队伍编号
202101800908

题号： A

基于 CIELAB 空间下考虑马赛克瓷砖表现力与低成本选色的研究

摘要

目前瓷砖的生产主要靠人眼对颜色的判定来决定，其结果通常会受到主观因素的影响，并且检测效率也比较低，所以通过颜色色差分析，可以根据所提供的原始图片色彩，找到颜色最相似的瓷砖。这种模型不仅能大大减少客户人工选色的工作量，还能大大提高分辨效率以及个性化的生活品质。

在问题一中，通过查阅文献，我们了解到 RGB 空间相对于人眼的局限性，因此我们将 RGB 色彩空间映射到与人眼更为契合的 CIELAB 色彩空间，计算并给出了在不同色彩空间下的色彩差异。通过色谱比较，我们证明了在 CIELAB 色彩空间下通过一种计算色彩间 ΔE 的方式，能精准找到与原始色彩最为相似的瓷砖色彩，且此种方法的效果远高于以 RGB 空间的欧氏距离进行衡量的方法。

在问题二中，我们研究了美学相关的色彩理论，确立了用以衡量色彩所能拥有的表现力的多项指标，同时我们结合和谐色彩理论，最终成功将问题二转化为有约束条件的单目标优化模型。针对该模型，我们提出了一种改进的多球分裂增量聚类（BKM）算法。并且考虑到聚类算法的初值敏感性，我们又提出了三种不同的样本空间方案以供厂商选择。在规定的约束条件下运行算法，最终我们分别找到了三个样本空间的 1-10 种应增加的颜色。

在问题三中，基于问题二的模型与数据，我们很轻易地就能得出每个样本空间最应增加的性价比最高的色彩数量。同时，我们结合现实室内装修设计的业务环境，分别针对三个样本空间提出了它们所适用的场景与风格。其中，冷色系方案更适用于印象派油画的艺术类作品，暖色系方案更适用于现代的人物摄影作品而针对符合正态分布的色彩空间提出的方案则更适用于各类画面色彩平衡中庸的风景摄影作品。

最后我们对模型做出推广，针对厂商未来可能会需要再次研发新型瓷砖颜色以处理日益扩大的业务需求的情况，我们假设其一开始选择我们的冷色系方案，而业务需求为暖色系图案。在此种扩大的样本空间上运行算法，结果证明，我们的模型仍然能在较低研发数量内达到收敛，且在冷暖色混合的作品上取得不错的效果。

本文基于 CIELAB 色彩空间的 ΔE 评价体系，利用改进的多球分裂增量聚类（BKM）算法建立了一个具有较强稳定性且可在未来根据业务需求进行持续优化的模型，具有较大的参考价值和现实意义。

关键词: CIELAB 色彩空间，色彩相似度，多球分裂增量聚类算法，Chameleon 算法

目录

1 问题重述 1

1.1 问题背景与意义 1

1.2 需解决的问题 1

2 问题假设 2

3 主要符号说明 2

4 模型建立与求解 2

4.1 问题一 3

4.1.1 颜色空间的转化 3

4.1.2 颜色差异比对 5

4.1.3 模型求解 7

4.2 问题二 9

4.2.1 评价模型的构造 9

4.2.2 色彩和谐理论的应用 9

4.2.3 冷暖色系对比所营造的表现力 11

4.2.4 改进的多球分裂增量聚类算法 13

4.2.5 模型求解及结果 16

4.3 基于三种色系和曲线下降速率考虑成本与表现力最优规划 . . 18

5 模型的评价与展望 22

5.1 评价颜色差异的模型应用 22

5.2 评价图案表现力的模型应用 22

5.2.1 颜色和谐参数和的应用 23

5.2.2 基于多球分裂的增量式 k-means 聚类算法 23

5.3 模型优缺点 23

5.3.1 以 RGB 标准的色差评价 23

5.3.2 以 LAB 标准的 CIEDE2000 色差评价 24

5.3.3 基于多球分裂的增量式 k-means 聚类算法: 25

5.4 基于改进 BKM 算法的推广 26

参考文献 29

附录 29

1 问题重述

1.1 问题背景与意义

80 年代初，马赛克瓷砖常被用于许多家庭铺设卫生间墙面、地面的铺设材料，而到如今，随着个性化设计成为现代人的一种品味追求，设计者可以通过马赛克瓷砖给予消费者个性化的生活环境，马赛克瓷砖由数十块小块的砖组成一个相对的大砖，以小巧玲珑、色彩斑斓的特点被广泛使用于室内小面积地面、墙面和室外大小幅墙面和地面，以拼接图案的新形式卷土重来，重新成为装饰界的时尚宠儿，备受青睐。

消费者在拼接图案时，需要根据图案的原始色彩找到颜色相近的瓷砖，方可进行拼接。目前瓷砖的生产主要靠人眼对颜色的判定来决定，其结果通常会受到主观因素的影响，检测结果不一定准确，并且检测效率也比较低，所以通过颜色色差分析，来建立一种能确定原始颜色与瓷砖颜色相对应的算法以及数学模型进行智能分析，可以根据所提供的原始图片色彩，找到颜色最相似的瓷砖。这种模型不仅能大大减少客户人工选色的工作量，还能大大提高分辨效率以及个性化的生活品质。

受到工艺和成本的限制，瓷砖的颜色只能是有限的几种。随着消费者对产品质量和设计要求的提高，马赛克瓷砖厂家需要花费大量的精力在产品研发上。在控制成本的基础上，确定需要研发新瓷砖色彩，以追求更高的表现效果与品质，也就成了目前瓷砖生产领域里的一主要问题。根据已有的瓷砖颜色，在相同的成本内，确定新的几种瓷砖颜色，让拼接图案具有更强的表现力，也就是我们的主要研究所在。

1.2 需解决的问题

为解决这些问题，我们将采取以下措施:

问题一，首先我们需要根据附件 2 和附件 3 所提供的总共 416 个 RGB 值，来确定每个值对应最相近的瓷砖厂已有的瓷砖颜色，并且将输出结果。

问题二，我们在瓷砖厂已有的 22 种瓷砖颜色的基础上，找出工厂需要优先增加研发的 1 种瓷砖颜色，能使得附件 2 和附件 3 给定的拼接图像在 23 种颜色

下，表现力效果最好。并且将优先研发的瓷砖颜色数量从 1 考虑到 10，给出每一种分别对应的 RGB 编码值。

问题三，基于问题二，我们需要考虑瓷砖厂的成本问题。由于研发颜色数量的增加，成本也在线性增加，而图案的表现效果并不一定越来越明显，所以我们需要从所做的成果中，总结出建议，给出新增研发瓷砖颜色的种数及其 RGB 值。使得瓷砖工厂在保证成本和表现效果之间做出最优规划。我们也会将此推广到更常见的生活应用中，即不基于附件 2 和附件 3 所给定的 RGB 值，而是通过更深入的色彩理论研究来得出结论。

2 问题假设

为了能够合理的简化问题，我们在模型建立的过程中提出如下假设：

1. 研发一种新颜色瓷砖的成本是相同的，与颜色本身无关。
2. 拼接图像的表现力只与我们下面模型所用计算的色差 ΔE 相关，与其他因素无关。
3. 研发的成本都按单位计算。

3 主要符号说明

以下是我们论文中的变量及其含义:

Symbol	Meaning
C_i	附件给定的原始颜色的 RGB 值
cur_i	瓷砖厂已有颜色的 RGB 值
E_l	新研发 l 种颜色图案的表现力
σ_k	满足第 k 种和谐颜色的加分值
$\Delta E_{C_j,cur_i}$	原始颜色与瓷砖颜色的差异值
ΔE_{cur_i}	归类于第 i 种瓷砖颜色的所有原始颜色的 $\Delta E_{C_j,cur_i}$ 之和

4 模型建立与求解

4.1 问题一

我们先进行 RGB 颜色空间到 CIELAB 颜色空间的转换，计算在不同颜色空间下的颜色差异，并进行比较，得出了在 CIELAB 颜色空间下的，能精准找到与原始颜色最为相似的瓷砖颜色。

4.1.1 颜色空间的转化

由于附件 2 和附件 3 所给的图像颜色特征是基于 RGB 颜色空间的，RGB 颜色空间因其颜色分布不均匀，并且它的颜色范围有限，拼接图案所表示的颜色与对应已有的瓷砖颜色不能只通过计算欧氏距离来确定。正式定义一个色彩空间时所采用的标准是 CIELAB 或 CIEXYZ 色彩空间，他们涵盖了正常人可见范围所有颜色所设计提出的，因此是最精准的色彩空间。在本问题中，我们主要考虑 CIELAB 色彩空间和 HSV 色彩空间。我们将转换颜色空间，并且比较几种颜色空间的具体效果，来确定转换的所对应的算法。

- CIELAB **色彩空间**

CIELAB 色彩空间将颜色用三个值表达示 [1]：“L^*” 代表感知的亮度、“a^*” 和 “b^*” 代表人类视觉的四种独特颜色：红色、绿色、蓝色和黄色。三个基本坐标表示颜色的亮度（L^*, $L^*=0$ 生成黑色而 $L^*=100$ 指示白色），它在红色 / 品红色和绿色之间的位置（a* 负值指示绿色而正值指示品红）和它在黄色和蓝色之间的位置（b* 负值指示蓝色而正值指示黄色）CIELAB 旨在作为一个感知上统一的空间，其中给定的数字变化对应于相似的感知颜色变化。转换步骤如下：

我们先将 RGB 颜色空间转换到 XYZ 颜色空间，

$$\begin{aligned}\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} &= \frac{1}{b_{21}} \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \\ &= \frac{1}{0.17697} \begin{bmatrix} 0.49 & 0.31 & 0.20 \\ 0.17697 & 0.81240 & 0.01063 \\ 0.00 & 0.01 & 0.99 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}\end{aligned} \tag{1}$$

其中 X 代表红色，Y 代表绿色，Z 代表蓝色。

我们再把 XYZ 颜色空间转换到 CIELAB 颜色空间，其转换公式如下。正向变

换：

$$\begin{aligned} L^* &= 116f(Y/Y_n) - 16 \\ a^* &= 500f(X/X_n) - f(Y/Y_n) \\ b^* &= 200f(Y/Y_n) - f(Z/Z_n) \end{aligned} \tag{2}$$

其中，

$$f(t) = \begin{cases} t^{1/3}, & \text{if } t > (6/29)^3 \\ \frac{1}{3}\left(\frac{29}{6}\right)^2 t + \frac{16}{116}, & \text{otherwise} \end{cases} \tag{3}$$

这里 X_n、Y_n、Z_n 是参照白点的 CIEXYZ 三色刺激值。

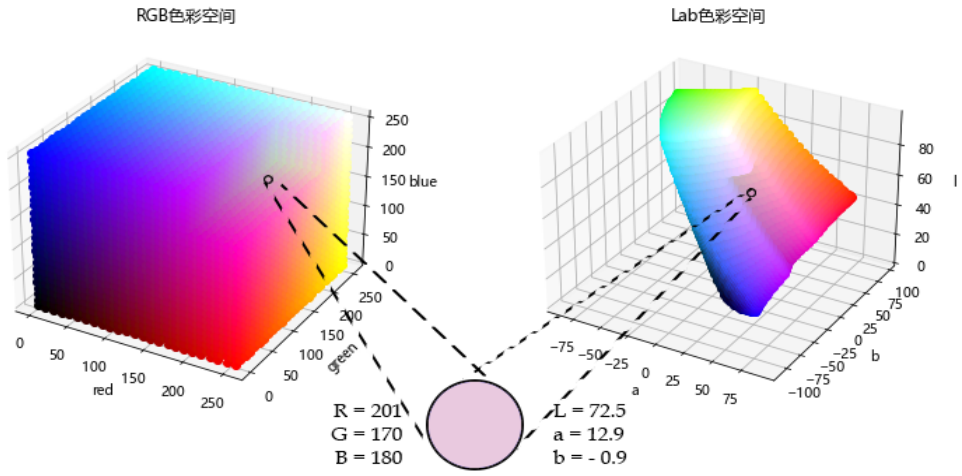

图 1: RGB 色彩空间转换到 CIELAB 色彩空间

反向变换如下（$\delta = \frac{6}{29}$）：

1. 定义 $f_y \overset{\text{def}}{=} (L^* + 16)/116$
2. 定义 $f_x \overset{\text{def}}{=} f_y + a^*/500$
3. 定义 $f_z \overset{\text{def}}{=} f_y - b^*/200$
4. 如果 $f_y > \delta$ 则 $Y = Y_n f_y^3$ 否则 $Y = (f_y - 16/116)\, 3\delta^2 Y_n$
5. 如果 $f_x > \delta$ 则 $X = X_n f_x^3$ 否则 $X = (f_x - 16/116)\, 3\delta^2 X_n$
6. 如果 $f_z > \delta$ 则 $Z = Z_n f_z^3$ 否则 $Z = (f_z - 16/116)\, 3\delta^2 Z_n$

