有损图像压缩——采用高效聚类的频繁序列挖掘视角

Avinash Kadimisetty、C. Oswald、B. Sivaselvan计算机工程系，IIITDM Kancheepuram，泰米尔纳德邦，印度电子邮件：{coe12b009, coe13d003, sivaselvanb}@iiitdm.ac.in

Kadimisetty Alekhya计算机科学与工程系，MVGRCE，安得拉邦，印度电子邮件：kalekhya366@gmail.com

arXiv:2601.18821v1 [eess.IV] 24 Jan 2026

摘要——本文探讨了频繁序列挖掘在有损图像压缩领域的应用前景。所提出的方法基于对像素进行聚类，并在压缩过程中使用簇标识符。JPEG中的DCT阶段被闭合频繁序列挖掘与*k*均值聚类相结合的方法取代，以有效处理冗余数据。本方法主要致力于对图像各分量的所有图像块并行应用*k*均值聚类，以缩短压缩时间。通过一种新颖的剪枝策略改进传统GSP算法，以优化模式基数，从而显著缩减编码表的大小。所提算法的仿真结果表明，与现有替代方法相比，该算法在压缩比和质量方面均有显著提升。

索引词—**闭合频繁序列挖掘、聚类、压缩比、图像质量、有损压缩**

I. 引言

技术进步催生了大量以文本、图像、音频、视频等形式存在的数据。每天都有海量数据通过互联网传输，而传输时间在很大程度上受数据量影响。存储如此大量的数据是一项艰巨的任务，因此产生了对压缩的需求。压缩过程会减少表示实际数据所需的比特数 [9]。压缩可分为无损压缩和有损压缩。在无损压缩中，可以从压缩后的数据还原出实际数据；压缩通过去除冗余来实现。常见示例包括霍夫曼编码、算术编码、游程编码、PNG 等 [1], [5], [13], [14]。相较之下，在有损压缩中，只能重建出与实际数据近似的数据。它通过去除冗余以及不相关信息来实现压缩。有损压缩的压缩效果优于无损压缩。典型示例包括 MP3、MPEG、JPEG、JPEG 2000、codec2、有损预测编码、GIF 等 [9], [10], [12]。图像压缩是指减少表示图像所需的数据量，其依据是相邻像素之间具有很强的相关性，这种现象称为空间冗余。

数据挖掘是从大型数据库中提取隐藏且有用信息的过程。Naren Ramakrishnan 等人提出了数据挖掘的五种视角：压缩、搜索、查询、归纳和近似 [8]。本文聚焦于数据挖掘的压缩视角。由于挖掘出的数据可以看作原始数据的精简表示，因此数据挖掘也可以视为一种压缩技术。数据挖掘技术包括关联规则挖掘、聚类、分类等 [4]。本文使用聚类和闭合频繁序列挖掘来压缩图像。

聚类或数据分割用于根据相似性将大型数据集划分为若干簇 [4]。一个簇是由数据对象构成的集合，其中的对象彼此相似，并且与其他簇中的对象不同。聚类算法可分为划分、层次、密度、网格和基于模型的方法。知名的聚类算法包括 k-均值、k中心点、AGNES、DIANA、BIRCH、DBSCAN 等 [4], [6], [7]。频繁序列挖掘是指在序列数据库中挖掘满足最小支持度阈值的完整序列集的过程 [16]。设 $J = \{j_1, j_2, \ldots, j_p\}$ 为项集。项集是 J的子集。序列 $\sigma = \langle s_1, s_2, \ldots, s_m \rangle (s_i \subseteq J)$ 是一个有序列表 [15]。序列$l(\sigma)$的长度是该序列中项集的总数。如果序列的长度为 l，则称其为 l序列。序列 $\sigma = \langle x_1, x_2, x_3, \ldots, x_m \rangle$ 是另一个序列 $\theta = \langle y_1, y_2, \ldots, y_n \rangle$ 的子序列，记为 $\sigma \sqsubseteq \theta$，当且仅当$\exists k_1, k_2, \ldots, k_m$，使得 $1 \leq k_1 < k_2 < \cdots < k_m \leq n$且 $x_1 \subseteq y_{k_1}, x_2 \subseteq y_{k_2}, \ldots, x_m \subseteq y_{k_m}$。$\theta$ 称为 σ 的超序列，因此 θ 包含 σ。在本研究中，序列被视为相对于索引有序且连续的。以文本“encyclopaedia”为例，按照传统定义，“ecy”是一个序列，但在我们的研究中，“ency”才是一个序列。

