更好的得到斜率的基础值。

参考文献

[1] 徐林生, 王执铨, 戴跃伟, 评审专家可信度评价模型及应用, 南京理工大学学报, 34(1):30-34, 2010.

[2] 郭东威, 丁根宏, 群组决策主观评分型竞赛名次的优化模型, 工程数学学报, 39(3):379-388, 2022.

[3] 朱彤, 纪新如, 黄晶晶, 等, 互联网 + 时代学科竞赛评审综合评价, 南通职业大学学报, 2019.

[4] 陈媛, 樊治平, 科技类评审中项目选择的两阶段综合集成方法, 中国管理科学, 18(2):127-133, 2010.

[5] 韩响玲, 陈志刚, 董婷婷, 竞赛评分结果的偏移量修正法, 湘潭工学院学报: 社会科学版, 5(1):77-80, 2003.

[6] 丁元耀, 贾让成, 均值一级类型组合投资优化选择模型, 预测, 18(4):64-65, 1999.

[7] 胡日东, 组合证券投资优化模型的比较研究, 运筹与管理, 10(1):98-103, 2001.

附录 A 我的 python 源程序

```
# 问题一
import numpy as np
import copy
def generate_sparse_vector(size, num_ones):
    if num_ones > size:
        raise ValueError("Number of ones cannot exceed vector
            size.")
    # 生成一个全为0的向量
    sparse_vector = np.zeros(size, dtype=float)
    # 随机选择要设置为1的位置
    ones_indices = np.random.choice(size, num_ones, replace=False)
    # 在选定的位置上设置为1
    sparse_vector[ones_indices] = 1
    return sparse_vector
con_array = np.zeros(3000,float)
CC = 71#每个专家可以审查的数量
while sum(5-con_array) != 0:#当仍有位置可以填充1时
    CC = CC+50
    p1 = np.zeros([125,3000],float)
    con_array = np.zeros(3000,float)
    zj_jiont = np.zeros([125,125],float)#与其他专家相较的个数
    zj_score = np.zeros(125,float)#与其相较的专家，相较作品的总数。
    zj_num = np.zeros(125,float)#与其相较的专家具体的数目。
    alpha = 0.5
    M = 10000 #解空间
    N = 40 #测试空间
    temp = np.zeros([125,3000],float)
    con_array_temp = np.zeros(3000,float)
    for j in range(125):
        T = np.zeros([M,3000],float)#创造10000个
        TT = np.zeros([N,3000],float)#创造40个待选
        score1 = np.zeros(N,float)
        score2 = np.zeros(N,float)
        temp_score = np.zeros(M,float)
        if j <= 50:#前t次
            for k in range(M):
                con_array_temp = copy.deepcopy(con_array)
```

```
        # 指定向量大小、1的数量和随机种子
            vector_size = 3000
            num_ones = CC # 指定1的数量
            # 生成稀疏向量
            sparse_vector = generate_sparse_vector(vector_size,
                num_ones)
            if j == 0:
                best = sparse_vector
                break
            # 定义两个数组向量
            A = 5-con_array_temp#剩余需插入1的向量
            B = sparse_vector
            # 计算余弦相似度 随机向量和剩余稀疏向量是否相近
            cosine_similarity = np.dot(A, B) /
                (np.linalg.norm(A) * np.linalg.norm(B))
            temp_score[k] = cosine_similarity
            T[k] = sparse_vector #生成10000个稀疏向量
            #print(np.where(sparse_vector!=0))
    else:
        #50-125次的时候主要是往里填
        con_array_temp = copy.deepcopy(con_array)
        g_index = np.argsort(con_array_temp)
        print(g_index[-1],con_array_temp[g_index[-1]])
        start = 0#视窗块
        end = CC
        for gg in range(30): # 流动窗：每个方案都选择一部分进行填充
            slice_temp = g_index[start:end] #取出排序过后的顺序索引
            for kk in slice_temp:
                TT[gg,kk] = 1 #将这些位置上的改为1，方便填充
            start = start +100
            end = end +100
            if end > 3000:
                break
        T[:40,:] = TT #前40行的所有列
    #检查爆不爆
    if j !=0:
        if j > 50:
            cos = np.arange(gg+1)
        else:
```

```
        cos = np.argsort(temp_score)[-N:]#相似系数最大的40个
    for ii,cos_index in enumerate(cos):
        temp = copy.deepcopy(p1)
        print("123",np.where((np.sum(p1,axis = 0))>5)[0])
        temp[j,:] = T[cos_index]
        print(np.where((np.sum(p1,axis = 0))>5)[0])
        if len(np.where((np.sum(p1,axis = 0) +
            T[cos_index])>5)[0]) > 0:#哪几个索引的作品不满足约束
            tt = np.where((np.sum(p1,axis = 0) +
                T[cos_index])>5)[0]
            temp = copy.deepcopy(p1)
            for gg in tt:
                T[cos_index][gg] = 0#将对应索引的元素改为0
                print(np.where((np.sum(p1,axis = 0) +
                    T[cos_index])>5)[0])
            temp[j,:] = T[cos_index]
            print(len(np.where((np.sum(p1,axis = 0) +
                T[cos_index])>5)[0]))
            for i in range(j+1):
                set1 = set(np.where(temp[j,:]!=0)[0])
                set2 = set(np.where(temp[i,:]!=0)[0])
                inter = set1.intersection(set2)
                zj_jiont[j,i] = len(inter)
                zj_jiont[i,j] = len(inter)
            for i in range(j+1):
                zj_score[i] = sum(zj_jiont[i,:])
                zj_num[i] = len(np.where(zj_jiont[i,:] !=
                    0)[0])
            #保证相较的作品量大且均衡
            score1[ii] =
                zj_score.mean()/(zj_score.std()+0.001)
            score2[ii] = 1/(zj_num.std()+0.001)
        else:#均满足
            for i in range(j+1):
                set1 = set(np.where(temp[j,:]!=0)[0])
                set2 = set(np.where(temp[i,:]!=0)[0])
                inter = set1.intersection(set2)
                zj_jiont[j,i] = len(inter)
                zj_jiont[i,j] = len(inter)
```

```
            for i in range(j+1):
                zj_score[i] = sum(zj_jiont[i,:])
                zj_num[i] = len(np.where(zj_jiont[i,:] !=
                    0)[0])
            #保证相较的作品量大且均衡
            score1[ii] =
                zj_score.mean()/(zj_score.std()+0.001)
            score2[ii] = 1/(zj_num.std()+0.001)

    if j !=0:
        for ii in range(N):
            score1[ii] = np.exp(score1[ii]/(score1.max()))#归一化
            score2[ii] = np.exp(score2[ii]/(score2.max()))
        Score = alpha*score1+(1-alpha)*score2
        index =np.argsort(Score)[-1]
        print(index)
        print(Score[index])
        print(j)
        best = T[index]#每个j迭代中最好的候选方案
    p1[j,:] = best
    con_array = np.sum(p1,axis = 0)
    if len(np.where(con_array>5)[0])>0:
        tt = np.where(con_array>5)[0]
        best[tt] = 0
        p1[j,:] = best#如果爆了，全改为0
