基于二分图完全匹配的图案变换最优路径编排

摘要

学校要求一定数量的学生组成图形阵列来编排指定图案。人员图案编排活动参与人员数量多、场地广，且对每一位人员排布有严格要求，任何一名人员出现失误都可能导致整个图案编排失败，而一场完整的团编排活动往往不止编排单一图案，而是对多个图案进行连续编排，要完成这类活动除了对编排图案进行精密设计之外还需要灵活地调动编排人员，规划好每一次图案变换的人员运动路径。当前活动需要相同数量的多名学生连续编排多张图像，为了提高图案编排效率，缩短图案变换时多人员运动所需的时间，同时以更合理的路径规划人员运动轨迹，严防编排过程中发生碰撞跌倒等安全事故，十分有必要为每一次图案编排的所有参与人员规划设计出最优路径。

对于问题一，要让 15 位参与人员编排出数字一到数字十图案的最优路径，需要确定图案点位的具体坐标，为了突出人员的坐标显示位置，我们建立 15×15 的**平面网格直角坐标系**来刻画所有数字图案的点位坐标。在编排图案变换过程的最优路径中，使用暴力枚举的方法列出所有人员运动的可能状态可以求出最优解，但穷举法计算复杂度过高，在实际生活中无法使用。由于编排前所有人员的站位和编排后所有人员的占位可以看作是所有参与人员**两种不同的存在状态**，故可以在每一次的图案编排中，将前后编排图案的坐标点位抽象为二**分图结构**，这样可以将图案编排的问题视作为将原图像的所有人员分配到新图像中的坐标当中去，且恰好能够将所有人员无重复地分配完成，即二**分图完全匹配问题**。为了解决此问题，我们使用与**贪心算法**相结合的**回溯匹配方法**——**Kuhn-Munkres 算法**来对二分图中的所有结点进行分配，即对所有人员进行最优路径的编排。建立模型之后，我们解得了所有参与人员在数字一到数字十的九次图像变换中的运动轨迹坐标，编排出每张图案不同编号人员的站位情况并进行可视化分析，解得所有人员移动的总路程为 **266.9** 个单位长度，平均每位参与人员在每次图案的变换中只需移动 **1.97** 个单位长度。

对于问题二，使用 20 名参与人员编排正方形、右箭头、十字架三个图案。采用与问题一类似的平面网格直角坐标系来刻画参与人员的编排图案，同样采用 **Kuhn-Munkres 算法**来编排每一次图案变换的人员最优移动路径。我们最终得到了三次图案变换所有参与人员的运动轨迹，编排出每张图案所有编号人员的站位情况并进行可视化分析，结果显示这 20 名人员在三次的图案变换过程中移动的总路径长度为 **173.3**，由于我们在新设定的三个图案编排过程中缺乏精心的思考与设计，平均每位参与人员在每一次的图案变换中移动 **2.8** 个单位长度。

关键字：**最优路径　二分图　完全分配　Kuhn-Munkres**

目录

一、问题重述 **3**

1.1 问题背景 3

1.2 问题描述 3

二、问题分析 **3**

三、模型假设 **3**

四、符号说明 **4**

五、模型的建立与求解 **4**

5.1 问题一模型的建立与求解 4

5.1.1 基于数字图案建立平面网格直角坐标系 4

5.1.2 二分图结构 4

5.1.3 Kuhn-Munkres 算法编排数字图案变换最优路径 5

5.1.4 最优路径编排结果 11

5.2 问题二模型的建立与求解 13

5.2.1 图案构造 13

5.2.2 Kuhn-Munkres 算法编排自定义图案最优路径 14

六、模型评价 **18**

6.1 模型优点 18

6.2 模型缺点 18

参考文献 **18**

附录 A　源代码 **19**

一、 问题重述

1.1 问题背景

学校组织若干个学生在操场进行数字图案变形游戏。首先固定若干个学生排成一排，组成数字“1”的图案，然后学生在操场上运动变换为数字“2”的图案，再由“2”变换为“3”，以此类推，一直变换到数字“10”为止。

1.2 问题描述

问题一以 15 个学生为例，建立数学模型计算出编排路径，给出编排过程中的最优路径，并列出相应的人员坐标。

问题二利用上问所建立的模型，任意确定三个图案进行模型求解，包括计算出编排路径，给出最优路径，并列出相应的人员坐标。

二、 问题分析

对于问题一，我们首先确定好数字“1”到数字“10”的图案坐标。由于 15 个参与人员在每一个图案当中都要有自己的坐标站位，即为变换前图案中的 15 个参与人员以最小的路径长度之和让他们分别运动到新图案的 15 个坐标中。因此可以将图案变换问题视作为**二分图完全匹配问题**，将运动前后的图案都抽象为**图结构**，然后使用 **Kuhn-Munkres 算法**编排出参与人员运动的最优路径。具体来说，先把所有要变换的数字图案刻画在坐标轴上，然后将变换前图案的 15 个坐标点抽象为一个图结构，变换后图案的 15 个坐标点抽象为另外一个图结构，构成一个二分图，最后再以各坐标点之间的直线距离来连接这两个图，构成这个二分图的边结构，将最优路径编排的问题转化为最小边权值的完全分配问题，使用 Kuhn-Munkres 算法进行求解。

对于问题二，首先确定好三个需要输入到模型中进行计算的图案，在坐标轴上进行刻画并标记好坐标点。仿照问题一，使用二分图完全匹配的 Kuhn-Munkres 算法编排出最优路径，并给出相应每个人员的坐标。

三、 模型假设

- 假设在图案的过程中所有参与人员运动速度相同，不存在个体运动状况的差异。
- 假设参与人员均能直线到达指定点位，在移动过程中无碰撞、跌倒等意外情况发生。
- 在本文建立的平面直角坐标系中，假设每个参与人员站位为一个方格，且周围参与人员站位只限于与其相邻的方格。
- 将坐标轴上的直线距离视为参与人员需要移动的真实距离。

四、 符号说明

符号	符号解释	符号单位
d_{ij}	i 点与 j 点之间的直线距离	/
G_i	数字 i 图案中各坐标点抽象出的图结构	/
$G_i \to G_j$	G_i 变换为 G_j	/
$G_i \to\to G_j$	G_i 变换为 G_j 的中间过程所有图像	/
w_{ij}	图结构中两结点边的权重	/
L_i	结点 i 的可行结点编号	/

五、 模型的建立与求解

5.1 问题一模型的建立与求解

5.1.1 基于数字图案建立平面网格直角坐标系

参与人员需要以数字 “1” 为第一个图案，依次完成 “2、3、4、5、6、7、8、9、10” 的图像变换。以 15 个参与人员为例，我们建立如图 1 所示的大小为 15×15 的平面网格直角坐标系，横纵坐标范围均为 1 到 15 且取整数。有颜色的方格即代表参与人员的站位。

基于图 1 建立的平面网格直角坐标系，所有的数字图案排布如图 2 所示。其中所有图案每一个参与人员占位点的坐标见支撑材料。

5.1.2 二分图结构

二分图是一种特殊的图结构，它的结点可以分为两个互不相交的顶点集合，记作 U 与 V，集合 U 内的结点与集合 V 内的结点在其所属结点内无互相连接的边，而图中的每条边的两个端点分别属于集合 U 与 V。

在参与人员站位排布的图案变换当中，由于参与人员是要将某一张数字图案的造型改变成另一张数字图案，那么参与人员要运动的路径则只与在当前图案中所站的坐标和新图案中的坐标有关，与当前图案中的其他坐标没有关系，因此我们可以将平面网格直角坐标系上的这种变换关系抽象成一种二分图结构，如图 3 所示。

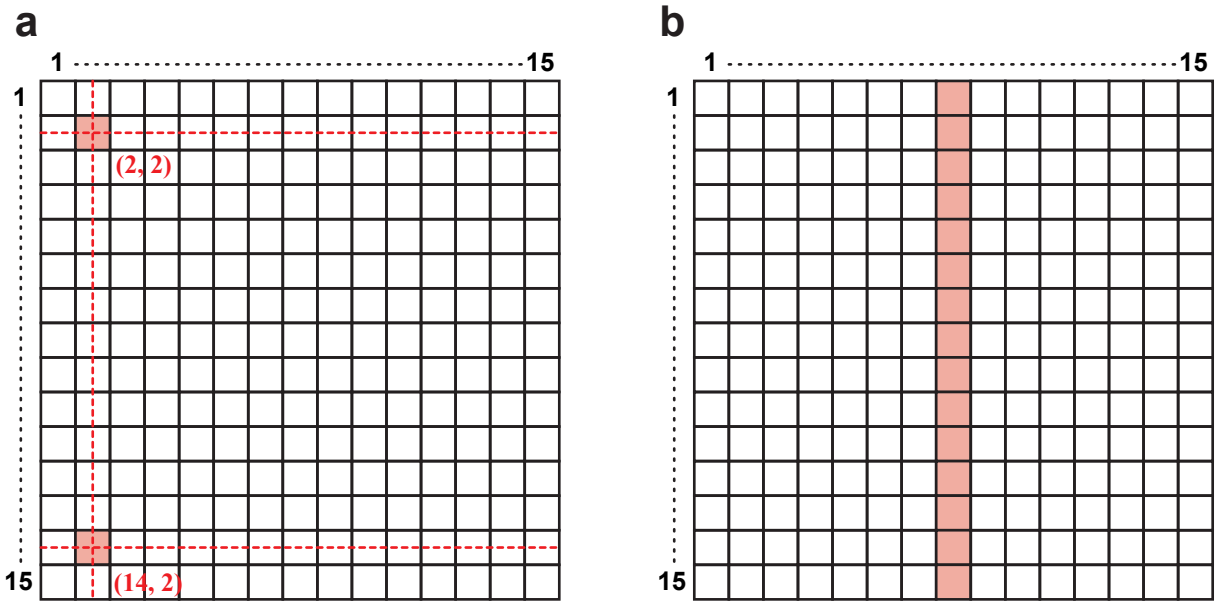

图 1　平面网格直角坐标系。（a）本题建立的空白坐标系，标注有坐标读取方法。（b）参与人员摆成数字“1”图案的初始队列。

将所有数字图案抽象成图结构后，结合题意可知需完成如式 1 所示的图形转换。

$$G_1 \to G_2 \to G_3 \to G_4 \to G_5 \to G_6 \to G_7 \to G_8 \to G_9 \to G_{10} \tag{1}$$