- HSV 色彩空间

HSV 都是一种将 RGB 色彩模型中的点在圆柱坐标系中的表示法。这个圆柱的中心轴取值为自底部的黑色到顶部的白色而在它们中间的是灰色，绕这个轴的角度对应于“色相”，到这个轴的距离对应于“饱和度”，而沿着这个轴的高度对应于“明度”。这两种表示法试图做到比基于笛卡尔坐标系的几何结构 RGB 更加直观。以下是 RGB 到 HSV 的转换方法：

正向转换：

$$h = \begin{cases} 0^\circ \text{ if } \max = \min \\ 60^\circ \times \frac{g-b}{\max - \min} + 0^\circ, \text{ if } \max = r \text{ and } g \geq b \\ 60^\circ \times \frac{g-b}{\max - \min} + 360^\circ, \text{ if } \max = r \text{ and } g < b \\ 60^\circ \times \frac{b-r}{\max - \min} + 120^\circ, \text{ if } \max = g \\ 60^\circ \times \frac{r-g}{\max - \min} + 240^\circ, \text{ if } \max = b \end{cases} \tag{4}$$

$$s = \begin{cases} 0, \text{ if } \max = 0 \\ \frac{\max - \min}{\max} = 1 - \frac{\min}{\max}, \text{ otherwise} \end{cases} \qquad v = \max \tag{5}$$

其中设（r，g，b）分别是一个颜色的红、绿和蓝坐标，它们的值是在 0 到 1 之间的实数。$max = max\{r, g, b\}, min = min\{r, g, b\}$。$h \in [0, 360)$，而 $s, \in [0, 1]$。

反向转换如下：

$$\begin{aligned} h_i &= \lfloor \tfrac{h}{60} \rfloor \\ f &= \tfrac{h}{60} - h_i \\ p &= v \times (1 - s) \\ q &= v \times (1 - f \times s) \\ t &= v \times (1 - (1 - f) \times s) \end{aligned} \tag{6}$$

对于每个颜色向量 (r, g, b) 有：

$$(r, g, b) = \begin{cases} (v, t, p), \text{ if } h_i = 0 \\ (q, v, p), \text{ if } h_i = 1 \\ (p, v, t), \text{ if } h_i = 2 \\ (p, q, v), \text{ if } h_i = 3 \\ (t, p, v), \text{ if } h_i = 4 \\ (v, p, q), \text{ if } h_i = 5 \end{cases} \tag{7}$$

4.1.2 颜色差异比对

颜色差异可以通过色彩空间内的欧氏距离简单计算得出，也可以使用国际照

明委员会较为复杂、均匀的人类知觉公式计算 [2]。本模型中，我们主要使用了后者。

- CIEDE2000

我们通过前面已经讲述的方法，将 RGB 颜色空间转换到 CIELAB 颜色空间后，利用如下步骤，进行颜色差异的定量分析。首先给出总的公式：

$$\Delta E_{12}^{*}=\sqrt{\left(\frac{\Delta L'}{k_L S_L}\right)^2+\left(\frac{\Delta C'}{k_C S_C}\right)^2+\left(\frac{\Delta H'}{k_H S_H}\right)^2+R_T\frac{\Delta C'}{k_C S_C}\frac{\Delta H'}{k_H S_H}} \tag{8}$$

下面我们来解释一下这个公式（注：公式使用角度而非弧度，k_L、k_C、k_H 一般取 1，L^*,a^*,b^* 是经过颜色空间转换的值）具体如下：

$$\Delta L'=L_2^*-L_1^* \tag{9}$$

$$\bar{L}=\frac{L_1^*+L_2^*}{2}\quad \bar{C}=\frac{C_1^*+C_2^*}{2} \tag{10}$$

$$a_1'=a_1^*+\frac{a_1^*}{2}\left(1-\sqrt{\frac{\bar{C}^7}{\bar{C}^7+25^*}}\right)\quad a_2'=a_2^*+\frac{a_2^*}{2}\left(1-\sqrt{\frac{\bar{C}^7}{\bar{C}^7+25^*}}\right) \tag{11}$$

$$\begin{gathered}\bar{C}'=\frac{C_1'+C_2'}{2}\ \text{and}\ \Delta C'=C_2'-C_1'\\ \text{where}\ C_1'=\sqrt{a_1'^2+b_1^*}\quad C_2'=\sqrt{a_2'^2+b_2^*}\end{gathered} \tag{12}$$

$$h_1'=\operatorname{atan2}\left(b_1^*,a_1'\right)\mod 360^\circ,\quad h_2'=\operatorname{atan2}\left(b_2^*,a_2'\right)\mod 360^\circ \tag{13}$$

其中 $arctan$ 的范围通常是从 π 到 $-\pi$ 弧度；颜色规格为 $0-360^\circ$，需要调整。如果 a' 和 b 都为零 (这也意味着相应的 C' 为零)，则 tan 是不确定的；在这种情况下，设置色调的角度为零。

$$\Delta h'=\begin{cases}h_2'-h_1' & \left|h_1'-h_2'\right|\le 180^\circ\\ h_2'-h_1'+360^\circ & \left|h_1'-h_2'\right|>180^\circ, h_2'\le h_1'\\ h_2'-h_1'-360^\circ & \left|h_1'-h_2'\right|>180^\circ, h_2'>h_1'\end{cases} \tag{14}$$

当 C_1' 或 C_2' 为零时，则 $\Delta h'$ 是不相关的，可以设置为零.

$$\Delta H'=2\sqrt{C_1'C_2'}\sin\left(\Delta h'/2\right),\quad \bar{H}'=\begin{cases}\left(h_1'+h_2'+360^\circ\right)/2 & \left|h_1'-h_2'\right|>180^\circ\\ \left(h_1'+h_2'\right)/2 & \left|h_1'-h_2'\right|\le 180^\circ\end{cases} \tag{15}$$

当 C_1' 或 C_2' 为 0 时，$H' = H_1' + H_2'$(不能被 2 整除；本质上，如果一个角是不确定的，那么就用另一个角作为平均值；依赖于不定角被设为零)。

$$T = 1 - 0.17\cos\left(\bar{H}' - 30°\right) + 0.24\cos\left(2\bar{H}'\right) + 0.32\cos\left(3\bar{H}' + 6°\right) - 0.20\cos\left(4\bar{H}' - 63°\right) \tag{16}$$

$$S_L = 1 + \frac{0.015(\bar{L} - 50)^2}{\sqrt{20 + (\bar{L} - 50)^2}} \quad S_C = 1 + 0.045\bar{C}' \quad S_H = 1 + 0.015\bar{C}'T \tag{17}$$

$$R_T = -2\sqrt{\frac{\bar{C}'^7}{\bar{C}^7 + 25^7}}\sin\left[60° \cdot \exp\left(-\left[\frac{\bar{H}' - 275°}{25°}\right]^2\right)\right] \tag{18}$$

我们也进行了欧氏距离的计算，用来验证 CIELAB 颜色空间转换的优越性。给定一个 RGB（红绿蓝）的色彩空间，最简单的差异计算方式就是在这个三维空间里求两个点间的距离：

$$\Delta R = \sqrt{(R_2 - R_1)^2 + (G_2 - G_1)^2 + (B_2 - B_1)^2} \tag{19}$$

4.1.3 模型求解

通过上述的公式理论推演，我们将以这些基础建立原始颜色对应瓷砖颜色的模型，公式如下：

$$C_i \rightarrow \{cur_i \mid min\Delta E_{C_i,cur_i}\} \tag{20}$$

以及通过比对 RGB 颜色空间的最小距离：

$$C_i \rightarrow \{cur_i \mid min\Delta R_{C_i,cur_i}\} \tag{21}$$

其中 C_i 是附件所给定的原始颜色的 RGB 值，cur_i 是瓷砖厂已有颜色的 RGB 值，对于原始颜色，我们利用如上公式，寻找到使得 ΔE 最小的已有瓷砖颜色，即与原始颜色最相近的瓷砖颜色。结果如图，以及输出相应的 RGB 值在附件里。

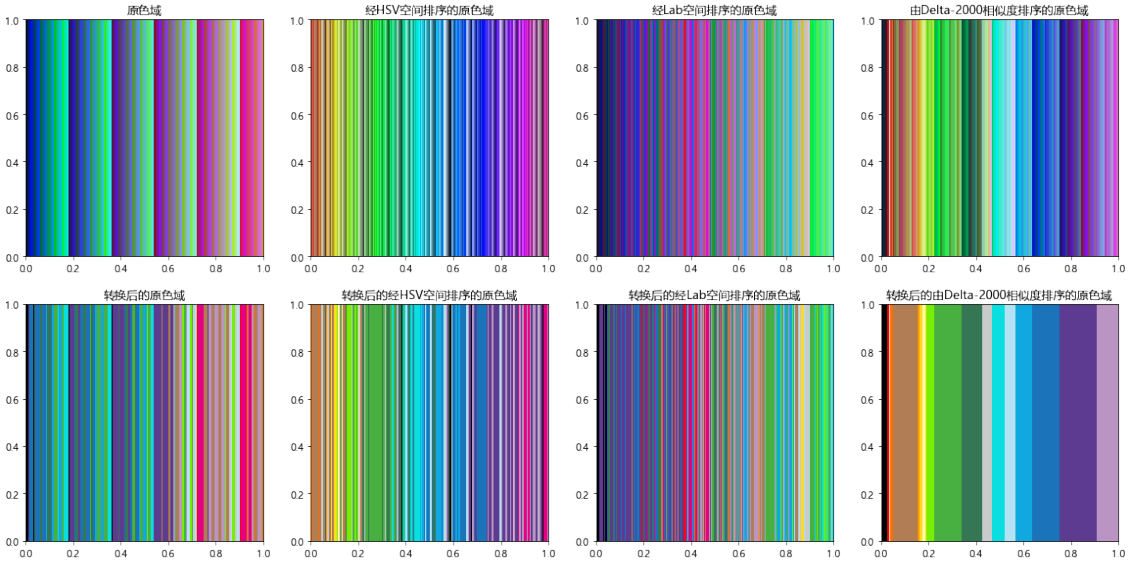

图 2: 最小 RGB 欧氏距离颜色对应方法

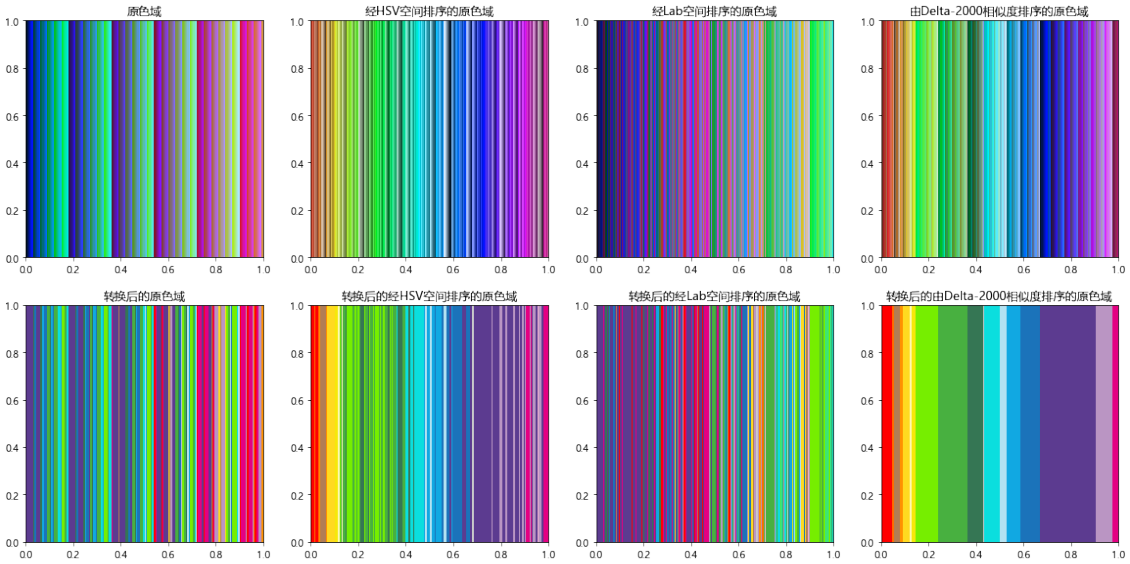

图 3: 最小 ΔE 颜色对应方法

从上图我们能很轻易地看出，由单纯的 RGB 欧氏距离所转换的色彩不管在 HSV 空间，或是在 Lab 空间的排列中，颜色的聚合度都明显不如由 δE 所转化的色彩。同时，将转换后的颜色以哈密尔顿最短路的方法最小排列起来，使其相互间的 ΔE 之和最小，显然 RGB 欧氏距离比 ΔE 包含了更多不同色系的颜色，显得更加杂乱无章。因此 RGB 空间中欧氏距离在用以形容色彩相似度时效果是完全比

