研究生课程考核试卷

（适用于课程论文、提交报告）

科 目：计算问题求解　教 师：吴全旺

姓 名：小墩兽　学 号：

专 业：电子信息　类 别：专业硕士

上课时间：2022 年 9 月至 2022 年 11 月

考 生 成 绩：

卷面成绩	平时成绩	课程综合成绩

阅卷评语：

阅卷教师 (签名)

重庆大学研究生院制

目录

本文为一篇论文研读报告，所选论文为：何胜学. 公平性指派问题及均值逼近求解算法[J]. 系统科学与数学, 2022, 42(9): 2399-2411.

我将从问题描述、求解方法、代码复刻、可能的改进四个方面来具体介绍本文所介绍的计算问题求解领域的算法。

1 问题描述

本文解决的问题为指派问题，指派问题可以分为三种：经典线性指派问题、二次指派问题、三次指派问题。

本文介绍了公平性指派问题的一种解法，这是三次指派问题中的一种特殊形式。下面将详细介绍这几种问题的文字定义和数学定义。

本文中涉及的所有符号及所指代含义，已总结如下表 1.1 所示。

表 1.1 符号总览

Table 1.1 Notations

符号	含义
$x_{i,j}$	代理i是否执行了任务j，如果执行了，值为 1，若没执行，值为 0
$f_1(x)$	经典指派问题的目标函数
$c_{i,j}$	代理i执行任务j所需要付出的代价
$f_2(x)$	二次指派问题的目标函数
$d_{i,j}$	代理i执行任务j所需要付出的代价，此代价与$c_{(i,j)}$ 相互独立
$f_3(x)$	三次指派问题的目标函数
$e_{i,j}$	代理i执行任务j所需要付出的代价，此代价与$c_{(i,j)}, d_{(i,j)}$ 相互独立
c_{av}	平均工作负荷
n	代理数或任务数，这两个数必须相等
$\boldsymbol{C}$	工作负荷矩阵，这个矩阵的元素为$c_{i,j}$
$\underline{c}$	c_{av}的下限
$\overline{c}$	c_{av}的上限
$c_{av}(r)$	第r次迭代下的c_{av}
c_{fore}	c_{av}的前一次值
δ	算法迭代步长
$x^*(r)$	第r次迭代下，匈牙利算法输出的可行解
$c_{av}^*(r)$	第r次迭代下，基于$x^*(r)$计算出的平均工作负荷
γ	收敛判定常数
Δ	步长修正参数

1.1 指派问题

① 问题描述与基本结论

将 n 项任务分配给 n 个代理，每个代理只能完成一个任务，所有任务必须被完成，所有的任务不可分割。如果以数学语言来描述，可以定义成如下的几个式子：

$$\sum_i x_{i,j} = 1, \forall j \in J; \tag{1.1}$$

$$\sum_j x_{i,j} = 1, \forall i \in I; \tag{1.2}$$

$$x_{i,j} \in \{0,1\}, \forall i \in I, j \in J \tag{1.3}$$

式中I是代理的集合，i是它的一个元素；J是任务的集合，j是它的一个元素；$x_{i,j}$描述了代理i是否执行了任务j，如果执行了，其值为 1，如果没执行，其值为 0。

式子（1.1）的含义为任意一个任务有且仅有一个代理执行；式子（1.2）的含义为任意一个代理有且执行一个任务；式子（1.3）的含义是$x_{i,j}$的取值只能为 0 或 1。

根据（1.1）（1.2）（1.3），可以得到如下结论。

$$x_{i,j}^2 = x_{i,j} \tag{1.4}$$

$$x_{i,j}x_{i,k} = 0, \forall j \neq k, i \in I \tag{1.5}$$

$$x_{i,j}x_{k,j} = 0, \forall i \neq k, j \in I \tag{1.6}$$

$$x_{i,j}x_{k,l} \in \{0,1\}, \forall i \neq k, j \neq l \tag{1.7}$$

$$x_{i,j}x_{k,l}x_{m,n} \in \{0,1\}, \forall i \neq k \neq m, j \neq l \neq n \tag{1.8}$$