由式 1 可知一共要进行 9 次数字图像变换任务，将变换前图案坐标的集合视作二分图中的 U，变换后图案坐标的集合视作 V，那么对于每一次变换而言，都是给 U 中的每一个结点分配一个 V 中的结点，直到 U 和 V 中的结点都被分配完毕。

因为图像变换的过程即为人员从当前坐标点运动到下一图形中的某一坐标点的过程，也就是为每一个当前图案的结点坐标分配一个将要变换的图案的坐标，并确保没有任何结点重复与遗漏，这样才能完成图案的变换。

5.1.3 Kuhn-Munkres 算法编排数字图案变换最优路径

为 U 中的每一个结点都分配一个 V 中的结点，也就是确定好原图案中每一个人员将要运动到下一个图案的坐标，那就能够编排出所有人员的运动路径。因此本文关键在于如何以最短的全体人员异移动路径长度总和，也就是以最小的二分图中边的权重和完成图中结点的分配任务。

Kuhn-Munkres 算法能够解决二分图最优权重匹配问题，它的基本思想是根据理想最优值给二分图宗的结点进行编号，通过不断调整结点的标号，寻找增广路径并扩展匹配，最终得到二分图的最优权重匹配。算法的关键步骤包括初始化标号矩阵、增广路径搜索和标号调整。在增广路径搜索中，通过遍历邻接顶点和匹配关系的判断，找到增广路径并进行匹配或标号调整。通过迭代执行这些步骤，直到无法找到增广路径为止，最终获得最优权重匹配。

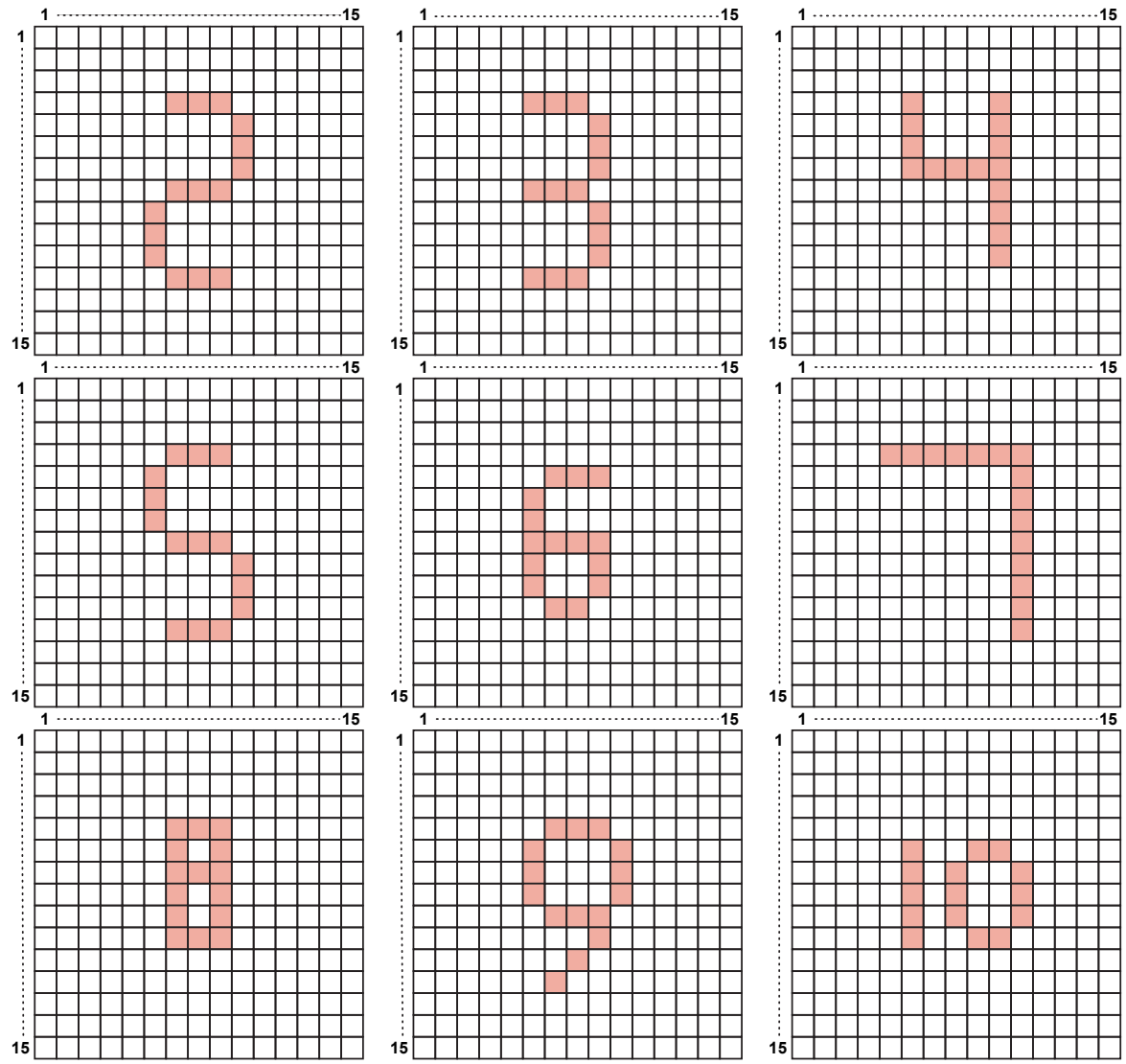

图 2　平面网格坐标直角系中“2-9”的数字图案排布

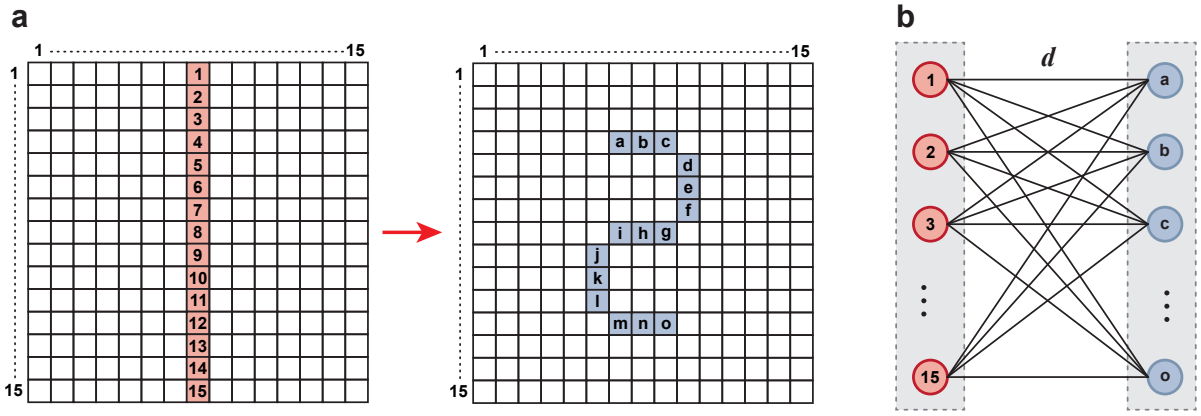

图 3　将平面网格直角坐标系上的图形抽象为二分图结构。（a）数字图案的转换。（b）二分图结构，其中左区域内的结点为原图案的坐标点，右区域内的结点为变换后图案的坐标点，边的权重为两点间的直线距离。

Step1：初始化二分图与邻接矩阵 假设数量为 N 的人员进行图案编排，将当前图案所有结点坐标与下一图案所有结点坐标分别化为如式 (2) 所示的集合 U 与集合 V。U 存放当前图案结点坐标，V 存放下一图案结点坐标。

$$\begin{cases} U &= \{U_1, U_2, \ldots, U_N\} \\ V &= \{V_1, V_2, \ldots, V_N\} \end{cases} \tag{2}$$

由于该二分图是一个由 U 向 V 分配的问题，因此将其视作有向图。创建一个 $N \times N$ 大小的矩阵，矩阵中存储 U 中元素到 V 中元素的距离，如式 (3) 所示。

$$A = \begin{bmatrix} d_{U_1V_1} & d_{U_1V_2} & \ldots & d_{U_1V_N} \\ d_{U_2V_1} & d_{U_2V_2} & \ldots & d_{U_2V_N} \\ \vdots & \vdots & \ddots & \vdots \\ d_{U_NV_1} & d_{U_NV_2} & \ldots & d_{U_NV_N} \end{bmatrix} \tag{3}$$