不上 ΔE 的，这证明了我们评价体系的优越性。

4.2 问题二

在问题二中，我们需要在瓷砖厂已有的 22 种瓷砖颜色的基础上，找出工厂需要优先增加研发的 1 种瓷砖颜色，能使得附件 2 和附件 3 给定的拼接图像在 23 种颜色下，表现力效果最好。并且将优先研发的瓷砖颜色数量从 1 考虑到 10，给出每一种分别对应的 RGB 编码值。

我们是先研究了一些色彩理论，再根据问题一已经构建的求相似颜色的模型基础上，通过改进多球分裂增量聚类的方法，以及结合和谐颜色理论，冷暖色系对比所营造的表现力等，得出了一种恰当的评价模型。在规定的约束条件下运行我们的模型，从而找到优先考虑研发的瓷砖颜色。下面我们将具体说明我们的算法实现和模型构造。

4.2.1 评价模型的构造

$$E_l = \sum_{i=1}^{22+l}\sum_{j=1}^{n}\Delta E_{C_j,cur_i} - \sum_{m=1}^{l}\sum_{k=1}^{5}\alpha_l\sigma_k - \sum_{i=1}^{l}W_iS_iD_i \tag{22}$$

其中 E_l 是指新研发 l 种瓷砖颜色后，图案拼接后的表现能力（数值越小，表现力越强）；$\Delta E_{C_j,cur_i}$ 是指在 $22+l$ 种瓷砖基础上，归类于第 i 种瓷砖颜色的 C_J 种原始原始颜色与其的差异值；σ_k 是指满足第 k 种和谐颜色的加分值;W_i, S_i, D_i 冷暖色系对比的三个维度系数。整体上来看，当我们要研发新的 l 种新颜色时，应当寻找使 E_l 值最小的 l 种颜色，这样拼接的图案的表现力最强。下面我们将具体讲述约束条件以及改进多球分裂增量聚类算法的优化方法。

4.2.2 色彩和谐理论的应用

色彩和谐是以和谐的方式将色彩与眼睛和谐结合的理论，也就是什么和什么颜色可以很好地结合在一起，这是几乎所有颜色设计决策背后的原因。而我们此次模型的约束条件之一，就来源于色彩和谐理论的应用，从而可以让马赛克瓷砖拼接出来图案更具有表现力。

- **彩色轮盘**

色彩和谐是基于色轮的概念，本质上，它是一个由所有颜色组成的圆圈。原色在轮子的三个距离相等的点上。通常是红色、蓝色和黄色。在彩轮产生的绘画领域，

这三种原色被用来混合几乎所有的颜色。在轮子上的三种原色之间是它们的混合色——紫色在红色和蓝色之间，橙色在红色和黄色之间，绿色在黄色和蓝色之间。从理论上讲，所有的颜色都在方向盘上的某个地方。

- **和谐颜色的五种类型**

1. **对立和谐**：这是最基本的和谐。它是一个与轮子上的主色相反的点。这种“相对”色称为互补色，因此直接的和谐也可称为互补和谐。几乎所有的颜色和谐（除了类似的）都是直接和谐的变化。互补色的高对比度创造了一个充满活力的外观，特别是在饱和使用时，但如果管理不当可能会不和谐。这是最常见的配色方案，在各种设计中都很容易找到。

如果我们新增的颜色 i 与原有的瓷砖颜色 j 满足对立和谐，那么在这里，我们给出 σ_1 的定义：

$$\sigma_1 = \frac{\Delta E_{i,j}}{2} \tag{23}$$

2. **分离互补和谐**：这允许一个更好的颜色范围，同时仍然不偏离主色和互补色之间的基本和谐。这种配色方案与互补配色方案具有同样强烈的视觉对比，但张力较小，总是看起来很好。

如果我们新增的颜色 i 与原有的瓷砖颜色 j,k 满足分离互补和谐，那么在这里，我们给出 σ_2 的定义：

$$\sigma_2 = \frac{\Delta E_{i,j} + \Delta E_{i,k} + \Delta E_{k,j}}{6} \tag{24}$$

3. **三角和谐**：这是指颜色的两个空间的任何一方的主色的补色。从本质上说，在三合一的和声中，在色轮上使用了三种距离相等的颜色。因此，这是在延伸颜色和谐的基本理念，这种和谐最好只用少量色彩。

如果我们新增的颜色 i 与原有的瓷砖颜色 j,k 满足三角和谐，那么在这里，我们给出 σ_3 的定义：

$$\sigma_3 = \sigma_2 \tag{25}$$

4. **相邻和谐**：也被称为相关颜色，这些颜色直接位于主色的左边和右边。通常搭配得很好，创造出一种宁静和舒适的设计，让人赏心悦目。

如果我们新增的颜色 i 与原有的瓷砖颜色 j 满足相邻和谐，而这与我们计算 ΔE 的出发方向是一致的，那么在这里，我们给出 σ_1 的定义：

$$\sigma_4 = 0 \tag{26}$$

5. **四角和谐**：类似于三位一体，在色轮上有四个相同距离的点。

如果我们新增的颜色 i 与原有的瓷砖颜色 j,k,l 满足四角和谐，那么在这里，我们给出 σ_5 的定义：

$$\sigma_5 = \frac{\Delta E_{i,j} + \Delta E_{i,k} + \Delta E_{k,j} + \Delta E_{l,j} + \Delta E_{l,k} + \Delta E_{l,i}}{12} \tag{27}$$

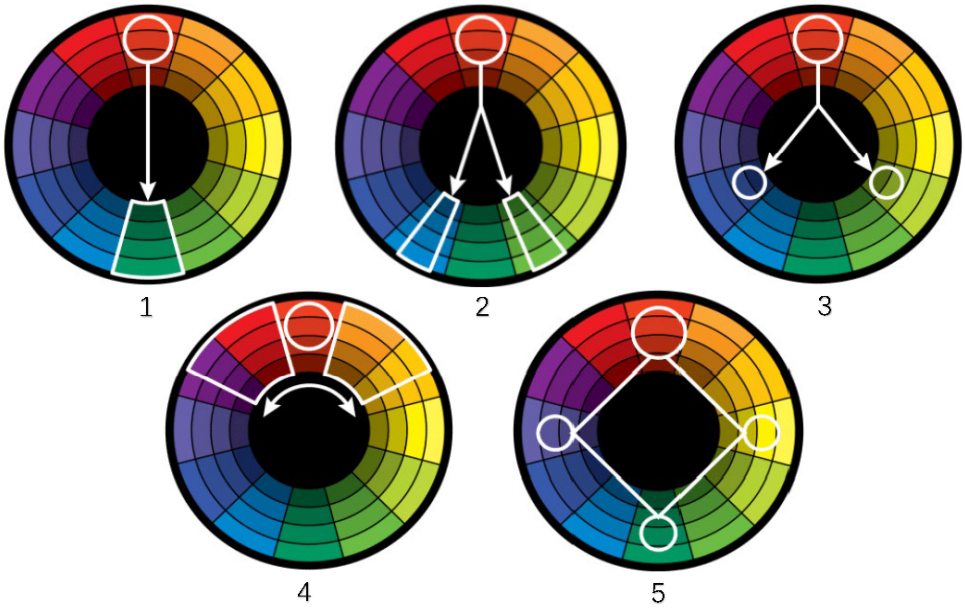

图 4: 五种和谐颜色示意图

通过上述的和谐颜色的比对，我们将得到 $\Sigma_{m=1}^{l}\Sigma_{k=1}^{5}\alpha_l\sigma_k$ 的具体数值，其中 α_l 是我们根据冷暖色调以及正态分布色调进行多次试验得到的常系数。

4.2.3 冷暖色系对比所营造的表现力

色彩有各种分类方式，我们选择用色相将所有颜色区分为冷色、暖色和中性色。下面给出本文结合色彩学理论对冷暖色调的定义：

1. **暖色**：光谱的暖色包括红色、橙红色、橙色、黄橙色和黄色。一般来说，假如一种复色以红色和黄色为主导，我们称其为暖色，并给出其在 RGB 色彩空间的定义：

$$\frac{R+G}{2} > B \tag{28}$$

2. **冷色**：光谱的冷色包括蓝色、蓝绿色、绿色、紫色和蓝紫色。在一种复色

中假如蓝色占据主导地位，我们称其为冷色，并给出定义：

$$\frac{R+G}{2} \leqslant B \tag{29}$$

3. **中性色和中性化的色彩：**黑与白以及黑白之间的各种不同明度的灰都叫中性色，也就是传统意义上的“五彩斑斓的灰”。从色彩光学上来说，黑白灰属于无彩色系，对光源的光波反射没有选择性，所以是中性色。考虑到中性色更多的作为一种中庸的选择应用于室内设计装修中，是不带有太多个人情感色彩的一种色系，同样不具备强烈的表现力，因此我们不予太多考虑。

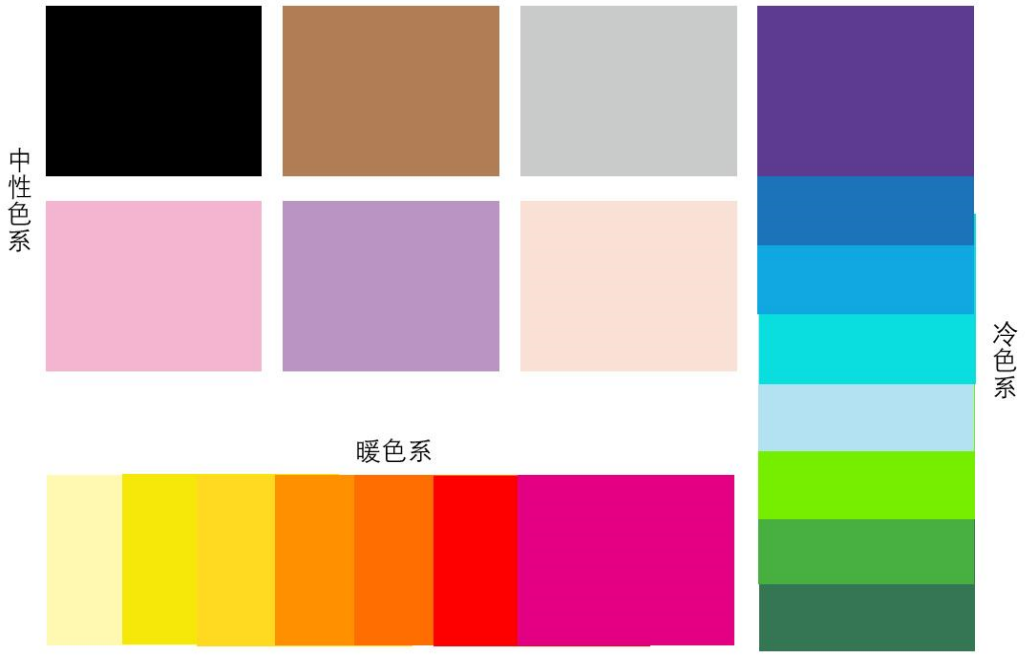

图 5: 原 22 种瓷砖颜色所划分的冷暖色系

设计界中有一句话：任何设计中对比都是增加易读性的关键。而冷暖色系对比则是其中最经典的搭配方案，我们经过查找多方资料后从中筛选出了几个指标来衡量冷暖色系对比所营造的表现力，下面给出定义：

距离 D：色彩可以使人感觉进退、凹凸、远近的不同。暖色系和明度高的色彩具有前进、凸出、接近的效果；冷色系和明度较低的色彩则具有后退、凹进、远离的效果。因此合理运用冷暖色对比可以是画面更具空间感和层次感。我们定义 D 为：

$$D = ln(\Delta E_{C_j,cur_i}) \tag{30}$$

重量 W：色彩的重量感主要取决于明度和纯度。明度和纯度高显得轻；明度和纯度低显得重；在室内设计的构图中常以此达到平衡和稳定的要求，以及表现性格的需要。我们定义 W 为：

$$W = \frac{1}{S_{HSV}V_{HSV} + 1} \tag{31}$$

其中 S_{HSV}, V_{HSV} 是 HSV 颜色空间里相对应的明度和纯度。

尺度 S：人眼对尺度的衡量会在一定程度上受到色彩对比的影响。暖色和明度高的色彩具有扩散作用，因此物体会显得大；冷色和明度低的色彩具有内聚的作用，因此物体会显得小。我们定义 S 为：

$$S = e^{B - \frac{R+G}{2}} \tag{32}$$

毫无疑问，一幅优秀的画作会具有层次感鲜明的构图，适中的重量感以及合理的尺度，结合色彩和谐理论，我们可以很轻易的将问题二看作是一个具有约束条件的单目标规划模型。因此我们针对该模型提出一种改进的多球分裂增量聚类算法。