        con_array = np.sum(p1,axis = 0)
    print(np.where(con_array>5)[0])
    print("结果",sum(5-con_array))
    for i in range(j+1):
        set1 = set(np.where(p1[j,:]!=0)[0])
        set2 = set(np.where(p1[i,:]!=0)[0])
        inter = set1.intersection(set2)
        zj_jiont[j,i] = len(inter)
        zj_jiont[i,j] = len(inter)
    for i in range(j+1):
        zj_score[i] = sum(zj_jiont[i,:])
        zj_num[i] = len(np.where(zj_jiont[i,:] != 0)[0])
    if j == 124:
        print(123)
```

```
file_name = "data2.csv"
delimiter = ","
# 使用np.savetxt保存二维数组为CSV文件
np.savetxt(file_name, p1, delimiter=delimiter)
```

```
#问题二预处理
import pandas as pd
import numpy as np
P2 = pd.read_csv("C:/Users/1/Desktop/B.csv",header=0)
PP2 =
   pd.read_csv("C:/Users/1/Desktop/guosai/my_array1.csv",header=0)
P2 = P2.iloc[2:,:22]
PP2 = PP2.iloc[:,0]
tt = [5,8,11,14,17]
zj = list()
for i in range(len(P2)):
   for j in tt:
      zj.append(P2.iloc[i,j])
zj=np.array(zj)
zj = np.unique(zj)
O_score = np.zeros([len(P2),len(zj)])
for i in range(len(P2)):
   for index,j in enumerate(zj):
      for k in tt:
         if P2.iloc[i,k] == j:
            O_score[i,index] = P2.iloc[i,k+1]
O_score1 = np.zeros([len(P2),len(zj)+1])
O_score1[:,1:] = O_score[:]
O_score1[:,0] = P2.iloc[:,1]

file_name = "C:/Users/1/Desktop/p2.csv"
delimiter = ","
# 使用np.savetxt保存二维数组为CSV文件
np.savetxt(file_name, O_score1, delimiter=delimiter)
zj.astype(object)
file_name = "C:/Users/1/Desktop/p2_name.csv"
delimiter = ","
import numpy as np
# 创建包含数据的NumPy数组
```

```
data = np.array(['P003', 'P037', 'P117', 'P125', 'P154', 'P183',
    'P223', 'P239',
              'P253', 'P255', 'P260', 'P262', 'P271', 'P275',
                 'P280', 'P288',
              'P290', 'P301', 'P308', 'P322', 'P326', 'P340',
                 'P345', 'P370',
              'P372', 'P376', 'P378', 'P381', 'P388', 'P394',
                 'P407', 'P431',
              'P434', 'P440', 'P460', 'P493', 'P530', 'P565',
                 'P593', 'P611',
              'P625', 'P704'], dtype=object)
# 使用numpy.savetxt()保存为CSV文件
np.savetxt(file_name, data, delimiter=',', fmt='%s')
```

```
import pandas as pd
import numpy as np
from scipy.cluster.hierarchy import linkage, dendrogram, cut_tree
import matplotlib.pyplot as plt
from collections import Counter
from scipy.spatial.distance import squareform

# 读取CSV文件，假设行名在文件的第一列
df = pd.read_csv("deal_data2.csv", index_col=0)

# 初始化一个字典来存储每个队伍的总和
team_sums = {}

# 遍历每一行并计算总和
for index, row in df.iterrows():
    team_sum = row.sum() # 求取该行（队伍）的数据总和
    team_sums[index] = team_sum # 将结果存储在字典中

# 初始化一个字典，用于存储每一列中非零元素所在行的名称及其对应值
expert_dict = {}

# 遍历每一列
for col in df.columns:
    # 使用`loc`方法找到非零元素的行索引和值
    non_zero_data = df.loc[df[col] != 0.00, col]
```

```
    # 将结果（行名称及其对应值）转化为字典
    non_zero_dict = non_zero_data.to_dict()

    # 将这个列的结果添加到总字典
    expert_dict[col] = non_zero_dict

# print(expert_dict['专家1'])
# 初始化一个字典，用于存储每个专家列的差值绝对值之和
expert_difference_sums = {}

# 遍历 expert_dict 的所有键（即专家列）和对应的值（即包含队伍和分数的字典）
for expert, teams_scores in expert_dict.items():
    # 初始化变量用于存储当前专家列的差值绝对值之和
    diff_sum = 0

    # 遍历当前专家列的所有队伍和分数
    for team, score in teams_scores.items():
        # 计算差值的绝对值
        diff = abs(score - team_sums.get(team,
            0))/team_sums.get(team, 0)

        # 累加差值绝对值
        diff_sum += diff

    # 将当前专家列的差值绝对值之和存储到字典
    expert_difference_sums[expert] = diff_sum/len(teams_scores)

# print("Sum of absolute differences for each expert:",
    expert_difference_sums)

# 初始化一个新的字典用于存储排名信息
ranked_expert_dict = {}

# 遍历每个专家及其对应的队伍和评分
for expert, teams_scores in expert_dict.items():
    # 对队伍根据分数进行排序，并获取排名
    sorted_teams = sorted(teams_scores.items(), key=lambda x:
        x[1], reverse=True)
```

```
    ranked_teams = {team: rank + 1 for rank, (team, _) in
        enumerate(sorted_teams)}

    # 将排名信息存入新的字典
    ranked_expert_dict[expert] = ranked_teams

# 输出结果
# print("Ranked expert dict:", ranked_expert_dict['专家1'])

# 初始化一个新的字典用于存储按team_sums值排序后的专家内部队伍的排名
ranked_by_team_sums_expert_dict = {}

# 遍历每个专家及其对应的队伍和评分
for expert, teams_scores in expert_dict.items():
    # 使用team_sums的值替换专家内部队伍的评分
    replaced_by_team_sums = {team: team_sums.get(team, 0) for
        team in teams_scores.keys()}

    # 对队伍根据team_sums的值进行排序，并获取排名
    sorted_teams = sorted(replaced_by_team_sums.items(),
        key=lambda x: x[1], reverse=True)
    ranked_teams = {team: rank + 1 for rank, (team, _) in
        enumerate(sorted_teams)}

    # 将排序后的队伍排名存入新的字典
    ranked_by_team_sums_expert_dict[expert] = ranked_teams

# 输出结果
# print("Ranked by team_sums expert dict:",
    ranked_by_team_sums_expert_dict['专家1'])
# 初始化一个字典用于存储每个专家的平均差值
average_difference_per_expert = {}

# 遍历ranked_expert_dict中的每一个专家
for expert, ranked_teams in ranked_expert_dict.items():
    # 获取此专家内包含的队伍总数
    total_teams = len(ranked_teams)

    # 初始化变量用于累加差值
```

```
    total_difference = 0.0

    # 遍历每个队伍并计算差值
    for team, rank in ranked_teams.items():
        # 获取在ranked_by_team_sums_expert_dict中相应的排名
        other_rank = ranked_by_team_sums_expert_dict.get(expert,
            {}).get(team, 0)

        # 计算差值的绝对值并累加
        total_difference += abs(rank - other_rank)/(total_teams-1)
    average_difference = total_difference/total_teams
    # 存储结果
    average_difference_per_expert[expert] = average_difference

# 输出结果
# print("Average difference per expert:",
    average_difference_per_expert)

for expert, difference in expert_difference_sums.items():
    expert_difference_sums[expert] = 1 - difference

# print('p0值为',expert_difference_sums)

# 更新average_difference_per_expert中的每一个值
for expert, average_difference in