再创建一个空的匹配边集合 E_{match}，用来存储已经匹配好的边，同时将 U 与 V 中的所有结点标记为未访问，并且将所有未访问的结点放入到集合 U_{nov} 与 V_{nov} 中。

Step2：设置初始顶标 初始顶标即为每一个结点分配的最佳情况，而本题所求为二分图的最小权重和分配，故 U 中顶标就是每一名人员到达 N 的不同坐标中的最短运动距离，如式 (4) 所示。

$$L_{U_i} = min(d_{U_iV_j}) \quad i, j \in [1, n] \tag{4}$$

V 所有元素的顶标初始化为 0，如式 5所示。

$$L_{V_i} = 0 \quad j \in [1, n] \tag{5}$$

Step3：寻找最佳路径 遍历 U 中每一个顶点，每一轮循环中将所有顶点均设置为未访问，再将 U_i 设置为已访问，接着对 U_i 遍历 V_{nov}，在 V_{nov} 找到符合式 (6) 的 V_j。由于初始顶标是按照最短路径设置的，因此倘若两点间距离等于两点顶标值之和，则说明此时寻找到的路径为最短路径。

$$A(i, j) = L_{U_i} + L_{V_j} \tag{6}$$

为方便起见，对于 $G_1 \to G_2$，给出如表 1 所示 G_1 三个坐标点到 G_2 每一个坐标点的距离。因此在不考虑全局最优情况下，这三点的最短路径如式 7 所示。完整的顶标初始化数据见支撑材料，正文仅给出表 1 中的三点。

$$\begin{cases} G_1(1) &\to G_2(b) \quad d = 3 \\ G_1(8) &\to G_2(h) \quad d = 0 \\ G_1(15) &\to G_2(n) \quad d = 3 \end{cases} \tag{7}$$

表 1 G_1 **中点位编号为“3”、“4”、“8”的结点到** G_2 **中各点的距离。加粗部分为该结点直线距离最短的点。**

标号	距离														
	d_{i1}	d_{i2}	d_{i3}	d_{i4}	d_{i5}	d_{i6}	d_{i7}	d_{i8}	d_{i9}	d_{i10}	d_{i11}	d_{i12}	d_{i13}	d_{i14}	d_{i15}
3	1.4	**1.0**	1.4	2.8	3.6	4.5	5.1	5.0	5.1	6.3	7.3	8.2	9.1	9.0	9.1
8	4.1	4.0	4.1	3.6	2.8	2.2	1.0	**0.0**	1.0	2.2	2.8	3.6	4.1	4.0	4.1
15	11.0	11.0	11.0	10.2	9.2	8.2	7.1	7.0	7.1	6.3	5.4	4.5	3.2	**3.0**	3.2

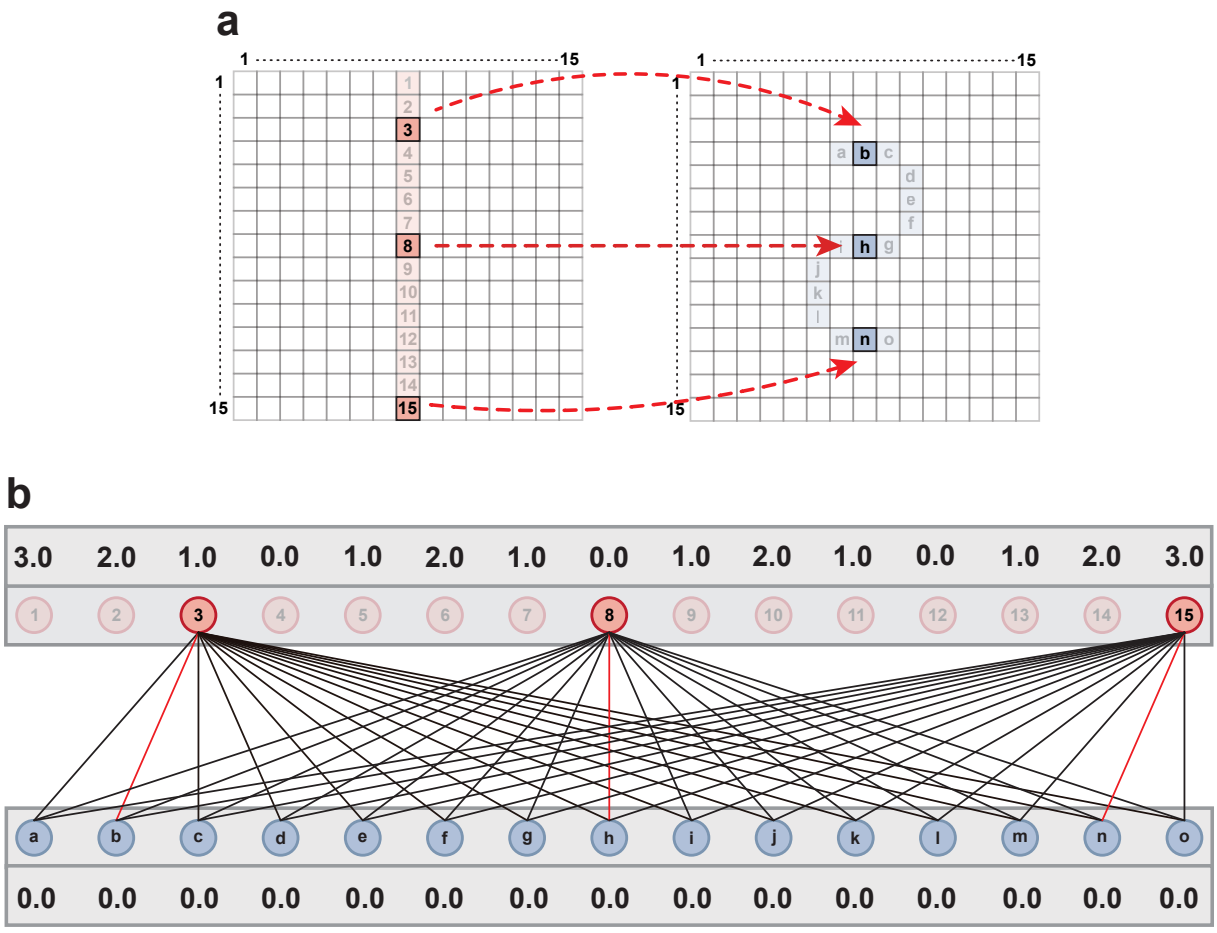

图 4 **寻找** G_1 **与** G_2 **之间的最短路径。(a) 在初始化不考虑全局最优的阶段，人员以最短路径运动的目的坐标。(b) 确定最短路径。**

以 15 个结点中的 3 点为示例，根据顶标值确定最短路径的具体过程如图 4 所示。

对每一个 U_i，如果寻找到满足式 (6) 的 V_j，则将 V_j 标记为已访问，若 V_j 未被 U 中其他结点所匹配，则将 V_j 标记为已匹配，并将该条边纳入到集合 E_{match} 中，如式 8 所

示。

$$E_{match} \longleftarrow (U_i, V_j) \tag{8}$$

Step4：更新顶标值 在初始化阶段，顶标值都是在不考虑全局最优结果下的当前最优值，G_1 中所有点都能按照各自初始化的最优结果运动到 G_2 当中是一种理想情况。实际上，G_1 中可能会有多个点在 G_2 中的最优目标点是同一个，如图 5所示。

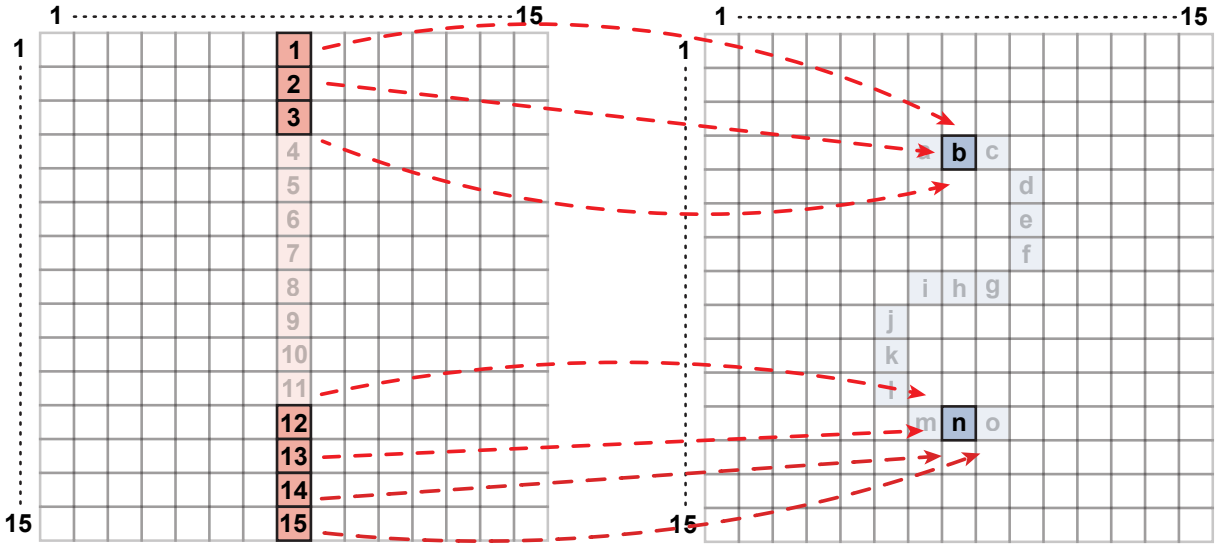

图 5　G_1 中多个点在 G_2 中的最优目标点是同一个

而在二分图完全分配问题中，G_1 中的点必须全部分配到 G_2 中且在此图案排布问题中必须一一对应，多个 G_1 中的点不能同时到达 G_2 中的同一个点。此时需要对 G_1 中在分配最优路径上有冲突的结点进行顶标志的更新。

