JEI Journal of Electronic Imaging

RESEARCH PAPER

Color quantization using an accelerated Jancey *k*-means clustering algorithm

Harrison Bounds, M. Emre Celebi,* and Jordan Maxwell
University of Central Arkansas, Department of Computer Science and Engineering, Conway, Arkansas, United States

ABSTRACT. Color quantization (CQ) is a fixed-rate vector quantization developed for color images to reduce their number of distinct colors while keeping the resulting distortion to a minimum. Various clustering algorithms have been adapted to the CQ problem over the past 40 years. Among these, hierarchical algorithms are generally more efficient (i.e., faster), whereas partitional ones are more effective (in minimizing distortion). Among the partitional algorithms, the effectiveness and efficiency of the Lloyd (or batch) *k*-means algorithm have been shown by multiple recent studies. We investigate an alternative, lesser-known *k*-means algorithm proposed by Jancey, which differs from Lloyd *k*-means (LKM) in the way it updates the cluster centers at the end of each iteration. To obtain a competitive color quantizer, we develop a weighted variant of Jancey *k*-means (JKM) and then accelerate the weighted algorithm using the triangle inequality. Through extensive experiments on 100 color images, we demonstrate that, with the proposed modifications, JKM outperforms LKM significantly in terms of efficiency without sacrificing effectiveness. In addition, the proposed JKM-based color quantizer is as straightforward to implement as the popular LKM color quantizer.

 [DOI: 10.1117/1.JEI.33.5.053052]

Keywords: color quantization; clustering; Lloyd *k*-means; batch *k*-means; Jancey *k*-means; triangle inequality

Paper 240382G received Apr. 14, 2024; revised Aug. 15, 2024; accepted Sep. 23, 2024; published Oct. 22, 2024.

1 Introduction

Color images have become ubiquitous over the past 20 years.[1] These images often contain a great many colors, posing a challenge to display, store, transmit, process, and analyze them. Color quantization (CQ) is an image processing technique designed to reduce the number of distinct colors in a color image while preserving its visual quality.[2] By reducing the number of colors in an image, CQ can generate replicas that are difficult to distinguish from the original.

CQ is composed of two subproblems:[3] (1) palette design and (2) pixel mapping. The former subproblem is concerned with choosing a limited set of colors, called the color palette, to represent the colors of the input image. This subproblem is usually formulated as a large-scale data clustering problem,[4] which is computationally hard in most settings.[5] On the other hand, the latter subproblem involves mapping each pixel in the input image to its nearest palette color, which can be solved exactly using a simple linear-time algorithm. Thus, the vast majority of the CQ literature deals with the palette design subproblem, which is also the main focus of this paper.

*Address all correspondence to M. Emre Celebi, ecelebi@uca.edu

1017-9909/2024/$28.00

CQ originally aimed to overcome the color depth limitations of early color display devices.[2] Although 24-bit color displays have become prevalent, CQ is still used in many visual computing applications, including[6] non-photorealistic rendering, image matting, image dehazing, image compression, color-to-grayscale conversion, image watermarking/steganography, image segmentation, content-based image retrieval, color analysis, color-texture analysis, saliency detection, and skin detection.

CQ algorithms can be categorized in various ways based on their palette design phase. For example, color palettes can be designed to be image-independent or image-dependent.[7] An image-independent palette is a universal color palette designed to represent a variety of images. In contrast, an image-dependent palette is a custom color palette designed to represent the color distribution of a specific image. The majority of CQ algorithms belong to the latter category. CQ algorithms can also be categorized based on the clustering algorithm[5] used in their palette design phase. Many clustering algorithms have been adapted to the palette design subproblem over the past 40 years.[6] Hierarchical clustering algorithms find nested clusters in a top-down or bottom-up manner. On the other hand, partitional clustering algorithms discover clusters simultaneously without imposing a hierarchical structure on the data. Hierarchical algorithms dominated the early CQ literature mainly due to their efficiency. The more recent literature, however, focuses primarily on partitional algorithms that promise greater effectiveness at the expense of more computation.

Among the many partitional algorithms used for palette design, the Lloyd (or batch) k-means algorithm[8–11] has been shown to be both effective and efficient by multiple studies.[12–20] The most comprehensive study among these was conducted by Celebi and Pérez-Delgado,[20] where the authors determined that a variant of Lloyd k-means (LKM) proposed by Celebi[15,16] was one of the best among 21 CQ algorithms published between 1980 and 2022. In fact, this k-means–based CQ algorithm has been independently integrated into the Android Open Source Project (available at Ref. 21).

In this paper, we investigate a lesser-known (at the time of this writing, Lloyd's paper[11] has been cited over 20,000 times, whereas Jancey's paper[9] has been cited only ≈250 times) k-means algorithm proposed by Jancey.[9] The Lloyd and Jancey k-means (JKM) algorithms have identical assignment steps, where each data point is assigned to the nearest cluster center. The two algorithms differ in their update steps. Lloyd k-means updates each cluster center to be the centroid of the data points assigned to it, whereas Jancey k-means updates each cluster center as a linear combination of itself and the centroid of the data points assigned to it. This linear combination is controlled by a user-defined parameter (α), which allows Jancey k-means to take larger steps toward a local minimum. Accordingly, Jancey k-means is expected to converge faster than Lloyd k-means. Our comprehensive experiments show that this is indeed the case. In fact, we demonstrate that, with suitable modifications, Jancey k-means outperforms Lloyd k-means significantly in terms of efficiency without sacrificing effectiveness.

The rest of this paper is structured as follows. Section 2 introduces the two k-means variants, their adaptation to weighted data, their acceleration using the triangle inequality, and their initialization. Section 3 presents the objective and subjective assessment of the two k-means variants and discusses our findings. Finally, Sec. 4 concludes the paper and provides directions for future work.

2 *k*-Means and Its Variants

In this section, we present an overview of the Lloyd and Jancey k-means algorithms, their extensions for weighted data, a practical strategy to accelerate the two algorithms and their weighted variants using the triangle inequality, and, finally, their initialization.

2.1 Lloyd *k*-Means Algorithm

Lloyd k-means is the simplest and most popular partitional clustering algorithm.[22] Given a dataset $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\} \subset \mathbb{R}^D$ and a positive integer $K > 1$, the goal of LKM is to partition $\mathcal{X}$ into K non-overlapping clusters $\{\mathcal{P}_1, \ldots, \mathcal{P}_K\}$. Each cluster $\mathcal{P}_i$ has a center $\mathbf{c}_i$, which is usually selected uniformly at random from $\mathcal{X}$. Starting from this initial configuration, the algorithm proceeds with an assignment step followed by an update step. In the assignment step, each data point

is assigned to its nearest center with respect to the squared Euclidean (ℓ_2^2) distance, whereas in the update step, each center is recomputed by averaging the data points assigned to it. These steps are alternated until the algorithm converges. It can be shown that each iteration either decreases the sum of squared error (SSE), $\mathrm{SSE} = \sum_{\mathbf{x}\in\mathcal{X}} d_{\mathrm{SE}}(\mathbf{x}, \{\mathbf{c}_1, \ldots, \mathbf{c}_K\})$, where $d_{\mathrm{SE}}(\mathbf{x}, \mathcal{C})$ is the ℓ_2^2 distance between data point $\mathbf{x}$ and its nearest center in $\mathcal{C}$, or leaves it unchanged (indicating convergence).

Consider an input image I in a CQ application. In this context, the dataset $\mathcal{X}$ represents the pixels in I, N is the number of pixels in I, D is the number of color channels (for RGB images, we have $D = 3$), and K is the number of desired colors in the quantized output image.

The pseudocode of the LKM algorithm is given below (superscripts denote iteration numbers).

1. Initialization: Initialize the K cluster centers $\mathcal{C}^{(0)} = \{\mathbf{c}_1^{(0)}, \ldots, \mathbf{c}_K^{(0)}\}$ and set the iteration counter to one ($t = 1$).
2. Assignment: Set $n_1 = \cdots = n_K = 0$. For each $j \in \{1, \ldots, N\}$, assign data point $\mathbf{x}_j$ to its nearest center in $\mathcal{C}^{(t-1)}$, that is, $\mathbf{c}_{i^*}$ with $i^* = \arg\min_{i\in\{1,\ldots,K\}} \|\mathbf{x}_j - \mathbf{c}_i^{(t-1)}\|_2^2$, where $\|\cdot\|_2$ denotes the ℓ_2 norm, and increment the size n_{i^*} of the corresponding cluster $\mathcal{P}_{i^*}^{(t)}$ ($n_{i^*} = n_{i^*} + 1$).
3. Update: Update each cluster center $\mathbf{c}_i$ to be the centroid of the data points assigned to it, that is, $\mathbf{c}_i^{(t)} = (1/n_i)\sum_{\mathbf{x}\in\mathcal{P}_i^{(t)}} \mathbf{x}$.
4. Termination: If the termination criterion is satisfied (see below), terminate the algorithm. Otherwise, increment the iteration counter ($t = t + 1$) and return to step 2.

In this study, we terminate LKM once the clusters stabilize (i.e., the data points assigned to each cluster stop changing). It is important to stress that, for LKM, the clusters stabilize if and only if their centers stabilize. Upon termination, LKM converges to a local minimum of its objective. As SSE is a nonsmooth and nonconvex objective with numerous local minima, step 1, initialization, is the most crucial step.[23] Poorly initialized centers can lead to empty clusters, slower convergence, or getting trapped in a shallow local minimum.

LKM's popularity can be attributed to multiple reasons. In addition to being a simple algorithm to implement, its time complexity is linear in N, D, and K, meaning that it can be used to cluster large datasets or initialize computationally more expensive clustering algorithms. LKM is also guaranteed to converge to a local minimum in a finite number of iterations[24] and is insensitive to the order in which the data points are processed.

As LKM is an iterative algorithm, its execution time depends on the number of times it iterates until reaching convergence. This number can vary remarkably based on initialization, dataset characteristics, and termination criterion. Although it has a linear time complexity, LKM can be computationally expensive due to its iterative nature. There are many ways to accelerate the algorithm, including sampling, data compression, dimensionality reduction, numerical approximations, geometric identities, and better initialization. Unfortunately, most of these approaches sacrifice simplicity, accuracy, or convergence guarantee for efficiency.

2.2 Jancey *k*-Means Algorithm

Jancey[9] and later Ostresh,[25] proposed an algorithm very similar to LKM. Let $\mathcal{P}_i^{(t)}$ denote cluster i ($i \in \{1, \ldots, K\}$) with center $\mathbf{c}_i^{(t)}$ at the end of iteration t ($t = 1, 2, \ldots$). Recall that in LKM, at the end of iteration t, each center is recomputed as the centroid of its cluster, that is

$$\mathbf{c}_i^{(t)} = \mathbf{m}_i^{(t)}, \tag{1}$$

where $\mathbf{m}_i^{(t)} = (1/n_i)\sum_{\mathbf{x}\in\mathcal{P}_i^{(t)}} \mathbf{x}$ denotes the centroid of $\mathcal{P}_i^{(t)}$.

In Jancey's k-means algorithm, each center is recomputed as a linear combination of itself and the centroid of its cluster, that is

$$\mathbf{c}_i^{(t)} = \alpha\mathbf{m}_i^{(t)} + (1-\alpha)\mathbf{c}_i^{(t-1)}, \tag{2}$$

$$= \mathbf{c}_i^{(t-1)} + \alpha[\mathbf{m}_i^{(t)} - \mathbf{c}_i^{(t-1)}], \tag{3}$$

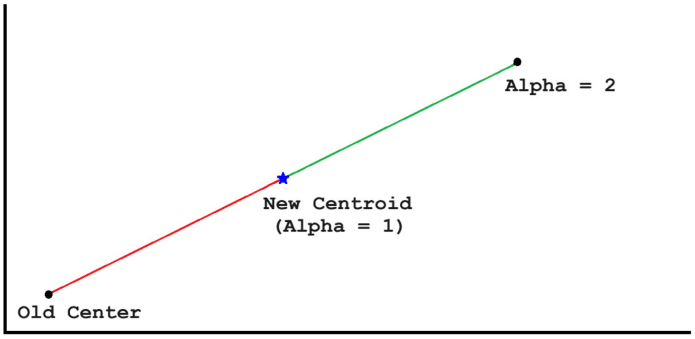

Fig. 1 Illustration of the Jancey center update equation in two dimensions.

where α is a "relaxation factor." Figure 1 illustrates the above equation in two dimensions. Note that this equation can be viewed as an example of the well-known successive over-relaxation method[26] for solving a linear system of equations.

The relaxation factor α influences JKM's behavior as follows:

- The algorithm is guaranteed to converge for $\alpha \in (0,2)$.[27,28]
- Jancey's choice of $\alpha = 2$ corresponds to reflecting the old center $\mathbf{c}_i^{(t-1)}$ through the new centroid $\mathbf{m}_i^{(t)}$ [Ref. 29, pp. 161–162], as illustrated in Fig. 1. However, with this particular α value, JKM may converge more slowly than LKM or even fail to converge due to points oscillating among clusters.
- The α value yielding optimal convergence rates[25,27] can be shown to be between 1 and 2 (the green line segment in Fig. 1), in which case Eq. (2)/(3) is said to be an "over-relaxation." In particular, the optimal α is near 1 if the clusters are well separated. On the other hand, the optimal α is near 2 if the clusters are poorly separated. Therefore, JKM with $\alpha \in (1,2)$ takes larger steps toward a local minimum and thus is expected to converge faster than LKM unless the dataset is well-clusterable.
- For $\alpha \in (0,1]$, Eq. (2)/(3) denotes a convex combination, that is, the new center $\mathbf{c}_i^{(t)}$ lies on the line segment joining the old center $\mathbf{c}_i^{(t-1)}$ and the new centroid $\mathbf{m}_i^{(t)}$ (the red line segment in Fig. 1). For the special case of $\alpha = 1$, Eq. (2)/(3) reduces to the LKM center update equation, Eq. (1). On the other hand, for $\alpha \in (0,1)$, Eq. (2)/(3) is said to be an under-relaxation. In this paper, we do not consider this regime, as it can be shown to yield a suboptimal convergence rate.[28]

The pseudocode of JKM is identical to that of LKM with one exception: the center update equation in step 3 should be Eq. (2)/(3). As the two algorithms are so similar, many acceleration techniques designed for LKM can also be adapted to JKM.