    average_difference_per_expert.items():
    # 用1 - 该值替代该值
    average_difference_per_expert[expert] = 1 - average_difference

# 输出更新后的结果
# print("p1值为", average_difference_per_expert)

# 获取专家列表（假设两个字典具有相同的键）
all_experts = list(expert_difference_sums.keys())

# 初始化矩阵
matrix = np.zeros((len(all_experts), 2))

# 填充矩阵
```

```
for i, expert in enumerate(all_experts):
    matrix[i, 0] = expert_difference_sums[expert]
    matrix[i, 1] = average_difference_per_expert[expert]

# 获取专家数量
num_experts = matrix.shape[0]

# 初始化相似矩阵，尺寸为 (num_experts x num_experts)
similarity_matrix = np.zeros((num_experts, num_experts))

# 计算 M 的值，这里我们使用所有 P_{0,u} * P_{0,v} + P_{1,u} * P_{1,v}
    的最大值
M = -float('inf')
for i in range(num_experts):
    for j in range(i+1, num_experts):
        M = max(M, matrix[i, 0] * matrix[j, 0] + matrix[i, 1] *
            matrix[j, 1])

# 填充相似矩阵
for u in range(num_experts):
    for v in range(num_experts):
        if u == v:
            # 对角线元素设置为1
            similarity_matrix[u, v] = 1
        else:
            # 使用给定的公式计算 X_{u,v}
            P_0_u, P_1_u = matrix[u]
            P_0_v, P_1_v = matrix[v]
            X_uv = (P_0_u * P_0_v + P_1_u * P_1_v) / M
            similarity_matrix[u, v] = X_uv

print(similarity_matrix)

Z = linkage(1 - similarity_matrix, method='ward')

# 绘制树状图 (Dendrogram)
plt.figure(figsize=(10, 7))
dendrogram(Z)
plt.title('Hierarchical Clustering Dendrogram')
```

```
plt.xlabel('Expert')
plt.ylabel('Distance')
plt.savefig('1.png')

# 使用cut_tree函数确定聚类标签
labels = cut_tree(Z, n_clusters=3).flatten()
# print("Cluster labels:", labels)

counter = Counter(labels)
# print("Number of instances in each cluster:", counter)

# 初始化一个空字典
expert_labels_dict = {}

# 使用循环来填充字典
for i, expert_name in enumerate(all_experts):
    expert_labels_dict[expert_name] = labels[i]

# print("Expert labels dictionary:", expert_labels_dict)

# 遍历字典并按照条件更新值
for expert, value in expert_labels_dict.items():
    if value == 0:
        expert_labels_dict[expert] = 1
    elif value == 1:
        expert_labels_dict[expert] = 0.8
    elif value == 2:
        expert_labels_dict[expert] = 1.2

print(expert_labels_dict)

expert_weight = list(expert_labels_dict.values())

score = {}

for index,row in df.iterrows():
    team_sc = sum(a * b for a,b in
        zip(row.tolist(),expert_weight))
    score[index] = team_sc
```

```
# print(score)

# 创建一个新的Pandas Series，与DataFrame的行索引对应
total_score_series = pd.Series(score)

# 向DataFrame中添加新的列
df['总得分'] = total_score_series

# 保存更新后的DataFrame回CSV文件
df.to_csv('附件1原始分权重总得分(改).csv')

# # 初始化用于存储不同分类的专家集合的字典
# class_experts_dict = {0: [], 1: [], 2: []}

# # 分类专家
# for expert, label in expert_labels_dict.items():
#     class_experts_dict[label].append(expert)

# # 初始化最终输出的字典
# final_output = {}

# # 获取每一类专家在 matrix 矩阵中的两个值
# for label, experts in class_experts_dict.items():
#     num_experts = len(experts)
#     class_matrix = np.zeros((num_experts, 2))

#     for i, expert in enumerate(experts):
#         row_index = all_experts.index(expert)
#         class_matrix[i, 0] = matrix[row_index, 0]
#         class_matrix[i, 1] = matrix[row_index, 1]

#     final_output[label] = class_matrix

# all_sum = []
# for i in range(3):
#     Sum = 0
#     for item in final_output[i]:
#         Sum += sum(item)
```

```
#    Sum = Sum/len(final_output[i])
#    all_sum.append(Sum)

# print(all_sum)
```

```
import pandas as pd
import numpy as np
P1 = pd.read_csv("C:/Users/1/Desktop/A.csv",header=0)#附件1

P1_ZJ = P1.iloc[2:2017,:19]
ZJ = list()
for i in range(len(P1_ZJ)):
    for j in range(5,19,3):
        ZJ.append(P1_ZJ.iloc[i,j])



ZJ_name = np.unique(ZJ)
Z_name = dict()
for i in ZJ_name:
    Z_name[i] = []
TT = np.zeros([len(P1_ZJ),len(ZJ_name)])#原始分数矩阵
t = [5,8,11,14,17]
for i in range(len(P1_ZJ)):
    for index,j in enumerate(ZJ_name):
        for k in t:
            if j == P1_ZJ.iloc[i,k]:
                TT[i,index] = P1_ZJ.iloc[i,k+1]
                Z_name[j].append(P1_ZJ.iloc[i,k+1])
Z_Z = dict()
for i in ZJ_name:
    Z_Z[i] = []

for index,i in enumerate(Z_Z.keys()):
    z = dict()
    for k in ZJ_name:
        z[k] = []
    for ii in range(TT.shape[0]):
        for jj in range(len(ZJ_name)):
```

```
            if (TT[ii,index] != 0) and (TT[ii,jj] != 0):
                z[ZJ_name[jj]].append(ii)
    Z_Z[i].append(z)

import matplotlib.pyplot as plt
import numpy as np
from matplotlib.font_manager import FontProperties
font_path = "C:/Windows/Fonts/SimHei.ttf" # 指定中文字体文件路径
font = FontProperties(fname=font_path, size=12) # 选择合适的字体大小

import matplotlib.pyplot as plt
from matplotlib.font_manager import FontProperties

# 指定中文字体
font_path = "C:/Windows/Fonts/SimHei.ttf" # 指定中文字体文件路径
font = FontProperties(fname=font_path, size=12) # 选择合适的字体大小

# 两个矩阵数据
data1 = [70.0, 70.0, 75.0, 75.0, 85.0, 40.0, 50.0, 30.0, 40.0,
    25.0, 55.0, 25.0, 70.0, 45.0, 45.0, 30.0, 60.0, 55.0, 80.0,
    60.0]
data2 = [68.0, 60.0, 54.0, 64.0, 55.0, 31.0, 40.0, 35.0, 25.0,
    17.0, 20.0, 21.0, 46.0, 43.0, 29.0, 33.0, 42.0, 58.0, 44.0,
    51.0]

# 创建x轴数据 (可以是简单的0到n的整数，也可以是其他标签)
x = range(1, len(data1) + 1)

# 绘制折线图
plt.plot(x, data1, label='数据1', marker='o', linestyle='-')
plt.plot(x, data2, label='数据2', marker='s', linestyle='--')

# 添加图例
plt.legend(prop=font)

# 添加标题和轴标签
plt.title('两个矩阵的折线图', fontproperties=font)
plt.xlabel('数据点', fontproperties=font)
plt.ylabel('数值', fontproperties=font)
```

```
# 显示图形
plt.show()


for index,i in enumerate(Z_Z.keys()):
    temp1 = list()
    temp2 = list()
    for index1,i1 in enumerate(Z_Z.keys()):