II. 相关工作

霍夫曼编码和算术编码通过消除像素编码中的冗余来压缩图像。霍夫曼码是一种最优码，出现概率较高的符号所对应的编码比出现概率较低的符号所对应的编码更短 [5]。虽然霍夫曼编码简单高效，因此广受欢迎，但由于它会为待编码文本字母表中的每个符号分配整数长度的编码，因此并非总能生成理想的编码。这个问题可以通过

算术编码技术则为整个输入文件分配一个编码。在算术编码中，最初会将每个输入文件表示为 0 到 1 之间的一个实数区间；每从输入文件中读取一个符号，该区间就会缩小 [14]。出现概率较高的符号所导致的区间缩小幅度小于出现概率较低的符号。较窄的区间需要更多位来表示，因此出现概率较高的符号会分配到较短的编码。

Lempel-Ziv-Welch（LZW）算法是一种基于字典的无损压缩技术，其字典初始化为待编码文本字母表中的所有符号[17]。LZW 通过将新短语保存到字典的空条目中，并用指向相应字典条目的指针替换输入文本中的短语来实现压缩。

图形交换格式（GIF）是对 LZW 编码技术进行修改后开发而成的，用于压缩图像数据。从本质上说，GIF 是一维的，它通过独立处理图像数据的每一行来压缩二维图像数据。然而，GIF 被认为效率低下，因为它无法发现相邻行像素之间的关联。变换编码是一种有损图像压缩技术，其做法是将原始图像数据划分为较小的图像块。每个图像块都会经过可逆变换处理，随后对变换后的数值进行量化，并使用游程编码、霍夫曼编码或其他类似技术对图像块进行编码。变换编码最广为人知的应用是用于有损图像压缩的 JPEG（联合图像专家组）标准，该标准采用离散余弦变换（DCT） [12]。JPEG 将彩色图像从 RGB 转换到亮度/色度颜色空间，并将图像组织为 8×8 图像块。对每个图像块应用 DCT 以获取频率分量，随后使用量化矩阵进行量化。最后，使用游程编码和霍夫曼编码的组合对结果进行编码。JPEG 压缩能够在保持良好质量的同时实现出色的压缩率。然而，有时 JPEG 算法重建出的图像会呈现块状外观。

III. 所提出的架构

A. 压缩器

1) 源图像：所提算法的流程如算法1所示。 I（源图像）在 RGB颜色空间中选取，并转换为 $m \times n \times 3$ 矩阵。I 被拆分为三个组件：红色（I_1）、绿色（I_2）和蓝色（I_3）。本文利用了各组件彼此独立这一特点。该算法并行应用于每个组件，以缩短压缩时间。下面以图像矩阵 $A_{8\times8}$ 为例，说明算法的执行过程。用于说明该算法的输入参数为：块大小 $b = 4$、聚类数$k = 4$、最小支持度 $\psi = 2$。图像被拆分为$n_b = 4$ 个图像块，例如 B_{A_1}、B_{A_2}、 B_{A_3}、 B_{A_4}。

算法1 压缩器

1: **输入：**

1) 源图像 $I_{m\times n\times 3}$**2) 块大小** b**、聚类数** k**、最小支持度**ψ**2: 输出：** C_I**，即压缩编码图像 3: 将** I **拆分为** I_1**、** I_2**、** I_3 **，分别对应红色、绿色和蓝色组件。4: 块数** n_b **=** $^{m\times n}_{b\times b}$ **5:** $i \leftarrow 1$**6: 对于 [1, 3]**

7 I_i n_b B_{i_1} $B_{i_2} \ldots$ $B_{i_{n_b}}$**中的 i 并**

8**行执行：拆分为图像块，，，：**$\overline{[T_{B_1}, T_{B_2}, \ldots, T_{B_{n_b}}]}$ **=并行聚类(**B_{i1}**,**B_{i2}**,. . . ,** B_{in_b}**,** k**) 9: 将每个图像块中的每个像素值替换为该像素所属簇的簇标识符。**