2.3 Weighted *k*-Means Algorithms

LKM can be modified to handle weighted data, $\mathcal{X} = \{\mathbf{x}_1 \ldots \mathbf{x}_N\} \subset \mathbb{R}^D$, where each data point $\mathbf{x}_j$ is assigned a nonnegative weight w_j. Without loss of generality, we assume that the weights are normalized and add up to one, that is, $\sum_{j=1}^{N} w_j = 1$.

The objective of the weighted LKM (WLKM) algorithm is identical to that of its unweighted counterpart, LKM, except the distances are weighted multiplicatively. In the weighted case, the optimal center $\mathbf{c}_i^{(t)}$ for cluster $\mathcal{P}_i^{(t)}$ at the end of iteration t is given by the weighted centroid of $\mathcal{P}_i^{(t)}$

$$\mathbf{c}_i^{(t)} = \frac{1}{\sum_{\mathbf{x}_j \in \mathcal{P}_i^{(t)}} w_j} \sum_{\mathbf{x}_j \in \mathcal{P}_i^{(t)}} w_j \mathbf{x}_j. \tag{4}$$

The pseudocode of WLKM is identical to that of LKM, with two exceptions: the cluster size update equation in step 2 should be $n_{i^*} = n_{i^*} + w_j$, whereas the center update equation in step 3 should be Eq. (4). On the other hand, given the pseudocode of WLKM, the pseudocode of

weighted JKM (WJKM) can be obtained by substituting the center update equation with the following:

$$\mathbf{c}_i^{(t)} = \mathbf{c}_i^{(t-1)} + \alpha\left(\frac{1}{\sum_{\mathbf{x}_j \in \mathcal{P}_i^{(t)}} w_j} \sum_{\mathbf{x}_j \in \mathcal{P}_i^{(t)}} w_j \mathbf{x}_j - \mathbf{c}_i^{(t-1)}\right).$$

Given a color image I, a time- and memory-efficient way to compute the "frequency" f_j of a pixel $\mathbf{x}_j \in I$ (i.e., the number of times the color $\mathbf{x}_j$ occurs in I) is by means of a hash table that uses chaining for collision resolution and a universal hash function of the form $h_{\mathbf{a}}(r,g,b) = (a_r r + a_g g + a_b b) \bmod P$, where (r,g,b) are the red, green, and blue components of an input color, respectively; P is a prime number; and the elements of sequence $\mathbf{a} = (a_r, a_g, a_b)$ are selected randomly from the set $\{0, 1, \ldots, P-1\}$.[6] Once the hash table is populated (by inserting the color of each pixel in I into the table), the "normalized weight" w_j of $\mathbf{x}_j$ is computed by $w_j = f_j/N$, where N is the number of pixels in I.

2.4 Triangle Inequality Elimination Technique

The triangle inequality elimination (TIE) technique accelerates LKM by reducing the number of distance computations it performs in each iteration. For a given data point $\mathbf{x}$, centers $\mathbf{c}_i$ and $\mathbf{c}_{\hat{i}}$, and a metric $d(\cdot)$, the triangle inequality states that $d(\mathbf{c}_i, \mathbf{c}_{\hat{i}}) \leq d(\mathbf{x}, \mathbf{c}_i) + d(\mathbf{x}, \mathbf{c}_{\hat{i}})$. Therefore, if $2d(\mathbf{x}, \mathbf{c}_i) \leq d(\mathbf{c}_i, \mathbf{c}_{\hat{i}})$, we can omit the computation of $d(\mathbf{x}, \mathbf{c}_{\hat{i}})$ knowing that $d(\mathbf{x}, \mathbf{c}_i) \leq d(\mathbf{x}, \mathbf{c}_{\hat{i}})$. This inequality is valid for metrics such as $d = \ell_2$. For the non-metric case of $d = \ell_2^2$, the inequality becomes $4\|\mathbf{x} - \mathbf{c}_i\|_2^2 \leq \|\mathbf{c}_i - \mathbf{c}_{\hat{i}}\|_2^2$.

At the beginning of each iteration, TIE precomputes the $\binom{K}{2}$ pairwise ℓ_2^2 distances between the K centers and stores them in a $K \times K$ matrix. The rows of this matrix are then sorted individually in increasing order. In other words, each center's distances to the other $(K-1)$ centers are sorted. To determine the nearest center to a given data point $\mathbf{x}$, TIE begins with the current nearest center for $\mathbf{x}$, say $\mathbf{c}_i$ (all data points can be assigned to an arbitrary center, e.g., $\mathbf{c}_1$, before the first iteration) and searches through the centers in order of increasing distance from $\mathbf{c}_i$ using the aforementioned triangle inequality test. If the test succeeds, the search can be aborted, as the test will succeed for the remaining centers in the sorted list. Otherwise, the ℓ_2^2 distance between $\mathbf{x}$ and the current center under consideration is computed. As TIE accelerates the time-consuming assignment step of LKM, which is identical to that of JKM, the same technique can also be used to accelerate JKM.

LKM/JKM can be accelerated using a variety of geometric techniques,[30,31] many of which are based on binary space partitioning trees such as k-d trees[32] or more elaborate formulations of TIE.[30] Some of these techniques[33] pay off only in high dimensions, making them unsuitable for low-dimensional data such as color image data. Others are difficult to understand (or implement) and require computationally expensive preprocessing.[32,34,35] By contrast, TIE is simple, intuitive, and practical, making it an ideal strategy for accelerating LKM/JKM for CQ. Originally proposed by Chen and Pan[36] for accelerating vector quantization, TIE was adapted to CQ by Hu and Su,[13] Hu et al.,[14] and Celebi.[15,16] For a detailed pseudocode, refer to Celebi.[6]

2.5 Cluster Center Initialization

As explained in Sec. 2.1, cluster center initialization is crucial for the success of LKM/JKM. To achieve a successful initialization, we utilize the maximin algorithm.[37,38] Maximin begins with an arbitrary data point taken as the first center $\mathbf{c}_1$. The next center $\mathbf{c}_i$ $(i \in \{2, \ldots, K\})$ is selected as the data point with the largest distance from centers $\{\mathbf{c}_1, \ldots, \mathbf{c}_{i-1}\}$, that is, $\mathbf{c}_i = \arg\ \max_{\mathbf{x} \in \mathcal{X}} \min(d(\mathbf{x}, \mathbf{c}_1), \ldots, d(\mathbf{x}, \mathbf{c}_{i-1}))$, where $d(\cdot,\cdot)$ is a metric, which is usually taken as ℓ_2. The algorithm can be implemented in $\mathcal{O}(NK)$ time.

The pseudocode of maximin is given below. Note that we use the centroid of $\mathcal{X}$ as the first center to ensure determinism.

1. Let d_j denote the distance of $\mathbf{x}_j$ to its nearest center (initially, $d_j = \infty$), and $d_{\max}$ denote the maximum such distance in $\mathcal{X}$. Set $\mathbf{c}_1 = (1/N)\sum_{\mathbf{x} \in \mathcal{X}} \mathbf{x}$, and the index i of the next center (to be selected) to $i = 2$.

2. Set $d_{\max} = -\infty$. For each $j \in \{1, \ldots, N\}$, update d_j, if necessary, as follows: if $d(\mathbf{x}_j, \mathbf{c}_{i-1}) < d_j$, then set $d_j = d(\mathbf{x}_j, \mathbf{c}_{i-1})$. Update $d_{\max}$, if necessary, as follows: if $d_{\max} < d_j$, set $d_{\max} = d_j$, and save the index of the current point as $j^* = j$.
3. Set $\mathbf{c}_i = \mathbf{x}_{j^*}$. If $i < K$, increment index i $(i = i + 1)$ and return to step 2; otherwise, terminate the algorithm.

Many algorithms can be used to initialize k-means–like partitional clustering algorithms.[23] Among these, maximin is one of the most popular for several reasons, including its simple, parameter-free, and deterministic nature, and good performance in a variety of applications. Maximin was introduced to the CQ literature by Houle and Dubois[39] and popularized by Goldberg[40] and Xiang.[41] More recently, the algorithm has been used as an effective initializer for k-means–based CQ algorithms by Thompson et al.[42] and Abernathy and Celebi.[19]

3 Objective and Subjective Assessment

In this section, we first describe our experimental setup, which includes the image set, performance measures, statistical tests, and parameter configuration. We then present an objective assessment of the LKM and JKM algorithms. We conclude the section with a brief subjective assessment of the two k-means variants.

3.1 Experimental Setup

The plain algorithms, namely, LKM and JKM, and their weighted variants accelerated using TIE, namely, accelerated LKM (ALKM) and accelerated JKM (AJKM), were tested on CQ100 (available at Ref. 43), a diverse dataset of 24-bit color images specifically curated for CQ research.[20] CQ100 contains 100 images, each with a resolution of 768×512 or 512×768 pixels.

The effectiveness of an algorithm was quantified using two fidelity measures: mean squared error (MSE) and multi-scale structural similarity (MS-SSIM).

MSE is the most popular fidelity measure in CQ[6] and is defined as follows:

$$\mathrm{MSE}(I, \tilde{I}) = \frac{1}{HW} \sum_{r=1}^{H} \sum_{c=1}^{W} \|I(r,c) - \tilde{I}(r,c)\|_2^2,$$

where I and $\tilde{I}$ denote the original input image and quantized output image, respectively, each with a resolution of $W \times H$ pixels. MSE values fall into the range $[0, 3 \times 255^2]$, with smaller values indicating a better match among the two images. Observe that MSE is simply a normalized variant of SSE, which is precisely the objective LKM and JKM minimize locally.

MSE is a simple fidelity measure, both conceptually and computationally. However, it disregards the characteristics of the human visual system. Structural similarity (SSIM)[44] addresses this shortcoming of MSE by combining the luminance, contrast, and structural differences among the two images that are being compared. MS-SSIM,[45] on the other hand, is a multi-scale variant of SSIM that incorporates variations in image resolution and viewing conditions. Note that, unlike MSE, SSIM and MS-SSIM are similarity measures whose values fall into the range [0, 1], with larger values indicating a better match between the two images. For mathematical and computational details of SSIM and MS-SSIM, refer to Refs. 44–47.

MS-SSIM is a fidelity measure recommended by both Ramella[48] and Pérez-Delgado and Celebi,[49] the only studies to date that focus on the objective evaluation of CQ algorithms. Therefore, in this study, we employed MS-SSIM along with MSE to quantify the effectiveness of a CQ algorithm.

The efficiency of an algorithm was quantified using two measures: central processing unit (CPU) time in milliseconds (ms) averaged over 10 independent runs and number iterations until reaching convergence. Unlike CPU time, the number of iterations is an implementation-independent efficiency measure. Note that we performed multiple runs of each algorithm solely to obtain more accurate time measurements, as there is no randomness in any of the algorithms presented in this paper. All algorithms were implemented in C++ and executed on a 2.60-GHz Intel Core i7-8850H CPU.

To determine if there are any statistically significant differences among the algorithms, we employed two nonparametric statistical tests:[50] the Friedman test[51] and the Iman–Davenport

test.[52] These tests are alternatives to the parametric two-way analysis of variance (ANOVA) test. Their advantage over ANOVA is that they do not require normality or homoscedasticity, assumptions that are often violated in machine learning or optimization studies.[53–57]

Given B blocks (subjects) and T treatments (measurements), the null hypothesis (H_0) of the Friedman test is that populations within a block are identical. The alternative hypothesis (H_1) is that at least one treatment tends to yield larger (or smaller) values than at least one other treatment. The test statistic is computed as follows.[58] In the first step, the observations within each block are ranked separately, so each block contains a separate set of T ranks. If ties occur, the tied observations are given the mean of the rank positions for which they are tied. If H_0 is true, the ranks in each block should be randomly distributed over the columns (treatments). Otherwise, we expect the lack of randomness in this distribution. For example, if a particular treatment is better than the others, we expect small (or large) ranks to favor that column. In the second step, the ranks in each column are summed. If H_0 is true, we expect the sums to be fairly close—so close that we can attribute differences to chance. Otherwise, we expect to see at least one difference between pairs of rank sums so large that we cannot reasonably attribute it to sampling variability. The test statistic is given as

$$\chi_r^2 = \frac{12}{BT(T+1)} \sum_{j=1}^{T} R_j^2 - 3B(T+1), \tag{5}$$

where R_j ($j \in \{1, 2, \ldots, T\}$) is the rank sum of the j'th column. χ_r^2 is approximately chi-square with $(T-1)$ degrees of freedom. H_0 is rejected at the α level of significance if the value of (5) is greater than or equal to the critical chi-square value for $(T-1)$ degrees of freedom.

Iman and Davenport[52] proposed the following alternative statistic:

$$F_r = \frac{(B-1)\chi_r^2}{B(T-1) - \chi_r^2}, \tag{6}$$

which is distributed according to the F-distribution with $(T-1)$ and $(T-1)(B-1)$ degrees of freedom. Compared with χ_r^2, this statistic is not only less conservative but also more accurate for small sample sizes.[52]

In this study, blocks and treatments correspond to images and algorithms, respectively. For each $K \in \{4, 16, 64, 256\}$, our goal was to determine if at least one algorithm is significantly better than at least one other algorithm at the $\alpha = 0.05$ level of significance. If this is the case, we performed multiple comparison testing to determine which pairs of algorithms differ significantly. For this purpose, we employed the Bergmann–Hommel test[59] (also at the $\alpha = 0.05$ level), a powerful multiple comparison test that has been used successfully in various machine learning studies.[23,50,60–62] Bergmann–Hommel is a dynamic test that considers the logical relations among the hypotheses and is strictly more powerful[63] than various alternative tests that control the family-wise error rate such as the Nemenyi,[64] Holm,[65] and Shaffer[66] tests. Note that we performed nonparametric statistical tests only for the accelerated algorithms. This is because the Bergmann–Hommel test takes exponential time[67] in the number of hypotheses (for T treatments/algorithms, we have $T \times (T-1)/2$ pairwise hypotheses); thus, it cannot handle more than ten algorithms, even on a high-performance CPU.

All algorithms have a common parameter K that denotes the number of desired colors in the quantized image. In the experiments, we tested the following four K values: 4, 16, 64, and 256. The lower bound was set to 4 because a typical natural image can hardly be quantized to fewer than four colors. On the other hand, most natural images can be represented faithfully with no more than 256 colors. Hence, the upper bound was set to 256.