        temp1 = list()
        temp2 = list()
        if len(Z_Z[i][0][i1]) > 5 and i != i1:
            for jj in Z_Z[i][0][i1]:
                temp1.append(TT[jj,index])
                temp2.append(TT[jj,index1])
            # 两个矩阵数据
            data1 = temp1
            data2 = temp2
            # 创建x轴数据（可以是简单的0到n的整数，也可以是其他标签）
            x = range(1, len(data1) + 1)

            # 绘制折线图
            plt.plot(x, data1, label=i, marker='o', linestyle='-')
            plt.plot(x, data2, label=i1, marker='s', linestyle='--')

            # 添加图例
            plt.legend(prop=font)

            # 添加标题和轴标签
            plt.title('两个矩阵的折线图', fontproperties=font)
            plt.xlabel('数据点', fontproperties=font)
            plt.ylabel('数值', fontproperties=font)

            # 显示图形
            plt.show()

# 单独专家的独立同分布

int_Z = dict()
```

```
for i in ZJ_name:
    int_Z[i] = []
for index,j in enumerate(ZJ_name):
    for k in Z_name[j]:
        int_Z[j].append(int(k))

import matplotlib.pyplot as plt
import numpy as np
from matplotlib.font_manager import FontProperties
font_path = "C:/Windows/Fonts/SimHei.ttf" # 指定中文字体文件路径
font = FontProperties(fname=font_path, size=12) # 选择合适的字体大小

for index,j in enumerate(ZJ_name):
    # 输入的分数列表
    scores = int_Z[j]

    # 指定中文字体
    font_path = "C:/Windows/Fonts/SimHei.ttf" # 指定中文字体文件路径
    font = FontProperties(fname=font_path, size=12) #
        选择合适的字体大小

    # 设置直方图的区间间隔为10
    bin_width = 10

    # 计算直方图的频率和区间
    hist, bins = np.histogram(scores, bins=range(0, 101,
        bin_width))

    # 绘制直方图
    plt.hist(scores, bins=range(0, 101, bin_width),
        edgecolor='k', alpha=0.7)
    plt.xlabel('分数区间', fontproperties=font)
    plt.ylabel('频率', fontproperties=font)
    plt.title('分数分布直方图', fontproperties=font)
    plt.grid(axis='y', linestyle='--', alpha=0.7)
    plt.xticks(bins)
    plt.show()

# 第二部分的分析
```

```
# 方案一标准分
# 方案二去掉最大最小，剩余分数求和排序。

import pandas as pd
import numpy as np
P1 = pd.read_csv("C:/Users/1/Desktop/B.csv",header=0)#附件
P1_ZJ = P1.iloc[:885,:22]
ZJ = list()
for i in range(len(P1_ZJ)):
    for j in range(5,19,3):
        ZJ.append(P1_ZJ.iloc[i,j])
ZJ_name = np.unique(ZJ)
Z_name = dict()
for i in ZJ_name:
    Z_name[i] = []
TT = np.zeros([len(P1_ZJ),len(ZJ_name)])#原始分数矩阵
t = [5,8,11,14,17]
for i in range(len(P1_ZJ)):
    for index,j in enumerate(ZJ_name):
        for k in t:
            if j == P1_ZJ.iloc[i,k]:
                TT[i,index] = P1_ZJ.iloc[i,k+1]
                Z_name[j].append([i,P1_ZJ.iloc[i,k+1]])
ZJ_name = np.unique(ZJ)
Z_name = dict()
for i in ZJ_name:
    Z_name[i] = []
TT = np.zeros([len(P1_ZJ),len(ZJ_name)])#原始分数矩阵
t = [5,8,11,14,17]
for i in range(len(P1_ZJ)):
    for index,j in enumerate(ZJ_name):
        for k in t:
            if j == P1_ZJ.iloc[i,k]:
                TT[i,index] = P1_ZJ.iloc[i,k+1]
                Z_name[j].append([i,P1_ZJ.iloc[i,k+1]])

import matplotlib.pyplot as plt
from matplotlib.font_manager import FontProperties
```

```
# 设置中文显示的字体
font_path = "C:/Windows/Fonts/SimHei.ttf" # 替换为你的字体文件路径
# 两个向量的数据
font = FontProperties(fname=font_path, size=12) # 选择合适的字体大小

for index,j in enumerate(ZJ_name):
    temp = list()
    temp1 = list()
    for k in Z_name[j]:
        if k[0] <=240:
            temp.append(k[1])
            temp1.append(P1_ZJ.iloc[k[0],0])
    vector1 = np.array(temp)
    vector2 = np.array(temp1)
    #vector1 = np.exp(vector1/vector1.max())
    #vector2 = np.exp(vector2/vector2.max())

    print(vector1.mean(),vector1.std())
    print(vector2.mean(),vector2.std())
    print("专家作品长度",len(temp))

    x = np.arange(len(vector1))
    # 创建图形
    plt.figure()

    # 绘制图形并使用设置的字体
    plt.plot(x, vector1, label=j)
    plt.plot(x, vector2, label='对应总分')

    # 添加标题和标签，使用设置的字体
    plt.title('折线图', fontproperties=font)
    plt.xlabel('数据点', fontproperties=font)
    plt.ylabel('数值', fontproperties=font)

    # 添加图例，使用设置的字体
    plt.legend(prop=font)

    # 显示图形
```

```
    plt.show()

import pandas as pd
import numpy as np
P1 = pd.read_csv("C:/Users/1/Desktop/A.csv",header=0)#附件1

P1_ZJ = P1.iloc[:2017,:22]
ZJ = list()
for i in range(len(P1_ZJ)):
    for j in range(5,19,3):
        ZJ.append(P1_ZJ.iloc[i,j])

P1_ZJ.iloc[:352]

ZJ_name = np.unique(ZJ)
Z_name = dict()
for i in ZJ_name:
    Z_name[i] = []
TT = np.zeros([len(P1_ZJ),len(ZJ_name)])#原始分数矩阵
t = [5,8,11,14,17]
for i in range(len(P1_ZJ)):
    for index,j in enumerate(ZJ_name):
        for k in t:
            if j == P1_ZJ.iloc[i,k]:
                TT[i,index] = P1_ZJ.iloc[i,k+1]
                Z_name[j].append([i,P1_ZJ.iloc[i,k+1]])

import matplotlib.pyplot as plt
from matplotlib.font_manager import FontProperties

# 设置中文显示的字体
font_path = "C:/Windows/Fonts/SimHei.ttf" # 替换为你的字体文件路径
# 两个向量的数据
font = FontProperties(fname=font_path, size=12) # 选择合适的字体大小

for index,j in enumerate(ZJ_name):
    temp = list()
    temp1 = list()
    for k in Z_name[j]:
```

```
        if k[0] > 352:
            temp.append(k[1])
            temp1.append(P1_ZJ.iloc[k[0],0])
    vector1 = np.array(temp)
    vector2 = np.sort(np.array(temp1))
    #vector1 = np.exp(vector1/vector1.max())
    #vector2 = np.exp(vector2/vector2.max())

    print(vector1.mean(),vector1.std())
    print(vector2.mean(),vector2.std())

    x = np.arange(len(vector1))
    # 创建图形
    plt.figure()

    # 绘制图形并使用设置的字体
    plt.plot(x, vector1, label=j)
    plt.plot(x, vector2, label='对应总分')

    # 添加标题和标签，使用设置的字体
    plt.title('折线图', fontproperties=font)
    plt.xlabel('数据点', fontproperties=font)
    plt.ylabel('数值', fontproperties=font)

    # 添加图例，使用设置的字体
    plt.legend(prop=font)