② 经典指派问题

指派问题下最常见的问题是经典指派问题，也称线性指派问题，属于线性纯整数规划问题，可以使用匈牙利算法在时间复杂度$O(n^3)$的条件下有效求解。经典线性指派问题的目标是最小化指派后的工作负荷。它的数学语言如下所示：

$$f_1(x) = \sum_{i \in I}\sum_{j \in J} c_{i,j}x_{i,j} \tag{1.9}$$

式中$f_1(x)$是目标函数，经典线性指派问题的目标是最小化$f_1(x)$，当然这种说法并不绝对，如果我们给$c_{i,j}$加上一个负号就能将问题转换为求最大化的$f_1(x)$，因此目标函数的定义取决于问题的实际场景；$c_{i,j}$是指代理i执行任务j时的工作负荷，作者假设它存在一个上下限，即$c_{i,j} \in [\underline{c}, \overline{c}]$。

③ 二次指派问题

二次指派问题加入了一个新的矩阵来衡量最终花费（收益），也就是说目标

函数多了一个影响量。它的数学语言如下所示：

$$f_2(x) = \sum_{i\in I}\sum_{j\in J} c_{i,j}d_{i,j}x_{i,j} \tag{1.10}$$

式中$f_2(x)$是目标函数；$c_{i,j}$是指代理i执行任务j时的第一类工作负荷；$d_{i,j}$是指代理i执行任务j时的第二类工作负荷。两种负荷共同构成了最终的工作负荷。

该问题是 NP-hard 问题，因此没有已知的算法可以在多项式时间内解决此问题，即使是很小的实例也可能需要较长的计算时间。

④ 三次指派问题

基于二次指派问题，我们可以类比推理出三次指派问题的数学语言：

$$f_3(x) = \sum_{i\in I}\sum_{j\in J} c_{i,j}d_{i,j}e_{i,j}x_{i,j} \tag{1.11}$$

式中$f_3(x)$是目标函数；$c_{i,j}$是指代理i执行任务j时的工作负荷；$d_{i,j}$是指代理i执行任务j时，单位$c_{i,j}$下的花费；$e_{i,j}$是指代理i执行任务j时，单位$d_{i,j}$下的花费。

显然，该问题也是 NP-hard 问题。

1.2 公平性指派问题

① 基本概念

作者提出的 “公平性指派问题”，需要定义一个公平性指标。首先需要衡量某种分配下的平均工作负荷，它用c_{av}表示，并且可以使用下式计算：

$$c_{av} = \frac{f_1(x)}{n} = \frac{\sum_{i\in I}\sum_{j\in J} c_{i,j}x_{i,j}}{n} \tag{1.12}$$

基于c_{av}，作者定义公平性指标$f_4(x)$如下式所示：

$$f_4(x) = \sum_{i\in I}\sum_{j\in J}\left(c_{i,j} - c_{av}\right)^2 x_{i,j} \tag{1.13}$$

将（1.12）带入（1.13），并作代数展开，$f_4(x)$的展开式中将包含$x_{i,j}x_{k,l}x_{m,n}$,$\forall i \neq k \neq m, j \neq l \neq n$的三次指派问题。因此，公平性指派问题从本质来看是一种三次指派问题，但是公平性指派问题的目标函数是（1.13）的格式，和常规的三次指派问题不同。此时函数的优化目标是平衡任务分配后，代理之间的工作负荷。

② 分类

1） 第一类公平性指派问题

这种情形的平均工作负荷c_{av}为定值，下面会介绍它的具体求法。此时需要定义代理和任务自身工作所引发的工作负荷。此时的工作负荷矩阵C如下式所

示（以 3×3 的情况为例）：

$$\begin{aligned}\boldsymbol{C}=\boldsymbol{A}+\boldsymbol{B}&=\begin{bmatrix}a_1 & a_1 & a_1\\ a_2 & a_2 & a_2\\ a_3 & a_3 & a_3\end{bmatrix}+\begin{bmatrix}b_1 & b_2 & b_3\\ b_1 & b_2 & b_3\\ b_1 & b_2 & b_3\end{bmatrix}\\&=\begin{bmatrix}a_1+b_1 & a_1+b_2 & a_1+b_3\\ a_2+b_1 & a_2+b_2 & a_2+b_3\\ a_3+b_1 & a_3+b_2 & a_3+b_3\end{bmatrix}\end{aligned} \tag{1.14}$$