4.2.4 改进的多球分裂增量聚类算法

为了求出 E_l 的最优解，我们需要在全色域的范围内进行分析，而这种方法导致计算负担过重，尤其是当 l 增加到 10 的时候，变量维度急剧增加，导致算法的效率和速度大幅度降低。我们需要找到一种优化算法，来降低计算量。而问题二要求是要基于已有的 22 种瓷砖颜色的基础上，再增加研发新的颜色。这就不由得想到了聚类这一种方法，我们在 k-means 的算法以及多球分裂增量聚类的基础上进行进一步的优化 [3]，即把已有的瓷砖颜色当做聚类中心，我们的工作就是在这些以聚类中心为中心，以对应的原始颜色的最大半径所包含的范围内，用最快又高效的方法，找到新的 l 个聚类点，从而降低 E_l 的大小。

这种算法的优点非常明显，缩小随机选定的中心范围能，从而改善 k-means 的计算效率，高效得出新的聚类中心；k-means 的聚类结果会受到从初始化中心的影响，这种算法能避免影响，不会陷入到局部最优的困局；一次性可以产生多个增量聚类中心，避免了传统增量聚类中心选择过程中的距离计算，将中心选取过程的计算量有 $O(n^s)$ 降低至 $o(klogk)(k < n)$，迭代的计算量更小。在具体说明这种算法的步骤及实现之前，我们先解释一下上述概念。

- **增量聚类**

增量聚类是维持或改变 k 个瓷砖颜色的结构的问题 [4]。比如给定一个新的原始颜色，这个颜色可能被划分到已有 k 个瓷砖颜色中，也有可能被划分到新的

瓷砖颜色中。其缺点也比较明显，因为没选择一个增量中心，就需要计算所有两两不同数据之间的距离，计算量超过了 k-means 一次完整迭代的时间复杂度。并且每确定一次增量中心，都需要运行完整的 k-means 算法到收敛，计算速度比较低下。

- Ball k-means(BKM) BKM 是一种改进 k-means 加速的算法，由于使用超球体来量化空间簇，获得了更加精确的近邻关系，该近邻关系不需要额外参数，消除了现有大多数优秀加速算法中单个样本上下界。

BKM 聚类算法基本思想：首先初始化簇表 S，使 S 包含所有点组成的簇。然后随机从 S 中选取一个簇 c_i，通过 k-means 算法，对选定的簇 c_i 二分，并计算每个簇的最小误差值，选择误差最小的两个簇，再将得到的两个簇加人 S 中，更新簇表 S。这样反复下去，直到最终产生 k 个簇这里通过误差平方和（ SSE ）作为目标函数，来衡量聚类质量。SSE 的定义如下：

$$SSE = \sum_{i=1}^{K} \sum_{x \in c_i} \mathrm{dist}\,(c_i, x)^2 \tag{33}$$

由于单个样本点的计算次数远小于现有的 k-means，从而减少距离计算次数，提高计算效率。但 BKM 容易受到初始颜色中心位置的影响，当初始中心位置较差时，聚类解的精度将随之下降。

针对上面分析增量聚类和 BKM 存在的问题，我们改进的多球分裂的增量算法将基于这些缺点。

- **改进的** BKM

改进的 BKM 主要体现在新的中心的选取方式上，其步骤如下：选择一个聚类中心，作为初始中心，并将该聚类中心加入到簇表中，[5] 选取 SSE 最大的簇，找出该簇中与簇中心最远的点以及与该点最远的点，将这两个点作为新的聚类中心，将这两个新的聚类中心加到簇集 S 中，删除旧的簇中心，计算当前聚类准则函数的值，与判断值进行比较。

- **球聚类的定义**

对于给定聚类 C，如果指定其质心为 c，半径为 r，则称 C 为球聚类。其质心和半径公式如下：

$$c = \frac{1}{|C|} \sum_{i=1}^{|C|} x_i \quad r = \max_{x_i \in C} (\|x_i - c\|) \tag{34}$$

其中，x_i 是分配给 c 的点，而 $|c|$ 表示 c 中的点个数。我们只需要记录球心与半径，而我们将用球聚类来完成对增量中心颜色的选择。

- **多球分裂增量聚类基本思想及步骤**

我们结合 BKM 聚类自身的特点，先确定已有聚类中半径最大的聚类 C_{rmax}，然后我们在该聚类中的范围内寻找新的几种聚点，从而使 ΔE_{curi} 的数值降低。重复上述步骤，就可以根据我们需要新增的聚点个数，找到最适合的聚点。

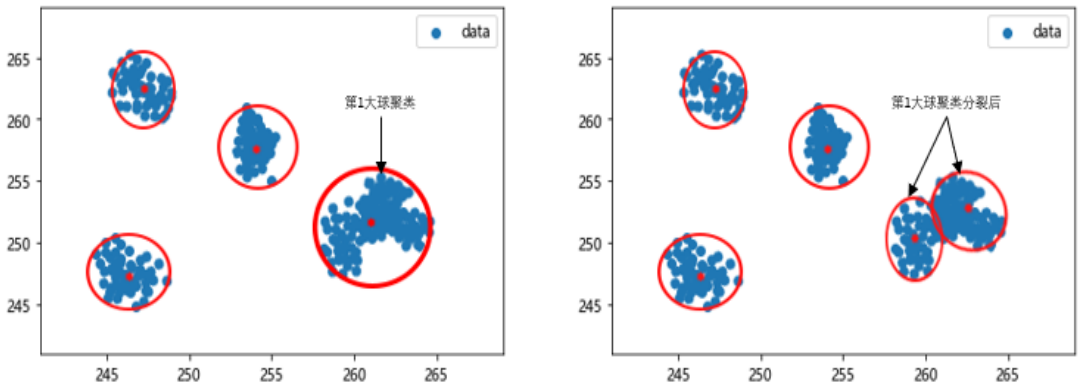

图 6: 多球分裂示意图

我们定义 k_{max} 为我们需要研发颜色的个数和已有瓷砖颜色的个数之和；k_0 为已有瓷砖颜色的个数（$k_0 < k_{max}$）;Δk 为增加颜色种类的步长；具体步骤如下：

第一步，令 $k_t = k_0$，聚类的个数为 k_i，最大迭代次数为 iter，运行改进的 BKM 算法，得到 k_t 个聚类中心，记此中心集合为 $V_t = \{v_1, v_2, \ldots, v_{k_t}\}$，对应的聚类集合记为 $C_t = \{c_1, c_2, \ldots, c_{k_t}\}_0$。

第二步，计算分裂个数 k_{split} 与分裂后总聚类个数 k'_{t_0} 若 $k_t + \Delta k \leq k_{\max}$，令 $k_{\text{split}} = \Delta k, k'_t = k_t + \Delta k$; 否则，令 $k_{split} = k_{\max} - k_t, k'_t = k_{\max}$.

第三步，计算新的增量中心。在 C_t 中选择具有最大球形半径的前 k_{split} 个聚类，将这些聚类的中心进行分裂，每个中心分裂为两个新的聚类中心。令 V_{split} 表示分裂中心集合，V_{new} 表示分裂后产生的所有新中心集合，V_{new} 即为增量中心集合。

第四步，利用新中心进行 k-means 聚类。令 $V_0 = (V_t - V_{\text{split}}) \cup V_{\text{new}}, k_t = k'_{t_0}$ 以聚类个数 k_t 、初始中心 V_0 、最大迭代次数 $iter$ 运行改进的 BKM 算法，得到 k_t 个聚类中心，记此中心集合为 $V_t = \{v_1, v_2, \ldots, v_{k_t}\}$，对应的聚类集合记为 $C_t = \{c_1, c_2, \ldots, c_{k_t}\}_0$.

第五步，终止判断。如果 $k_t = k_{\max}$，运行改进的 BKM 聚类至收敛，算法结束。否则，转至 step2。

流程图如下：

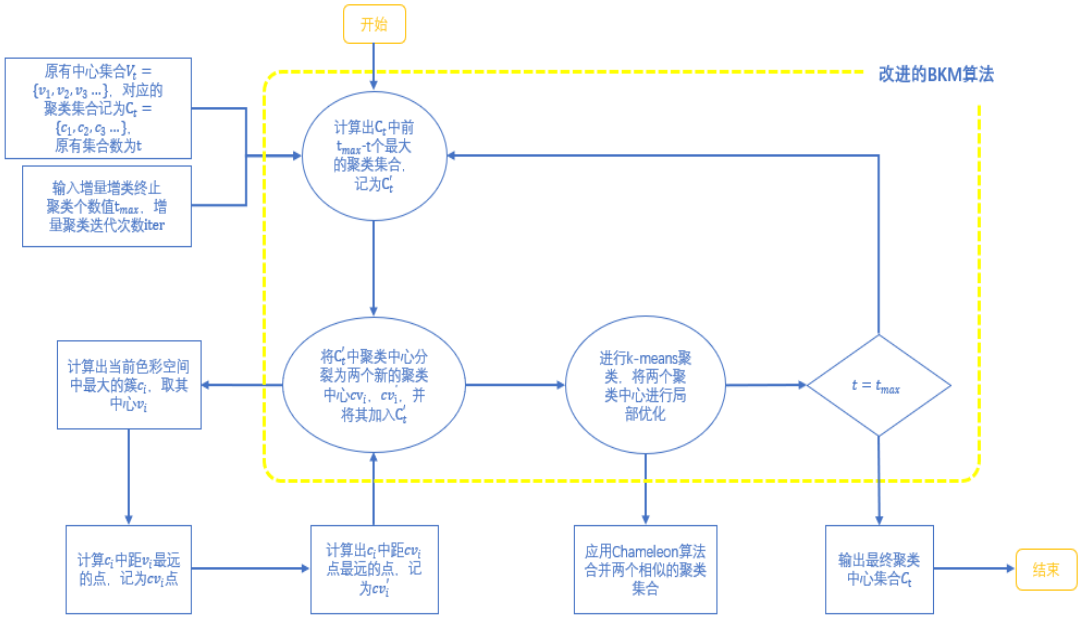

图 7: 多球分裂增量聚类流程图

4.2.5 模型求解及结果

我们根据评价模型的 E_l 作为目标函数:

$$E_l=\sum_{i=1}^{22+l}\sum_{j=1}^{n}\Delta E_{C_j,cur_i}-\sum_{m=1}^{l}\sum_{k=1}^{5}\alpha_l\sigma_k-\sum_{i=1}^{l}W_iS_iD_i \tag{35}$$

根据上述理论，我们的约束条件为：

$$\begin{cases} R_{new}\le maxR_i, G_{new}\le maxG_i, B_{new}\le maxB_i \\ (0<R_i,G_i,B_i<255) \\ D_i>W_i>ln(S_i) \\ E_l'\le 0 且 E_l'\le E_{l-1}' \end{cases} \tag{36}$$

其中 R_{new},G_{new},B_{new} 是新瓷砖颜色的 RGB 值，$maxR_i$ 是指除掉 0 和 255 之外的，在给定原始颜色的 R 值中挑选最大的值；E_l' 是对 E_l 求导，保证表现力曲线是递减的。

考虑到 BKM 算法的初值敏感性，换言之，样本空间的选择对于 BKM 算法最后的结果有极其明显的导向作用，而同时我们无法去认为一幅优秀的艺术作品会有必要涵盖色彩空间中的所有色系。富有创造力与表现力的艺术作品通常非常具有个人情感色彩，如巴勃罗·毕加索所创作的以灰色和淡蓝色为主导的《格尔尼卡》，如文森特·威廉·梵高所创作的以深蓝与鹅黄色为主导的《星月夜》，又如现代摄影作品最经典的人物—玛丽莲·梦露，她的作品通常以红唇、金发作为主元素。

这些艺术作品都或多或少包含了作者的私人情感，因此其必然会有色系上独特的选择。受到颜色数量的限制，我们无法给出一个完美的方案，让它既可以表现出冷色系的内敛、清静，又能渲染出暖色系的热情与豪迈。因此我们退而求其次，分别针对不同色系给出以下三种初值方案：