JKM and its accelerated variant AJKM have an additional parameter α that controls the degree of over-relaxation in the center updates. Recall that $\alpha \in [1, 2)$ ensures rapid convergence, and for $\alpha = 1$, JKM reduces to LKM. Accordingly, we tested the following five α values: 1.2, 1.4, 1.6, 1.8, and 1.99. We also tested the borderline α value of 2.0 (i.e., Jancey's choice) for which JKM is not guaranteed to converge. Indeed, in this case, JKM failed to converge in 194 (48.5%) out of 400 runs (100 images $\times$ $|\{4, 16, 64, 256\}|$ colors), confirming the theoretical results mentioned in Sec. 2.2.

3.2 Objective Assessment

In this section, we present the experimental results and discuss our findings for each performance measure separately.

3.2.1 *Mean squared error*

Figure 2 shows the box plots of the MSE distributions for ALKM, AJKM12 (AJKM with $\alpha = 1.2$), AJKM14 (AJKM with $\alpha = 1.4$), AJKM16 (AJKM with $\alpha = 1.6$), AJKM18 (AJKM with $\alpha = 1.8$), and AJKM199 (AJKM with $\alpha = 1.99$) for $K \in \{4, 16, 64, 256\}$. To facilitate comparisons, the MSE values were normalized as follows. For each input image and K value, the MSE values obtained by the six algorithms were divided by the smallest MSE obtained on that image. Therefore, in each case, the smaller the normalized MSE, the better (or more effective) the algorithm, with the best algorithm attaining a normalized MSE of one. Observe that the figure compares only the accelerated algorithms. This is because, for a given input image and K value, each such algorithm gives an identical MSE to its plain counterpart (e.g., ALKM versus LKM).

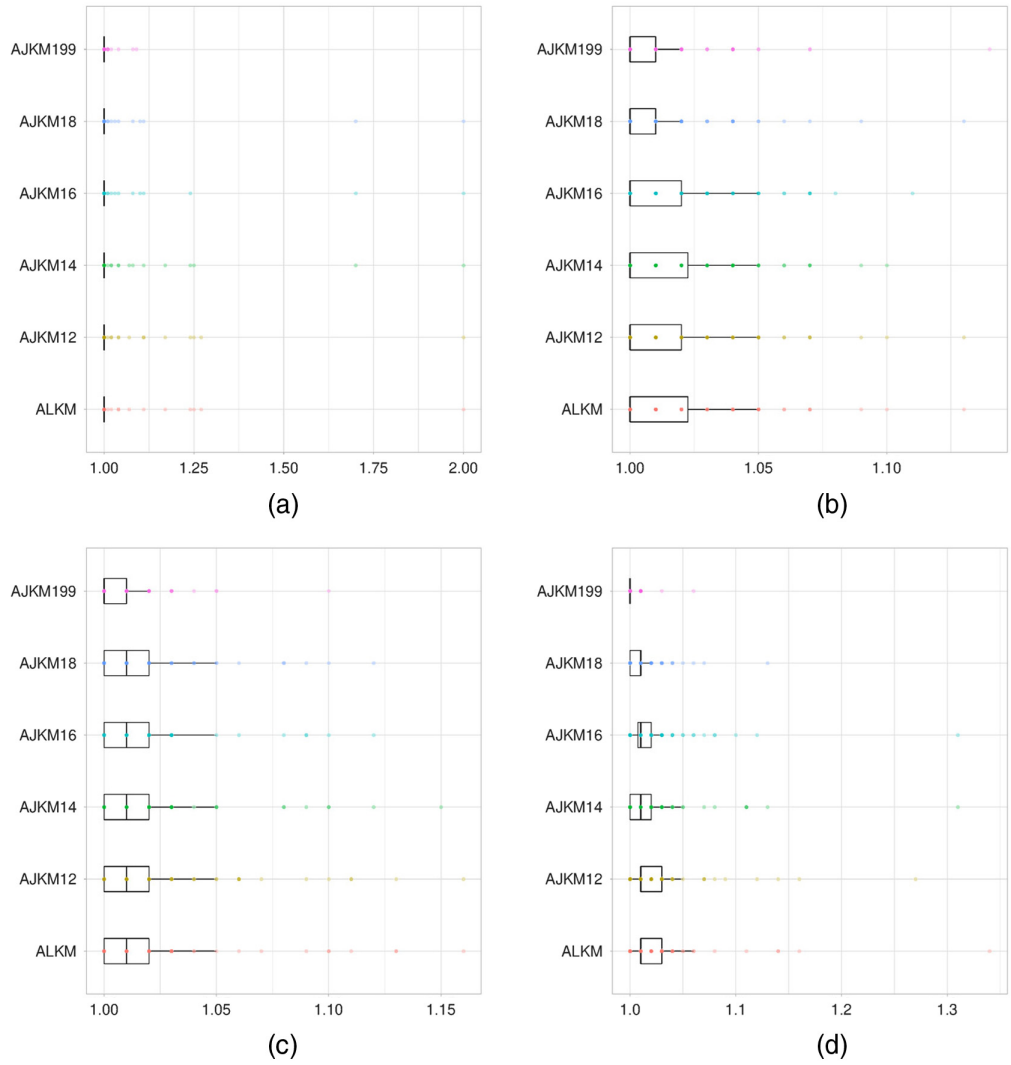

Fig. 2 Box plots of the normalized MSE distributions for each accelerated algorithm. (a) Four colors, (b) 16 colors, (c) 64 colors, and (d) 256 colors (in each subfigure, the *x* and *y* axes represent the normalized MSE values and the CQ algorithms, respectively).

Table 1 Mean MSE rank of each CQ algorithm over the dataset for $K \in \{4, 16, 64, 256\}$ (higher ranks are better).

Algorithm	$K = 4$	$K = 16$	$K = 64$	$K = 256$	Mean
ALKM	3.61	3.36	3.26	2.88	3.28
AJKM12	3.53	3.36	3.16	2.88	3.23
AJKM14	3.54	3.26	3.22	3.28	3.32
AJKM16	3.57	3.47	3.28	3.50	3.45
AJKM18	3.59	3.85	3.60	3.98	3.75
AJKM199	3.18	3.72	4.49	4.50	3.97

Table 1 gives the mean MSE rank of each CQ algorithm over the dataset for $K \in \{4, 16, 64, 256\}$ (higher ranks are better). The last column gives the mean of the (mean) ranks over the four K values. Based on this last column, generally speaking, the larger the α value, the better the algorithm.

Table 2 gives the results of the Friedman and Iman–Davenport tests for $K \in \{4, 16, 64, 256\}$. For $K = 4$ and $K = 16$, it can be seen that both tests failed to detect a statistically significant difference in MSE among the algorithms. In other words, the algorithms perform similarly when K is small. This can be explained by the fact that the problem of minimizing SSE is relatively easier for small K values because there are fewer local minima in which LKM/JKM can be trapped.

Table 2 shows that both the Friedman and Iman–Davenport tests detected a statistically significant difference in MSE among the algorithms for $K = 64$ and $K = 256$. Thus, we performed the Bergmann–Hommel test to determine which pairs of algorithms differ significantly in each case. The results are given in Table 3. We examine the two cases (i.e., $K = 64$ and $K = 256$) separately below.

For $K = 64$, only the five null hypotheses that involve AJKM199, namely, "ALKM versus AJKM199," "AJKM12 versus AJKM199," "AJKM14 versus AJKM199," "AJKM16 versus AJKM199," and "AJKM18 versus AJKM199," are rejected. As we know from Table 1 that AJKM199 has the highest (or best) mean rank for $K = 64$, the following relationship holds:

$$\text{AJKM199} > \{\text{ALKM}, \text{AJKM12}, \text{AJKM14}, \text{AJKM16}, \text{AJKM18}\},$$

where a notation such as $\{A, B\} > C$ indicates that there is no statistically significant difference between algorithms A and B, and these two are significantly better than algorithm C. Therefore, the above relationship means that AJKM199 is the best algorithm with respect to MSE for $K = 64$, whereas the remaining five algorithms perform similarly.

Table 2 Results of the Friedman and Iman–Davenport tests for MSE for $K \in \{4, 16, 64, 256\}$ (✓: rejected; ✗: not rejected).

	Friedman ($\alpha = 0.05$)			Iman–Davenport ($\alpha = 0.05$)		
K	$\chi_F^2(5)$	p	H_0	$F_r(5,495)$	p	H_0
4	3.754	0.585	✗	0.749	0.587	✗
16	7.694	0.174	✗	1.547	0.174	✗
64	37.021	5.93×10^{-07}	✓	7.916	3.45×10^{-07}	✓
256	58.923	6.23×10^{-11}	✓	13.225	4.16×10^{-12}	✓

Table 3 Results of the Bergmann–Hommel test for MSE for $K \in \{64, 256\}$ (✓: rejected; ✗: not rejected).

Null hypothesis	$K = 64$	$K = 256$
ALKM versus AJKM12	✗	✗
ALKM versus AJKM14	✗	✗
ALKM versus AJKM16	✗	✗
ALKM versus AJKM18	✗	✓
ALKM versus AJKM199	✓	✓
AJKM12 versus AJKM14	✗	✗
AJKM12 versus AJKM16	✗	✗
AJKM12 versus AJKM18	✗	✓
AJKM12 versus AJKM199	✓	✓
AJKM14 versus AJKM16	✗	✗
AJKM14 versus AJKM18	✗	✓
AJKM14 versus AJKM199	✓	✓
AJKM16 versus AJKM18	✗	✗
AJKM16 versus AJKM199	✗	✓
AJKM18 versus AJKM199	✓	✗

For $K = 256$, the situation is not as clear. In this case, the null hypotheses that involve only one of AJKM18 and AJKM199 (except "AJKM16 versus AJKM18") are rejected. Combined with the mean rank information for $K = 256$ given in Table 1, we can infer the following relationships:

$$\{\text{AJKM18}, \text{AJKM199}\} > \{\text{ALKM}, \text{AJKM12}, \text{AJKM14}\},$$
$$\text{AJKM199} > \{\text{ALKM}, \text{AJKM12}, \text{AJKM14}, \text{AJKM16}\}.$$

The above relationships can be interpreted as follows:

- $\{\text{AJKM18}, \text{AJKM199}\}$ is the best group of algorithms.
- $\{\text{ALKM}, \text{AJKM12}, \text{AJKM14}\}$ is the worst group of algorithms.
- AJKM16 is in between (it cannot be unambiguously classified because the Bergmann–Hommel test rejects "AJKM16 versus AJKM199" but not "AJKM16 versus AJKM18").

3.2.2 *Multi-scale structural similarity*

Figure 3 shows the box plots of the MS-SSIM distributions for ALKM, AJKM12, AJKM14, AJKM16, AJKM18, and AJKM199 for $K \in \{4, 16, 64, 256\}$. As in Fig. 2, this figure compares only the accelerated algorithms. This is because, for a given input image and K value, each such algorithm gives an identical MS-SSIM to its plain counterpart (e.g., ALKM versus LKM). Recall that, unlike MSE, MS-SSIM values are normalized in $[0, 1]$, and the larger the MS-SSIM, the better (or more effective) the algorithm.

Table 4 gives the mean MS-SSIM rank of each CQ algorithm over the dataset for $K \in \{4, 16, 64, 256\}$ (lower ranks are better). The last column gives the mean of the (mean) ranks over the four K values. Based on this last column, the larger the α value, the better the algorithm is on average.

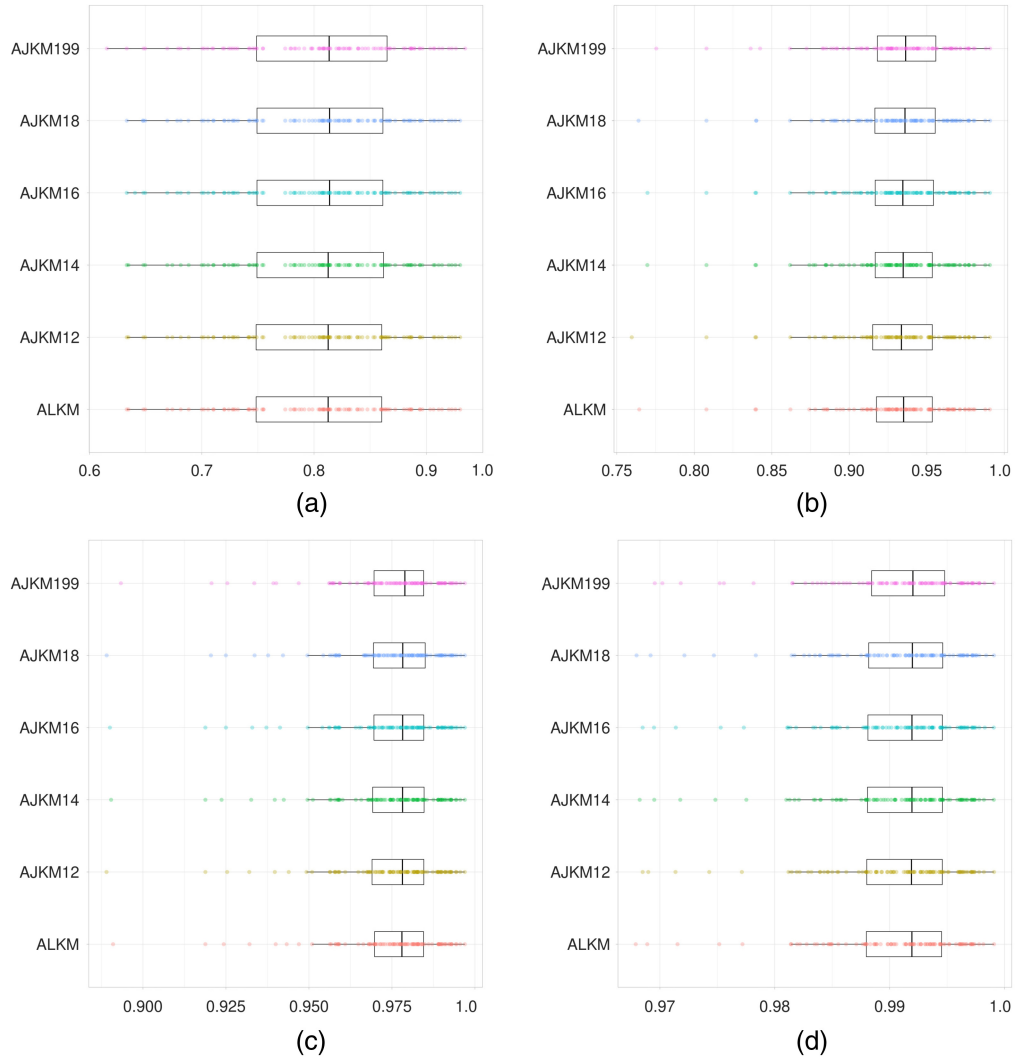

Fig. 3 Box plots of the MS-SSIM distributions for each accelerated algorithm. (a) Four colors, (b) 16 colors, (c) 64 colors, and (d) 256 colors (in each subfigure, the x and y axes represent the MS-SSIM values and the CQ algorithms, respectively).