10 B_{i_1} $B_{i_2} \ldots -B_{i_{n_b}}$**：** $I_i^{'}$ **=合**

11 S_i **=** $I_i^{'}$ ψ

并所有图像块，，，：改进的GSP(,) 12: $S_i^{'}$ **=支持度改进(**S_i**,** $I_i^{'}$**) 13:** CT_i **=霍夫曼编码(**$S_i^{'}$**) 14: 对于**$S_i^{'}$**中的每个序列** σ**，将**$I_i^{'}$**中的** σ **替换为其霍夫曼码，以形成** C_I**15: 结束循环**

$$A = \begin{bmatrix} 139 & 108 & 108 & 108 & 108 & 108 & 108 & 121 \\ 177 & 165 & 121 & 139 & 139 & 139 & 121 & 95 \\ 239 & 183 & 139 & 139 & 165 & 139 & 108 & 108 \\ 225 & 165 & 121 & 165 & 165 & 151 & 108 & 108 \\ 165 & 139 & 121 & 152 & 151 & 134 & 121 & 121 \\ 113 & 121 & 108 & 121 & 139 & 165 & 183 & 139 \\ 108 & 90 & 95 & 134 & 121 & 121 & 165 & 151 \\ 95 & 121 & 108 & 108 & 108 & 121 & 139 & 121 \end{bmatrix}$$

$$B_{A_1} = \begin{bmatrix} 139 & 108 & 108 & 108 \\ 177 & 165 & 121 & 139 \\ 239 & 183 & 139 & 139 \\ 225 & 165 & 121 & 165 \end{bmatrix} \quad B_{A_2} = \begin{bmatrix} 108 & 108 & 108 & 121 \\ 139 & 139 & 121 & 95 \\ 165 & 139 & 108 & 108 \\ 165 & 151 & 108 & 108 \end{bmatrix}$$

$$B_{A_3} = \begin{bmatrix} 165 & 139 & 121 & 152 \\ 113 & 121 & 108 & 121 \\ 108 & 90 & 95 & 134 \\ 95 & 121 & 108 & 108 \end{bmatrix} \quad B_{A_4} = \begin{bmatrix} 151 & 134 & 121 & 121 \\ 139 & 165 & 183 & 139 \\ 121 & 121 & 165 & 151 \\ 108 & 121 & 139 & 121 \end{bmatrix}$$

$$A^{'} = \begin{bmatrix} 3 & 1 & 1 & 1 & 3 & 3 & 3 & 0 \\ 2 & 2 & 1 & 3 & 2 & 2 & 0 & 3 \\ 0 & 2 & 3 & 3 & 1 & 2 & 3 & 3 \\ 0 & 2 & 1 & 2 & 1 & 1 & 3 & 3 \\ 3 & 3 & 0 & 3 & 3 & 1 & 2 & 2 \\ 2 & 0 & 2 & 0 & 1 & 0 & 0 & 1 \\ 2 & 1 & 1 & 0 & 2 & 2 & 0 & 3 \\ 1 & 0 & 2 & 2 & 2 & 2 & 1 & 2 \end{bmatrix}$$

2）图像聚类：*k*-均值聚类旨在将给定的数据对象集划分为*k* 个簇。在图像中，每个像素都被视为一个数据对象。k-均值的第一步是最大化簇内相似度

标识符	Mean
0	232
1	113
2	171
3	139

(a) T_{B_1}

标识符	Mean
0	121
1	160
2	139
3	106

(b) T_{B_2}

标识符	Mean
0	123
1	93
2	109
3	152

(c) T_{B_3}

标识符	Mean
0	171
1	137
2	119
3	151

(d) T_{B_4}

表 I：簇标识符表

第二步是最小化簇间相似度。*k*-均值聚类定义k 个质心，每个簇对应一个质心。质心的选取应使它们彼此相距较远。接下来，将每个数据对象分配给质心距离该对象最近的簇。每次分配后，都会更新簇的质心。所有对象完成聚类后，计算出全局质心，并将每个数据对象重新分配给质心距离其最近的簇，以最小化簇间相似度。重复此过程，直至不再进行重新分配。由于每个分量中的所有图像块彼此独立，因此对所有图像块并行应用*k*-均值聚类。使用块簇标识符表，将每个图像块中的像素替换为相应的簇标识符。将所有图像块合并，形成矩阵 $I^{'}$。对图像块 B_{A_1}, B_{A_2}, B_{A_3}, B_{A_4} 执行*k*-均值聚类，相应的块簇标识符表分别为表 Ia、Ib、Ic 和 Id。