    # 显示图形
    plt.show()

method2_score = np.zeros(len(TT),float)
for i in range(len(TT)):
    temp = TT[i,np.where(TT[i] != 0)]
    temp = temp[0]
    ind = np.argsort(temp)
    print(sum(temp[ind[1:-1]]))
    method2_score[i] = sum(temp[ind[1:-1]])

np.where( method2_score==method2_score.max())
```

```
o_score = np.array(P1_ZJ.iloc[:,0])
o_score = np.exp(o_score/(o_score.max()))
method2_score = np.exp(method2_score/(method2_score.max()))

method2_score = np.exp(method2_score/(method2_score.max()))

talking = np.where( method2_score==method2_score.max())[0]

talking_zj = np.where(TT[talking][0] != 0)#f方案二异常的作品对应的专家

ZJ_name = np.unique(ZJ)
talking_zj_name = ZJ_name[talking_zj]

ZJ_name = np.unique(ZJ)
talking_zj_name = ZJ_name[talking_zj]
int_Z = dict()
for i in ZJ_name:
    int_Z[i] = []
for index,j in enumerate(ZJ_name):
    for k in Z_name[j]:
        int_Z[j].append(int(k))

import matplotlib.pyplot as plt
import numpy as np
from matplotlib.font_manager import FontProperties
font_path = "C:/Windows/Fonts/SimHei.ttf" # 指定中文字体文件路径
font = FontProperties(fname=font_path, size=12) # 选择合适的字体大小

for i in talking_zj_name:
    # 输入的分数列表
    scores = int_Z[i]
    # 指定中文字体
    font_path = "C:/Windows/Fonts/SimHei.ttf" # 指定中文字体文件路径
    font = FontProperties(fname=font_path, size=12) #
        选择合适的字体大小
    # 设置直方图的区间间隔为10
    bin_width = 10
    # 计算直方图的频率和区间
```

```
    hist, bins = np.histogram(scores, bins=range(0, 101,
        bin_width))
    # 绘制直方图
    plt.hist(scores, bins=range(0, 101, bin_width),
        edgecolor='k', alpha=0.7)
    plt.xlabel('分数区间', fontproperties=font)
    plt.ylabel('频率', fontproperties=font)
    plt.title('分数分布直方图', fontproperties=font)
    plt.grid(axis='y', linestyle='--', alpha=0.7)
    plt.xticks(bins)
    plt.show()

import matplotlib.pyplot as plt
from matplotlib.font_manager import FontProperties

# 设置中文显示的字体
font_path = "C:/Windows/Fonts/SimHei.ttf" # 替换为你的字体文件路径
# 两个向量的数据
font = FontProperties(fname=font_path, size=12) # 选择合适的字体大小
vector1 = o_score[:352]
vector2 = method2_score[:352]

x = np.arange(len(vector1))
# 创建图形
plt.figure()

# 绘制图形并使用设置的字体
plt.plot(x, vector1, label='原始')
plt.plot(x, vector2, label='方案2')

# 添加标题和标签，使用设置的字体
plt.title('折线图', fontproperties=font)
plt.xlabel('数据点', fontproperties=font)
plt.ylabel('数值', fontproperties=font)

# 添加图例，使用设置的字体
plt.legend(prop=font)

# 显示图形
```

```
plt.show()

method2_score

# 方案1
import pandas as pd
import numpy as np
P1 = pd.read_csv("C:/Users/1/Desktop/A.csv",header=0)#附件1

P1_ZJ = P1.iloc[2:2017,:22]
ZJ = list()
for i in range(len(P1_ZJ)):
    for j in range(5,19,3):
        ZJ.append(P1_ZJ.iloc[i,j])
ZJ_name = np.unique(ZJ)

Z_name = dict()
for i in ZJ_name:
    Z_name[i] = []
TT = np.zeros([len(P1_ZJ),len(ZJ_name)])#原始分数矩阵
t = [5,8,11,14,17]
for i in range(len(P1_ZJ)):
    for index,j in enumerate(ZJ_name):
        for k in t:
            if j == P1_ZJ.iloc[i,k]:
                TT[i,index] = P1_ZJ.iloc[i,k+2]
                Z_name[j].append(P1_ZJ.iloc[i,k+2])
int_Z = dict()
for i in ZJ_name:
    int_Z[i] = []
for index,j in enumerate(ZJ_name):
    for k in Z_name[j]:
        int_Z[j].append(k)

import matplotlib.pyplot as plt
import numpy as np
from matplotlib.font_manager import FontProperties
font_path = "C:/Windows/Fonts/SimHei.ttf" # 指定中文字体文件路径
font = FontProperties(fname=font_path, size=12) # 选择合适的字体大小
```

```
for index,j in enumerate(ZJ_name):
    # 输入的分数列表
    scores = int_Z[j]

    # 指定中文字体
    font_path = "C:/Windows/Fonts/SimHei.ttf" # 指定中文字体文件路径
    font = FontProperties(fname=font_path, size=12) #
        选择合适的字体大小
    # 设置直方图的区间间隔为10
    bin_width = 10
    # 计算直方图的频率和区间
    hist, bins = np.histogram(scores, bins=range(0, 101,
        bin_width))

    # 绘制直方图
    plt.hist(scores, bins=range(0, 101, bin_width),
        edgecolor='k', alpha=0.7)
    plt.xlabel('分数区间', fontproperties=font)
    plt.ylabel('频率', fontproperties=font)
    plt.title('分数分布直方图', fontproperties=font)
    plt.grid(axis='y', linestyle='--', alpha=0.7)
    plt.xticks(bins)
    plt.show()
method1_score = np.zeros(len(TT),float)
for i in range(len(TT)):
    temp = TT[i,np.where(TT[i] != 0)]
    temp = temp[0]
    ind = np.argsort(temp)
    print(sum(temp[ind[1:-1]]))
    method1_score[i] = sum(temp[ind[1:-1]])

o_score = np.array(P1_ZJ.iloc[:,0])
o_score = np.exp(o_score/(o_score.max()))
method1_score = np.exp(method1_score/(method1_score.max()))

import matplotlib.pyplot as plt
from matplotlib.font_manager import FontProperties
```

```
# 设置中文显示的字体
font_path = "C:/Windows/Fonts/SimHei.ttf" # 替换为你的字体文件路径
# 两个向量的数据
font = FontProperties(fname=font_path, size=12) # 选择合适的字体大小
vector1 = o_score[:352]
vector2 = method1_score[:352]

x = np.arange(len(vector1))
# 创建图形
plt.figure()

# 绘制图形并使用设置的字体
plt.plot(x, vector1, label='原始')
plt.plot(x, vector2, label='方案1')

# 添加标题和标签，使用设置的字体
plt.title('折线图', fontproperties=font)
plt.xlabel('数据点', fontproperties=font)
plt.ylabel('数值', fontproperties=font)

# 添加图例，使用设置的字体
plt.legend(prop=font)

# 显示图形
plt.show()

# 新的标准公式

import pandas as pd
import numpy as np
PP = pd.read_csv("C:/Users/1/Desktop/AAA.csv",header=0)#附件1

PP1 = PP.iloc[:,1:-1]
PP.iloc[:,-1]

o_score = np.array(P1_ZJ.iloc[:,0])
o_score = np.exp(o_score/(o_score.max()))
method3_score = np.exp(method3_score/(method3_score.max()))
```

```
np.where(method3_score==method3_score.max())

np.where(PP.iloc[203,1:-1] != 0)

P1_ZJ = P1.iloc[2:2017,:22]
ZJ = list()
for i in range(len(P1_ZJ)):
    for j in range(5,19,3):
        ZJ.append(P1_ZJ.iloc[i,j])
ZJ_name = np.unique(ZJ)

talking_zj1 = np.where(PP.iloc[203,1:-1] !=