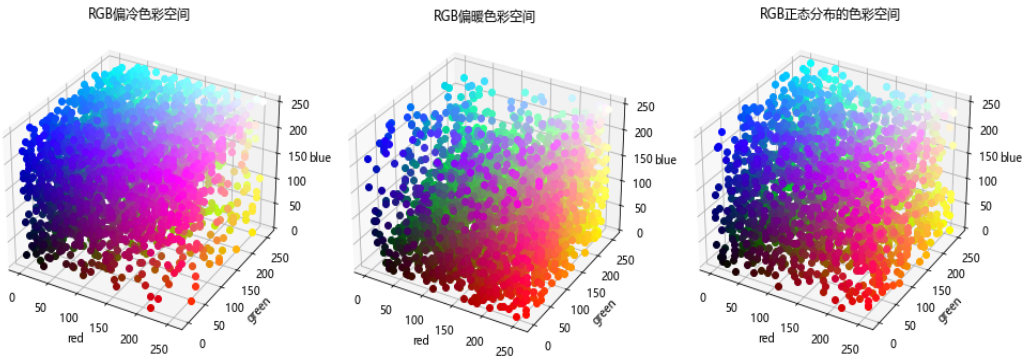

图 8: 三种不同色系的色彩空间

分别在三个色彩空间上运行改进后的 BKM 算法，我们可以得到最终的结果（详细见附录）。下面给出偏冷色系空间上的 1-10 个点的 RGB 值。

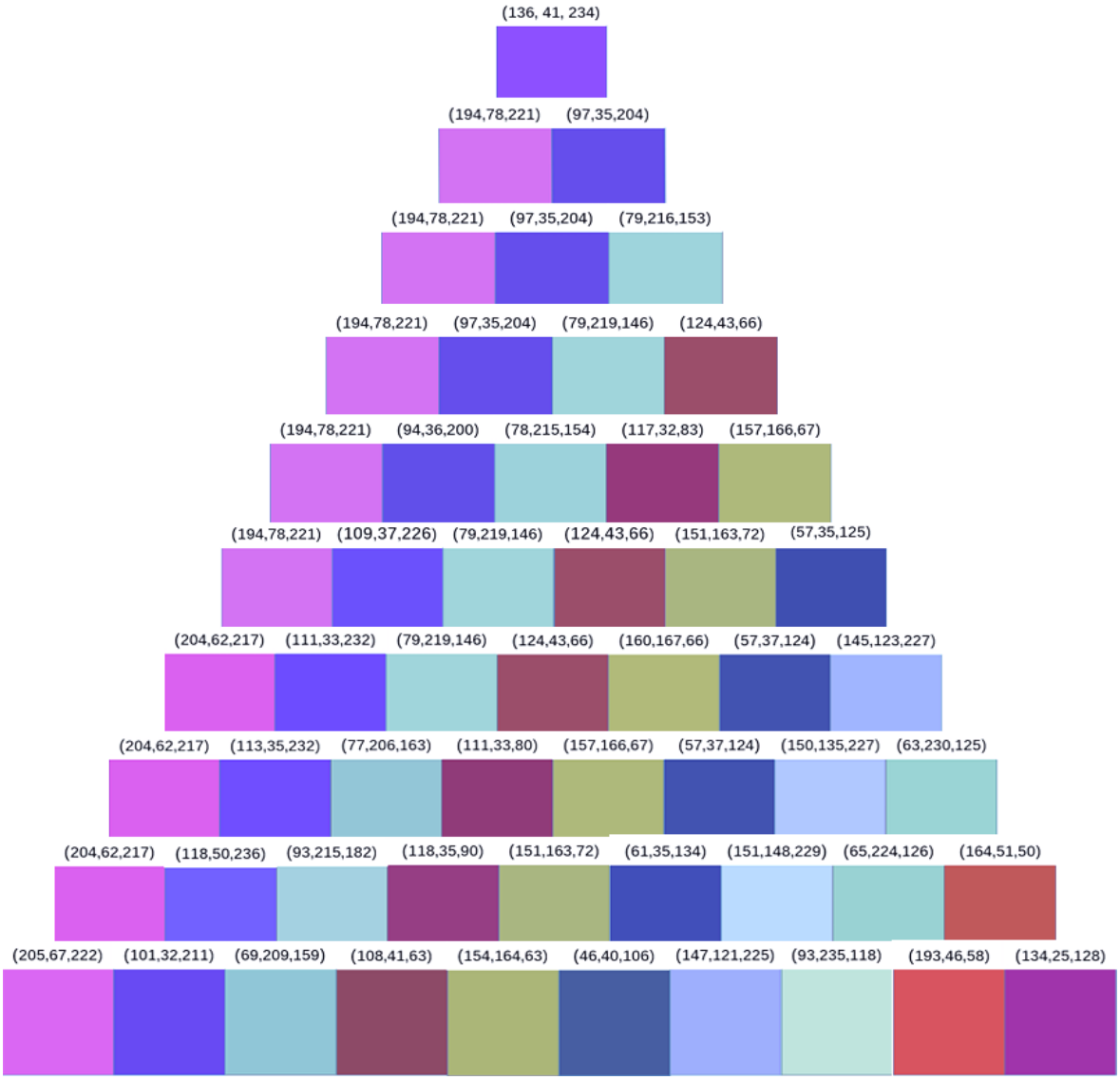

图 9: 偏冷色系空间上的 1-10 个点的 RGB 值

4.3 基于三种色系和曲线下降速率考虑成本与表现力最优规划

由于不考虑研发颜色难度，只考虑研制数量成本，问题三可视作以最少的颜色表现出最丰富的色彩的问题。我们由问题二提出的目标优化函数可以为三种方案画出不同的目标函数下降曲线如下：

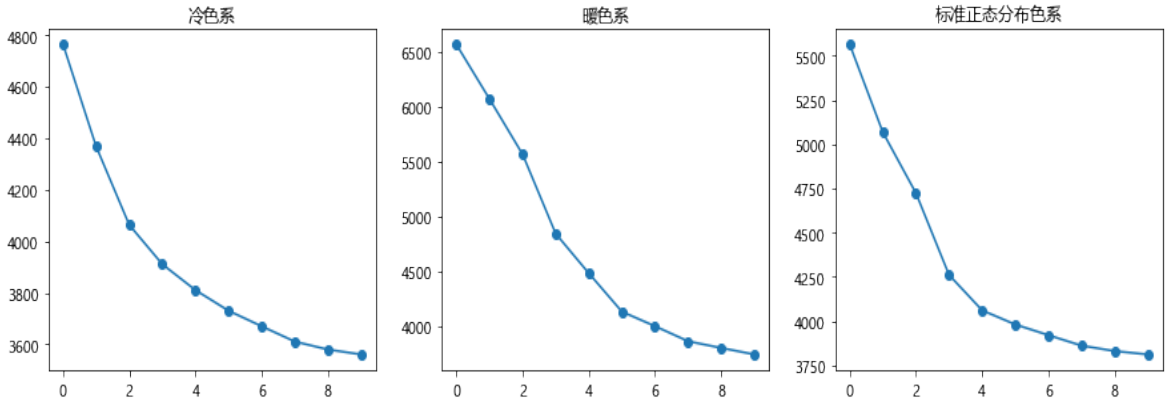

图 10: 三种色系目标函数下降曲线

上图横轴为研制数量，可以明显看到，在各色彩空间中，随着研制数量的增加，目标优化函数下降的值在不断减少，换言之，每增加一种颜色，厂商生产的马赛克瓷砖表现力就会多一分，但是随着增加数量越来越多，马赛克瓷砖表现力会趋于一个极限，这样就导致了后续的研发性价比不高。

图 11: 从左至右分别为 8 色-256 色-sRGB 色系

因此根据“手肘法”原则，我们可以清楚的得到冷色系肘点为 4 种颜色，暖色系肘点为 6 种颜色，标准正态分布色系肘点为 5 种颜色。

问题三的解我们已经获得，考虑到该厂是作为业务需求提出的问题，接下来我们不妨来对各色系所适用的范围做一个深入的探讨。

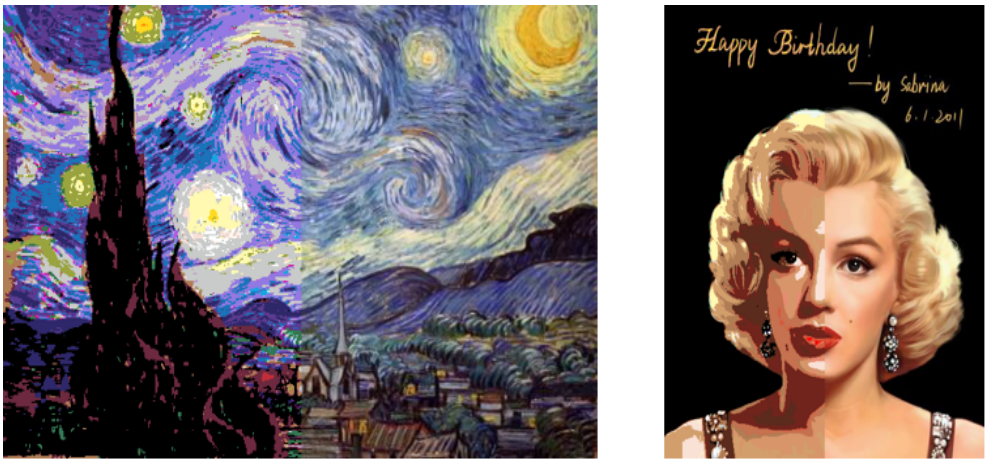

图 12：左冷色系方案，图左为经由 26 种颜色转换后的原图，右为原图
右暖色系方案，图左为经由 28 种颜色转换后的原图，右为原图

图 13：正态分布方案，图左为经由 27 种颜色转换后的原图，右为原图

基于现实装修风格考虑，人物肖像画由于人体皮肤色彩通常偏暖色系，而油画类等艺术作品则通常偏冷色系，一般的风景照片画面色彩则会更加趋于平衡系色彩，因此更符合正态分布的色彩空间。因此该厂选择研发的颜色种类时应考虑到自身的业务需求以及客户喜好，由于冷暖色系之间差别较大，如下图：

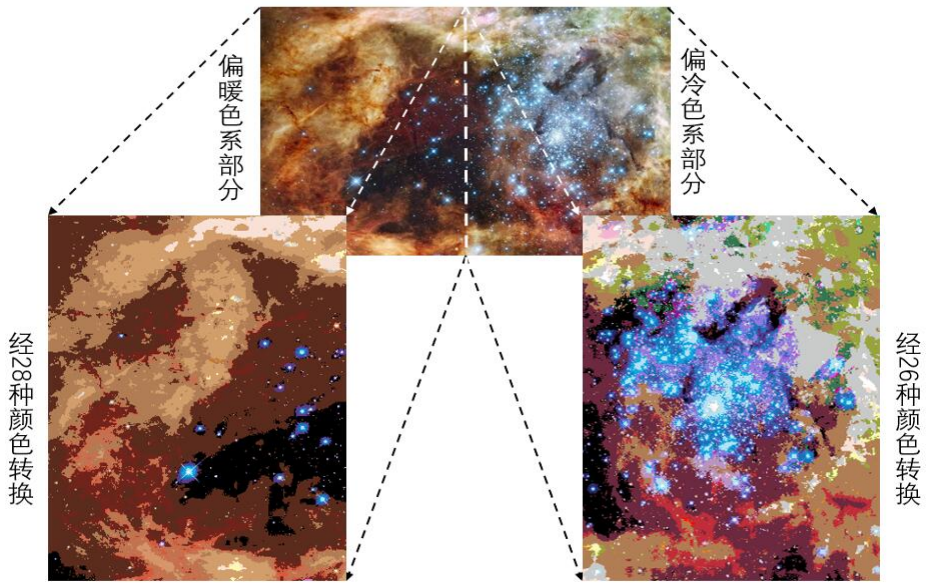

图 14

这张图的左半部分偏暖色，同时右半部分偏冷色。若是以冷暖色系分别处理，效果则会大大增强，而单单以一种色系来进行转换则无异于一场灾难。所以该厂还应考虑图片本身的色系不应过于复杂与糅杂，有色彩偏向性的图案往往能取得很不错的效果。