3) 闭合频繁序列挖掘：序列 σ 在序列数据库 D 中的支持度，定义为 D 中包含 σ 的序列数量，支持度(σ)=$|\{s|s \in D \wedge \sigma \sqsubseteq s\}|$。支持度不低于给定支持度阈值$\psi$ 的序列称为频繁序列***FS***。所有闭合频繁序列***CFS***的集合定义如下：$\boldsymbol{CFS} = \{\sigma|\sigma \in FS$且$\nexists\gamma \in FS$，满足 $\sigma \sqsubseteq \gamma$且$support(\sigma) = support(\gamma)\}$。由于闭合频繁序列集合不包含那些支持度与其超序列相同的序列，因此有 $|CFS| \leq |FS|$。闭合频繁序列挖掘（CFSM）旨在生成所有频繁且闭合的序列，可通过两种方式实现：1）挖掘所有频繁序列，然后剪除所有非闭合序列；2）通过剪除非闭合序列，逐层生成闭合序列[3]。本文采用方法 2。为完成此任务，对用于挖掘频繁序列的广义序贯模式（GSP）挖掘算法进行修改，在每一层执行一个额外步骤，以生成闭合频繁序列

算法2 闭合频繁序列挖掘

输入： 序列数据库 I $^{'}$，最小支持度 ψ
输出： 闭合频繁序列S
过程改进的*GSP* _
$C_1 \leftarrow \{\sigma|\sigma\ is\ a\ 1-sequence\}$
$C_2 \leftarrow C_1 \times C_1$
$\forall\gamma \in I^{'}$，对于 $\sigma \in C_2$，如果 $\sigma \sqsubseteq \gamma$，支持度$(\sigma)$++
$F_2 \leftarrow \{\sigma|\sigma \in C_2 \wedge support(\sigma) \geq \psi\}$
$l \leftarrow 3$
当 $F_{l-1}!$ =空值 **时**
C_l = $\emptyset$
对 每个 ($\sigma \in F_{l-1}$) **执行**
对 每个 ($\gamma \in F_{l-1}$) **执行**
若$(l-2)$ 是 σ ==$(l-2)$ 的长度后缀 γ 的前缀**则**
$\theta = \sigma + \gamma[l-1]$
$C_l = C_l \cup \theta$
结束 if
结束 for
结束 for
对 每个 ($\sigma \in C_l$) **执行**
如果 $support(\sigma) \geq \psi$ **则**
$F_l \leftarrow F_l \cup \sigma$
结束 if
结束循环
对每个 $\gamma \in F_l$ **执行**
对每个 $\sigma \in F_{l-1}$ **执行**
如果 ($\sigma \sqsubseteq \gamma$)$\wedge$($support(\gamma)$ == $support(\sigma)$) **then**
$F_{l-1} \leftarrow F_{l-1} - \sigma$
结束条件判断
结束循环
结束循环
$l++$;
结束 while 循环
S = $C_1 \cup F_2 \cup F_3 \cup \cdots \cup F_{l-1}$
结束过程

用于剪除非闭合序列 [11]的层级。在图像中，图像矩阵被视为一个序列数据库，其中每一行都是一个像素序列。序列 σ 的支持度是包含σ的行数。对 $I^{'}$ 执行 CFSM，以生成闭合频繁序列。算法2从 $I^{'}$ 中挖掘闭合频繁序列。表II从 A 中挖掘闭合频繁序列。

生成用于编码的序列中，可能有少数字符不会作为子序列出现在任何序列中。这些字符可能无法编码，从而导致编码含糊不清。为解决这一问题， C_1 中的序列不会被剪枝。某些序列及其子序列也都是频繁的。在此示例中，*3312* 和 *33* 的支持度分别为 2 和 4，且二者均为频繁序列。由于序列 *3312* 包含*33*的 2 次出现，因此需要修正*33* 的支持度。如果这种重叠