    0)#f方案二异常的作品对应的专家
ZJ_name = np.unique(ZJ)
talking_zj_name1 = ZJ_name[talking_zj1]
int_Z = dict()
for i in ZJ_name:
    int_Z[i] = []
for index,j in enumerate(ZJ_name):
    for k in Z_name[j]:
        int_Z[j].append(int(k))

PP1.iloc[20,0]

np.where(PP1.iloc[:,0]!=0)[0]

Z_3 = dict()
for i in range(97):
    Z_3[i] = []
    for j in np.where(PP1.iloc[:,i]!=0)[0]:
        print(j)
        print(PP1.iloc[j,i])
        Z_3[i].append(PP1.iloc[j,i])

import matplotlib.pyplot as plt
import numpy as np
from matplotlib.font_manager import FontProperties
font_path = "C:/Windows/Fonts/SimHei.ttf" # 指定中文字体文件路径
font = FontProperties(fname=font_path, size=12) # 选择合适的字体大小
```

```
for index,j in enumerate(ZJ_name):
    # 输入的分数列表
    scores = Z_3[index]

    # 指定中文字体
    font_path = "C:/Windows/Fonts/SimHei.ttf" # 指定中文字体文件路径
    font = FontProperties(fname=font_path, size=12) #
        选择合适的字体大小

    # 设置直方图的区间间隔为10
    bin_width = 10

    # 计算直方图的频率和区间
    hist, bins = np.histogram(scores, bins=range(0, 101,
        bin_width))

    # 绘制直方图
    plt.hist(scores, bins=range(0, 101, bin_width),
        edgecolor='k', alpha=0.7)
    plt.xlabel('分数区间', fontproperties=font)
    plt.ylabel('频率', fontproperties=font)
    plt.title('分数分布直方图', fontproperties=font)
    plt.grid(axis='y', linestyle='--', alpha=0.7)
    plt.xticks(bins)
    plt.show()


import matplotlib.pyplot as plt
import numpy as np
from matplotlib.font_manager import FontProperties
font_path = "C:/Windows/Fonts/SimHei.ttf" # 指定中文字体文件路径
font = FontProperties(fname=font_path, size=12) # 选择合适的字体大小
for i in talking_zj_name1:
    # 输入的分数列表
    scores = int_Z[j]
    # 指定中文字体
    font_path = "C:/Windows/Fonts/SimHei.ttf" # 指定中文字体文件路径
    font = FontProperties(fname=font_path, size=12) #
        选择合适的字体大小
```

```
    # 设置直方图的区间间隔为10
    bin_width = 10
    # 计算直方图的频率和区间
    hist, bins = np.histogram(scores, bins=range(0, 101,
        bin_width))
    # 绘制直方图
    plt.hist(scores, bins=range(0, 101, bin_width),
        edgecolor='k', alpha=0.7)
    plt.xlabel('分数区间', fontproperties=font)
    plt.ylabel('频率', fontproperties=font)
    plt.title('分数分布直方图', fontproperties=font)
    plt.grid(axis='y', linestyle='--', alpha=0.7)
    plt.xticks(bins)
    plt.show()

temp =method3_score[:352]
temp1 = method1_score[:352]

np.argsort(temp)[-23:]

np.argsort(temp1)[-23:]

import matplotlib.pyplot as plt
from matplotlib.font_manager import FontProperties

# 设置中文显示的字体
font_path = "C:/Windows/Fonts/SimHei.ttf" # 替换为你的字体文件路径
# 两个向量的数据
font = FontProperties(fname=font_path, size=12) # 选择合适的字体大小
vector1 = o_score[:352]
vector2 = method1_score[:352]

# 假设 method3_score 包含了你的数据

# 使用 argsort 方法对 method3_score 进行排序
sorted_indices = np.argsort(vector2)[::-1]

# 使用排序后的索引获取排序后的向量
vector2 = vector2[sorted_indices]
```

```
# 如果你只需要前 352 个值，你可以这样截取
vector2 = vector2
x = np.arange(len(vector1))
# 创建图形
plt.figure()

# 绘制图形并使用设置的字体
plt.plot(x, vector1, label='原始')
plt.plot(x, vector2, label='方案3')

# 添加标题和标签，使用设置的字体
plt.title('折线图', fontproperties=font)
plt.xlabel('数据点', fontproperties=font)
plt.ylabel('数值', fontproperties=font)

# 添加图例，使用设置的字体
plt.legend(prop=font)

# 显示图形
plt.show()

# 其它

np.argsort(TT[0,np.where(TT[0] != 0)])

file_name = "C:/Users/1/Desktop/data.csv"
delimiter = ","

# 使用np.savetxt保存二维数组为CSV文件
np.savetxt(file_name, ttt, delimiter=delimiter)

Z_name = dict()
for i in ZJ_name:
    Z_name[i] = []

P1_ZJ.iloc[:,1]

for i in range(len(P1_ZJ)):
```

```
    for j in range(5,19,3):
        for k in Z_name.keys():
            if k == (P1_ZJ.iloc[i,j]):
                Z_name[k].append(P1_ZJ.iloc[i,1])
Z_array = np.zeros([len(ZJ_name),len(ZJ_name)],float)


 for index1,k1 in enumerate(Z_name.keys()):
    for index2,k2 in enumerate(Z_name.keys()):
        inter=np.intersect1d(Z_name[k1],Z_name[k2])
        Z_array[index1,index2] = len(inter)

Total = np.zeros([3,97],float)

for index1,k1 in enumerate(Z_name.keys()):
    Total[0,index1] = len(Z_name[k1])
    print(len(Z_name[k1]))

for index1,k1 in enumerate(Z_name.keys()):
    Total[1,index1] = 97-len(np.where(Z_array[index1,:]==0)[0])
    Total[2,index1] = sum(Z_array[index1,:])
    print(97-len(np.where(Z_array[index1,:]==0)[0]))
    print(sum(Z_array[index1,:]))

Z_3

import matplotlib.pyplot as plt
import numpy as np

# 三行数据
data1 = np.array([104., 104., 103., 103., 104., 104., 104.,
    104., 103., 105., 104.,
       104., 104., 104., 104., 104., 104., 103., 103., 103.,
          105., 109.,
       105., 103., 104., 103., 103., 104., 104., 104., 107.,
          103., 104.,
       104., 107., 103., 104., 104., 102., 104., 103., 104.,
          103., 107.,
       104., 103., 104., 103., 104., 104., 103., 104., 104.,
```

```
        103., 103.,
       103., 103., 105., 104., 103., 102., 108., 104., 104.,
        103., 103.,
       104., 103., 103., 104., 104., 103., 103., 103., 102.,
        104., 105.,
       105., 103., 104., 103., 103., 104., 104., 104., 104.,
        104., 103.,
       103., 104., 105., 104., 105., 104., 104., 104., 104.])

data2 = np.array([18., 19., 25., 19., 19., 19., 14., 16., 18.,
    16., 26.,
        21., 17., 20., 21., 14., 15., 16., 16., 14., 21., 19.,
        14., 17., 13., 20., 17., 17., 24., 15., 24., 23., 31.,
        23., 31., 18., 13., 20., 16., 12., 13., 31., 18., 22.,
        35., 15., 24., 16., 17., 23., 22., 16., 16., 28., 22.,
        17., 22., 20., 17., 13., 19., 26., 18., 17., 17., 15.,
        15., 27., 19., 21., 16., 17., 21., 22., 18., 23., 15.,
        28., 16., 29., 16., 17., 19., 18., 16., 17., 20., 18.,
        14., 18., 17., 30., 17., 25., 13., 18., 20.])