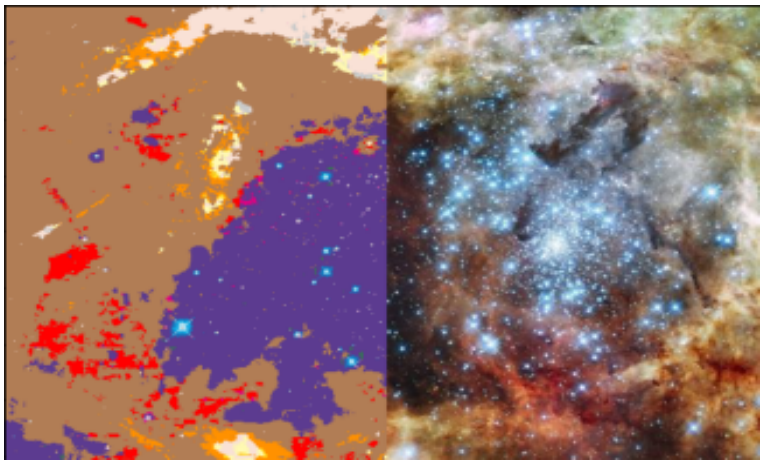

图 15: 左为以冷色系色彩转换偏暖色系区域的“灾难”，右为原图

5 模型的评价与展望

5.1 评价颜色差异的模型应用

- sRGB

由于大多数色彩差异的定义是色彩空间中的距离，确定距离的标准方法是欧几里得距离度量。如果目前有一个 RGB（红，绿，蓝）元组，并希望找到颜色差异，计算上最简单的是考虑 R,G,B 线性维度定义的颜色空间。当一种颜色与另一种颜色相比较时，这种方法就可以奏效，只需要知道距离是否更大即可。如果这些颜色距离的平方被求和，这样的度量有效地成为颜色距离的方差，公式如下:

$$\Delta R = \sqrt{(R_2 - R_1)^2 + (G_2 - G_1)^2 + (B_2 - B_1)^2} \tag{37}$$

- non-linear sRGB

但为了更好地适应人类的感知，人们曾多次尝试权衡 RGB 值，这些成分通常是加权的（红色 30% ，绿色 59% ，蓝色 11%），然而，这些成分在颜色测定方面明显较差，而且恰好是这些颜色亮度的贡献，而不是人类视觉对这些颜色的容忍度较低的程度。更接近的近似值会更准确（对于非线性的 sRGB，使用 0-255 的颜色范围），改进的公式如下:

$$\begin{cases} \sqrt{2 \times \Delta R^2 + 4 \times \Delta G^2 + 3 \times \Delta B^2} & \bar{R} < 128 \\ \sqrt{3 \times \Delta R^2 + 4 \times \Delta G^2 + 2 \times \Delta B^2} & \text{otherwise} \end{cases} \tag{38}$$

- redmean

低成本近似，更加平滑，公式如下:

$$\begin{aligned} &\bar{r} = \tfrac{R_1 + R_2}{2} \\ &\Delta C = \sqrt{\left(2 + \tfrac{\bar{r}}{256}\right) \times \Delta R^2 + 4 \times \Delta G^2 + \left(2 + \tfrac{255 - \bar{r}}{256}\right) \times \Delta B^2} \end{aligned} \tag{39}$$

- CIEDE2000

公式和用法在上面已经叙述过了，主要是通过 l*c*h 的差异引起的价值和亮度，色度，色调的补偿来评价颜色差异。

5.2 评价图案表现力的模型应用

5.2.1 颜色和谐参数和的应用

通过添加和谐参数中的对立和谐，分离互补和谐，三角和谐，相邻和谐，四角和谐来给总模型添加新的损失函数，由于这些和谐取值越大，说明越互补，那么用于生成具有更富有表现力的颜色来说是更好的，这个参数可以在颜色差异较小的时候让模型感知更富有表现力的颜色，让在仅有的成本下更大的选择较小的颜色作为聚类中心。

5.2.2 基于多球分裂的增量式 k-means 聚类算法

算法从给定的初始聚类中心个数 k_0 开始，按照固定步长 k 逐渐增加中心数目，直至得到 k 个初始中心。与传统增量聚类利用距离关系来逐一增加聚类中心不同，算法根据球聚类大小来一次性产生多个增量聚类中心，避免了所有 n 个 s 维数据之间的距离计算，将传统增量聚类中心选取过程的计算量由对比实验结果表明，算法降低了 BKM 的初始化中心敏感性，而该方法改用于球聚类半径的大小关系进行增量聚类中心的选取，分裂的球聚类对其它球聚类的大小没有影响，而且不影响其他增量聚类中心的选择，所以可以同时分裂多个球聚类，一次同时产生多个增量中心。

5.3 模型优缺点

5.3.1 以 RGB 标准的色差评价

以 RGB 标准在已有瓷砖的基础上通过三个色差评价后近似的图，几乎一样，放一张供参考:

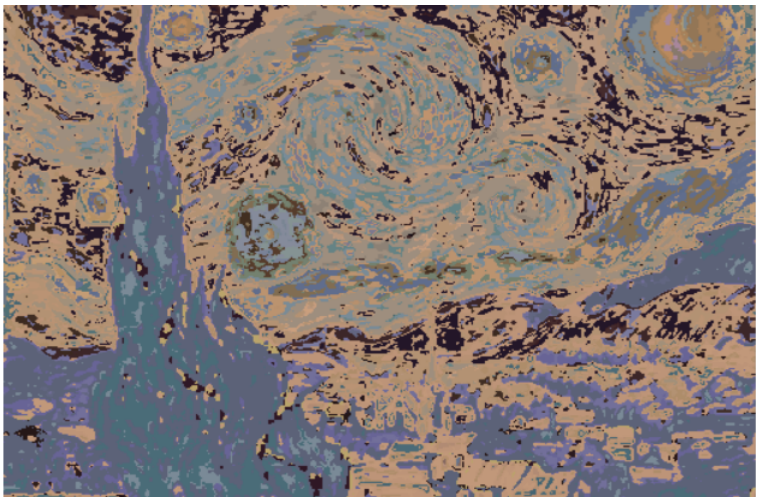

图 16: 以 RGB 颜色空间标准转换的图案

优点:

(1) 这种呈现方式更适合打印，在颜色成本上付出的要少（因为选择的色域有限）。

(2) 以冷色调为主，在一些冷色调为主的图片上，且能使用的色域有限的情况下，用 sRGB 类似模型可能是个比较节省开支的选择。

(3) 模型时间复杂度很低,O(1) 下就可以计算得出，不会用到矩阵运算和各种变换。

缺点:

(1) 我们的模型任务是在已有的颜色下且考虑成本下选择较少的新颜色来提高表现图片时的表现力，所以 sRGB 模型非常的劣势，我们仅在测试的时候使用。

5.3.2 以 LAB 标准的 CIEDE2000 色差评价

以 LAB 标准的 CIEDE2000 色差评价后近似的图，放一张供参考:

图 17: 以 LAB 颜色空间标准转换的图案

优点:

(1) 该模型颜色较为丰富，在已有的颜色下表现力提高不少。

(2) 冷暖色平均，对于一幅颜色复杂的图片可以更好的拼凑，成本要少很多。

缺点:

(1) 模型时间复杂度有点大,$O(n^2)$ 下就可以计算得出，会用到矩阵运算转换 RGB 色域到 LAB 色域和各种变换，计算精度要求很高，跑模型时间较长。

5.3.3 基于多球分裂的增量式 k-means 聚类算法:

优点:

(1) 在考虑聚类中心初始化问题的时候，最新提出的增量聚类算法，能够更精确的计算出来，但是复杂度却远大于原始的 k-means 算法，单是选择一个增量中心，就要计算两两数据之间的距离，单个中心计算量就是 $O(n^2)$，而最近新的 BKM 算法可以通过聚类平移加快计算。

(2) 传统的增量聚类每次只产生一个新的中心，而 ICBMS 算法可以一次同时产生多个新中心，减少了增量过程中运行 BKM 的次数。

(3) 与传统增量聚类相比，迭代计算量更小。传统增量聚类迭代过程采用的是 k-means 算法，而我们采用的是 BKM 算法，其是一种具有良好加速效果的 k-means 算法，且具有效率优势，效率最大提升了 437 倍，这可以大大缩短模型计算时间。

缺点:

相比于比较传统的 k-means 等模型几乎没有任何方面的劣势，甚至在数据量极大的时候加速效果非常明显。

5.4 基于改进 BKM 算法的推广

考虑到工厂的业务需求是时时刻刻变化的，由问题三的求解我们可以知道，想在一开始就获得最优的解是不合理且不可能的。因此我们需要考虑，随着工厂业务扩大，即使已经研发过一次新颜色，不妨假设工厂选择了偏冷色系颜色的研发，工厂的瓷砖颜色却也已经在表现力上逐渐下降，我们不妨认为是由于扩宽业务所带来的偏暖色系图案较多，这一点同样在问题三得到了证明。这时我们就可视其为一个增量聚类问题。

用我们提出的改进的 BKM 算法处理即将到来的增量数据，即业务新增的 RGB 值。当然，增量数据会对聚类结果产生影响而不会对当前的类集合产生广泛的影响。色彩空间更新以后，有四种可能性，见下图：

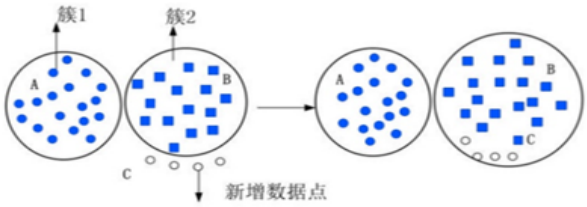

图 18: 增量 RGB 加入已有集合中

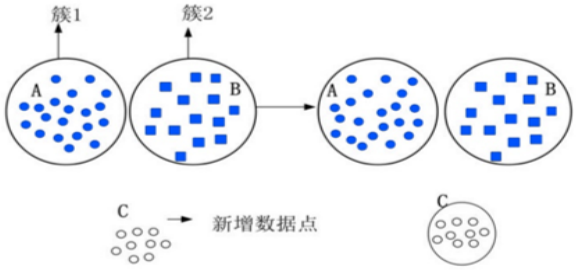

图 19: 增量 RGB 与已有集合相似性较低，生成一个新集合

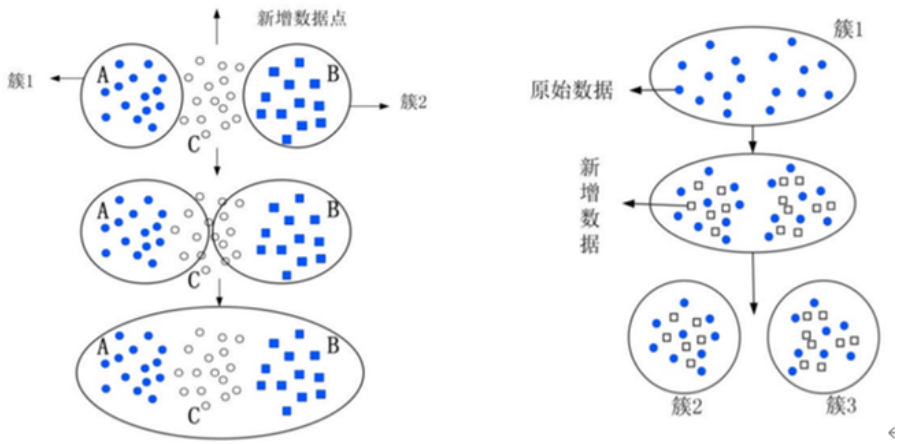

图 20: 左为增量 RGB 改变原有两个集合数据之间的相似性，合并两个已有集合的情况
右为增量 RGB 改变原有两个集合数据之间的相似性，合并两个已有集合

用改进的 BKM 算法进行聚类优化。目标优化函数由下图所示：

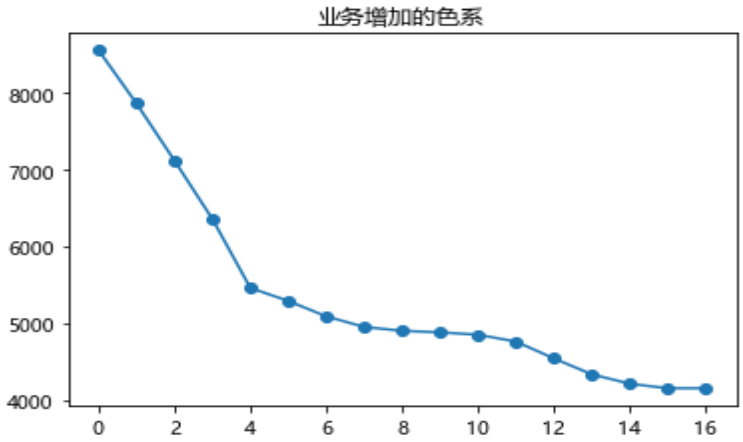

图 21

可以发现，我们设计的算法依然在较短的次数里达到了收敛的效果，这证明了我们算法的高效率与优越性。因此，我们有理由相信，我们改进的 BKM 算法并不只有纯理论上的价值，在实际业务场景中他也能发挥巨大作用。