支持度未被修正，闭合频繁序列集合 S 的基数可能会增加，导致许多具有这种重叠支持度的序列不会用于编码过程。编码表的大小将大幅增加，从而导致压缩比降低。设 H_s 为所有较长序列的集合， L_s 为 S中所有真子序列的集合。如果不修正 L_s 中序列的重叠支持度，那么L_s 中实际支持度低于H_s 中序列的序列，其支持度值就会偏高。因此，需要修正这些子序列的支持度。由此可见，显然有必要计算 S 中每个序列的修正支持度（ψ_{mod}）。编码时，H_s 中的序列比 L_s 中的序列优先级更高，因此 L_s 中序列的重叠支持度会降低。为获得 ψ_{mod}，按长度降序排列 S 中的序列。对于每个序列 $\sigma \in S$，计算其支持度，并从 $I^{'}$ 中移除 σ 的所有实例，然后更新 S 中其他序列的支持度。重复此过程，直到 $I^{'}$为空。闭合频繁序列及其修正支持度构成 $S^{'}$ 和 $|S^{'}| \leq |S|$。表 III 展示了闭合频繁序列的支持度和修正支持度。

设 E 为一个矩阵，其中每一行都是一个序列。 E 中的闭合频繁序列为 $\{333-2, 11-2, 44-2, 1-2, 2-1, 3-2, 4-2\}$。

$$E = \begin{bmatrix} 4 & 4 & 1 & 1 & 2 & 3 & 3 & 3 \\ 1 & 1 & 3 & 3 & 3 & 4 & 4 & 3 \end{bmatrix}$$

由于*333* 是长度最大的序列，因此从 E 中移除*333*的所有实例，并更新其他序列的支持度。闭合频繁序列及其支持度为 $\{333-2, 11-2, 44-2, 1-2, 2-1, 3-1, 4-2\}$。

$$E = \begin{bmatrix} 4 & 4 & 1 & 1 & 2 & & & \\ 1 & 1 & & & & 4 & 4 & 3 \end{bmatrix}$$

重复此过程，直到 E 为空，所得序列及其修正支持度如下： $\{333-2, 11-2, 44-2, 1-0, 2-1, 3-1, 4-0\}$。

4) **霍夫曼编码**：使用 $S^{'}$ 中带有修正支持度值的序列，通过霍夫曼编码为每个序列生成二进制码。序列及其对应的二进制码存储在编码表 CT 中。 $S^{'}$ 中的序列按长度降序排列。对于 $S^{'}$ 中的每个序列 σ ，将其在 $I^{'}$ 中的所有实例替换为 CT 中相应的二进制码，从而生成压缩二进制图像 C_I。对 $A^{'}$ 中的闭合频繁序列进行霍夫曼编码。A 的二进制编码图像大小为 111 位。

B. 解压器

算法 3执行解压缩。对 C_I 进行霍夫曼解码，得到 $I^{'}$。将 $I^{'}$ 中的每个簇标识符替换为相应块簇标识符表中的均值，从而得到解压缩后的图像 D_I. For A, the 解压缩 矩阵为 D_A.

$$D_A = \begin{bmatrix} 139 & 113 & 113 & 113 & 106 & 106 & 106 & 121 \\ 171 & 171 & 232 & 139 & 139 & 139 & 121 & 106 \\ 232 & 171 & 139 & 139 & 160 & 139 & 106 & 106 \\ 232 & 171 & 113 & 171 & 160 & 160 & 106 & 106 \\ 152 & 152 & 123 & 152 & 151 & 137 & 119 & 119 \\ 109 & 123 & 109 & 123 & 137 & 171 & 171 & 137 \\ 109 & 93 & 93 & 123 & 119 & 119 & 171 & 151 \\ 93 & 123 & 109 & 109 & 119 & 119 & 137 & 119 \end{bmatrix}$$

算法 3 解压器

1: **输入：**
1) 压缩图像 C_I2) 块数 n_b3) 各组件的编码表 CT_1, CT_2, CT_3各组件中每个图像块的簇标识符表 T_{B_1}, T_{B_2}, ..., T_{Bn_b}2: **输出：** 解压缩图像 D_I3: **过程**解压缩4: 使用编码表对压缩图像的每个组件进行解码 C_I CT_i($1 \leq i \leq 3$), 得到 $I_i^{'}$ 5: 对于每个组件，将 $I_i^{'}$ 中的每个簇标识符替换为相应的平均像素值。6: 合并所有组件，形成解压缩图像 D_I7: **过程结束**