式中a_i代表代理自身工作所引发的工作负荷，b_i代表任务所引发的工作负荷。

求解方法

$$c_{av}=\frac{\sum_{i\in I}\sum_{j\in J}(a_i+b_j)}{n}=\frac{\sum_{i=1}^{n}(a_i+b_i)}{n} \tag{1.15}$$

公式（1.15）的结果显然是一个定值，即所有a_i和b_i的累加。将（1.15）带入公式（1.13）。（1.13）可以化简为经典指派问题，如（1.9）所示，此时通过匈牙利算法可以有效求解。

2）第二类公平性指派问题

这种情形的平均工作负荷c_{av}不为定值，此时的工作负荷矩阵C如下式所示（以 3×3 的情况为例）：

$$\boldsymbol{C}=\begin{bmatrix}c_{1,1} & c_{1,2} & c_{1,3}\\ c_{2,1} & c_{2,2} & c_{2,3}\\ c_{3,1} & c_{3,2} & c_{3,3}\end{bmatrix} \tag{1.16}$$

此时每个代理对应每个任务都有独立的工作负荷。此时的公平性指派问题就是一类 NP-hard 问题。本文的讨论也基于这类问题进行。

2 求解方法

作者针对第二类公平性指派问题，提出了一种多项式时间内的求解算法，即均值逼近算法。该算法的核心思想是不断假定一个c_{av}，基于此c_{av}可以得到工作负荷矩阵C，此时原问题将转换为可用匈牙利算法求解的经典指派问题，将求解的结果再一个新的c_{av}^{*}，重复之前的步骤会出现收敛现象，便可得到一个局部最优解。将多个局部最优解中找出目标函数值最低的解，即得全局最优解。下面将详细介绍此算法。

2.1 匈牙利算法

匈牙利算法是均值逼近算法中子问题的解决方法，主要用于解决一些与二分图匹配有关的问题。前文所提到的经典指派问题，就可以看作是二分图的匹配问题，可以使用匈牙利算法。匈牙利算法的步骤如下。

① 矩阵初等变换

根据工作负荷$c_{i,j}$构造矩阵，随后对矩阵初等变换，使各行各列都出现 0 元素。下面给出一个案例。

$$\begin{bmatrix}6&7&11&2\\4&5&9&8\\3&1&10&4\\5&9&8&2\end{bmatrix}\xrightarrow{\text{行变换}}\begin{bmatrix}4&5&9&0\\0&1&5&4\\2&0&9&3\\3&7&6&0\end{bmatrix}\xrightarrow{\text{列变换}}\begin{bmatrix}4&5&4&0\\0&1&0&4\\2&0&4&3\\3&7&1&0\end{bmatrix}\tag{2.1}$$

② 尝试找寻最优解

1）寻找只有一个 0 元素的行，标记这个 0 元素，同时划去该列其它 0 元素

2）寻找只有一个 0 元素的列，标记这个 0 元素，同时划去该行其它 0 元素

3）若还存在没有标记的 0 元素，且找不到独立 0 元素的行（列），从剩余 0 元素最少的行（列）开始，比较这行 0 元素所在列中 0 元素的数目，选择 0 元素最少的那列的这个 0 元素画圈，同时划去该行该列其余 0 元素

$$\begin{bmatrix}4&5&4&0\\0&1&0&4\\2&0&4&3\\3&7&1&0\end{bmatrix}\xrightarrow{\text{寻找孤立 0 元素}}\begin{bmatrix}4&5&4&0\\0&1&~~0~~&4\\2&0&4&3\\3&7&1&~~0~~\end{bmatrix}\tag{2.2}$$