data3 = np.array([520., 520., 515., 515., 520., 520., 520.,
    520., 515., 525., 520.,
       520., 520., 520., 520., 520., 520., 515., 515., 515.,
        525., 545.,
       525., 515., 520., 515., 515., 520., 520., 520., 535.,
        515., 520.,
       520., 535., 515., 520., 520., 510., 520., 515., 520.,
        515., 535.,
       520., 515., 520., 515., 520., 520., 515., 520., 520.,
        515., 515.,
       515., 515., 525., 520., 515., 510., 540., 520., 520.,
        515., 515.,
       520., 515., 515., 520., 520., 515., 515., 515., 510.,
        520., 525.,
       525., 515., 520., 515., 515., 520., 520., 520., 520.,
        520., 515.,
       515., 520., 525., 520., 525., 520., 520., 520., 520.])

# 创建 x 轴坐标
```

```
x = np.arange(len(data1))


import matplotlib.pyplot as plt
from matplotlib.font_manager import FontProperties


# 设置中文显示的字体
font_path = "C:/Windows/Fonts/SimHei.ttf" # 替换为你的字体文件路径
# 两个向量的数据
font = FontProperties(fname=font_path, size=12) # 选择合适的字体大小
# 创建图形
plt.figure(figsize=(10, 6))

# 绘制三行数据的折线图，使用设置的字体
plt.plot(x, data1, label='数据 1')
plt.plot(x, data2, label='数据 2')
plt.plot(x, data3, label='数据 3')

# 添加标题和标签，使用设置的字体
plt.title('折线图', fontproperties=font)
plt.xlabel('数据点', fontproperties=font)
plt.ylabel('数值', fontproperties=font)

# 添加图例，使用设置的字体
plt.legend(prop=font)

# 显示图形
plt.show()


from pylab import *
from matplotlib.font_manager import FontProperties
import matplotlib.pyplot as plt

for index1,k1 in enumerate(Z_name.keys()):

    plt.plot(range(len(ZJ_name)),Z_array[index1,:])
    plt.show()
```

```
# 问题1

# 作品有3000个
# 专家有125个
#同一道题目有5名不同的专家负责

import numpy as np

def generate_sparse_vector(size, num_ones):
    if num_ones > size:
        raise ValueError("Number of ones cannot exceed vector
            size.")

    # 生成一个全为0的向量
    sparse_vector = np.zeros(size, dtype=int)

    # 随机选择要设置为1的位置
    ones_indices = np.random.choice(size, num_ones, replace=False)

    # 在选定的位置上设置为1
    sparse_vector[ones_indices] = 1

    return sparse_vector

# 指定向量大小和1的数量，不设置随机种子
vector_size = 20
num_ones = 5 # 指定1的数量

# 生成随机稀疏向量
sparse_vector = generate_sparse_vector(vector_size, num_ones)
print(sparse_vector)
sum(sparse_vector)
np.where(p1[0,:]!=0)
np.where(p1[1,:]!=0)

# 问题2
import pandas as pd
import numpy as np
```

```
P2 = pd.read_csv("C:/Users/1/Desktop/A.csv",header=0)#附件1

P2_ZJ = P1.iloc[2:2017,:19]
ZJ = list()
for i in range(len(P2_ZJ)):
   for j in range(5,19,3):
      ZJ.append(P2_ZJ.iloc[i,j])

ZJ = np.array(ZJ,str)
ZJ_name = np.unique(ZJ)
Z_name = dict()
for i in ZJ_name:
   Z_name[i] = []
for i in range(len(P2_ZJ)):
     for j in range(5,19,3):
         for k in Z_name.keys():
            if k == (P2_ZJ.iloc[i,j]):
               Z_name[k].append(P2_ZJ.iloc[i,1])
```

```
# 问题三
import random
import matplotlib.pyplot as plt
from matplotlib.font_manager import FontProperties

# 设置中文显示的字体
font_path = "C:/Windows/Fonts/SimHei.ttf" # 替换为你的字体文件路径
# 两个向量的数据
font = FontProperties(fname=font_path, size=12) # 选择合适的字体大小
import pandas as pd
import numpy as np
PP2 = pd.read_csv("C:/Users/1/Desktop/C.csv",header=0)#附件1
PP2 = PP2.iloc[:1500]


stand_score = list()
for index,j in enumerate(PP2.columns):
   if "标准" in j:
      #if pd.notna(PP2.iloc[i, index+1]) and k > 4:
      stand_score.append(np.array(PP2.iloc[:,index]))
```

```
stand_score = np.array(stand_score).reshape(-1,1500)

talk_score = np.zeros_like(stand_score)
temp = list()
for i in range(len(PP2)):
    k=0
    for index,j in enumerate(PP2.columns):
        if "标准" in j:
            if pd.notna(PP2.iloc[i, index+1]) and k > 4:
                talk_score[k,i]=PP2.iloc[i, index+1]
                temp.append([k,i])
            else:
                talk_score[k,i] = PP2.iloc[i, index]
            k = k+1
stand_score = np.array(stand_score).reshape(1500,8)
Q1 = stand_score[:,5:].max(axis=1)-stand_score[:,5:].min(axis=1)
temp1 = np.where(Q1>20)[0]
import copy
Q = result[:,5:].max(axis=1)-result[:,5:].min(axis=1)
temp = np.where(Q>20)[0]
S1 = np.zeros([3,len(temp)])

for index,i in enumerate(temp):
    S1[:,index] = result[i,5:].reshape(3)

MM = np.zeros_like(result.copy())
MM =copy.deepcopy(result)

m,n = S1.shape

r = np.zeros(n,float)
l = np.zeros(n,float)
T_mean = S1.mean(axis=0)
for i in range(n):
    for j in range(3):
        if S1[j,i] <= T_mean[i] :
            r[i] = r[i]+1
        if S1[j,i] >= T_mean[i]:
```

```
            l[i] = l[i]+1




FZ= np.zeros_like(S1,float)
T_mean = S1.mean(axis=0)

FZ1= np.zeros_like(S1,float)

for i in range(n):
    for j in range(3):
        FZ1[j,i] = np.abs(S1[j,i]-T_mean[i])

for aa in range(10):
    a1 = aa/10
    SS = 0
    #print(FZ)
    FZ = np.zeros_like(S1,float)
    for i in range(n):
        if i == 0:
            if r[i] > l[i]:
                temp_d = np.zeros(3,float)
                temp_im = np.zeros(3,float)

                index = np.argsort(S1[:,i])[0]
                index1 = np.argsort(FZ1[:,i])[-1]
                if index1 not in [index]:
                    l1 = np.abs(S1[:,i].mean()-S1[index1,i])
                    a =np.random.uniform(0.2*l1,l1+5 ,1)
                    temp_d[index1] = 1
                    if S1[:,i].mean()-S1[index1,i] >0:
                        temp_im[index1] = a
                    else:
                        temp_im[index1] = -a


                else:
                    l1 = S1[:,i].mean()-S1[index,i]
```

```
            a =np.random.uniform(0.2*l1,l1+5 ,1)
            temp_d[index] = 1
            temp_im[index] = a

        FZ[:,i] = (S1[:,i]+temp_d*temp_im)
        #FZ[:,i] = S2[:,i]
    else:
        temp_d = np.zeros(3,float)
        temp_im = np.zeros(3,float)


        index = np.argsort(S1[:,i])[-1]
        index1 = np.argsort(FZ1[:,i])[-1]
        if index1 not in [index]:

            l1 = np.abs(S1[:,i].mean()-S1[index1,i])