IV. 仿真结果与讨论

仿真采用标准测试图像 Lena、Peppers、Baboon、Boat 等，格式为*bmp* [2]。将结果与 JPEG 和 GIF 的结果进行比较。该算法使用 Python 3 实现，并在配备 1.70GHz 时钟频率和 4GB 主存、运行 LinuxUbuntu 15.10 版本的 Intel core i5-4120 CPU 的 2 个核心上进行仿真。结果展示了尺寸为 512×512的 Lena 和 Peppers 图像。压缩效率受参数 b、k 和 ψ 影响，而解压缩图像的质量仅受 b和 k 影响。因此，我们观察了 b、 k 和 ψ 对压缩效率的影响。压缩比C_r 定义为

$$C_r = \frac{\text{Uncompressed size of Image}}{\text{Compressed size of Image}}$$

图像质量使用峰值信噪比（PSNR）、结构相似性（SSIM）等指标进行衡量。若误差均值（均方误差（MSE））较低，则 PSNR 较高。

图像 S_{C_I} 的压缩后大小是所有簇标识符表、编码表和编码图像大小的总和。图 1 展示了 Lena 和 Peppers 在不同支持度（ψ）下的压缩比（C_r）效率。如图所示，对于 Lena 和 Peppers，压缩比会随着支持度增加而上升，达到峰值后再下降。其他数据集也观察到了类似的趋势。

长度	候选序列集	频繁序列集
1	C_1 = {0(8), 1(8), 2(7), 3(6)}	未进行剪枝
2	C_2 = {00(1), 01(1), 02(5), 03(3), 10(3), 11(3), 12(4), 13(3), 20(3), 21(4), 22(4), 23(1), 30(2)、31(3)、32(1)、33(4)}	L_2 = {02(5), 03(3), 10(3), 11(3), 12(4), 13(3), 20(3), 21(4), 22(4), 30(2), 31(3), 33(4)}
3	C_3 = {020(1), 021(1), 022(2), 030(0), 031(0), 033(1), 102(2), 103(0), 110(1), 111(1), 112(0), 113(2), 120(0), 121(1), 122(1), 130(0), 131(0), 133(2), 202(1), 203(2), 210(0), 211(2), 212(2), 213(1), 220(2), 221(2), 222(1), 310(0), 311(1), 312(2), 313(0), 330(2), 331(2),333(1)}	L_3 = {022(2), 102(2), 113(2), 133(2), 203(2), 211(2)、212(2)、220(2)、221(2)、312(2)、331(2)、330(2)} L_2 = {02(5), 03(3), 10(3), 11(3), 12(4), 13(3)} 20(3), 21(4), 22(4), 31(3), 33(4)}
4	C_4 = {0220(1), 0221(0), 1022(2), 1133(2), 1331(0), 1330(0), 2113(1), 2203(2), 2211(0), 2212(1), 3312(2)}	L_4 = {1022(2)、1133(2)、2203(2)、3312(2)} L_3 = {211(2), 212(2), 221(2), 330(2)}

表II： A' 上使用 $\psi = 2$的CFSM

序列	1022	1133	2203	3312	211	212	221	330	02	03	10	11	12	13	20	21	22	31	33	0	1	2	3
支持度	2	2	2	2	2	2	2	2	5	3	3	3	4	3	3	4	4	3	4	8	8	7	6
ψ_{mod}	2	2	1	2	0	2	1	1	2	1	1	0	0	0	0	1	0	1	1	4	1	3	2