③ 打勾画直线

1）给没有“标记 0”的行打勾。

2）打勾行中，含“删除 0”元素的列打勾。

3）打勾列中，含“标记 0”元素的行打勾。

4）未打勾行画横线，打勾列画竖线。

$$\begin{bmatrix}4&5&4&0\\0&1&0&4\\2&0&4&3\\3&7&1&0\end{bmatrix}\xrightarrow{(1)}\begin{bmatrix}4&5&4&0\\0&1&0&4\\2&0&4&3\\3&7&1&0\end{bmatrix}\xrightarrow{(2)}\begin{bmatrix}4&5&4&0\\0&1&0&4\\2&0&4&3\\3&7&1&0\end{bmatrix}\begin{matrix}\\\\\\\surd\end{matrix}$$
$$\begin{matrix}\\\\\\\surd\end{matrix}\qquad\begin{matrix}\surd\\\\\\\surd\end{matrix}$$
$$\xrightarrow{(3)}\begin{bmatrix}4&5&4&0\\0&1&0&4\\2&0&4&3\\3&7&1&0\end{bmatrix}\begin{matrix}\surd\\\\\\\surd\end{matrix}\xrightarrow{(4)}\begin{bmatrix}4&5&4&0\\0&1&0&4\\2&0&4&3\\3&7&1&0\end{bmatrix}\begin{matrix}\surd\\\\\\\surd\end{matrix}\qquad\begin{matrix}\\\\\\\surd\end{matrix}\tag{2.3}$$

④ 增加 0 元素

当我们画的直线数目小于矩阵的行数或者列数时，说明我们此时无法找到一个可行解。找不到可行解的根本原因是②中的 0 元素不够。因此我们需要增 0 元素。执行如下两个步骤。

1）寻找未被直线覆盖部分的最小元素

2）行减去最小元素，打勾列加上最小元素。

$$\begin{bmatrix}4&5&4&0\\0&1&0&4\\2&0&4&3\\3&7&1&0\end{bmatrix}\xrightarrow{(1)}1\xrightarrow{(2)}\begin{bmatrix}3&4&3&0\\0&1&0&5\\2&0&4&4\\2&6&0&0\end{bmatrix}\tag{2.4}$$

随后，重复②③步，直到找到一个可行解x。至此，匈牙利算法结束，最后找到的 0 元素的位置，就对应了代理和任务的对应关系。

2.2 均值逼近算法

① 基本思想

根据第一章的分析，第一类公平性指派问题可以转换为可用匈牙利算法求解的经典指派问题。那么如果我们在给定一个初始工作负荷均值，就可以将第二类公平性指派问题转换为第一类公平性指派问题，随后，通过求解对应的线性指派问题可得到一个新的工作负荷均值；利用新的工作负荷均值又可以形成新的线性指派问题，并对其加以求解；重复上述求解过程可形成一个具有收敛特性的工作负荷均值序列。

作者认为重复这个子问题的求解过程，最终可以得到一个收敛的解，且这个解是初始工作负荷均值下的一个局部最优解。如果我们让初始工作负荷均值在一定的范围里，并对这个范围进行一维搜索求解，就可以得到一个不断逼近原问题的全局最优解。

② 算法步骤

1）确定工作负荷均值c_{av}

首先需要确定c_{av}的取值范围。取值的下限$\underline{c}=\min\{c_{i,j}\},i\in I,j\in J$，上限$\overline{c}=\max\{c_{i,j}\},i\in I,j\in J$。令子问题的初始求解序号为$\tau=0$，定义$c_{av}(r)$为第$r$次迭代下的工作负荷均值，定义前一收敛均值为$c_{fore}$。初始状态下，各个参数的数学表达如下：

$$\begin{gathered}c\in[\underline{c},\overline{c}\,]\\ c_{av}(0)=\underline{c},c_{fore}=\underline{c}-1\end{gathered}\tag{2.5}$$

2）确定算法迭代步长δ

将工作负荷矩阵C的所有元素置入一个集合，并按照大小升序排列，得到一个有序集合C_M。迭代步长δ的计算如下式所示：

$$\begin{gathered}C_M=\{c_1,c_2,\dots,c_M\}\\ \delta=\max\left\{\frac{\min\limits_{m\le M-1}\{c_{m+1}-c_m\}}{n},\Delta\right\}\end{gathered}\tag{2.6}$$