Table 4 Mean MS-SSIM rank of each CQ algorithm over the dataset for $K \in \{4, 16, 64, 256\}$ (lower ranks are better).

Algorithm	$K = 4$	$K = 16$	$K = 64$	$K = 256$	Mean
ALKM	3.59	3.66	3.47	4.47	3.80
AJKM12	3.68	3.48	3.80	4.12	3.77
AJKM14	3.63	3.69	3.57	3.79	3.67
AJKM16	3.44	3.57	3.75	3.39	3.54
AJKM18	3.20	3.18	3.52	3.02	3.23
AJKM199	3.47	3.43	2.89	2.21	3.00

Table 5 gives the results of the Friedman and Iman–Davenport tests for $K \in \{4, 16, 64, 256\}$. For $K = 4$ and $K = 16$, it can be seen that both tests failed to detect a statistically significant difference in MS-SSIM among the algorithms. In other words, the algorithms perform similarly when K is small. Recall that such was also the case for MSE.

Table 5 Results of the Friedman and Iman–Davenport tests for MS-SSIM for $K \in \{4,16,64,256\}$ (✓: rejected; ✗: not rejected).

	Friedman ($\alpha = 0.05$)			Iman–Davenport ($\alpha = 0.05$)		
K	$\chi_r^2(5)$	p	H_0	$F_r(5,495)$	p	H_0
4	4.401	0.493	✗	0.879	0.495	✗
16	4.881	0.431	✗	0.976	0.432	✗
64	15.166	0.010	✓	3.097	0.009	✓
256	94.743	5.90×10^{-11}	✓	23.145	3.33×10^{-16}	✓

Table 5 shows that both the Friedman and Iman–Davenport tests detected a statistically significant difference in MS-SSIM among the algorithms for $K = 64$ and $K = 256$. Thus, we performed the Bergmann–Hommel test to determine which pairs of algorithms differ significantly in each case. The results are given in Table 6. We examine the two cases separately below.

For $K = 64$, only the following two null hypotheses are rejected: "AJKM12 versus AJKM199" and "AJKM16 versus AJKM199." Combined with the mean rank information for $K = 64$ given in Table 4, we can infer two alternative classifications:

- $\{\text{ALKM}, \text{AJKM14}, \text{AJKM18}, \text{AJKM199}\}$ and $\{\text{AJKM12}, \text{AJKM16}\}$
- $\{\text{ALKM}, \text{AJKM12}, \text{AJKM14}, \text{AJKM16}, \text{AJKM18}\}$ and AJKM199.

Unfortunately, the only relationship that can be ascertained among the algorithms is AJKM199 $>$ $\{\text{AJKM12}, \text{AJKM16}\}$, which means AJKM199 is better than both AJKM12 and AJKM16, which perform similarly.

Table 6 Results of the Bergmann–Hommel test for MS-SSIM for $K \in \{64,256\}$ (✓: rejected; ✗: not rejected).

Null hypothesis	$K = 64$	$K = 256$
ALKM versus AJKM12	✗	✗
ALKM versus AJKM14	✗	✓
ALKM versus AJKM16	✗	✓
ALKM versus AJKM18	✗	✓
ALKM versus AJKM199	✗	✓
AJKM12 versus AJKM14	✗	✗
AJKM12 versus AJKM16	✗	✓
AJKM12 versus AJKM18	✗	✓
AJKM12 versus AJKM199	✓	✓
AJKM14 versus AJKM16	✗	✗
AJKM14 versus AJKM18	✗	✓
AJKM14 versus AJKM199	✗	✓
AJKM16 versus AJKM18	✗	✗
AJKM16 versus AJKM199	✗	✓
AJKM18 versus AJKM199	✗	✓

For $K = 256$, all the null hypotheses are rejected but the following four: "ALKM versus AJKM12," "AJKM12 versus AJKM14," "AJKM14 versus AJKM16," and "AJKM16 versus AJKM18." Combined with the mean rank information for $K = 256$ given in Table 4, we can infer the following relationship:

$$\text{AJKM199} > \{\text{AJKM16}, \text{AJKM18}\} > \{\text{ALKM}, \text{AJKM12}\},$$

which can be interpreted as follows:

- AJKM199 is the best algorithm.
- $\{\text{ALKM}, \text{AJKM12}\}$ is the worst group of algorithms.
- $\{\text{AJKM16}, \text{AJKM18}\}$ and AJKM14 are in between (AJKM14 cannot be unambiguously classified because the Bergmann–Hommel test rejects "AJKM14 versus AJKM18" but not "AJKM12 versus AJKM14" or "AJKM14 versus AJKM16").

3.2.3 *CPU time*

Figure 4 shows the box plots of the CPU time distributions for LKM, JKM12 (JKM with $\alpha = 1.2$), JKM14 (JKM with $\alpha = 1.4$), JKM16 (JKM with $\alpha = 1.6$), JKM18 (JKM with $\alpha = 1.8$), JKM199 (JKM with $\alpha = 1.99$), and the corresponding accelerated algorithms, namely,

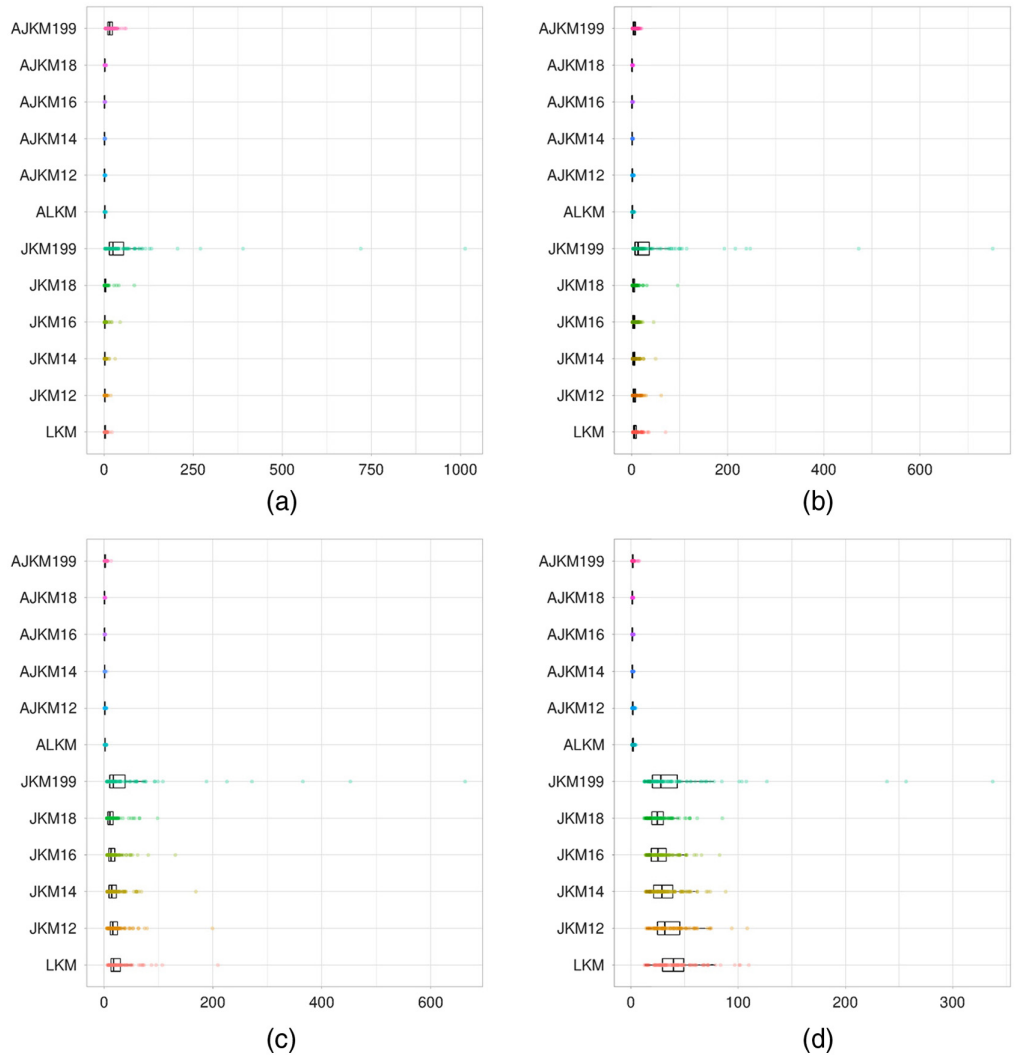

Fig. 4 Box plots of the normalized CPU time distributions for each algorithm. (a) Four colors, (b) 16 colors, (c) 64 colors, and (d) 256 colors (in each subfigure, the x and y axes represent the normalized CPU times and the CQ algorithms, respectively).

ALKM, AJKM12, AJKM14, AJKM16, AJKM18, and AJKM199, respectively. The CPU times were normalized as before, so that the fastest (or most efficient) algorithm took a normalized CPU time of one unit in each case. Observe that, for each K value, JKM199 is considerably slower than its rivals. In fact, JKM199's inefficiency stretches the x-axis so much that it becomes difficult to appreciate the differences among the other algorithms. Therefore, in Fig. 5, we give the same box plots with the JKM199 data removed.

Table 7 gives the mean CPU time rank of each CQ algorithm over the dataset for $K \in \{4, 16, 64, 256\}$ (higher ranks are better). The last column gives the mean of the (mean) ranks over the four K values.

Table 8 shows that both the Friedman and Iman–Davenport tests detected a statistically significant difference in CPU time among the algorithms for each K value tested. Thus, we performed the Bergmann–Hommel test to determine which pairs of algorithms differ significantly in each case. The results are given in Table 9. We examine the four cases separately below.

For $K = 4$, all the null hypotheses are rejected but the following two: "ALKM versus AJKM18" and "AJKM14 versus AJKM16." Combined with the mean rank information for $K = 4$ given in Table 7, we can infer the following relationship:

$$\{\text{AJKM14}, \text{AJKM16}\} > \text{AJKM12} > \{\text{ALKM}, \text{AJKM18}\} > \text{AJKM199},$$

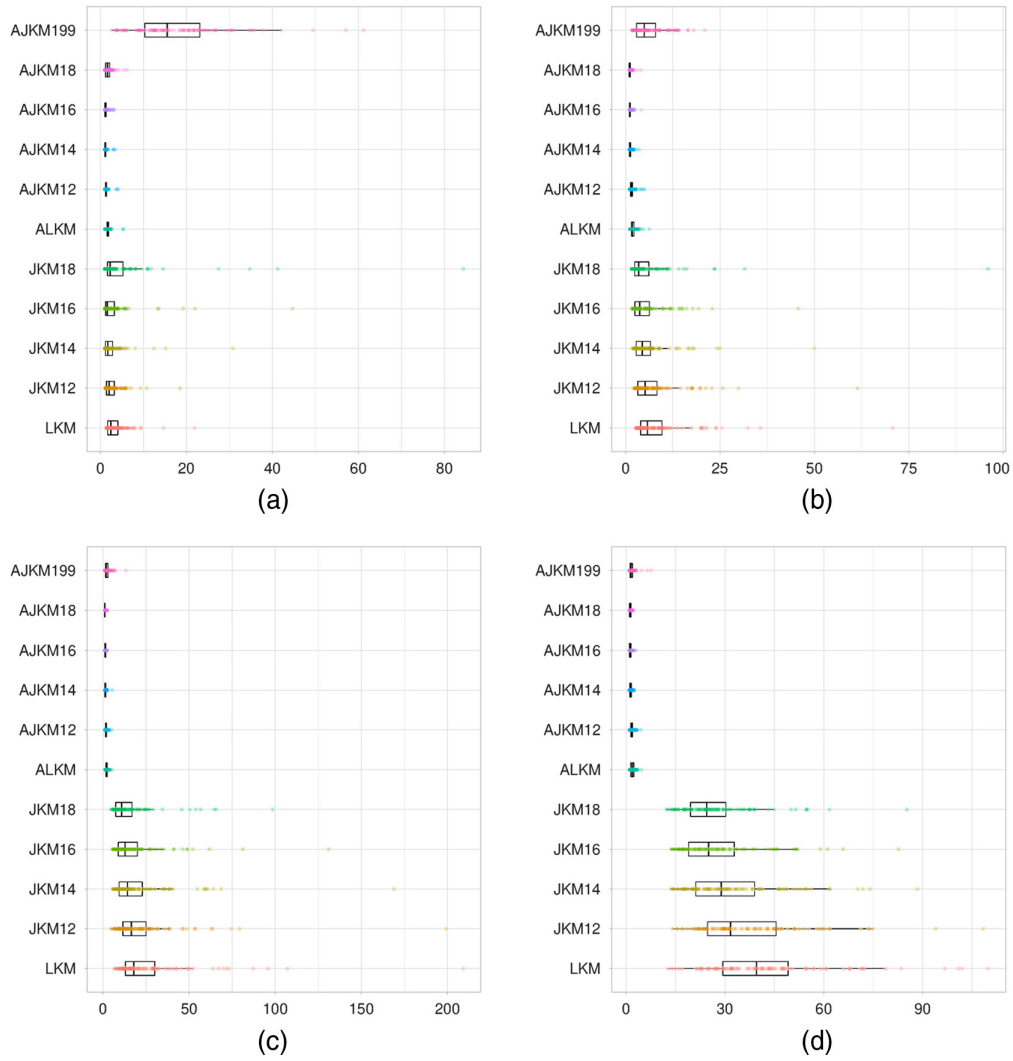

Fig. 5 Box plots of the normalized CPU time distributions for each algorithm but JKM199. (a) Four colors, (b) 16 colors, (c) 64 colors, and (d) 256 colors (in each subfigure, the x and y axes represent the normalized CPU times and the CQ algorithms, respectively).

Table 7 Mean CPU time rank of each CQ algorithm over the dataset for $K \in \{4, 16, 64, 256\}$ (higher ranks are better).