以 U_1、U_2 到 V 中各点距离如表 2 所示。

表 2　$G_1 \to G_2$ 的变化过程中，U_1、U_2 到 V 中各点距离

标号	距离														
	d_{i1}	d_{i2}	d_{i3}	d_{i4}	d_{i5}	d_{i6}	d_{i7}	d_{i8}	d_{i9}	d_{i10}	d_{i11}	d_{i12}	d_{i13}	d_{i14}	d_{i15}
1	3.2	**3.0**	3.2	4.5	5.4	6.3	7.1	7.0	7.1	8.2	9.2	10.2	11.0	11.0	11.0
2	2.2	**2.0**	2.2	3.6	4.5	5.4	6.1	6.0	6.1	7.3	8.2	9.2	10.0	10.0	10.0

如图 6 所示，我们首先确定好所有边的顶标，从 U_1 开始对 U 中所有点进行遍历查找最优路径，也即路径长度等于 U 中结点顶标值的路径。经过查找发现 U_1 的最优到达目标点为 V_b。

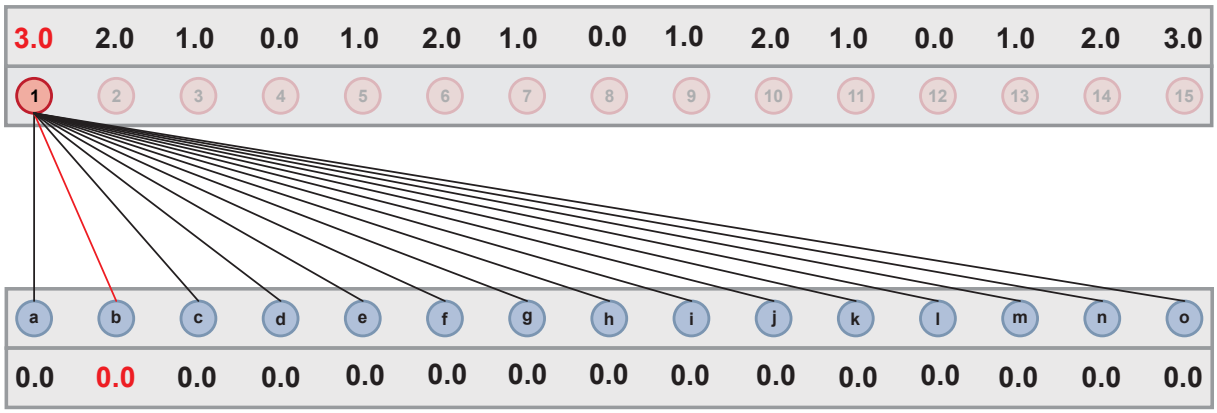

图 6　U_1 根据初始化顶标找到最短路径

但查找到 U_2 时发现，如图 7 所示，U_2 在 G_2 中的最优到达目标点也为 V_b。无法满足 U_1 与 U_2 两个点同时到达理想的指定位置，那么只能兼顾全局最优结果，改变其中的某条路径，使 G_1 中的结点能完全地排布到 G_2 中去。

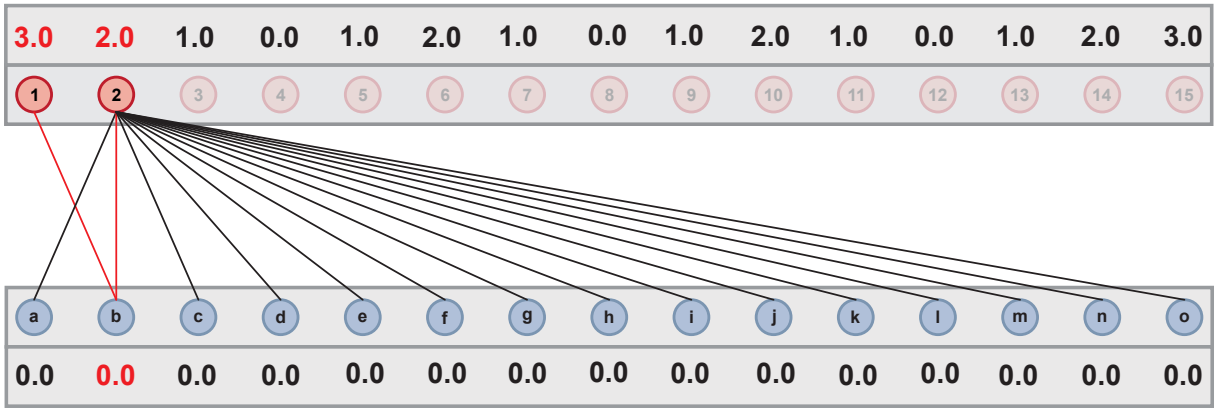

图 7　U_1 找到最短路径后，U_2 根据初始化顶标找到最短路径

涉及到路径冲突的是 U_1、U_2、V_b，由于需要求取的是最短的路径长度，而理想状态的最优结果在顶标初始化的过程中已经设置好，我们只能增加 U 中的顶标值，降低路径长度的要求，选择比顶标初始值更高的路径长度才有可能完成匹配任务。

首先设置集合 U_{conf} 与 V_{conf} 存储 U 与 V 中涉及路径冲突的结点，然后对涉及冲突的 U 中结点进行如式 9 所示的操作。

$$L_{U_i} = L_{U_i} + max(h, 0.1) \quad U_i \in U_{conf} \tag{9}$$

对涉及冲突的 V 中结点进行如式 (10) 所示的操作。这是为了让一条边的两结点之和始终为理想最小路径长度，如式 (11) 所示。

$$L_{V_i} = L_{V_i} - max(h, 0.1) \quad V_i \in V_{conf} \tag{10}$$

$$L_{U_i} + L_{V_j} = d_{ij} \quad (U_i, V_j) \in E_{match} \tag{11}$$

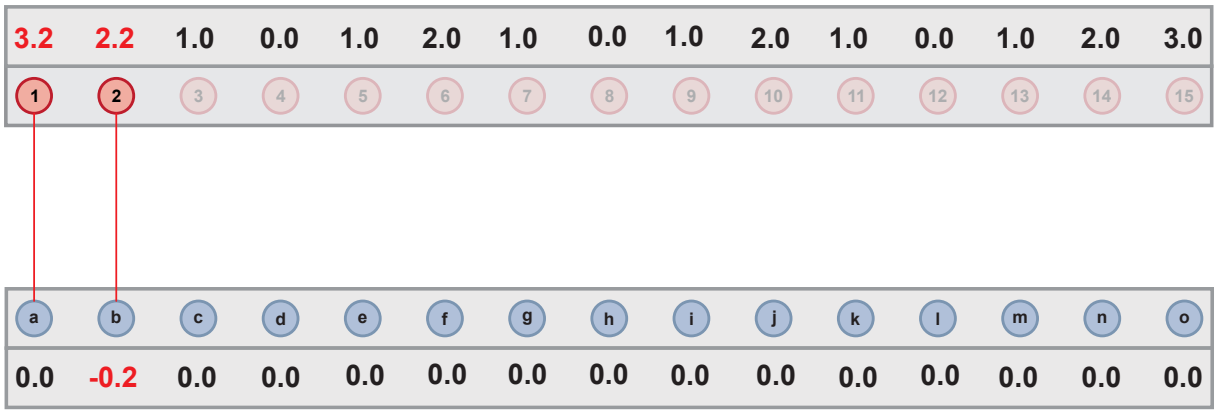

图 8　更新顶标后，U_1 匹配到新的路径。

其中 h 为超参数，可以自行设定，由于本文计算结果均保留 1 位小数，故设 $h \geq 1$。

设 $h = 0.2$，此时我们根据更新后的顶标值重复 **Step3**，直到为 U_1 或 U_2 找到了新的分配路径为止。如图所示，顶标更新后 $L_{U_1} = 3.2$，根据表 2 的数据，它与 V_a 构成最短路径，冲突解决后，(U_2, V_b) 这条边不需要再更新。

Step5: 输出最终分配结果　当 U 中所有结点被分配完毕，无法找到未分配结点时，Kuhn-Munkres 算法流程结束，返回 E_{match} 为最小权重和分配结果，也即每一个人员的运动路径。最终确定的分配结果与顶标值如图 9 所示。