式中M是工作负荷矩阵C的元素个数，n是代理或任务的数目，Δ是步长补偿参数，是一个预先设定好的正常数，避免C_M出现了两个非常接近的元素导致步长过小。

3）调用匈牙利算法解决问题

首先保存当前的工作负荷均值，$c_{initial}\coloneqq c_{av}(r)$。随后执行匈牙利算法，需要注意，此时的目标函数不是式（1.9），而是式（1.13）。根据当前的$c_{av}(r)$，基于式（1.13）对工作负荷矩阵$\boldsymbol{C}$进行一定变换后，再调用匈牙利算法求解这个线性指派问题，能够得到相应的可行解$x^*(r)$（r是迭代次数），和目标函数值f_r^*。随后利用这个可行解$x^*(r)$，根据式子（1.12）可以计算出在该可行解下的工作负荷均值$c_{av}^*(r)$。如果$|c_{av}^*(r)-c_{av}(r)|\le\gamma$（$\gamma$是判断是否已经收敛的判定常数），则转 4），否则令$c_{av}(r)\coloneqq c_{av}^*(r)$，随后重新调用匈牙利算法。

4）确定新的工作负荷均值

当收敛后，我们要确定一个新的c_{av}进行下一轮运算。如果$c_{av}^*(r)\neq c_{fore}$，则令$r\coloneqq r+1$, $c_{fore}\coloneqq c_{av}^*(r)$, $c_{av}(r)\coloneqq c_{av}^*(r)+\delta$，否则令$c_{av}(r)\coloneqq c_{initial}+\delta$

5）判断终止

如果$c_{av}(r)\le(\overline{c})$，转 3），否则，终止算法，输出对应的$f^*=\min\{f_r^*\}$的最优解$x^*$。

③ 算法伪代码

基于前文的算法原理，尽管原文没给出，但是可以写出如下的算法伪代码。

算法：均值逼近算法

Input: 工作负荷矩阵C，代理数目n，步长补偿参数Δ，收敛判断常数γ

Output: 最优解x^*

$r \leftarrow 0$;
$c_{av}(r) = \underline{c}$;
$c_{fore} = \underline{c} - 1$;
$C_M \leftarrow sort(C_M)$;
$\delta \leftarrow calculateDelta(C_M, \Delta)$;
repeat
  repeat
    $x^*(r), f_r^* \leftarrow Hungarian(c_{av}(r), \gamma)$;
    $c_{initial} \leftarrow c_{av}(r)$;
    $c_{av}^*(r) \leftarrow aveWorkload(x^*(r), n)$;
    if $|c_{av}^*(r) - c_{av}(r)| > \gamma$
      $c_{av}(r) \leftarrow c_{av}^*(r)$;
    else
      if $c_{av}^*(r) \neq c_{fore}$
        $c_{fore} \leftarrow c_{av}^*(r)$;
        $c_{av}(r+1) \leftarrow c_{av}^*(r) + \delta$;
        $r \leftarrow r + 1$;
      else
        $c_{av}(r) \leftarrow c_{initial} + \delta$;
  until $|c_{av}^*(r) - c_{av}(r)| \leq \gamma$
until $c_{av}(r) > (\overline{c})$

3 代码复现

作者并没有给出开源代码，但是基于作者的思想，我使用 Python 复写了作者的算法，并根据作者在原文的案例给出了验证，最后得到的指派结果与作者一致。并绘制了如下图 3.1。横坐标为增加单位步长的均值点，纵坐标为以横坐标为初始值，经过匈牙利算法后最终收敛的结果。