表III： A'的闭合频繁序列及其支持度和修正支持度

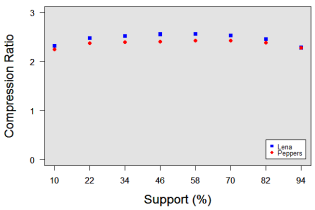

图1： ψ 与 C_r的对比

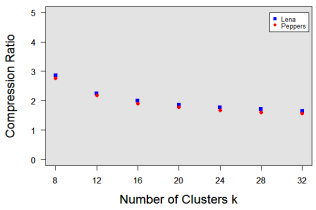

图 2： k 与 C_r

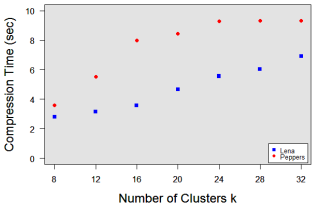

图 3： k 与 C_t

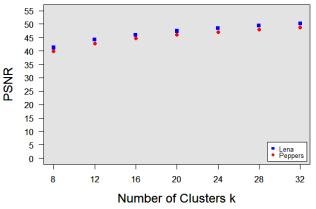

图 4： k 与峰值信噪比

对于较低的 ψ 值， $|CFS|$ 较大，因此编码表的大小也较大。对于较高的 ψ 值， $|CFS|$ 较小，因此编码图像的大小较大。对于 Lena，在 ψ =58% 时观察到最高压缩比 2.555。对于 Peppers，在 ψ = 70% 时观察到压缩比 2.425。当 $\psi \in [40\%, 70\%]$ 时，压缩比偏高。此外，随着块大小增加，压缩比也会增加。这是因为随着 b 增大，块簇标识符表的数量会减少。由于每个簇标识符表的大小会随着 k 增大而增加， S_{C_I} 也会增加，因此 C_r 这一数值下降，这一点可从图 2 中观察到。随着 k 增大，对像素进行聚类所需的时间也随之增加，因此压缩时间也会增加，这一点可从图 3 中观察到。对于不同的 b值，平均压缩时间最低可达 2 秒。这得益于块聚类带来的并行性。图 4 展示了 k 对 Lena 和 Peppers 的峰值信噪比的影响。随着簇数增加，由于像素被分组到关联性更强的簇中，图像质量也会提高。对于 $k = 32$，Lena 的峰值信噪比为 50.26，Peppers 的峰值信噪比为 48.70。还可以观察到，随着块大小增加，峰值信噪比会降低。



(a) 所提方法



(b) JPEG

图 5：大小为 224 千字节的 Lena 压缩图像

表 IV 和表 V 分别比较了所提算法与 JPEG 和 GIF 算法在 Lena 和 Peppers 图像上的表现。输入图像大小为 786.6 千字节。对于 Lena，在大小为 224 千字节时，所提算法的峰值信噪比为 39.49，而 JPEG 的峰值信噪比为 41.82。结果发现，在 224 千字节时，解压缩图像的质量与 JPEG 图像相近。图像如图 5a 和 5b 所示。

对于 Peppers，在压缩图像大小为 241 千字节时，所提算法与 JPEG 的峰值信噪比差异可以忽略，结构相似性数值也很接近。

图像	输入大小 (kB)	所提方法大小 (kB)	JPEG大小 (kB)	所提方法峰值信噪比 无单位	JPEG峰值信噪比 无单位	所提方法的结构相似性 无单位	JPEG结构相似性 无单位	$(b,\ k,\ \psi)$
Lena	786.6	166.7	162	37.72	41.19	0.9697	0.9870	(128,8,70)
	786.6	224	224	39.49	41.82	0.978	0.992	(256,12,70)
	786.6	359	341.3	45.99	47.92	0.997	0.995	(32, 16, 82)
Peppers	786.6	241	242	38.09	38.45	0.973	0.989	(256,12,82)
	786.6	254	262	38.99	39.19	0.982	0.99	(128,12,70)
	786.6	365	363	44.58	46.27	0.996	0.996	(32,16,94)

表 IV：所提算法与 JPEG 在 Lena 和 Peppers 上的比较

图像	输入大小 (kB)	所提方法大小 (kB)	GIF 大小 (kB)	所提方法峰值信噪比 无单位	GIF 的峰值信噪比 无单位	所提方法的结构相似性 无单位	GIF 的结构相似性 无单位	$(b,\ k,\ \psi)$
Lena	786.6	221.3	226	40.64	29.55	0.985	0.78	(128, 12, 58)
	786.6	205	204	39.49	28.78	0.978	0.77	(256, 12, 46)
	786.6	173.45	173.71	37.72	27.12	0.969	0.75	(128, 8, 46)
Peppers	786.6	220.13	216	38.096	28.94	0.973	0.81	(256, 12, 46)
	786.6	207.59	204	35.14	28.13	0.94	0.79	(256, 8, 94)
	786.6	185.38	192	36.088	27.94	0.95	0.78	(128, 8, 46)