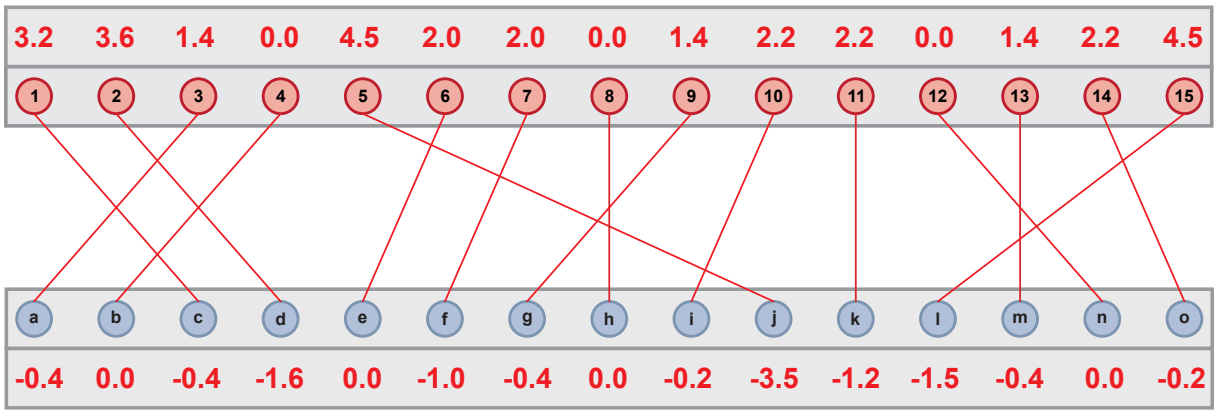

图 9　$G_1 \to G_2$ 所有人员坐标以全局最短路径分配

5.1.4 最优路径编排结果

分别对 $G2 \to G_3$、$G_3 \to G_4$、$G_4 \to G_5$、$G_5 \to G_6$、$G_6 \to G_7$、$G_7 \to G_8$、$G_8 \to G_9$、$G_9 \to G_{10}$ 重复 Kuhn-Munkres 算法，编排出所有图案变化的最短路径，所有人员的运动路径如表 3。

同时，我们对所有 1-15 编号人员的运动路径进行可视化，G_1 到 G_{10} 的所有图案人员排布顺序如图 10 所示，同时展现了 1 号人员、8 号人员、15 号人员的运动轨迹。

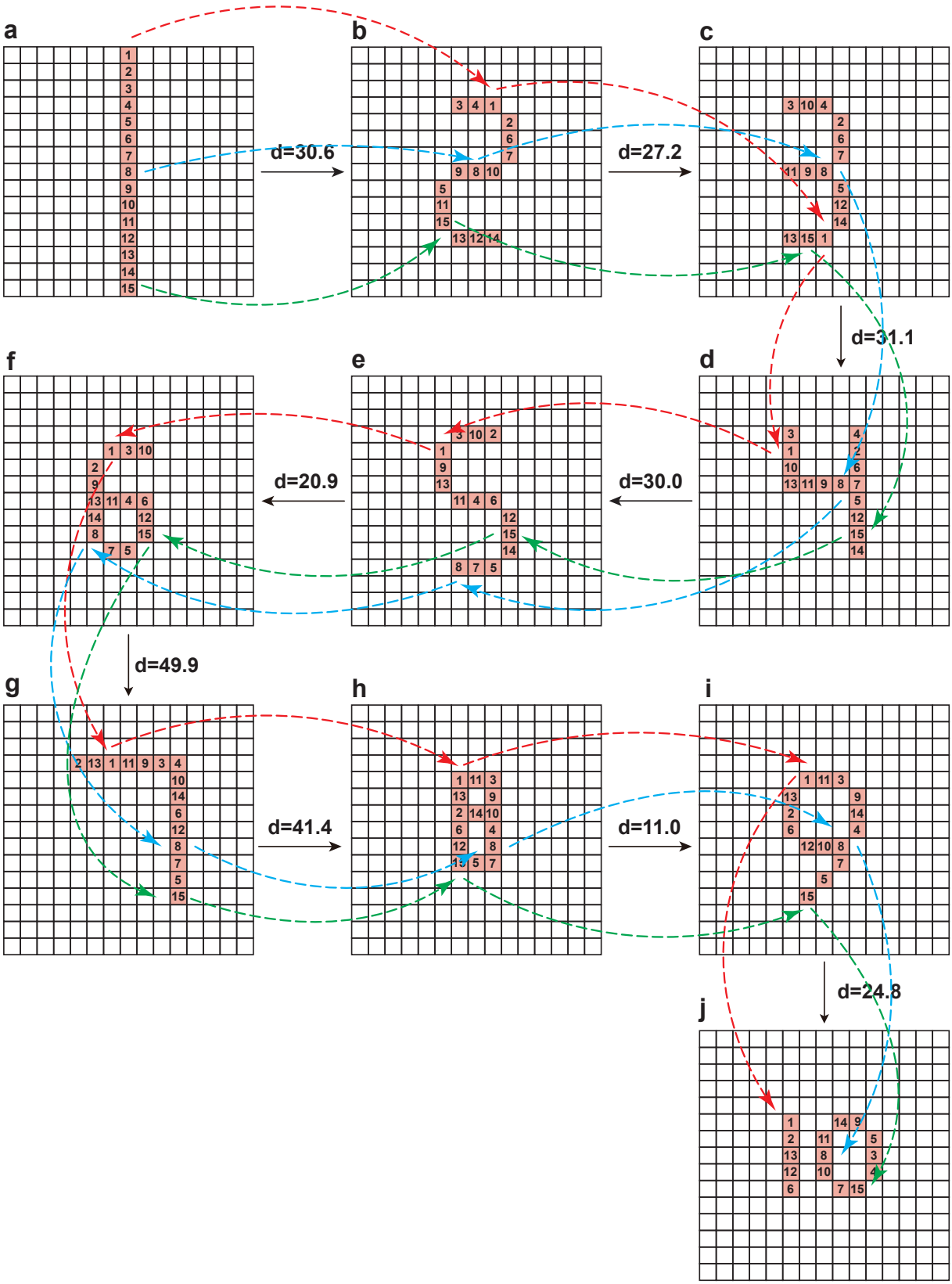

图 10　所有数字图案 1-15 编号人员排布结果，其中展示了 1 号人员、8 号人员、15 号人员的运动轨迹。a、b、c、d、e、f、g、h、i、j 分别表示 G_1、G_2、G_3、G_4、G_5、G_6、G_7、G_8、G_9、G_{10} 的人员排布结果。

表 3　编号 1-15 号人员由 G_1 运动到 G_{10} 的全部运动轨迹

编号	运动路径
1	$(1,8) \to (4,9) \to (12,8) \to (5,6) \to (5,6) \to (5,7) \to (4,7) \to (5,7) \to (5,7) \to (6,6)$
2	$(2,8) \to (5,10) \to (5,9) \to (5,10) \to (4,9) \to (6,6) \to (4,5) \to (7,7) \to (7,6) \to (7,6)$
3	$(3,8) \to (4,7) \to (4,6) \to (4,6) \to (4,7) \to (5,8) \to (4,10) \to (5,9) \to (5,9) \to (8,11)$
4	$(4,8) \to (4,8) \to (4,8) \to (4,10) \to (8,8) \to (8,8) \to (4,11) \to (8,9) \to (8,10) \to (9,11)$
5	$(5,8) \to (9,6) \to (9,9) \to (8,10) \to (12,9) \to (11,8) \to (11,11) \to (10,8) \to (11,8) \to (7,11)$
6	$(6,8) \to (6,10) \to (6,9) \to (6,10) \to (8,9) \to (8,9) \to (7,11) \to (8,7) \to (8,6) \to (10,6)$
7	$(7,8) \to (7,10) \to (7,9) \to (7,10) \to (12,8) \to (11,7) \to (10,11) \to (10,9) \to (10,9) \to (10,9)$
8	$(8,8) \to (8,8) \to (8,8) \to (7,9) \to (12,7) \to (10,6) \to (9,11) \to (9,9) \to (9,9) \to (8,8)$
9	$(9,8) \to (8,7) \to (8,7) \to (7,8) \to (6,6) \to (7,6) \to (4,9) \to (6,9) \to (6,10) \to (6,10)$
10	$(10,8) \to (8,9) \to (4,7) \to (6,6) \to (4,8) \to (5,9) \to (5,11) \to (7,8) \to (9,8) \to (9,8)$
11	$(11,8) \to (10,6) \to (8,6) \to (7,7) \to (8,7) \to (8,7) \to (4,8) \to (5,8) \to (5,8) \to (7,8)$
12	$(12,8) \to (12,8) \to (10,9) \to (9,10) \to (9,10) \to (9,9) \to (8,11) \to (9,7) \to (9,7) \to (9,6)$
13	$(13,8) \to (12,7) \to (12,6) \to (7,6) \to (7,6) \to (8,6) \to (4,6) \to (6,7) \to (6,6) \to (8,6)$
14	$(14,8) \to (12,9) \to (11,9) \to (11,10) \to (11,10) \to (9,6) \to (6,11) \to (7,9) \to (7,10) \to (6,9)$
15	$(15,8) \to (11,6) \to (12,7) \to (10,10) \to (10,10) \to (10,9) \to (12,11) \to (10,7) \to (12,7) \to (10,10)$

$G_1 \to\to G_{10}$ 变化过程中，每一次图案变化中，所有人员的运动路径长度之和如表 4所示。其中使用 $G_{i,j}$ 作为 $G_i \to G_j$ 的简写。

表 4　$G_1 \to\to G_{10}$ 每一次图案变换的运动路径长度

图案	$G_{1,2}$	$G_{2,3}$	$G_{3,4}$	$G_{4,5}$	$G_{5,6}$	$G_{6,7}$	$G_{7,8}$	$G_{8,9}$	$G_{9,10}$	总和
路径长度	30.6	27.2	31.1	30.0	20.9	49.9	41.4	11.0	24.8	**266.9**