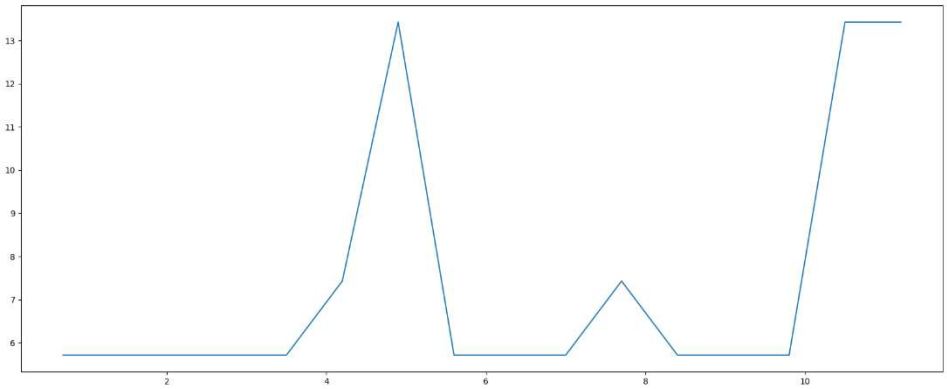

图 3.1 案例 1 的代码复写结果

Figure 3.1: The result of code replicates in case 1

对于作者论文中另一个的矩阵，我也进行了验证，该矩阵如图 3.2 所示，该矩阵在我复写的程序中输出的结果为：

最佳目标函数: 46.19999999999998

此时对应解: [(0, 11), (3, 0), (7, 13), (9, 16), (10, 5), (11, 19), (14, 6), (15, 10), (16, 4), (1, 17), (17, 15), (5, 1), (12, 3), (18, 8), (2, 9), (4, 7), (19, 12), (13, 2), (6, 14), (8, 18)]

此时对应的平均工作负荷： 39.3

$$C_{15\times15}=\begin{bmatrix}
49&56&53&82&35&53&76&86&46&32&33&41&53&43&49&73&83&43&83&36\\
78&56&46&38&39&73&69&80&66&54&82&39&34&49&71&38&65&39&37&44\\
60&63&62&51&71&86&73&61&37&37&35&67&85&75&76&71&86&64&32&72\\
40&67&72&74&39&49&49&69&85&73&81&56&64&38&32&52&88&76&52&36\\
57&69&63&82&68&31&55&40&82&40&38&60&86&83&88&63&84&59&47&80\\
83&38&66&77&85&31&54&67&83&35&31&61&46&58&68&41&76&46&47&76\\
72&32&39&46&88&51&34&88&70&79&79&84&81&61&38&83&30&81&52&37\\
59&64&40&81&82&84&59&52&84&39&64&74&36&40&39&34&87&47&66&83\\
77&46&58&68&45&78&50&70&44&50&71&80&74&57&39&72&69&31&38&55\\
86&81&75&60&78&65&50&55&81&64&74&50&71&78&51&49&43&72&67&68\\
45&83&51&42&50&37&63&32&44&83&87&72&88&40&31&88&84&51&60&83\\
77&46&67&56&51&88&75&85&60&83&57&46&49&68&53&85&65&80&78&40\\
33&40&48&40&44&78&33&42&62&63&76&89&36&67&41&65&83&62&82&65\\
72&63&39&84&33&76&61&82&73&79&34&38&38&86&56&33&62&33&83&45\\
65&76&64&51&77&36&39&80&58&53&79&48&71&30&33&65&57&83&55&60\\
52&60&56&44&37&74&73&67&71&63&38&44&83&67&79&35&32&88&48&67\\
59&78&31&83&42&75&71&53&88&84&86&46&49&31&46&73&59&84&89&46\\
55&56&59&77&57&75&61&56&43&75&88&66&88&79&70&40&85&70&58&80\\
38&86&79&55&58&34&77&30&39&36&44&82&51&70&48&63&41&82&51&71\\
81&48&75&87&38&86&63&56&62&57&60&73&38&67&58&31&38&86&31&80
\end{bmatrix}$$