表 V：所提算法与 GIF 在 Lena 和 Peppers 上的比较

对于所测试的数据集，所提算法生成的图像质量接近 JPEG。对于 Lena，所提算法生成的压缩图像大小为 221.3 千字节，峰值信噪比为 40.64，结构相似性为 0.985；而 GIF 格式的压缩图像大小为 229 千字节，峰值信噪比为 29.55，结构相似性为 0.78。对于 Peppers，所提算法生成的压缩图像大小为 220.13 千字节，峰值信噪比为 38.09，结构相似性为 0.973；而 GIF 格式的压缩图像大小为 204 千字节，峰值信噪比为 28.13，结构相似性为 0.81。就压缩比和质量而言，所提算法始终优于 GIF。其他数据集也呈现出类似趋势。

V. 结论

本文提出了一种高效且新颖的基于频繁序列挖掘的有损图像压缩算法。仿真结果证明，在挖掘最优像素序列方面，我们的方法优于其他一些替代方法，在视觉质量损失可接受的情况下实现了更高的压缩比。作为未来工作的一个方向，可以尝试使用其他高效的簇算法替代 k-means，以提高图像的视觉质量。还可以利用图像的邻域属性来提高压缩比。我们旨在针对大规模标准数据集开展广泛仿真，以研究包括压缩时间在内的其他多个参数。

参考文献

[1] Thomas Boutell。《PNG（便携式网络图形）规范》1.0 版。1997 年。[2] USC-SIPI 图像数据库。http://sipi.usc.edu/database/，2016 年。[3] Antonio Gomariz、Manuel Campos、Roque Marin 和 Bart Goethals。《知识发现与数据挖掘进展：第 17 届亚太会议（PAKDD 2013），澳大利亚黄金海岸，2013 年 4 月 14—17 日，论文集，第一部分》，第 50–61 页。Springer Berlin Heidelberg，德国柏林，2013 年。章节：ClaSP：一种用于挖掘频繁闭序列的高效算法。

[4] Jiawei Han 和 Micheline Kamber。《数据挖掘：概念与技术》。Morgan Kaufmann，2000 年。[5] David A Huffman 等。最小冗余码的构造方法。《IRE 论文集》，40(9):1098–1101，1952 年。[6] Anil K. Jain。数据聚类：超越 k-均值的 50 年。《Pattern Recognition Letters》，31(8):651–666，2010 年。国际模式识别会议（ICPR）第 19 届会议获奖论文。[7] Leonard Kaufman 和 Peter J. Rousseeuw。《围绕中心点划分（PAM 程序）》，第 68–125 页。John Wiley & Sons, Inc.，2008 年。[8] Naren Ramakrishnan 和 Ananth Grama。数据挖掘：从偶然发现到科学——客座编辑导言。《IEEE Computer》，32(8):34–37，1999 年。[9] David Salomon。《数据压缩：完整参考指南》。第二版，2004 年。[10] A. Skodras、C. Christopoulos 和 T. Ebrahimi。JPEG 2000 静态图像压缩标准。《IEEE Signal Processing Magazine》，18(5):36–58，2001 年 9 月。[11] Ramakrishnan Srikant 和 Rakesh Agrawal。《挖掘序列模式：推广与性能改进》。Springer，1996 年。[12] Gregory K. Wallace。JPEG 静态图像压缩标准。《Commun. ACM》，34(4):30–44，1991 年 4 月。[13] V. Watson。游程编码，2003 年 7 月 24 日。美国专利申请 10/143,542。[14] Ian H Witten、Radford M Neal 和 John G Cleary。用于数据压缩的算术编码。《Communications of the ACM》，30 540，1987 6 520( ): – 。[15] Xifeng Yan、Jiawei Han 和 Ramin Afshar。CloSpan：在大型数据集中挖掘闭合序列模式。载于《SDM》，第 166–177 页。SIAM，2003 年。[16] Shuigeng Zhou、Yue Zhao、Jihong Guan 和 Joshua Huang。一种基于邻域的聚类算法。载于 TuBao Ho、David Cheung 和 Huan Liu 编，《知识发现与数据挖掘进展》，Lecture Notes in Computer Science 第 3518 卷，第 361–371 页。Springer Berlin Heidelberg，2005 年。[17] Jacob Ziv 和 Abraham Lempel。一种用于序列数据压缩的通用算法。《IEEE 信息论汇刊》，23(3):337–343，1977 年。