同时通过式 (12) 的计算可知平均每个人员在每次图像变换过程中的平均移动路径长度为 **1.97**。

$$E_{person} = \frac{Path_{total}}{Num_{sport} \times N} \tag{12}$$

其中 E_{person} 表示人员在每次图案变换中的移动路径长度的期望值，$Path_{total}$ 表示图案变化的总次数，N 是参与图案变化的人员总数。

5.2 问题二模型的建立与求解

5.2.1 图案构造

我们以 20 名编号人员构造以下四个图案进行变换，如图 11 所示。每个图案详细坐标位点见附件。其中起始排列为“1”字形的队列命名为 G_a，其余三张图像命名为

G_b-G_d，需要完成如式 (13) 所示的图案变换。

$$G_a \rightarrow G_b \rightarrow G_c \rightarrow G_d \tag{13}$$

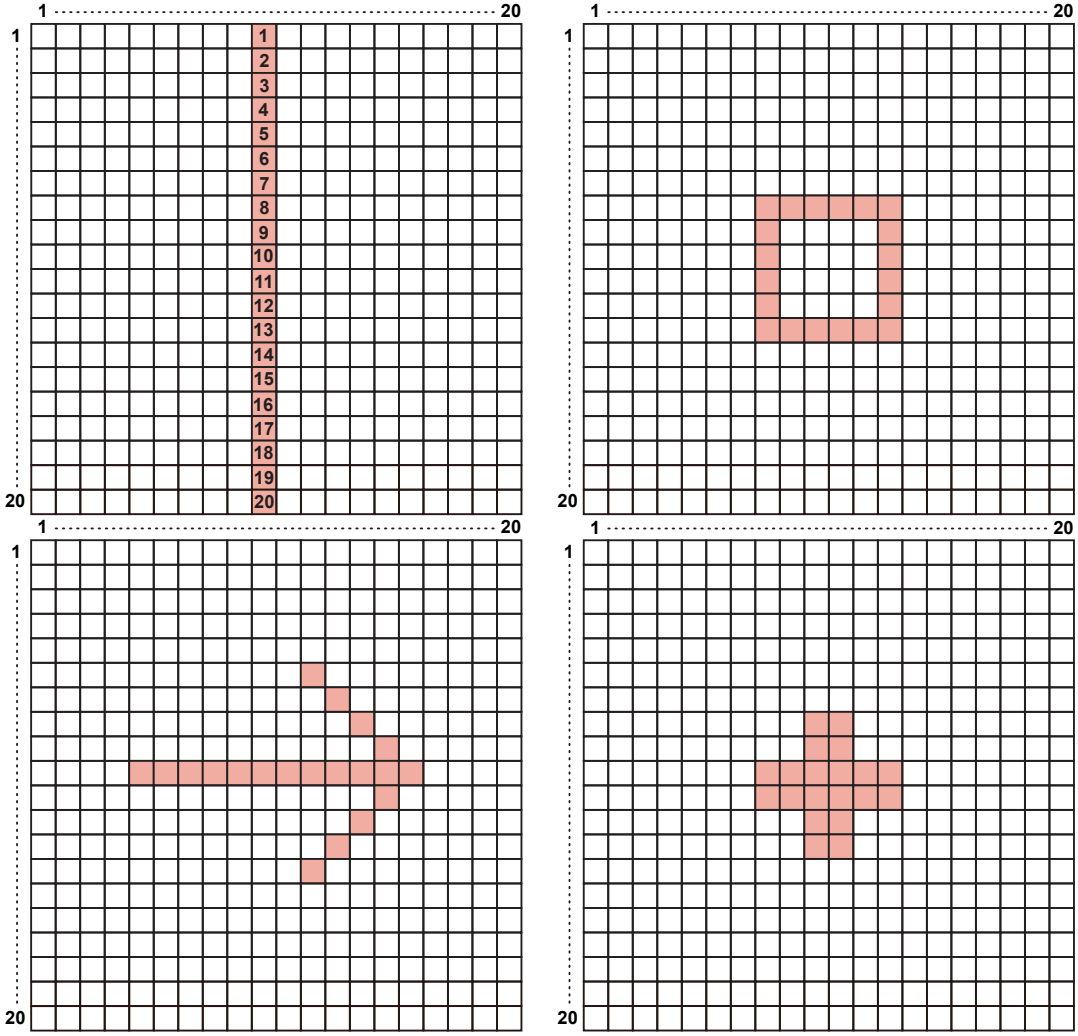

图 11　平面网格直角坐标系中 G_a-G_d 的数字图案排布

5.2.2 Kuhn-Munkres 算法编排自定义图案最优路径

在式 13 中需要完成三次图像变换，即 $G_a \rightarrow G_b$、$G_b \rightarrow G_c$、$G_c \rightarrow G_d$，那么需要分别对三次图像变换使用 Kuhn-Munkres 算法求解出最优路径。

求解步骤与问题二中的数字图案排布类似，构造好二分图结构后首先确定好图中结点的顶标值。$G_a \rightarrow G_b$、$G_b \rightarrow G_c$、$G_c \rightarrow G_d$ 的 U 中初始化顶标值如表 5 所示。其中 $G_{i,j}$ 表示 $G_i \rightarrow G_j$。

表 5　$G_a \rightarrow\rightarrow G_d$ 变换过程中的每一次待分配结点集合 U 中的 Kuhn-Munkres 算法初始顶标值。

图案	U_1	U_2	U_3	U_4	U_5	U_6	U_7	U_8	U_9	U_{10}	U_{11}	U_{12}	U_{13}	U_{14}	U_{15}	U_{16}	U_{17}	U_{18}	U_{19}	U_{20}
$G_{a,b}$	7.0	6.0	5.0	4.0	3.0	2.0	1.0	0.0	1.0	2.0	2.0	1.0	0.0	1.0	2.0	3.0	4.0	5.0	6.0	7.0
$G_{b,c}$	2.0	2.0	2.0	2.0	1.4	1.0	1.0	0.0	1.0	2.0	3.0	3.0	2.2	1.4	1.0	0.0	1.0	0.0	1.0	0.0
$G_{c,d}$	3.0	2.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	2.0	3.0	2.2	2.2	2.2	2.2	2.0	1.4	2.0	2.2

由初始顶标进行 Kuhn-Munkres 算法二分图完全分配，针对 $G_a \rightarrow G_b$、$G_b \rightarrow G_c$、$G_c \rightarrow G_d$，分别有如图 12　13　14 所示的最终结果。

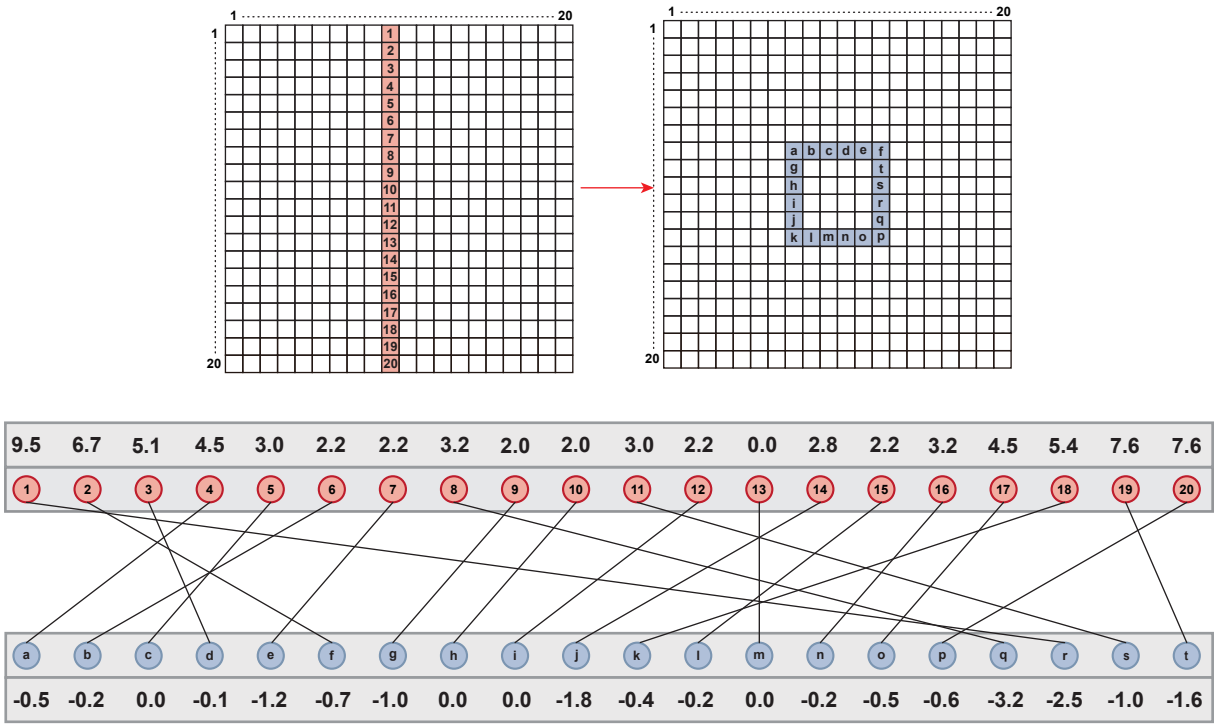

图 12　$G_a \rightarrow G_b$ Kuhn-Munkres 算法最终分配结果与二分图中各结点顶标值

图案变换过程中，以 Kuhn-Munkres 算法编排出的最优 1-20 编号人员运动路径的坐标轨迹如表 7 所示，每一轮图像变换的运动路劲长度如表 6 所示，由式 (12) 可得每名人员在每轮变换中平均移动路径长度为 **2.8**。