Algorithm	$K = 4$	$K = 16$	$K = 64$	$K = 256$	Mean
ALKM	2.75	2.34	1.95	2.17	2.30
AJKM12	3.94	3.25	3.01	2.63	3.21
AJKM14	5.02	4.50	3.97	3.83	4.33
AJKM16	5.03	4.78	4.59	4.39	4.69
AJKM18	3.27	4.95	4.87	4.58	4.41
AJKM199	1.00	1.19	2.63	3.41	2.06

Table 8 Results of the Friedman and Iman–Davenport tests for CPU time for $K \in \{4, 16, 64, 256\}$ (✓: rejected; ✗: not rejected).

	Friedman ($\alpha = 0.05$)			Iman–Davenport ($\alpha = 0.05$)		
K	$\chi_F^2(5)$	p	H_0	$F_r(5,495)$	p	H_0
4	334.359	1.39×10^{-10}	✓	199.838	-2.22×10^{-16}	✓
16	327.367	1.83×10^{-10}	✓	187.736	7.73×10^{-112}	✓
64	191.003	9.54×10^{-11}	✓	61.196	1.31×10^{-49}	✓
256	131.539	7.47×10^{-11}	✓	35.342	2.22×10^{-16}	✓

Table 9 Results of the Bergmann–Hommel test for CPU time for $K \in \{4, 16, 64, 256\}$ (✓: rejected; ✗: not rejected).

Null hypothesis	$K = 4$	$K = 16$	$K = 64$	$K = 256$
ALKM versus AJKM12	✓	✓	✓	✗
ALKM versus AJKM14	✓	✓	✓	✓
ALKM versus AJKM16	✓	✓	✓	✓
ALKM versus AJKM18	✗	✓	✓	✓
ALKM versus AJKM199	✓	✓	✓	✓
AJKM12 versus AJKM14	✓	✓	✓	✓
AJKM12 versus AJKM16	✓	✓	✓	✓
AJKM12 versus AJKM18	✓	✓	✓	✓
AJKM12 versus AJKM199	✓	✓	✗	✓
AJKM14 versus AJKM16	✗	✗	✓	✗
AJKM14 versus AJKM18	✓	✗	✓	✓
AJKM14 versus AJKM199	✓	✓	✓	✗
AJKM16 versus AJKM18	✓	✗	✗	✗
AJKM16 versus AJKM199	✓	✓	✓	✓
AJKM18 versus AJKM199	✓	✓	✓	✓

which can be interpreted as follows:

- {AJKM14, AJKM16} is the best group of algorithms.
- AJKM199 is the worst algorithm.
- AJKM12 and {ALKM, AJKM18} are in between, with the former being better than the latter.

For $K = 16$, all the null hypotheses are rejected but the following three: "AJKM14 versus AJKM16," "AJKM14 versus AJKM18," and "AJKM16 versus AJKM18." Combined with the mean rank information for $K = 16$ given in Table 7, we can infer the following relationship:

$$\{\text{AJKM14}, \text{AJKM16}, \text{AJKM18}\} > \text{AJKM12} > \text{ALKM} > \text{AJKM199},$$

which can be interpreted as follows:

- {AJKM14, AJKM16, AJKM18} is the best group of algorithms.
- AJKM199 is the worst algorithm.
- AJKM12 and ALKM are in between, with the former being better than the latter.

For $K = 64$, all the null hypotheses are rejected but the following two: "AJKM12 versus AJKM199" and "AJKM16 versus AJKM18." Combined with the mean rank information for $K = 64$ given in Table 7, we can infer the following relationship:

$$\{\text{AJKM16}, \text{AJKM18}\} > \text{AJKM14} > \{\text{AJKM12}, \text{AJKM199}\} > \text{ALKM},$$

which can be interpreted as follows:

- {AJKM16, AJKM18} is the best group of algorithms.
- ALKM is the worst algorithm.
- AJKM14 and {AJKM12, AJKM199} are in between, with the former being better than the latter.

For $K = 256$, all the null hypotheses are rejected but the following four: "ALKM versus AJKM12," "AJKM14 versus AJKM16," "AJKM14 versus AJKM199," and "AJKM16 versus AJKM18." Combined with the mean rank information for $K = 256$ given in Table 7, we can infer the following relationships:

$$\{\text{AJKM16}, \text{AJKM18}\} > \text{AJKM199} > \{\text{ALKM}, \text{AJKM12}\},$$
$$\text{AJKM18} > \{\text{AJKM14}, \text{AJKM199}\} > \{\text{ALKM}, \text{AJKM12}\}.$$

The above relationships can be interpreted as follows:

- {AJKM16, AJKM18} is the best group of algorithms.
- {ALKM, AJKM12} is the worst group of algorithms.
- {AJKM14, AJKM199} is in between.

3.2.4 *Number of iterations*

Figure 6 shows the box plots of the number of iterations distributions for ALKM, AJKM12, AJKM14, AJKM16, AJKM18, and AJKM199. As in Figs. 2 and 3, this figure compares only the accelerated algorithms. This is because each such algorithm iterates an identical number of times until reaching convergence as its plain counterpart (e.g., ALKM versus LKM). To facilitate comparisons, the number of iterations was also normalized as in the case of MSE.

Table 10 gives the mean number of iterations rank of each CQ algorithm over the dataset for $K \in \{4, 16, 64, 256\}$ (higher ranks are better). The last column gives the mean of the (mean) ranks over the four K values.

Table 11 shows that both the Friedman and Iman–Davenport tests detected a statistically significant difference in the number of iterations among the algorithms for each K value tested. Thus, we performed the Bergmann–Hommel test to determine which pairs of algorithms differ

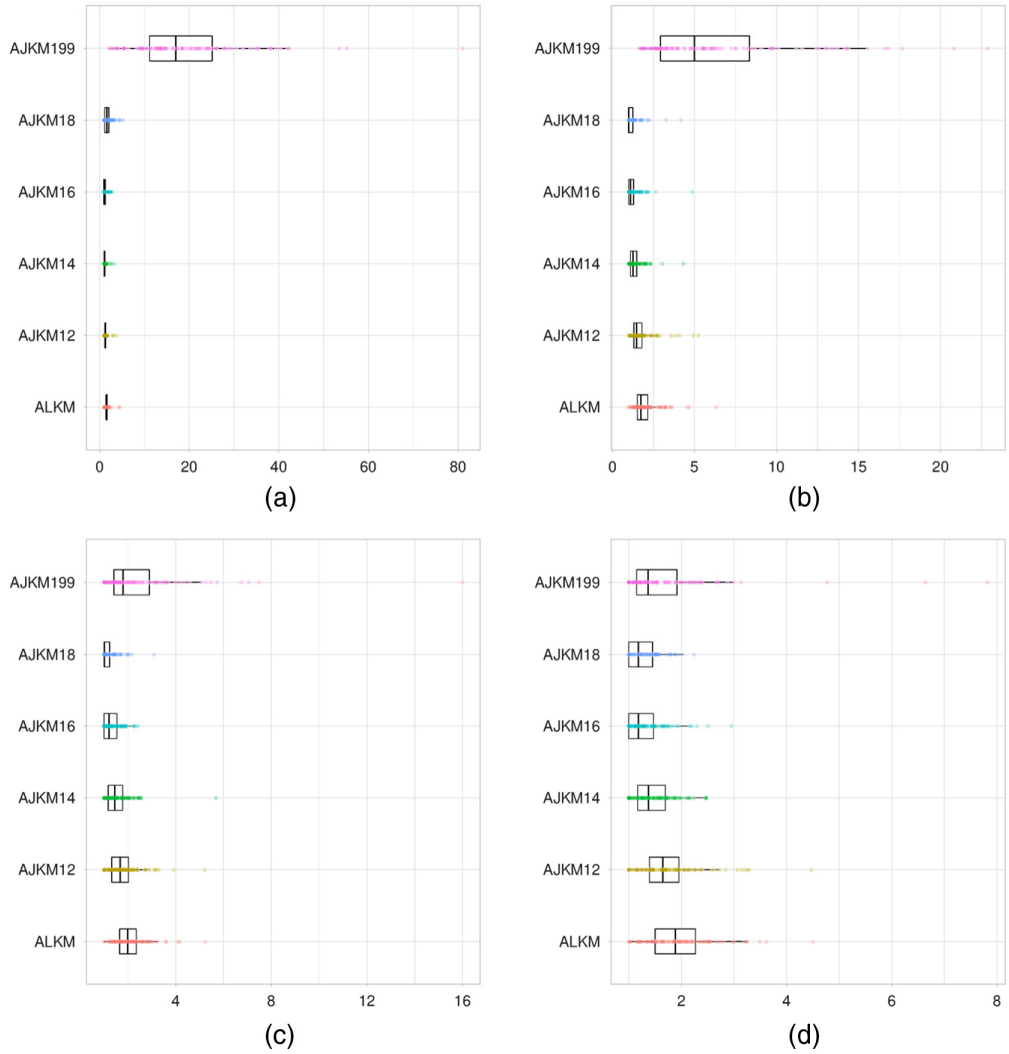

Fig. 6 Box plots of the distributions of the normalized number of iterations for each accelerated algorithm. (a) Four colors, (b) 16 colors, (c) 64 colors, and (d) 256 colors (in each subfigure, the x and y axes represent the normalized number of iterations and the CQ algorithms, respectively).

Table 10 Mean number of iterations rank of each CQ algorithm over the dataset for $K \in \{4, 16, 64, 256\}$ (higher ranks are better).

Algorithm	$K = 4$	$K = 16$	$K = 64$	$K = 256$	Mean
ALKM	2.80	2.36	2.06	2.21	2.35
AJKM12	3.92	3.28	3.06	2.75	3.25
AJKM14	4.91	4.31	3.73	3.63	4.14
AJKM16	5.12	4.87	4.60	4.38	4.74
AJKM18	3.26	5.01	4.89	4.61	4.44
AJKM199	1.00	1.19	2.68	3.43	2.07

significantly in each case. The results are given in Table 12. We examine the four cases separately below.

For $K = 4$, all the null hypotheses are rejected but the following two: "ALKM versus AJKM18" and "AJKM14 versus AJKM16." Combined with the mean rank information for $K = 4$ given in Table 10, we can infer the following relationship:

Table 11 Results of the Friedman and Iman–Davenport tests for number of iterations for $K \in \{4, 16, 64, 256\}$ (✓: rejected; ✗: not rejected).

	Friedman ($\alpha = 0.05$)			Iman–Davenport ($\alpha = 0.05$)		
K	$\chi_F^2(5)$	p	H_0	$F_r(5,495)$	p	H_0
4	330.521	1.81×10^{-10}	✓	193.072	-2.22×10^{-16}	✓
16	328.857	1.78×10^{-10}	✓	190.232	9.11×10^{-113}	✓
64	174.911	1.11×10^{-10}	✓	53.266	-2.22×10^{-16}	✓
256	121.959	8.95×10^{-11}	✓	31.938	-2.22×10^{-16}	✓

$$\{\text{AJKM14}, \text{AJKM16}\} > \text{AJKM12} > \{\text{ALKM}, \text{AJKM18}\} > \text{AJKM199},$$

which can be interpreted as follows:

- $\{\text{AJKM14}, \text{AJKM16}\}$ is the best group of algorithms.
- AJKM199 is the worst algorithm.
- AJKM12 and $\{\text{ALKM}, \text{AJKM18}\}$ are in between, with the former being better than the latter.

For $K = 16$, all the null hypotheses are rejected but the following: "AJKM16 versus AJKM18." Combined with the mean rank information for $K = 16$ given in Table 10, we can infer the following relationship:

$$\{\text{AJKM16}, \text{AJKM18}\} > \text{AJKM14} > \text{AJKM12} > \text{ALKM} > \text{AJKM199},$$

which can be interpreted as follows:

- $\{\text{AJKM16}, \text{AJKM18}\}$ is the best group of algorithms.
- AJKM199 is the worst algorithm.
- AJKM14, AJKM12, and ALKM are in between, where AJKM14 is better than AJKM12, which is better than ALKM.

For $K = 64$, all the null hypotheses are rejected but the following two: "AJKM12 versus AJKM199" and "AJKM16 versus AJKM18." Combined with the mean rank information for $K = 64$ given in Table 10, we can infer the following relationship:

$$\{\text{AJKM16}, \text{AJKM18}\} > \text{AJKM14} > \{\text{AJKM12}, \text{AJKM199}\} > \text{ALKM},$$

which can be interpreted as follows:

- $\{\text{AJKM16}, \text{AJKM18}\}$ is the best group of algorithms.
- ALKM is the worst algorithm.
- AJKM14 and $\{\text{AJKM12}, \text{AJKM199}\}$ are in between, where the former is better than the latter.

For $K = 256$, all the null hypotheses are rejected but the following three: "ALKM versus AJKM12," "AJKM14 versus AJKM199," and "AJKM16 versus AJKM18." Combined with the mean rank information for $K = 256$ given in Table 10, we can infer the following relationship:

$$\{\text{AJKM16}, \text{AJKM18}\} > \{\text{AJKM14}, \text{AJKM199}\} > \{\text{ALKM}, \text{AJKM12}\},$$

which can be interpreted as follows:

- $\{\text{AJKM16}, \text{AJKM18}\}$ is the best group of algorithms.
- $\{\text{ALKM}, \text{AJKM12}\}$ is the worst group of algorithms.
- $\{\text{AJKM14}, \text{AJKM199}\}$ is in between.

Table 12 Results of the Bergmann–Hommel test for number of iterations for $K \in \{4, 16, 64, 256\}$ (✓: rejected; ✗: not rejected).