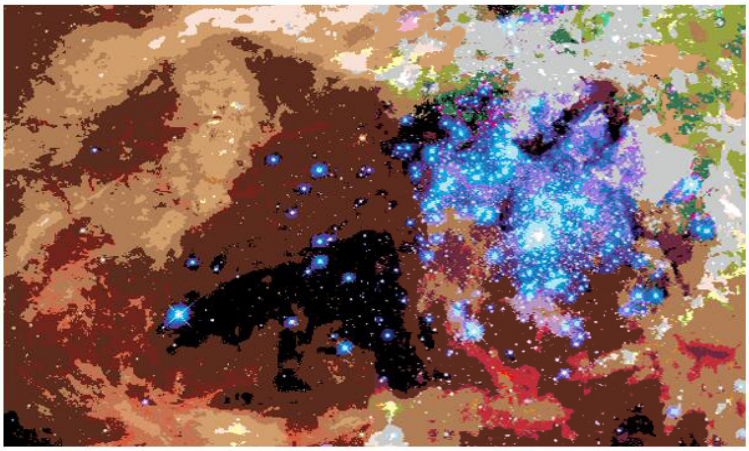

图 22：经再次研发后的 33 种颜色转换的星空图，无论在暖色系部分还是冷色系部分都取得了不错的效果

参考文献

[1] 维基百科, “Cielab 色彩空间 — 维基百科, 自由的百科全书,” 2021, [Online; accessed 204 2021]. [Online]. Available: -{R|https://zh.wikipedia.org/w/index.php?title=CIELAB%E8%89%B2%E5%BD%A9%E7%A9%BA%E9%97%B4&oldid=65033533}-

[2] ——, “颜色差异 — 维基百科, 自由的百科全书,” 2021, [Online; accessed 2104 2021]. [Online]. Available: -{R|https://zh.wikipedia.org/w/index.php?title=%E9%A2%9C%E8%89%B2%E5%B7%AE%E5%BC%82&oldid=65300269}-

[3] A. M. Bagirov, “Modified global k-means algorithm for minimum sum-of-squares clustering problems,” Pattern Recognition, vol. 41, no. 10, pp. 3192 - 3199, 2008. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031320308001362

[4] A. Likas, N. Vlassis, and J. J. Verbeek, “The global k-means clustering algorithm,” Pattern Recognition, vol. 36, no. 2, pp. 451 - 461, 2003, biometrics. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031320302000602

[5] S. Xia, D. Peng, D. Meng, C. Zhang, G. Wang, E. Giem, W. Wei, and Z. Chen, “A fast adaptive k-means with no bounds,” IEEE Transactions on Pattern Analysis and Machine Intelligence, pp. 1 - 1, 2020.

附录

问题二中，优先考虑研发的十种暖色和正态色系瓷砖的 RGB 值如下

暖：[238,248,241], [210,159,108], [233,139,116], [216,149,106], [113,36,28], [91,43,29], [216,89,61], [251,253,90], [59,250,124], [198,113,79]

正态：[254, 244, 226]，[96, 206, 210]，[237, 117, 85]，[165, 237, 217]，[243, 182, 60]，[233, 219, 201]，[90, 49, 85]，[236, 151, 54]，[238, 189, 199]，[128, 71, 126]