表 6　$G_a \rightarrow\rightarrow G_d$ 每一次图案变换的运动路径长度

图案	$G_a \rightarrow G_b$	$G_b \rightarrow G_c$	$G_c \rightarrow G_d$	总和
路径长度	78.9	47.4	47.0	**173.3**

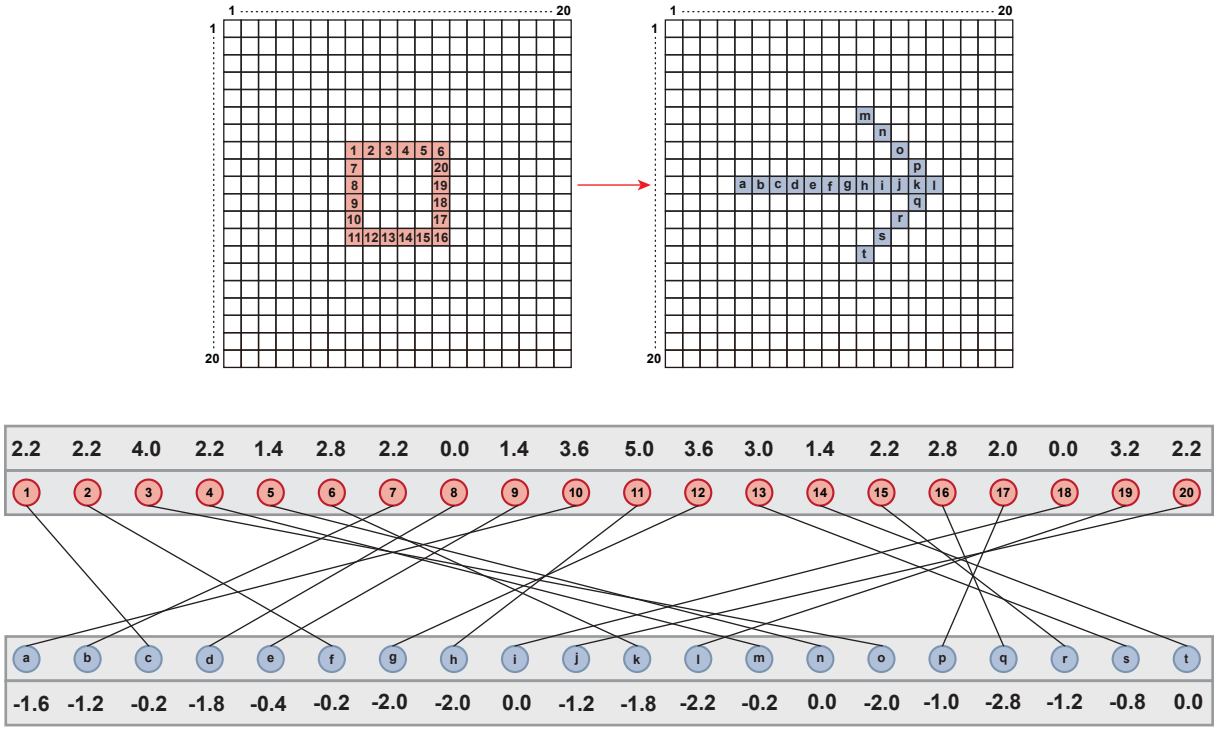

图 13 $G_b \to G_c$ **Kuhn-Munkres 算法最终分配结果与二分图中各结点顶标值**

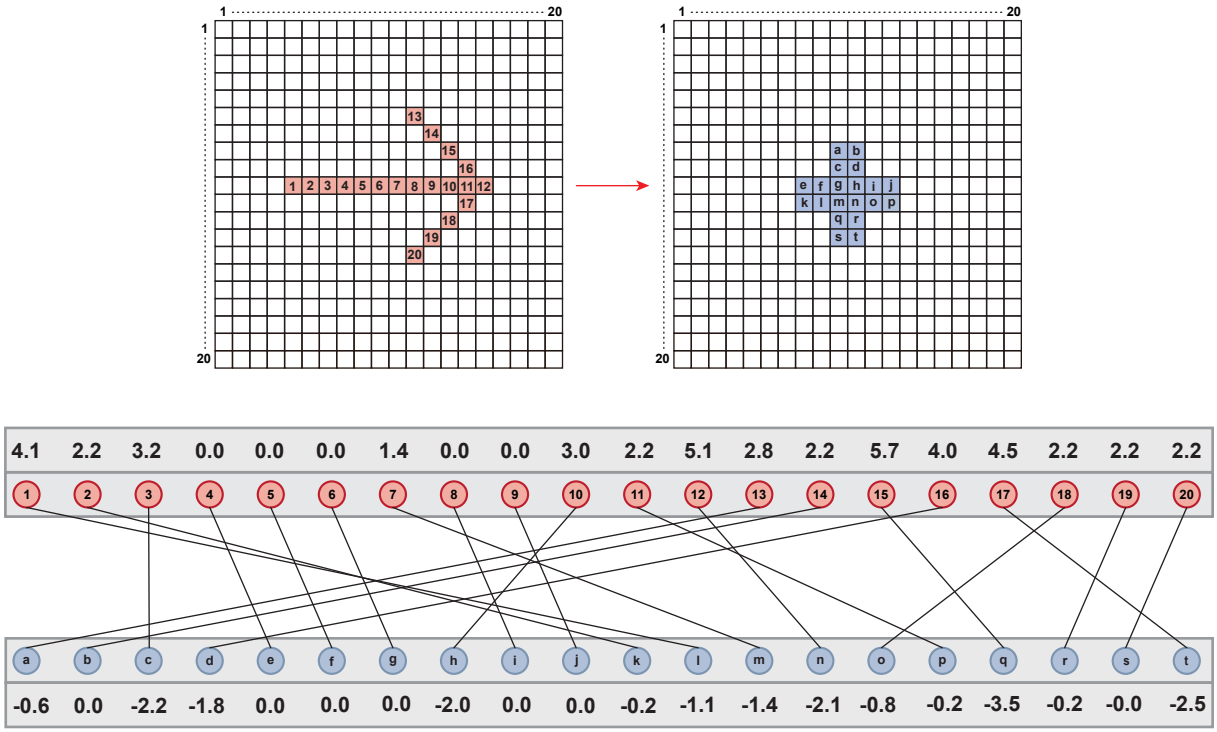

图 14 $G_c \to G_d$ **Kuhn-Munkres 算法最终分配结果与二分图中各结点顶标值**

表 7　编号 1-20 号人员由 G_a 运动到 G_d 的全部运动轨迹

编号	运动路径	编号	运动路径
1	$(1,10) \to (10,13) \to (10,13) \to (10,13)$	11	$(11,10) \to (11,13) \to (10,16) \to (11,11)$
2	$(2,10) \to (8,13) \to (10,15) \to (11,13)$	12	$(12,10) \to (11,8) \to (10,9) \to (10,9)$
3	$(3,10) \to (8,11) \to (6,12) \to (8,10)$	13	$(13,10) \to (13,10) \to (13,13) \to (12,11)$
4	$(4,10) \to (8,8) \to (10,7) \to (9,10)$	14	$(14,10) \to (12,8) \to (10,5) \to (11,9)$
5	$(5,10) \to (8,10) \to (8,14) \to (12,10)$	15	$(15,10) \to (13,9) \to (10,11) \to (11,10)$
6	$(6,10) \to (8,9) \to (10,10) \to (10,10)$	16	$(16,10) \to (13,11) \to (14,12) \to (13,10)$
7	$(7,10) \to (8,12) \to (7,13) \to (8,11)$	17	$(17,10) \to (13,12) \to (12,14) \to (11,12)$
8	$(8,10) \to (9,13) \to (9,15) \to (9,11)$	18	$(18,10) \to (13,8) \to (10,12) \to (10,12)$
9	$(9,10) \to (9,8) \to (10,6) \to (11,8)$	19	$(19,10) \to (12,13) \to (10,14) \to (10,11)$
10	$(10,10) \to (10,8) \to (10,8) \to (10,8)$	20	$(20,10) \to (13,13) \to (11,15) \to (13,11)$

对编号为 1-20 号所有人员的运动轨迹进行可视化，$G_a \to\to G_d$ 变换过程中每张图案的人员占位如图 15 所示。其中展示了 1 号人员、10 号人员、20 号人员的运动轨迹。