Null hypothesis	$K = 4$	$K = 16$	$K = 64$	$K = 256$
ALKM versus AJKM12	✓	✓	✓	✗
ALKM versus AJKM14	✓	✓	✓	✓
ALKM versus AJKM16	✓	✓	✓	✓
ALKM versus AJKM18	✗	✓	✓	✓
ALKM versus AJKM199	✓	✓	✓	✓
AJKM12 versus AJKM14	✓	✓	✓	✓
AJKM12 versus AJKM16	✓	✓	✓	✓
AJKM12 versus AJKM18	✓	✓	✓	✓
AJKM12 versus AJKM199	✓	✓	✗	✓
AJKM14 versus AJKM16	✗	✓	✓	✓
AJKM14 versus AJKM18	✓	✓	✓	✓
AJKM14 versus AJKM199	✓	✓	✓	✗
AJKM16 versus AJKM18	✓	✗	✗	✗
AJKM16 versus AJKM199	✓	✓	✓	✓
AJKM18 versus AJKM199	✓	✓	✓	✓

3.2.5 *Summary and additional comments*

To summarize, our statistical significance analyses demonstrated that LKM and JKM with $\alpha \in \{1.2, 1.4, 1.6, 1.8, 1.99\}$ have similar effectiveness when K is small (i.e., $K \in \{4, 16\}$). For larger K values (i.e., $K \in \{64, 256\}$), JKM199 is significantly more effective than its peers, and for $K = 256$, JKM18 shares the top spot with JKM199 or is a second best. On the other hand, the algorithms exhibit a noticeably more varied behavior in terms of efficiency. In general, AJKM16 and AJKM18 are the most efficient algorithms, whereas ALKM and AJKM199 are the least efficient. Hence, an accelerated JKM implementation with $\alpha = 1.8$ strikes the best balance between effectiveness and efficiency and is a strong alternative to an accelerated implementation of the popular LKM algorithm. Although AJKM18 is not significantly more effective than ALKM (except for $K = 256$), the former is significantly more efficient than the latter. For example, AJKM18 took, on average, ≈75, 200, 500, and 1500 ms to quantize a 768×512 (or 512×768) image to 4, 16, 64, and 256 colors, respectively, whereas the corresponding average CPU times for ALKM were ≈100, 400, 1000, and 2500 ms, respectively. Interestingly, Drezner[68] also empirically determined $\alpha = 1.8$ to be the best relaxation factor value in a different application domain (i.e., facility location).

At first glance, Figs. 5 and 6 appear to contradict each other, at least for $K \in \{16, 64, 256\}$. There is a simple explanation for this. For LKM and JKM, the per-iteration time complexity of the assignment and update steps are $\mathcal{O}(NK)$ and $\mathcal{O}(K)$, respectively. As $N \gg K$ in CQ applications, the algorithms spend most of their time in the assignment step. In fact, these algorithms perform exactly NK distance computations in every iteration. Therefore, there is a near-perfect correlation (>0.995) between CPU time and the number of iterations for plain algorithms. This phenomenon can be observed by comparing the bottom half of each box plot in Fig. 5 (corresponding to LKM and JKM with $\alpha \in \{1.2, 1.4, 1.6, 1.8\}$) to the corresponding box plot in Fig. 6. On the other hand, for ALKM and AJKM, thanks to the TIE technique, the number of distance computations performed per iteration decreases rapidly as the centers stabilize. In other words, these algorithms become progressively faster as they iterate. This is why CPU time is not necessarily proportional to the number of iterations for the accelerated algorithms.

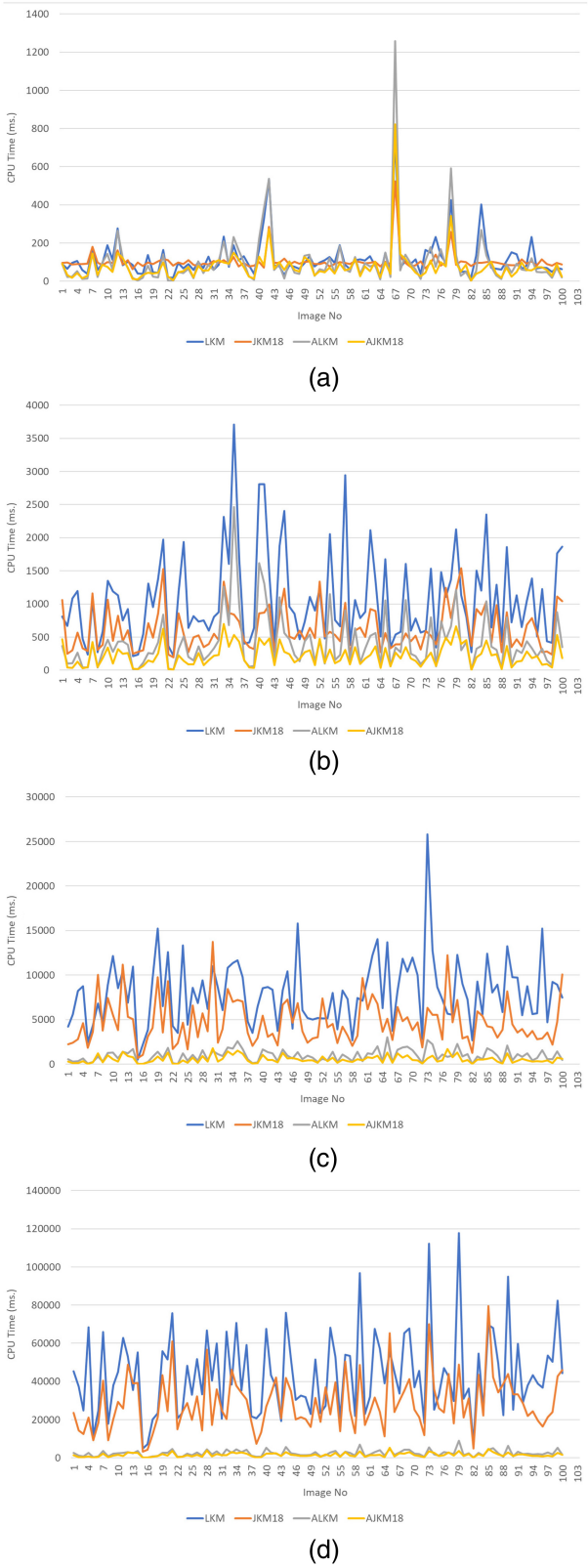

Fig. 7 CPU time (in milliseconds) for each image for LKM, JKM18, ALKM, and AJKM18. (a) Four colors, (b) 16 colors, (c) 64 colors, and (d) 256 colors (in each subfigure, the *x* and *y* axes represent the image numbers and the CPU times, respectively).

Now that we examined the effectiveness and efficiency of LKM and JKM (with various α values), let us compare the top performer, JKM with $\alpha = 1.8$, both in its plain (i.e., JKM18) and accelerated (i.e., AJKM18) variants against the popular LKM algorithm and its accelerated variant, ALKM. Figure 7 plots each algorithm's CPU time in milliseconds (averaged over 10 independent runs) for each image for $K \in \{4, 16, 64, 256\}$. From Fig. 5, the ordering of the algorithms (from the slowest to the fastest) appears to be roughly LKM, JKM18, ALKM, and AJKM18. Figure 7 confirms this observation and clearly shows that the efficiency differences between the plain and accelerated algorithms (i.e., LKM versus ALKM and JKM18 versus AJKM18) increase with K, which is not surprising because the overhead associated with the TIE technique pays off only if K is sufficiently large, and in general, the larger the K value, the fewer distance computations TIE performs compared with plain LKM/JKM.

3.3 Subjective Assessment

Figures 8, 10, 12, and 14 show the Columbia crew, Trade Fair Tower, coloring pencils, and color checker images, respectively, quantized using LKM and JKM with three different α values.

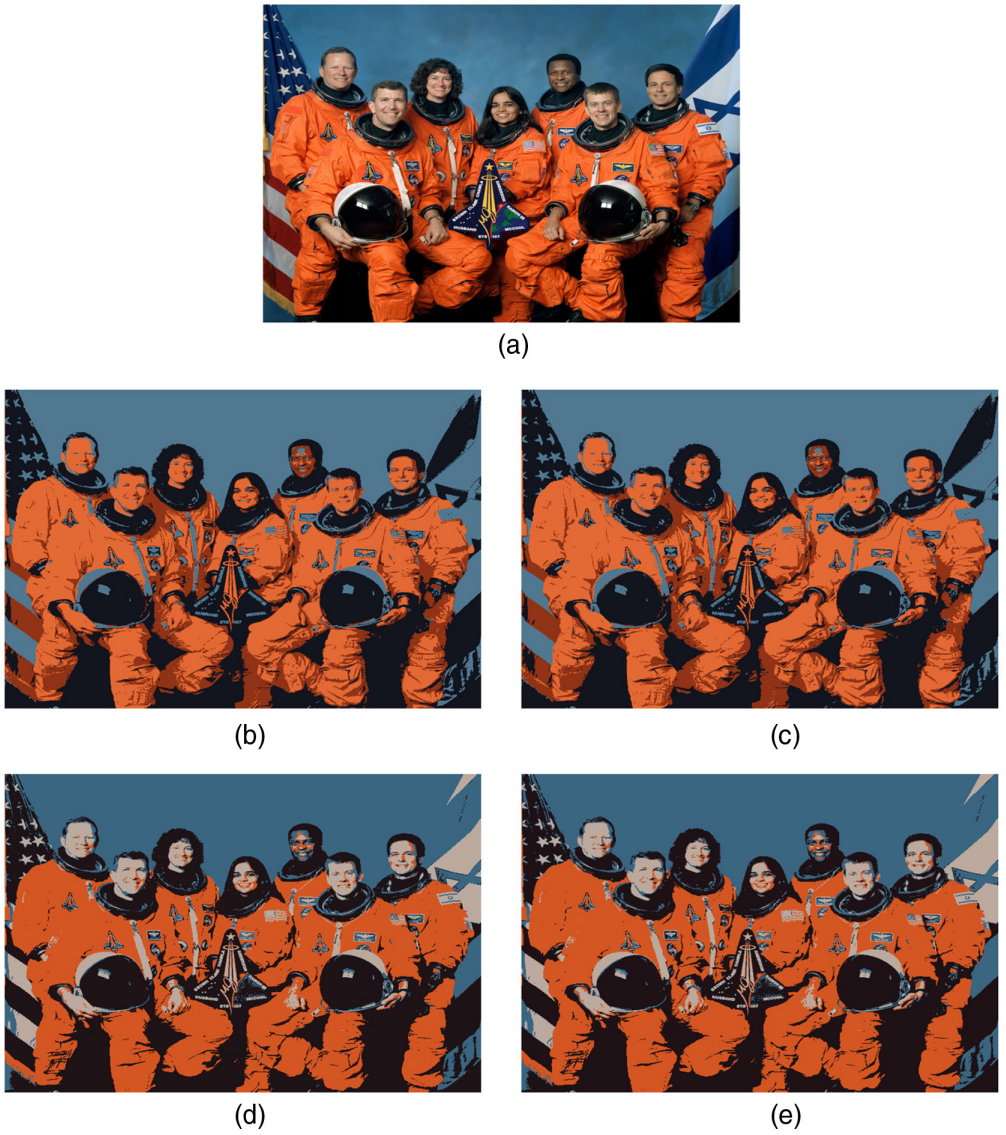

Fig. 8 Columbia crew (148,399) colors and its various quantized versions (four colors). (a) Columbia crew. (b) LKM (MSE = 2753). (c) JKM12 (MSE = 2753). (d) JKM18 (MSE = 2221). (e) JKM199 (MSE = 2221).

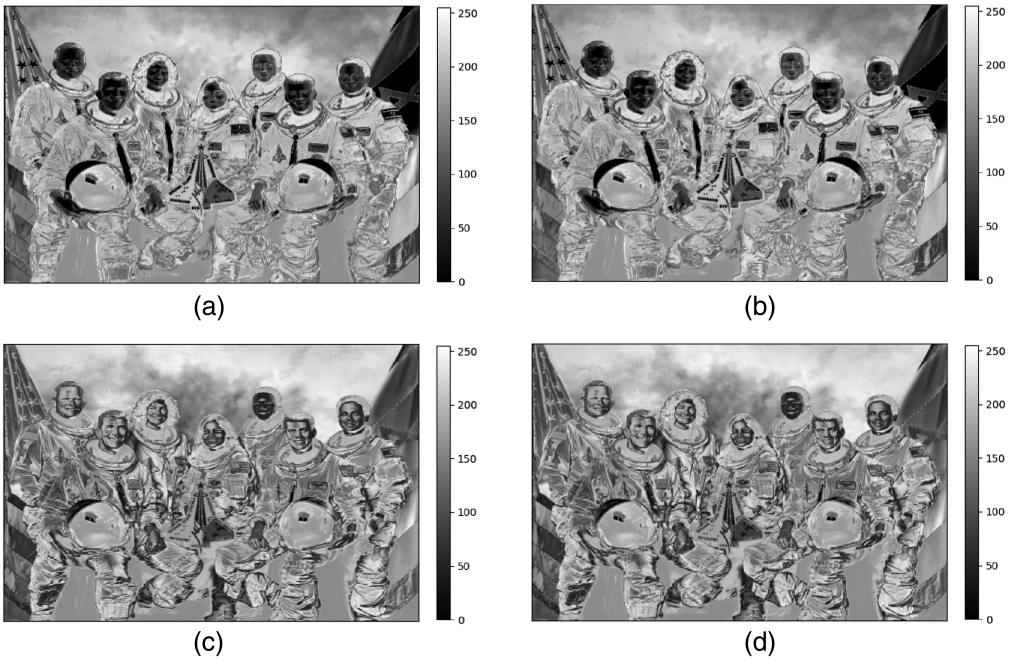

Fig. 9 Error images corresponding to Fig. 8. (a) LKM. (b) JKM12. (c) JKM18. (d) JKM199.

On the other hand, Figs. 9, 11, 13, and 15 show the error images for Figs. 8, 10, 12, and 14, respectively. In each case, the error image is obtained by amplifying the pixel-wise normalized Euclidean differences between the input and output by a factor of 4 and then negating them for better visualization.[20] Hence, the cleaner (or lighter) the error image, the better the reproduction of the input image. For small K values (i.e., $K \in \{4, 16\}$), we can observe the differences among the algorithms more easily. For example, both LKM and JKM12 cause significant color shifts/losses in the skin tones and the white parts of the spacesuits and helmets when quantizing the Columbia crew image to four colors (refer to Figs. 8 and 9). By contrast, JKM18 and JKM199 produce much better results despite the extremely small number of colors they are allowed. As another example, consider the task of quantizing the color checker image to 256 colors. At first, this task may appear to be much easier, as the algorithms are asked to represent 24 color patches with 256 colors. However, the large and uniformly colored patches are challenging to quantize without generating false contours. In addition, some of the achromatic colors at the bottom row are visually difficult to distinguish. In this case, all four quantized images appear nearly identical. However, an inspection of the corresponding error images reveals that the algorithms can be differentiated by the bottom rows of their outputs. In particular, it is evident that JKM18 and JKM199 reproduce the input more accurately than LKM and JKM14.