图 3.2 案例 2 的代码复写

Figure 3.2: Code replicates in case 2

作者对这个案例给出的程序运行答案如下表 3.1 所示：

表 3.1 作者的实验结果
Table 3.1 Experimental results of the authors

方法	问题	指派结果	负荷均值	目标函数值	计算时间 (s)	子问题求解次数 (求解器迭代次数)
3A	1	[6, 3, 7, 5, 8, 9, 2, 1, 4, 10]	22.7	8.1	< 0.001	50
	2	[1, 5, 4, 11, 15, 6, 8, 7, 12, 10, 3, 14, 2, 13, 9]	39.8	42.4	< 0.001	100
	3	[12, 18, 10, 1, 8, 2, 15, 14, 19, 17, 6, 20, 4, 3, 7, 11, 5, 16, 9, 13]	39.3	46.2	< 0.001	123
	4	[11, 18, 25, 14, 9, 7, 3, 12, 8, 15, 6, 2, 13, 10, 20, 22, 5, 17, 24, 4, 19, 21, 16, 1, 23]	65.76	58.56	< 0.001	172
	5	[30, 21, 8, 17, 3, 26, 4, 1, 2, 16, 13, 19, 24, 9, 15, 29, 7, 12, 25, 28, 18, 27, 22, 20, 14, 5, 23, 11, 6, 10]	84.97	128.97	< 0.001	198
	6	[35, 12, 25, 17, 3, 32, 29, 28, 16, 27, 1, 14, 31, 6, 9, 5, 19, 8, 26, 30, 20, 4, 2, 11, 21, 18, 24, 13, 33, 22, 15, 23, 34, 7, 10]	56.6	38.40	< 0.001	131
Lingo	1	[6, 3, 7, 5, 8, 9, 2, 1, 4, 10]	22.7	8.1	15	168383
	2	[1, 5, 4, 11, 15, 6, 8, 7, 12, 10, 3, 14, 2, 13, 9]	39.8	42.4	411	3883007
	3	[12, 18, 10, 1, 8, 2, 15, 14, 19, 17, 6, 20, 4, 3, 7, 11, 5, 16, 9, 13]	39.3	46.2	2137	14518548

需要注意图 3.2 中，作者对矩阵的标注是 15×15，但实际上这个矩阵是 20×20，作者标注出错了。对应了表 3.1 第三行的内容。经过比对，我的程序运行结果和表中的实验结果一致。

至此，代码复写相关的介绍结束。相关代码可前往如下链接进行下载：
https://github.com/xiao-dun-shou/Master-Course-Homework/tree/main/Fair%20Assignment%20Problem

4 可能的改进

本算法中作者提出的通过均值来寻找局部最优均值的算法，非常的巧妙，给出了一种应对 NP 难问题的求解思路。

① 关于算法本身的可能改进

我认为作者在算法第四步——确定新的负荷均值，还有优化的空间，我自己在实验过程中发现作者给出的算法会导致c_{av}陷入死循环。即使是从全文来看，作者在这里的程序设置似乎也缺乏理论基础，几乎没有相关文字的描述或者数学理论的证明。

下图 4.1 为我按照作者的算法思想，监控c_{av}变换的结果，横坐标是算法步骤 3 成功收敛的次数。可以发现c_{av}的值出现了循环。此现象会导致搜索空间不能完全覆盖，可能会有部分局部最优解将无法被搜索到。