            a =np.random.uniform(0.2*l1,l1+5 ,1)
            temp_d[index1] = 1
            if S1[:,i].mean()-S1[index1,i] >0:
                temp_im[index1] = a
            else:
                temp_im[index1] = -a


        else:
            l1 = S1[index,i]-S1[:,i].mean()
            l2 = S1[index,i]-S1[:,i].min()
            a =np.random.uniform(l1,l1+5,1)
            temp_d[index] = 1
            temp_im[index] = -a



        FZ[:,i] = (S1[:,i]+temp_d*temp_im).reshape(-1)
        #FZ[:,i] = S2[:,i]
else:
    TT = list()
    #处理
```

```
        if r[i] > l[i]:
            for jj in range(100):
                index1 = np.argsort(FZ1[:,i])[-1]
                index = np.argsort(S1[:,i])[:2]
                for ind,ii in enumerate(index):
                    temp_d = np.zeros(3,float)
                    temp_im = np.zeros(3,float)
                    if index1 not in index:
                        temp_d1 = np.zeros(3,float)
                        temp_im1 = np.zeros(3,float)
                        l1 = np.abs(S1[:,i].mean()-S1[index1,i])
                        a =np.random.uniform(0.2*l1,l1+5,1)
                        temp_d1[index1] = 1
                        if S1[:,i].mean()-S1[index1,i] >0:
                            temp_im1[index1] = a
                        else:
                            temp_im1[index1] = -a
                        TT.append([temp_d1,temp_im1])
                    temp_d[index[:ind+1]] = 1
                    l1 = S1[:,i].mean()-S1[ii,i]
                    l2 = S1[:,i].max()-S1[ii,i]
                    a =np.random.uniform(0.2*l1,l1+5,ind+1)
                    temp_im[index[:ind+1]] = a
                    TT.append([temp_d,temp_im])

            score1 = list()
            score2 = list()
            score3 = list()
            R = list()
            for ii in TT:
                t=0
                T=0
                r1 = S1[:,i]+ii[0]*ii[1]
                FZ[:,i] = r1
                R.append(r1)
```

```
            score1.append(np.abs(1/T))#极差
            score2 .append(-FZ[:,:(i+1)].std())
            i1 = [0,FZ[:,0].mean()]
            i2 = [i,FZ[:,i].mean()]
            tt1 = [0,S1[:,0].mean()]
            tt2 = [len(S1[0,:]),S1[:,-1].mean()]
            k_rate1 = ((i2[1]-i1[1])/(i2[0]-i1[0]))
            k_rate2 = ((tt2[1]-tt1[1])/(tt2[0]-tt1[0]))
            score3.append(1/(k_rate1-k_rate2)**2)

        score1 = np.array(score1)
        score3 = np.array(score3)
        score2 = np.array(score2).reshape(-1)
        R = np.array(R)
        score1 = np.exp(score1/score1.max())
        score2 = np.exp(score2/score2.max())
        score3 = np.exp(score3/score3.max())
        SS =(1-a1)*(score3)+a1*(score1*0.3+score2*(0.7))
        id1 = np.argsort(SS)[-1]

        FZ[:,i] = R[id1].reshape(3)
    else:
        for jj in range(100):

            index = np.argsort(S1[:,i])[-2:]

            index1 = np.argsort(FZ1[:,i])[-1]


            for ind,ii in enumerate(index):
                temp_d = np.zeros(3,float)
                temp_im = np.zeros(3,float)
                if index1 not in index:
                    temp_d1 = np.zeros(3,float)
                    temp_im1 = np.zeros(3,float)
                    l1 = np.abs(S1[:,i].mean()-S1[index1,i])
                    a =np.random.uniform(0.2*l1,l1+5 ,1)
                    temp_d1[index1] = 1
```

```
                if S1[:,i].mean()-S1[index1,i] >0:
                    temp_im1[index1] = a
                else:
                    temp_im1[index1] = -a
                TT.append([temp_d1,temp_im1])
            temp_d[index[:ind+1]] = 1
            l1 = S1[ii,i]-S1[:,i].mean()
            l2 = S1[ii,i]-S1[:,i].min()
            a =np.random.uniform(0.2*l1,l1+5, ind+1)
            temp_im[index[:ind+1]] = -a
            TT.append([temp_d,temp_im])
    score1 = list()
    score2 = list()
    score3 = list()
    R = list()
    for ii in TT:
        t=0
        T=0
        r1 = S1[:,i]+ii[0]*ii[1]
        FZ[:,i] = r1
        R.append(r1)



        score1.append(np.abs(1/T))#极差
        score2 .append(-FZ[:,:(i+1)].std())
        i1 = [0,FZ[:,0].mean()]
        i2 = [i,FZ[:,i].mean()]
        tt1 = [0,S1[:,0].mean()]
        tt2 = [len(S1[0,:]),S1[:,-1].mean()]
        k_rate1 = ((i2[1]-i1[1])/(i2[0]-i1[0]))
        k_rate2 = ((tt2[1]-tt1[1])/(tt2[0]-tt1[0]))
        score3.append(1/(k_rate1-k_rate2)**2)

    score1 = np.array(score1)
    score3 = np.array(score3)
    score2 = np.array(score2).reshape(-1)
```

```
            R = np.array(R)
            score1 = np.exp(score1/score1.max())
            score2 = np.exp(score2/score2.max())
            score3 = np.exp(score3/score3.max())

            SS =
                (1-a1)*(score3)+a1*(score1*0.3+score2*(0.7))

            id1 = np.argsort(SS)[-1]

            FZ[:,i] = R[id1].reshape(3)

for index,i in enumerate(temp):
    result[i,5:]= FZ[:,index].reshape(-1,3)

Q1 = result[:,5:].max(axis=1)-result[:,5:].min(axis=1)
temp1 = np.where(Q1>20)[0]
print(len(temp1))

T = result[:,:5].mean(axis=1)+result[:,5:].sum(axis=1)
T1 = PP2.iloc[:,0]
T2 = tt[:1500]
T3 = MM[:,:5].mean(axis=1)+MM[:,5:].sum(axis=1)

print(sum(np.abs(np.argsort(T)-np.argsort(T1))))
print(sum(np.abs(np.argsort(T2)-np.argsort(T1))))
print(sum(np.abs(np.argsort(T3)-np.argsort(T1))))
x = np.arange(len(T))
# 创建图形
plt.figure()
#vector3[np.where(vector3==vector3.max())] = vector3.mean()

# 绘制图形并使用设置的字体
#plt.plot(x, vector2, label='二阶段')



#plt.plot(x, vector1, label='复议')
plt.plot(x, T1, label='复议后总分')
```

```
plt.plot(x, T, label='处理极差后的总分')
plt.plot(x, T2, label='无复议后总分')
plt.plot(x, T3, label='处理极差前的总分')
plt.xlabel('排名', fontproperties=font)
plt.ylabel('分数', fontproperties=font)

  # 添加标题和标签，使用设置的字体
plt.title('数据2.2评审模型调整后数据分布', fontproperties=font)

# 添加图例，使用设置的字体
plt.legend(prop=font)

plt.grid(True)
# 显示图形
plt.show()
```