4 Conclusions and Future Work

In this paper, we investigated the performance of two k-means variants, namely, LKM and JKM, on the color quantization problem. LKM is a very popular partitional clustering algorithm, which alternates between an assignment step (where each data point is assigned to the nearest cluster center) and an update step (where each center is recomputed as the centroid of its cluster). On the other hand, JKM is a relatively unknown variant of LKM featuring a different update step (where each center is recomputed as a linear combination of itself and the centroid of its cluster). JKM is parameterized by the coefficient α of the linear combination, whose value primarily influences the algorithm's convergence speed. For each algorithm, we presented three implementations: plain (LKM and JKM), weighted (WLKM and WJKM), and accelerated and weighted (ALKM and AJKM, respectively). Although all three implementations of a given algorithm produce identical results, they each have markedly different computational requirements.

(a)

(b)

(c)

(d)

(e)

Fig. 10 Trade Fair Tower (72,299 colors) and its various quantized versions (16 colors). (a) Trade Fair Tower. (b) LKM (MSE = 216). (c) JKM12 (MSE = 216). (d) JKM18 (MSE = 191). (e) JKM199 (MSE = 191).

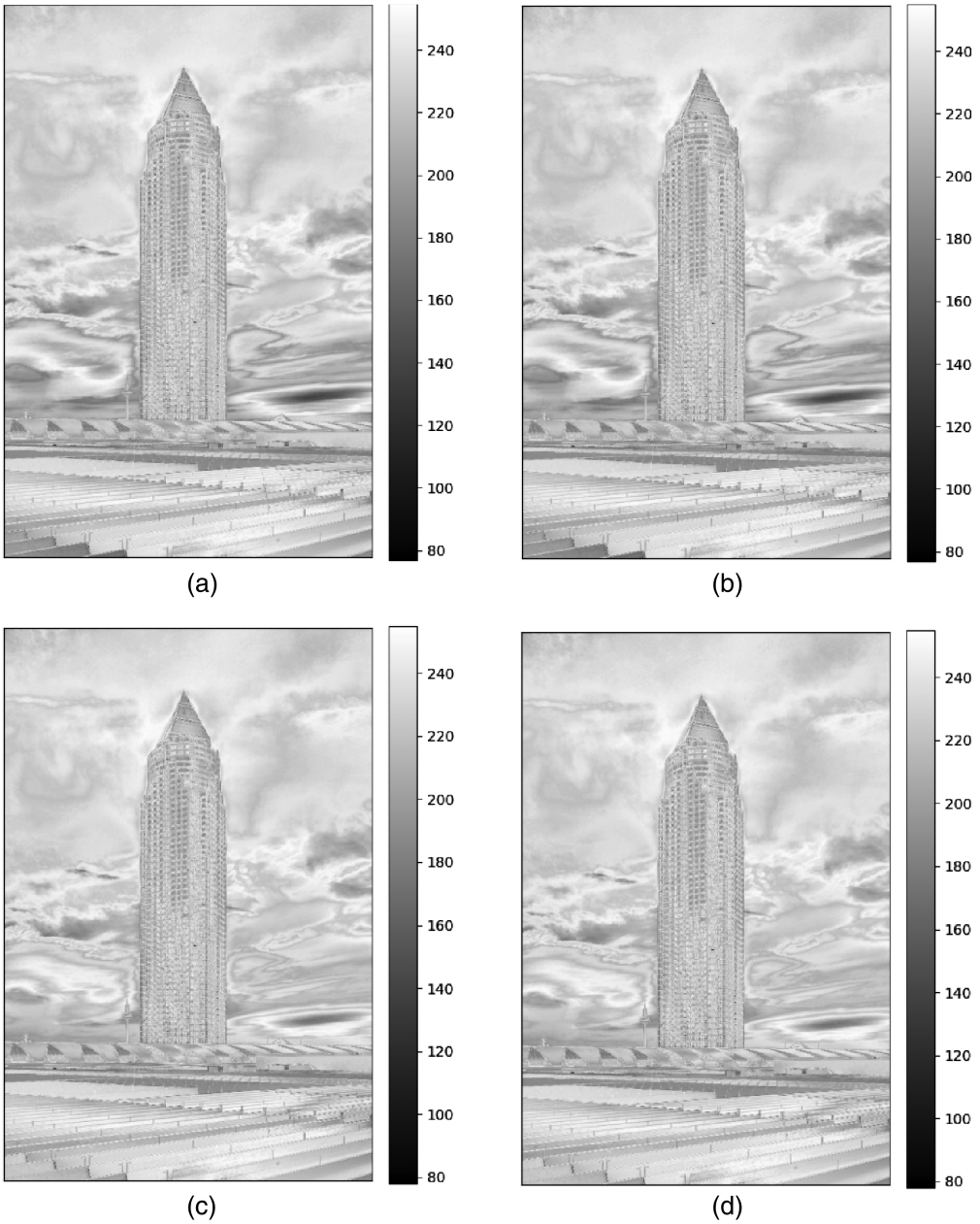

Fig. 11 Error images corresponding to Fig. 10. (a) LKM. (b) JKM12. (c) JKM18. (d) JKM199.

The weighted implementation is faster than the plain one because it operates on a smaller dataset, whereas the accelerated and weighted implementation is faster than the weighted one because it eliminates unnecessary distance computations (using the triangle inequality). Extensive experiments conducted on the CQ100 dataset allowed us to evaluate the effectiveness and efficiency of each algorithm for various numbers of colors. We determined that the choice of α primarily impacts JKM's efficiency, with larger α values often performing better (unless α is too large, e.g., 1.99). In our application, JKM with $\alpha = 1.8$ achieved the best trade-off between effectiveness and efficiency, leading to a color quantizer that is at least as effective as the popular LKM but significantly more efficient. In addition, JKM is no more difficult to implement than LKM. Future work includes developing an adaptive scheme to determine a near-optimal α value for a given input image and exploring the applicability of JKM to higher-dimensional clustering problems.

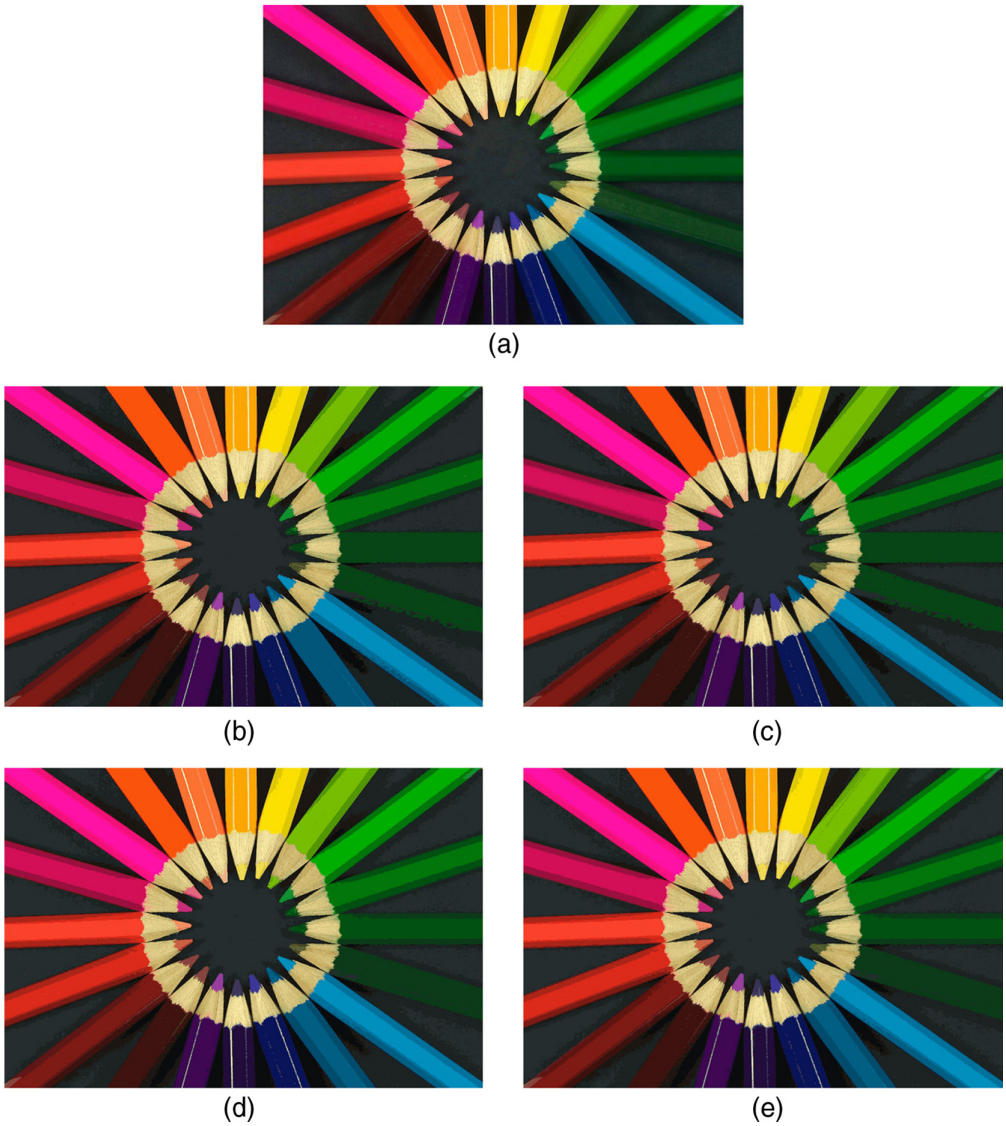

Fig. 12 Coloring pencils (115,201 colors) and its various quantized versions (64 colors). (a) Coloring pencils. (b) LKM (MSE = 149). (c) JKM16 (MSE = 152). (d) JKM18 (MSE = 137). (e) JKM199 (MSE = 135).

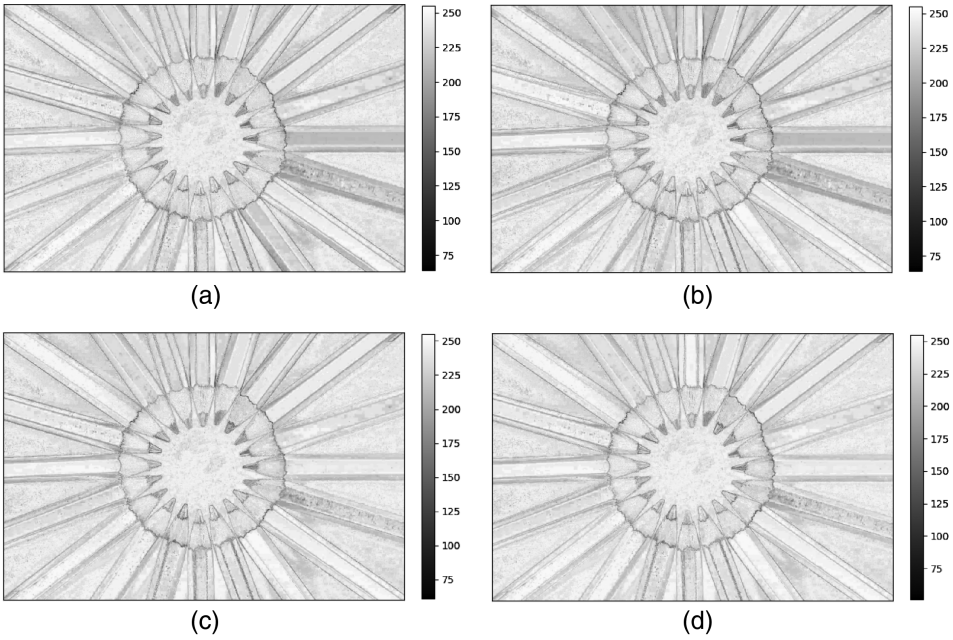

Fig. 13 Error images corresponding to Fig. 12. (a) LKM. (b) JKM16. (d) JKM18. (d) JKM199.

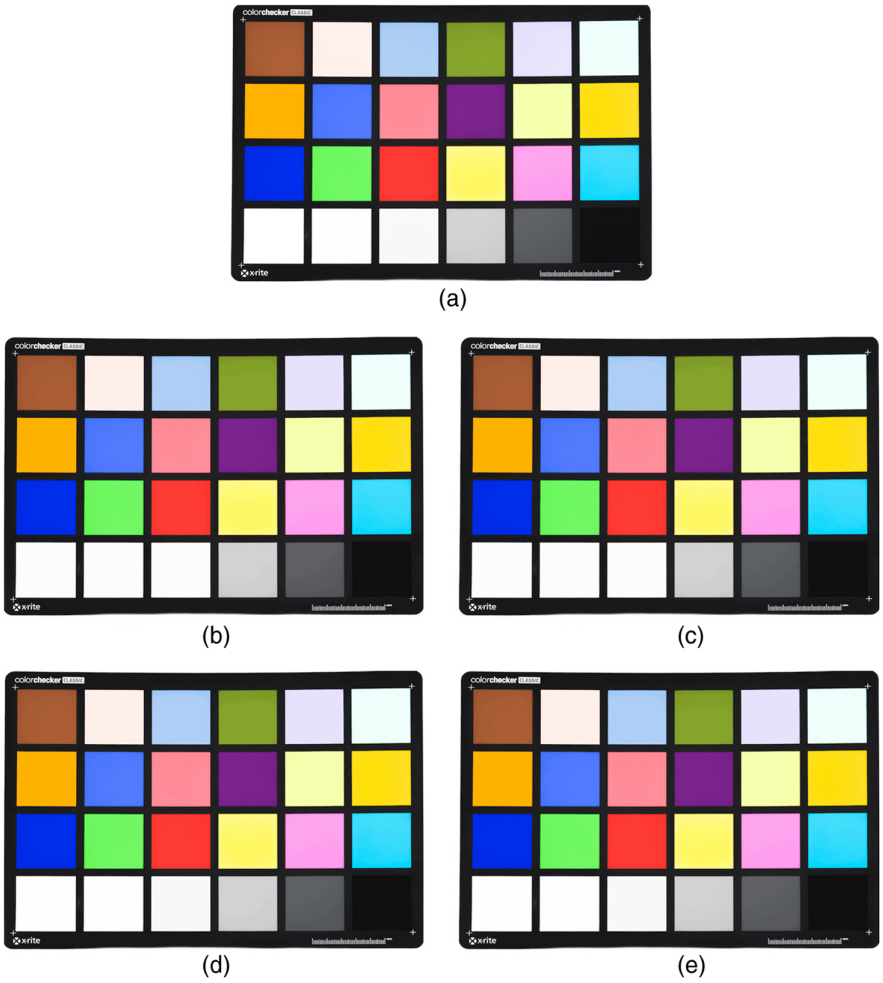

Fig. 14 Color checker (30,593 colors) and its various quantized versions (256 colors). (a) Color checker. (b) LKM (MSE = 14). (c) JKM14 (MSE = 14). (d) JKM18 (MSE = 11). (e) JKM199 (MSE = 10).