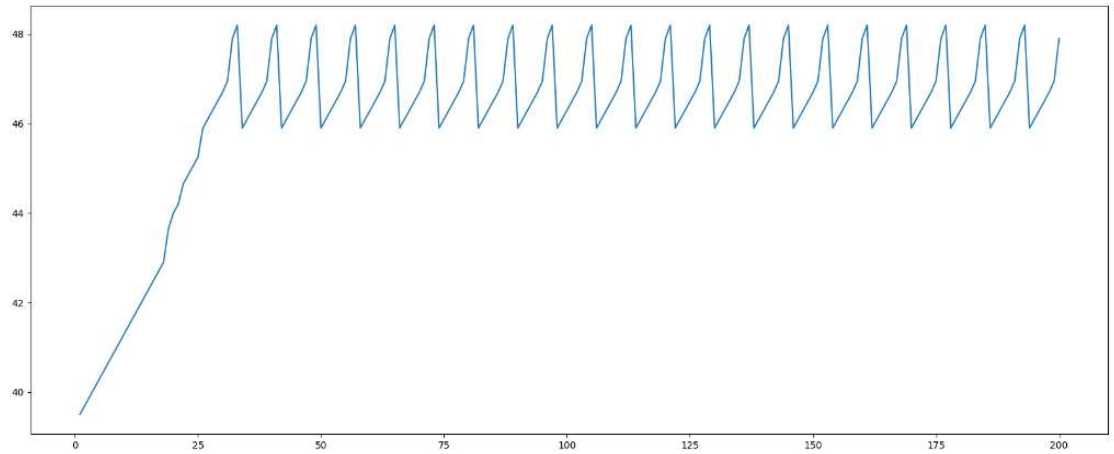

图 4.1 c_{av}变换结果

Figure 4.1 The transform result of c_{av}

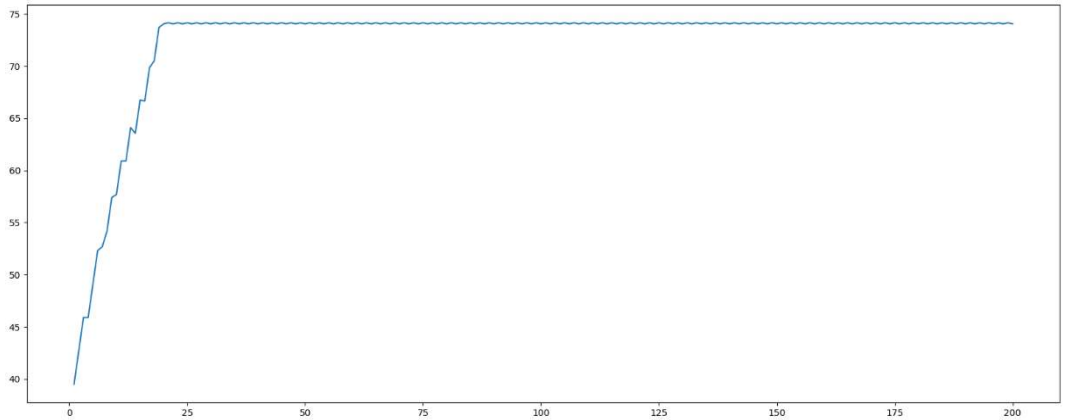

图 4.2 更改超参数后，c_{av}变换结果

Figure 4.2 The transform result of c_{av} after changes

通过修改超参数，比如增大步长，可以改良这种循环的情况，如图 4.2 所示。

我又尝试更改了代码逻辑，去除了代码中关于c_{fore}的定义（伪代码第 14 步），直接通过c_{av}与c_{av}*作为算法步骤 4 中的判断条件，又得到如下图 4.3 的结果。

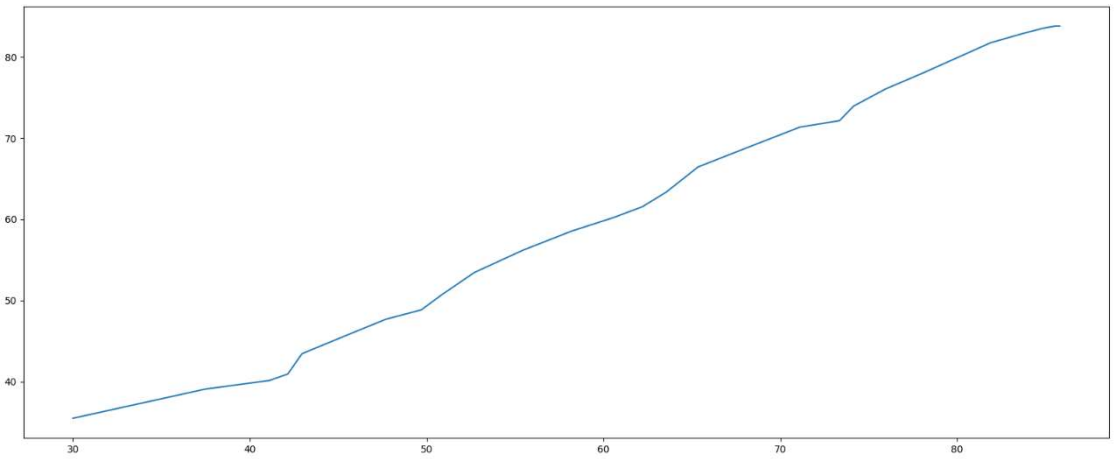

图 4.3 更改判断条件后，c_{av}变换结果

Figure 4.3 The transform result of c_{av} after changes

可见，此时的c_{av}的变化更加的平滑，范围也更广，更不会出现跳变的情况，但是当任务规模很大时，这可能会影响整个代码的运行效率。由于作者未公开原码，不知道作者在这里的实验中使用了怎样的小技巧（trick）。

② 关于算法应用场景的可能改进

另外，我认为这种算法虽然能保证每一个代理者都能公平的执行某项工作，这看似很公平，但是并不符合生活实际的应用。作为公司的任务分配者，在任务执行的公平程度和任务执行的总花费这两个维度上，应该综合考虑才更加合理。