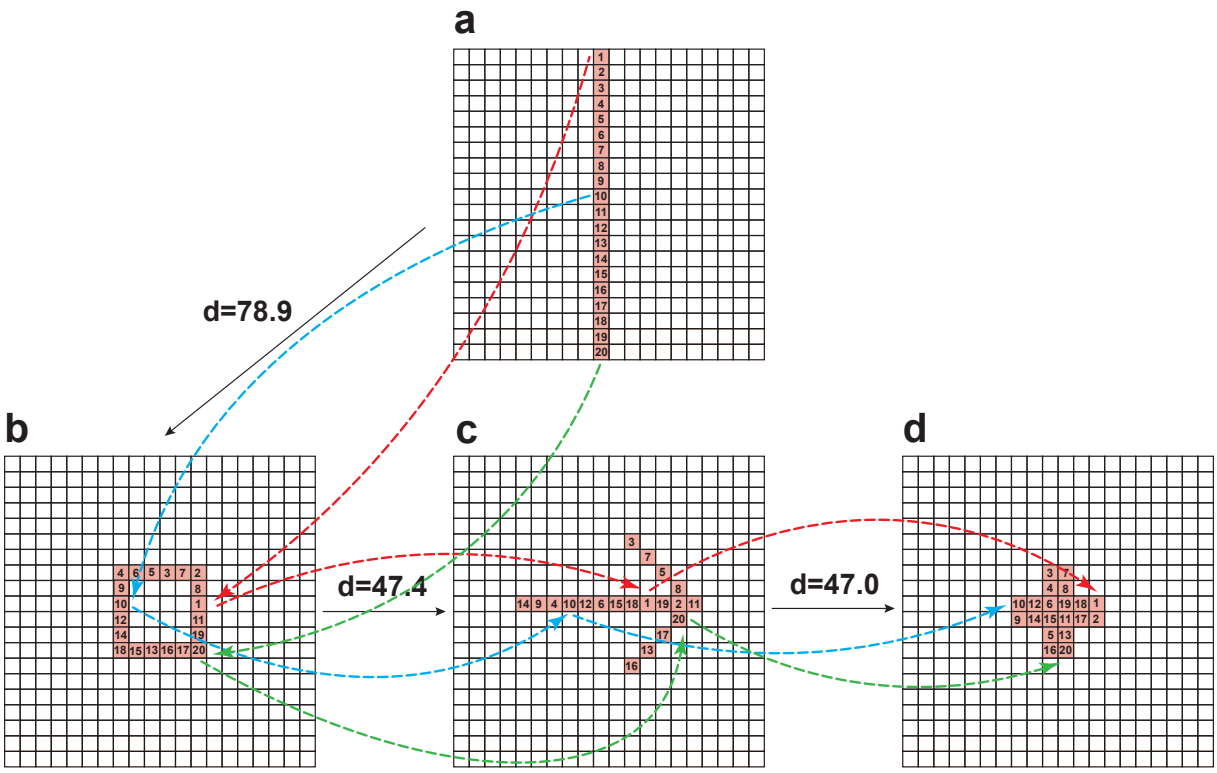

图 15　所有 $G_a \to\to G_d$ 图案 1-20 编号人员排布结果，其中展示了 1 号人员、10 号人员、20 号人员的运动轨迹。a、b、c、d 分别表示 G_a、G_b、G_c、G_d 的人员排布结果。

六、 模型评价

6.1 模型优点

- Kuhn-Munkres 算法能够找到给定二分图中的最佳匹配，即最佳权重的完美匹配，为图案编排问题提供具有确定性的解决方案，而不是给出近似解。
- 相比起暴力穷举的 $O(n!)$ 复杂度，Kuhn-Munkres 算法的复杂度仅为 $O(n^3)$，更加高效而节省计算资源，且此为最坏情况，实际计算中 Kuhn-Munkres 算法的复杂程度要小得多。
- 我们建立的模型不仅适用于二分图的最小权重匹配问题，还可以用于解决最大权重的完美匹配问题，也同时适用于带权图等多种数据结构。因此，我们建立的模型能够广泛地应用到许多实际场景，如任务分配、资源分配、指派问题等。

6.2 模型缺点

- 在编排最优路径时，我们只考虑从一个图案到下一个图案的变换过程，没有综合之后的所有图案变换过程进行考虑。
- Kuhn-Munkres 算法的性能高度依赖于输入的顺序，对于不同的输入顺序，算法的执行时间可能会有所不同。

参考文献

[1] Zhu H, Zhou M C. Efficient role transfer based on Kuhn–Munkres algorithm[J]. IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans, 2011, 42(2): 491-496.

[2] 彭波. 数据结构 [M]. 清华大学出版社有限公司, 2002.

[3] 应毅, 唐立, 刘定一, 等. 基于聚类和二分图匹配的物流派件调度方法 [J]. Journal of Jiangsu University (Natural Science Edition)/Jiangsu Daxue Xuebao (Ziran Kexue Ban), 2020, 41(3).

[4] 韩晓, 孟相如, 康巧燕, 等. 基于二分图最优匹配的虚拟网络映射算法 [J]. Systems Engineering & Electronics, 2019, 41(12).

[5] 田兴鹏, 朱晓荣, 朱洪波. 基于 KM 算法的分布式无线节点任务分配方法 [J]. 北京邮电大学学报, 2020, 43(6): 96.

附录 A　源代码

Kuhn-Munkres 算法源代码。

```
# encoding=utf-8

import numpy as np
import random
import math
import time

zero_threshold = 0.00000001


class KMNode(object):
    def __init__(self, id, exception=0, match=None, visit=False):
        self.id = id
        self.exception = exception
        self.match = match
        self.visit = visit


class KuhnMunkres(object):
    def __init__(self):
        self.matrix = None
        self.x_nodes = []
        self.y_nodes = []
        self.minz = 0.01
        self.x_length = 0
        self.y_length = 0
        self.index_x = 0
        self.index_y = 1

    def __del__(self):
        pass

    def set_matrix(self, x_y_values):
        xs = set()
        ys = set()
        for x, y, value in x_y_values:
            xs.add(x)
            ys.add(y)

        x_dic = {x: i for i, x in enumerate(xs)}
        y_dic = {y: j for j, y in enumerate(ys)}
        self.x_nodes = [KMNode(x) for x in xs]
        self.y_nodes = [KMNode(y) for y in ys]
```

```
        self.x_length = len(xs)
        self.y_length = len(ys)

        self.matrix = np.zeros((self.x_length, self.y_length))

        for row in x_y_values:
            x = row[self.index_x]
            y = row[self.index_y]
            value = row[2]
            x_index = x_dic[x]
            y_index = y_dic[y]
            self.matrix[x_index, y_index] = value

        # 将最小值设置为初始顶标
        for i in range(self.x_length):
            self.x_nodes[i].exception = min(self.matrix[i, :])


    def km(self):
        for i in range(self.x_length):
            while True:
                self.minz = 0.01
                self.set_false(self.x_nodes)
                self.set_false(self.y_nodes)

                if self.dfs(i):
                    break

                self.change_exception(self.x_nodes, self.minz)
                self.change_exception(self.y_nodes, -self.minz)

    def dfs(self, i):
        match_list = []
        while True:
            x_node = self.x_nodes[i]
            x_node.visit = True
            for j in range(self.y_length):
                y_node = self.y_nodes[j]
                if not y_node.visit:
                    t = x_node.exception + y_node.exception - self.matrix[i][j]
                    if abs(t) < zero_threshold:
                        y_node.visit = True
                        match_list.append((i, j))
                        if y_node.match is None:
                            self.set_match_list(match_list)
                            return True
                        else:
```

```
                    i = y_node.match
                    break
            else:
                if t >= zero_threshold:
                    self.minz = min(self.minz, t)
        else:
            return False

def set_match_list(self, match_list):
    for i, j in match_list:
        x_node = self.x_nodes[i]
        y_node = self.y_nodes[j]
        x_node.match = j
        y_node.match = i

def set_false(self, nodes):
    for node in nodes:
        node.visit = False

def change_exception(self, nodes, change):
    for node in nodes:
        if node.visit:
            node.exception += change

def get_connect_result(self):
    ret = []
    for i in range(self.x_length):
        x_node = self.x_nodes[i]
        j = x_node.match
        y_node = self.y_nodes[j]
        x_id = x_node.id
        y_id = y_node.id
        value = self.matrix[i][j]

        if self.index_x == 1 and self.index_y == 0:
            x_id, y_id = y_id, x_id
        ret.append((x_id, y_id, value))

    return ret

def get_max_value_result(self):
    ret = 0
    for i in range(self.x_length):
        j = self.x_nodes[i].match
        ret += self.matrix[i][j]

    return ret
```

```
def run_kuhn_munkres(x_y_values):
    process = KuhnMunkres()
    process.set_matrix(x_y_values)
    process.km()
    return process.get_connect_result()

def getValues(source_coordinates, target_coordinates):
    ret_values = []

    for index_i, i in enumerate(source_coordinates):
        for index_j, j in enumerate(target_coordinates):
            ret_values.append((index_i+1, index_j+1, getDistance(i, j)))
    return ret_values

def getDistance(n1, n2):
    return round(math.sqrt((n1[0]- n2[0]) ** 2 + (n1[1] - n2[1]) ** 2), 1)



if __name__ == '__main__':
    s_time = time.time()
    # ret = test()

    values1 = [(1, 10) , (2, 10) , (3, 10) , (4, 10) , (5, 10) , (6, 10) , (7, 10) , (8, 10) ,
        (9, 10) , (10, 10), (11, 10) , (12, 10) , (13, 10) , (14, 10) , (15, 10), (16, 10) ,
        (17, 10) , (18, 10) , (19, 10) , (20, 10)]
    values2 = [(8, 8) , (8, 9) , (8, 10) , (8, 11) , (8, 12) , (8, 13) , (9, 8) , (10, 8) ,
        (11, 8) , (12, 8) , (13, 8) , (13, 9) , (13, 10) , (13, 11) , (13, 12) , (13, 13) , (9,
        13) , (10, 13) , (11, 13) , (12, 13)]

    result = run_kuhn_munkres(getValues(values1, values2))
    minLen = 0
    for i in result:
       minLen += i[2]
    print(result)
    # print(minLen)
```