python 代码如下所示：

```
import numpy as np
import pandas as pd
import scipy as sci
import re
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
from mpl_toolkits.mplot3d import Axes3D
from colormath.color_objects import sRGBColor, LabColor
from colormath.color_conversions import convert_color
from colormath.color_diff import delta_e_cie2000
import matplotlib.patches as mpathes
from scipy.spatial.distance import pdist

warnings.filterwarnings('ignore')

f = open("附件2：图像1颜色列表.txt",encoding='utf8')
# f = open("附件3：图像2颜色列表.txt",encoding='utf8')
line = f.readline()
rgbs=[]
while line:
    nums=re.findall(r'\d+', line)
    if len(nums)==0:
        line = f.readline()
        continue
#     print(line,nums[1:4])
    rgb=list(map(int,nums[1:4]))
    rgbs.append(sRGBColor(rgb[0],rgb[1],rgb[2]))
    line = f.readline()
f.close()
cur_rgbs=[
    [0,0,0],
```

```
    [255,255,255],
    [255,0,0],
    [246,232,9],
    [72,176,64],
    [27,115,186],
    [53,118,84],
    [244,181,208],
    [255,145,0],
    [177,125,85],
    [92,59,144],
    [11,222,222],
    [228,0,130],
    [255,218,32],
    [118,238,0],
    [17,168,226],
    [255,110,0],
    [201,202,202],
    [255,249,177],
    [179,226,242],
    [249,225,214],
    [186,149,195]
]
for i in range(len(cur_rgbs)):
    cur_rgbs[i]=sRGBColor(cur_rgbs[i][0],cur_rgbs[i
      ][1],cur_rgbs[i][2])

def indexofMin(arr):
    minindex = 0
    currentindex = 1
    while currentindex < len(arr):
        if arr[currentindex] < arr[minindex]:
            minindex = currentindex
        currentindex += 1
    return minindex

def color_plot(rgbs,changed=True):
    fig=plt.figure()
    ax= Axes3D(fig)
```

```
    for rgb in rgbs:
        if changed==False:
            rgb=rgb.get_value_tuple()
        ax.plot(rgb[0],rgb[1],rgb[2], "o", color=np.
           array(rgb)/255)
    return ax

rgb_chi={}
rgb_chi1={}
for rgb in rgbs:
    lab=convert_color(rgb,LabColor)
    distance=[]
    distance2=[]
    for c_rgb in cur_rgbs:
        c_lab=convert_color(c_rgb,LabColor)
        distance.append(delta_e_cie2000(lab,c_lab))
        distance2.append(pdist([rgb.get_value_tuple(),
           c_rgb.get_value_tuple()]))
    print(indexofMin(distance))
    print(indexofMin(distance2))
    if cur_rgbs[indexofMin(distance)].get_value_tuple()
       in rgb_chi:
        rgb_chi[cur_rgbs[indexofMin(distance)].
           get_value_tuple()].append(rgb.
           get_value_tuple())
    else:
        rgb_chi[cur_rgbs[indexofMin(distance)].
           get_value_tuple()]=[]
        rgb_chi[cur_rgbs[indexofMin(distance)].
           get_value_tuple()].append(rgb.
           get_value_tuple())
    if cur_rgbs[indexofMin(distance2)].get_value_tuple
       () in rgb_chi1:
        rgb_chi1[cur_rgbs[indexofMin(distance2)].
           get_value_tuple()].append(rgb.
           get_value_tuple())
    else:
        rgb_chi1[cur_rgbs[indexofMin(distance2)].
```

```
            get_value_tuple()]=[]
        rgb_chi1[cur_rgbs[indexofMin(distance2)].
            get_value_tuple()].append(rgb.
            get_value_tuple())

import colorsys

def get_hsv(rgb):
    r=rgb[0]
    g=rgb[1]
    b=rgb[2]
    return colorsys.rgb_to_hsv(r, g, b)

def get_hsv1(rgb):
    rgb=rgb[0]
    r=rgb[0]
    g=rgb[1]
    b=rgb[2]
    return colorsys.rgb_to_hsv(r, g, b)

def get_lab(rgb):
    r=rgb[0]
    g=rgb[1]
    b=rgb[2]
    lab=convert_color(sRGBColor(r,g,b),LabColor).
        get_value_tuple()
    return lab

def get_lab1(rgb):
    rgb=rgb[0]
    r=rgb[0]
    g=rgb[1]
    b=rgb[2]
    lab=convert_color(sRGBColor(r,g,b),LabColor).
        get_value_tuple()
    return lab

r_hsv=list(map(lambda x:x.get_value_tuple(),rgbs))
```

```
r_hsv.sort(key=get_hsv)

r_lab=list(map(lambda x:x.get_value_tuple(),rgbs))
r_lab.sort(key=get_lab)

tmp=sorted(rgb_chi1.items(), key=lambda d:d[0])
tmp.sort(key=get_hsv1)

tmp1=sorted(rgb_chi.items(), key=lambda d:d[0])
tmp1.sort(key=get_lab1)

x=0
y=0
length=0.005
for key in tmp:
    r_s=key[1]
    for i in range(len(r_s)):
        rect = mpathes.Rectangle([x,y],length,1,color=
           np.array(r_s[i])/255)
        x+=length
    #     if i%14==0:
    #         x=0
    #         y+=length
        plt.gca().add_patch(rect)
plt.title("由Delta-2000相似度排序的原色域")
plt.show()

x=0
y=0
length=0.005
for key in tmp:
    r_s=key[1]
    for i in range(len(r_s)):
        rect = mpathes.Rectangle([x,y],length,1,color=
           np.array(key[0])/255)
        x+=length
    #     if i%14==0:
    #         x=0
```

```
    #         y+=length
        plt.gca().add_patch(rect)
plt.title("转换后的由Delta-2000相似度排序的原色域")
plt.show()

x=0
y=0
length=0.005
for key in tmp1:
    r_s=key[1]
    for i in range(len(r_s)):
        rect = mpathes.Rectangle([x,y],length,1,color=
           np.array(r_s[i])/255)
        x+=length
    #     if i%14==0:
    #         x=0
    #         y+=length
        plt.gca().add_patch(rect)
plt.title("由Delta-2000相似度排序的原色域")
plt.show()

x=0
y=0
length=0.005
for key in tmp1:
    r_s=key[1]
    for i in range(len(r_s)):
        rect = mpathes.Rectangle([x,y],length,1,color=
           np.array(key[0])/255)
        x+=length
    #     if i%14==0:
    #         x=0
    #         y+=length
        plt.gca().add_patch(rect)
plt.title("转换后的由Delta-2000相似度排序的原色域")

fig = plt.figure()  # 定义新的三维坐标轴
ax = Axes3D(fig)
```

```
size = 25
points = np.linspace(0, 255, size).astype(np.int32)

for x in points:
    for y in points:
        for z in points:
            if (x>200)&(y<200)&(z>200):
                continue
            ax.plot([x], [y], [z], "ro", color=(x /
               255, y / 255, z / 255, 1))
print('---')
ax.set_xlabel('red')
ax.set_ylabel('green')
ax.set_zlabel('blue')
plt.title("RGB色彩空间")
plt.show()

fig = plt.figure()  # 定义新的三维坐标轴
ax = Axes3D(fig)

rgbs=[]
labs=[]
for x in points:
    for y in points:
        for z in points:
            if (x>200)&(y<200)&(z>200):
                continue
            rgb=sRGBColor(x,y,z,is_upscaled=True)
            lab=convert_color(rgb,LabColor).
               get_value_tuple()
            ax.plot(lab[1],lab[2],lab[0], "o", color=(x
                / 255, y / 255, z / 255, 1))
print('---')
ax.set_xlabel('a')
ax.set_ylabel('b')
ax.set_zlabel('l')
plt.title("Lab色彩空间")
```

```
plt.show()

fig = plt.figure()  # 定义新的三维坐标轴
ax = Axes3D(fig)
import random

size = 25
points = np.linspace(0, 255, size).astype(np.int32)
print(random.random())
for x in points:
    for y in points:
        for z in points:
            if ((x+y)/2>z):
                if random.random()>0.05:
                    continue
            else:
                if random.random()>0.3:
                    continue
            ax.plot([x], [y], [z], "ro", color=(x /
              255, y / 255, z / 255, 1))
print('---')
ax.set_xlabel('red')
ax.set_ylabel('green')
ax.set_zlabel('blue')
plt.title("RGB偏冷色彩空间")
plt.show()

fig = plt.figure()  # 定义新的三维坐标轴
ax = Axes3D(fig)
import random

size = 25
points = np.linspace(0, 255, size).astype(np.int32)
print(random.random())
for x in points:
    for y in points:
        for z in points:
            if ((x+y)/2<z):
```

```
                if random.random()>0.05:
                    continue
            else:
                if random.random()>0.3:
                    continue
            ax.plot([x], [y], [z], "ro", color=(x /
               255, y / 255, z / 255, 1))
print('---')
ax.set_xlabel('red')
ax.set_ylabel('green')
ax.set_zlabel('blue')
plt.title("RGB偏暖色彩空间")
plt.show()

fig = plt.figure()  # 定义新的三维坐标轴
ax = Axes3D(fig)
import random

size = 25
points = np.linspace(0, 255, size).astype(np.int32)
print(random.random())
for x in points:
    for y in points:
        for z in points:
            if ((x+y)/2<z):
                if random.random()>0.15:
                    continue
            else:
                if random.random()>0.15:
                    continue
            ax.plot([x], [y], [z], "ro", color=(x /
               255, y / 255, z / 255, 1))
print('---')
ax.set_xlabel('red')
ax.set_ylabel('green')
ax.set_zlabel('blue')
plt.title("RGB正态分布的色彩空间")
plt.show()
```

```
import numpy as np
import pandas as pd
import scipy as sci
import re
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
from mpl_toolkits.mplot3d import Axes3D
from colormath.color_objects import sRGBColor, LabColor
from colormath.color_conversions import convert_color
from colormath.color_diff import delta_e_cie2000
import matplotlib.patches as mpathes
from scipy.spatial.distance import pdist
from scipy.optimize import basinhopping
import random
from sklearn.cluster import KMeans

warnings.filterwarnings('ignore')

f = open("附件2：图像1颜色列表.txt",encoding='utf8')
line = f.readline()
rgbs=[]
while line:
    nums=re.findall(r'\d+', line)
    if len(nums)==0:
        line = f.readline()
        continue
#     print(line,nums[1:4])
    rgb=list(map(int,nums[1:4]))
    rgbs.append(sRGBColor(rgb[0],rgb[1],rgb[2],
      is_upscaled=True))
    line = f.readline()
f.close()
f = open("附件3：图像2颜色列表.txt",encoding='utf8')
line = f.readline()
while line:
    nums=re.findall(r'\d+', line)
```

```
    if len(nums)==0:
        line = f.readline()
        continue
#     print(line,nums[1:4])
    rgb=list(map(int,nums[1:4]))
    rgbs.append(sRGBColor(rgb[0],rgb[1],rgb[2],
      is_upscaled=True))
    line = f.readline()
f.close()
cur_rgbs=[
    [0,0,0],
    [255,255,255],
    [255,0,0],
    [246,232,9],
    [72,176,64],
    [27,115,186],
    [53,118,84],
    [244,181,208],
    [255,145,0],
    [177,125,85],
    [92,59,144],
    [11,222,222],
    [228,0,130],
    [255,218,32],
    [118,238,0],
    [249,225,214],
    [186,149,195]
]
for i in range(len(cur_rgbs)):
    cur_rgbs[i]=sRGBColor(cur_rgbs[i][0],cur_rgbs[i
      ][1],cur_rgbs[i][2],is_upscaled=True)

cur_rgbs=[
    sRGBColor (11.0, 222.0, 222.0,is_upscaled=True),
    sRGBColor (118.0, 238.0, 0.0,is_upscaled=True),
    sRGBColor (27.0, 115.0, 186.0,is_upscaled=True),
    sRGBColor (72.0, 176.0, 64.0,is_upscaled=True),
    sRGBColor (92.0, 59.0, 144.0,is_upscaled=True),
```

```
    sRGBColor (53.0, 118.0, 84.0,is_upscaled=True),
    sRGBColor (17.0, 168.0, 226.0,is_upscaled=True),
    sRGBColor (179.0, 226.0, 242.0,is_upscaled=True),
    sRGBColor (255.0, 0.0, 0.0,is_upscaled=True),
    sRGBColor (177.0, 125.0, 85.0,is_upscaled=True),
    sRGBColor (201.0, 202.0, 202.0,is_upscaled=True),
    sRGBColor (228.0, 0.0, 130.0,is_upscaled=True),
    sRGBColor (246.0, 232.0, 9.0,is_upscaled=True),
    sRGBColor (255.0, 218.0, 32.0,is_upscaled=True),
    sRGBColor (186.0, 149.0, 195.0,is_upscaled=True),
    sRGBColor (249.0, 225.0, 214.0,is_upscaled=True),
    sRGBColor (255.0, 145.0, 0.0,is_upscaled=True),
    sRGBColor (255.0, 249.0, 177.0,is_upscaled=True),
    sRGBColor (255.0, 110.0, 0.0,is_upscaled=True),
    sRGBColor (244.0, 181.0, 208.0,is_upscaled=True)
]

def func(x):
    tmp_rgbs=list(cur_rgbs)
    distan_min=0
    t={} #与x距离最近的点数
    print(x)
    for xx in x:
#         print(x[i],x[i+1],x[i+2])
#         tmp_rgbs.append(sRGBColor(xx[0],xx[1],xx[2]))
        tmp_rgbs.append(LabColor(xx[0],xx[1],xx[2]))
        t[len(tmp_rgbs)-1]=[]
    for rgb in rgbs:
        lab=convert_color(rgb,LabColor)
        distance=[]
        for c_rgb in tmp_rgbs:
            c_lab=convert_color(c_rgb,LabColor)
            distance.append(delta_e_cie2000(lab,c_lab))
        distan_min+=min(distance)
        if indexofMin(distance)>len(tmp_rgbs)-1-len(x):
            t[indexofMin(distance)].append(np.array(lab
               .get_value_tuple()))
    for tt in t.keys():
```

```
            if len(t[tt])<2:
                t=False
                break
        print(distan_min)
        return distan_min,t

max(map(len,rgb_chi.values()))
def max_cluster(cl):
    maxx=0
    max_i=0
    for key in cl.keys():
        tmp=len(cl[key])
        if tmp>maxx:
            maxx=tmp
            max_i=key
    return max_i
def max_distance(cl,dot):
    dot=convert_color(tuple_rgb(dot),LabColor)
    maxx=0
    max_i=0
    for pot in cl:
        pot1=convert_color(tuple_rgb(pot),LabColor)
        tmp=delta_e_cie2000(dot,pot1)
        if tmp>maxx:
            maxx=tmp
            max_i=pot
    return max_i
def tuple_rgb(x):
    return sRGBColor(x[0],x[1],x[2])
def tuple_lab(x):
    return LabColor(x[0],x[1],x[2])
def rgb_lab(x):
    return convert_color(x,LabColor)
def lab_rgb(x):
    return convert_color(x,sRGBColor)

# print(rgb_chi[max_cluster(rgb_chi)])
tmp1=max_distance(rgb_chi[max_cluster(rgb_chi)],
```

```
   max_cluster(rgb_chi))
tmp2=max_distance(rgb_chi[max_cluster(rgb_chi)],tmp1)
value,poss=func([rgb_lab(tuple_rgb(tmp1)).
   get_value_tuple(),rgb_lab(tuple_rgb(tmp2)).
   get_value_tuple()])
tmp=[]
for pos in poss.values():
    tmp.append(np.mean(pos,axis=0))
value,poss=func(tmp)
tmp=[]
for pos in poss.values():
    tmp.append(np.mean(pos,axis=0))
value,poss=func(tmp)
tmp=[]
for pos in poss.values():
    tmp.append(np.mean(pos,axis=0))
value,poss=func(tmp)
tmp=[]
for pos in poss.values():
    tmp.append(np.mean(pos,axis=0))
value,poss=func(tmp)
tmp=[]
for pos in poss.values():
    tmp.append(np.mean(pos,axis=0))
value,poss=func(tmp)
tmp=[]
for pos in poss.values():
    tmp.append(np.mean(pos,axis=0))
value,poss=func(tmp)
tmp=[]
for pos in poss.values():
    tmp.append(np.mean(pos,axis=0))
value,poss=func(tmp)
tmp=[]
for pos in poss.values():
    tmp.append(np.mean(pos,axis=0))
value,poss=func(tmp)
tmp=[]
```

```
for pos in poss.values():
    tmp.append(np.mean(pos,axis=0))
value,poss=func(tmp)
tmp=[]
for pos in poss.values():
    tmp.append(np.mean(pos,axis=0))
value,poss=func(tmp)

class Particle:
    # 初始化
    def __init__(self, baseParticle, x_max, max_vel,
       dim):
#         self.__pos = [random.uniform(0, 255)for i in
   range(dim)]  # 粒子的位置
        self.__pos = [random.uniform(0, 100),random.
           uniform(-128, 127),random.uniform(-128, 127)
           ]  # 粒子的位置
        self.__vel = [random.uniform(-max_vel, max_vel)
            for i in range(dim)]  # 粒子的速度
        self.__bestPos = [0.0 for i in range(dim)]  #
           粒子最好的位置
        self.__fitnessValue = fit_fun(self.__pos)  # 适
           应度函数值

    def set_pos(self, i, value):
        self.__pos[i] = value

    def get_pos(self):
        return self.__pos

    def set_best_pos(self, i, value):
        self.__bestPos[i] = value

    def get_best_pos(self):
        return self.__bestPos

    def set_vel(self, i, value):
        self.__vel[i] = value
```

```
    def get_vel(self):
        return self.__vel

    def set_fitness_value(self, value):
        self.__fitnessValue = value

    def get_fitness_value(self):
        return self.__fitnessValue


class PSO:
    def __init__(self,baseParticle, dim, size, iter_num
       , x_max, max_vel, best_fitness_value=float('Inf'
       ), C1=2, C2=2, W=1):
        self.C1 = C1
        self.C2 = C2
        self.W = W
        self.dim = dim  # 粒子的维度
        self.size = size  # 粒子个数
        self.iter_num = iter_num  # 迭代次数
        self.x_max = x_max
        self.max_vel = max_vel  # 粒子最大速度
        self.best_fitness_value = func(baseParticle)
        self.best_position = [i for i in baseParticle]
            # 种群最优位置
        self.fitness_val_list = []  # 每次迭代最优适应
           值

        # 对种群进行初始化
        self.Particle_list = [Particle(baseParticle,
           self.x_max, self.max_vel, self.dim) for i in
            range(self.size)]

    def set_bestFitnessValue(self, value):
        self.best_fitness_value = value

    def get_bestFitnessValue(self):
```

```
        return self.best_fitness_value

    def set_bestPosition(self, i, value):
        self.best_position[i] = value

    def get_bestPosition(self):
        return self.best_position

    # 更新速度
    def update_vel(self, part):
        for i in range(self.dim):
            vel_value = self.W * part.get_vel()[i] +
               self.C1 * random.random() * (part.
               get_best_pos()[i] - part.get_pos()[i]) \
                        + self.C2 * random.random() * (
                           self.get_bestPosition()[i] -
                            part.get_pos()[i])
            if vel_value > self.max_vel:
                vel_value = self.max_vel
            elif vel_value < -self.max_vel:
                vel_value = -self.max_vel
            part.set_vel(i, vel_value)


    # 更新位置
    def update_pos(self, part):
        for i in range(self.dim):
            pos_value = part.get_pos()[i] + part.
               get_vel()[i]
            part.set_pos(i, pos_value)
        value = fit_fun(part.get_pos())
        if value < part.get_fitness_value():
            part.set_fitness_value(value)
            for i in range(self.dim):
                part.set_best_pos(i, part.get_pos()[i])
        if value < self.get_bestFitnessValue():
            self.set_bestFitnessValue(value)
            for i in range(self.dim):
```

```
                self.set_bestPosition(i, part.get_pos()
                    [i])
            print(self.get_bestFitnessValue(),self.
                fitness_val_list, self.get_bestPosition())

    def update(self):
        for i in range(self.iter_num):
            print(i)
            for part in self.Particle_list:
                self.update_vel(part)  # 更新速度
                self.update_pos(part)  # 更新位置
            self.fitness_val_list.append(self.
                get_bestFitnessValue())  # 每次迭代完把
                当前的最优适应度存到列表
        return self.fitness_val_list, self.
            get_bestPosition()


def fit_fun(X):  # 适应函数
    return func(X)

dim = 3
size = 20
iter_num = 100
x_max = 0.1
max_vel = 10

pso = PSO([41.623599583238224, 64.67420354604263,
    -81.18519615922087], dim, size, iter_num, x_max,
    max_vel)
fit_var_list, best_pos = pso.update()
print("最优位置:" + str(best_pos))
print("最优解:" + str(fit_var_list[-1]))
plt.plot(np.linspace(0, iter_num, iter_num),
    fit_var_list, alpha=0.5)
plt.show()

from PIL import Image
```

```
im=Image.open("玛丽莲.jpg")
im_array=np.array(im)
for w in range(len(im_array[0])):
    print(w)
    for l in range(len(im_array)):
        im_rgb=im_array[l][w]
        im_lab=convert_color(sRGBColor(im_rgb[0],im_rgb
           [1],im_rgb[2],is_upscaled=True),LabColor)
        distance=[]
        for c_rgb in cur_rgbs:
            c_lab=convert_color(c_rgb,LabColor)
            distance.append(delta_e_cie2000(im_lab,
               c_lab))
        im_array[l][w]=cur_rgbs[indexofMin(distance)].
           get_upscaled_value_tuple()
```