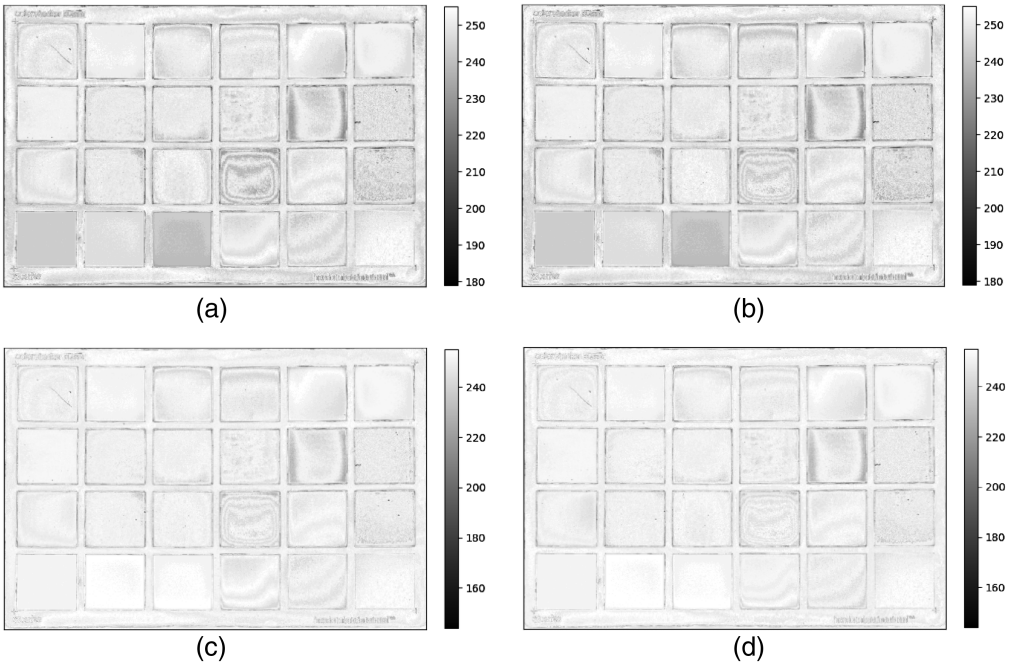

Fig. 15 Error images corresponding to Fig. 14. (a) LKM. (b) JKM14. (c) JKM18. (d) JKM199.

Disclosures

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Code and Data Availability

The source code of the *k*-means–based color quantizers described in this paper will be made available at https://github.com/HarrisonBounds. In addition, the 2400 output images (six CQ algorithms $\times 100$ input images $\times |\{4, 16, 64, 256\}|$ colors) and Microsoft Excel worksheets containing the MSE and MS-SSIM for each input/output image combination will be released as part of the next version of CQ100 (https://data.mendeley.com/datasets/vw5ys9hfxw/3).

Acknowledgments

This material is based upon work supported by the National Science Foundation (Grant No. OIA-1946391). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. The authors gratefully acknowledge the open-source web-based graphing software PlotsOfData.[69]

References

1. R. Ramanath et al., "Color image processing pipeline," *IEEE Signal Process. Mag.* **22**(1), 34–43 (2005).
2. P. Heckbert, "Color image quantization for frame buffer display," *ACM SIGGRAPH Comput. Graph.* **16**(3), 297–307 (1982).
3. M. Orchard and C. Bouman, "Color quantization of images," *IEEE Trans. Signal Process.* **39**(12), 2677–2690 (1991).
4. X. Wu, "Color quantization by dynamic programming and principal analysis," *ACM Trans. Graph.* **11**(4), 348–372 (1992).
5. A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: a review," *ACM Comput. Surv.* **31**(3), 264–323 (1999).
6. M. E. Celebi, "Forty years of color quantization: a modern, algorithmic survey," *Artif. Intell. Rev.* **56**, 13953–14034 (2023).
7. R. S. Gentile, J. P. Allebach, and E. Walowit, "Quantization of color images based on uniform color spaces," *J. Imaging Technol.* **16**(1), 11–21 (1990).

8. E. Forgy, "Cluster analysis of multivariate data: efficiency vs. interpretability of classifications," *Biometrics* **21**, 768 (1965).
9. R. C. Jancey, "Multidimensional group analysis," *Austr. J. Botany* **14**(1), 127–130 (1966).
10. Y. Linde, A. Buzo, and R. M. Gray, "An algorithm for vector quantizer design," *IEEE Trans. Commun.* **28**(1), 84–95 (1980).
11. S. Lloyd, "Least squares quantization in PCM," *IEEE Trans. Inf. Theory* **28**(2), 129–137 (1982).
12. Y. C. Hu and M. G. Lee, "K-means based color palette design scheme with the use of stable flags," *J. Electron. Imaging* **16**(3), 033003 (2007).
13. Y. C. Hu and B. H. Su, "Accelerated K-means clustering algorithm for colour image quantization," *Imaging Sci. J.* **56**(1), 29–40 (2008).
14. Y. C. Hu, M. G. Lee, and P. Tsai, "Colour palette generation schemes for colour image quantization," *Imaging Sci. J.* **57**(1), 46–59 (2009).
15. M. E. Celebi, "Fast color quantization using weighted sort-means clustering," *J. Opt. Soc. Am. A* **26**(11), 2434–2443 (2009).
16. M. E. Celebi, "Improving the performance of K-means for color quantization," *Image Vis. Comput.* **29**(4), 260–271 (2011).
17. Q. Wen and M. E. Celebi, "Hard vs. fuzzy C-means clustering for color quantization," *EURASIP J. Adv. Signal Process.* **2011**(1), 118–129 (2011).
18. S. C. Huang, "An efficient palette generation method for color image quantization," *Appl. Sci.* **11**(3), 1043 (2021).
19. A. D. Abernathy and M. E. Celebi, "The incremental online k-means clustering algorithm and its application to color quantization," *Expert Syst. Appl.* **207**, 117927 (2022).
20. M. E. Celebi and M. L. Pérez-Delgado, "CQ100: a high-quality image dataset for color quantization research," *J. Electron. Imaging* **32**(3), 033019 (2023).
21. "The android open source project," CelebiQuantizer.java, https://tinyurl.com/bd24un54 (2021).
22. X. Wu et al., "Top 10 algorithms in data mining," *Knowl. Inf. Syst.* **14**(1), 1–37 (2008).
23. M. E. Celebi, H. Kingravi, and P. A. Vela, "A comparative study of efficient initialization methods for the K-means clustering algorithm," *Expert Syst. Appl.* **40**(1), 200–210 (2013).
24. S. Z. Selim and M. A. Ismail, "K-means-type algorithms: a generalized convergence theorem and characterization of local optimality," *IEEE Trans. Pattern Anal. Mach. Intell.* **PAMI-6**(1), 81–87 (1984).
25. L. M. Ostresh, Jr., "On the convergence of a class of iterative methods for solving the weber location problem," *Oper. Res.* **26**(4), 597–609 (1978).
26. A. Hadjidimos, "Successive overrelaxation (SOR) and related methods," *J. Comput. Appl. Math.* **123**(1–2), 177–199 (2000).
27. B. C. Peters, Jr. and H. F. Walker, "An iterative procedure for obtaining maximum-likelihood estimates of the parameters for a mixture of normal distributions," *SIAM J. Appl. Math.* **35**(2), 362–378 (1978).
28. B. C. Peters, Jr. and H. F. Walker, "The numerical evaluation of the maximum-likelihood estimate of a subset of mixture proportions," *SIAM J. Appl. Math.* **35**(3), 447–452 (1978).
29. M. R. Anderberg, *Cluster Analysis for Applications*, Academic Press (1973).
30. G. Hamerly and J. Drake, "Accelerating Lloyd's algorithm for K-means clustering," in *Partitional Clustering Algorithms*, M. E. Celebi, Ed., pp. 41–78, Springer (2015).
31. S. Wang, Y. Sun, and Z. Bao, "On the efficiency of K-means clustering: evaluation, optimization, and algorithm selection," *Proc. VLDB Endowment* **14**(2), 163–175 (2020).
32. T. Kanungo et al., "An efficient K-means clustering algorithm: analysis and implementation," *IEEE Trans. Pattern Anal. Mach. Intell.* **24**(7), 881–892 (2002).
33. C. Elkan, "Using the triangle inequality to accelerate K-means," in *Proc. 20th Int. Conf. Mach. Learn.*, pp. 147–153 (2003).
34. J. Z. C. Lai and Y. C. Liaw, "Improvement of the K-means clustering filtering algorithm," *Pattern Recognit.* **41**(12), 3677–3681 (2008).
35. R. R. Curtin, "A dual-tree algorithm for fast k-means clustering with large k," in *Proc. SIAM Int. Conf. Data Mining*, pp. 300–308 (2017).
36. S. H. Chen and J. S. Pan, "Fast search algorithm for VQ-based recognition of isolated words," in *IEE Proc. I (Commun., Speech and Vis.)*, Vol. 136, pp. 391–396 (1989).
37. T. F. Gonzalez, "Clustering to minimize the maximum intercluster distance," *Theor. Comput. Sci.* **38**(2–3), 293–306 (1985).
38. M. E. Dyer and A. M. Frieze, "A simple heuristic for the P-centre problem," *Oper. Res. Lett.* **3**(6), 285–288 (1985).
39. G. Houle and E. Dubois, "Quantization of color images for display on graphics terminals," in *Proc. IEEE Glob. Telecommun. Conf.*, pp. 1138–1142 (1986).
40. N. Goldberg, "Colour image quantization for high resolution graphics display," *Image Vis. Comput.* **9**(5), 303–312 (1991).

41. Z. Xiang, "Color image quantization by minimizing the maximum intercluster distance," *ACM Trans. Graph.* **16**(3), 260–276 (1997).
42. S. Thompson, M. E. Celebi, and K. H. Buck, "Fast color quantization using Macqueen's K-means algorithm," *J. Real-Time Image Process.* **17**(5), 1609–1624 (2020).
43. "CQ100: a high-quality image dataset for color quantization research," https://data.mendeley.com/datasets/vw5ys9hfxw/3 (2023).
44. Z. Wang et al., "Image quality assessment: from error visibility to structural similarity," *IEEE Trans. Image Process.* **13**(4), 600–612 (2004).
45. Z. Wang, E. P. Simoncelli, and A. C. Bovik, "Multiscale structural similarity for image quality assessment," in *Proc. 37th Asilomar Conf. Signals, Syst. & Comput.*, Vol. 2, pp. 1398–1402 (2003).
46. Z. Wang and A. C. Bovik, "Mean squared error: love it or leave it? A new look at signal fidelity measures," *IEEE Signal Process. Mag.* **26**(1), 98–117 (2009).
47. D. Brunet, E. R. Vrscay, and Z. Wang, "On the mathematical properties of the structural similarity index," *IEEE Trans. Image Process.* **21**(4), 1488–1499 (2012).
48. G. Ramella, "Evaluation of quality measures for color quantization," *Multimedia Tools Appl.* **80**, 32975–33009 (2021).
49. M. L. Pérez-Delgado and M. E. Celebi, "A comparative study of color quantization methods using various image quality assessment indices," *Multimedia Syst.* **30**, 40 (2024).
50. S. García and F. Herrera, "An extension on "statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons," *J. Mach. Learn. Res.* **9**, 2677–2694 (2008).
51. M. Friedman, "The use of ranks to avoid the assumption of normality implicit in the analysis of variance," *J. Am. Stat. Assoc.* **32**(200), 675–701 (1937).
52. R. L. Iman and J. M. Davenport, "Approximations of the critical region of the friedman statistic," *Commun. Stat. - Theory Methods* **9**(6), 571–595 (1980).
53. J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *J. Mach. Learn. Res.* **7**, 1–30 (2006).
54. J. Luengo, S. García, and F. Herrera, "A study on the use of statistical tests for experimentation with neural networks: analysis of parametric test conditions and non-parametric tests," *Expert Syst. Appl.* **36**(4), 7798–7808 (2009).
55. S. García et al., "A study of statistical techniques and performance measures for genetics-based machine learning: accuracy and interpretability," *Soft Comput.* **13**, 959–977 (2009).
56. S. García et al., "A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the CEC'2005 special session on real parameter optimization," *J. Heuristics* **15**(6), 617–644 (2009).
57. J. Carrasco et al., "Recent trends in the use of statistical tests for comparing swarm and evolutionary computing algorithms: practical guidelines and a critical review," *Swarm Evol. Comput.* **54**, 100665 (2020).
58. W. W. Daniel, *Applied Nonparametric Statistics*, 2nd ed., PWS-KENT Publishing Company, (1990).
59. B. Bergmann and G. Hommel, "Improvements of general multiple test procedures for redundant systems of hypotheses," in *Multiple Hypotheses Testing*, P. Bauer and G. Hommel andE. Sonnemann, Eds., pp. 100–115, Springer (1988).
60. J. Derrac et al., "A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms," *Swarm Evol. Comput.* **1**(1), 3–18 (2011).
61. R. M. O. Cruz, R. Sabourin, and G. D. C. Cavalcanti, "Dynamic classifier selection: recent advances and perspectives," *Inf. Fusion* **41**, 195–216 (2018).
62. U. Johansson et al., "Rule extraction with guarantees from regression models," *Pattern Recognit.* **126**, 108554 (2022).
63. G. Hommel and G. Bernhard, "Bonferroni procedures for logically related hypotheses," *J. Stat. Plann. Inference* **82**(1–2), 119–128 (1999).
64. P. B. Nemenyi, "Distribution-free multiple comparisons," PhD thesis, Princeton University (1963).
65. S. Holm, "A simple sequentially rejective multiple test procedure," *Scand. J. Stat.* **6**(2), 65–70 (1979).
66. J. P. Shaffer, "Modified sequentially rejective multiple test procedures," *J. Am. Stat. Assoc.* **81**(395), 826–831 (1986).
67. G. Hommel and G. Bernhard, "A rapid algorithm and a computer program for multiple test procedures using logical structures of hypotheses," *Comput. Methods Programs Biomed.* **43**(3–4), 213–216 (1994).
68. Z. Drezner, "A note on the weber location problem," *Ann. Oper. Res.* **40**(1), 153–161 (1992).
69. M. Postma and J. Goedhart, "PlotsOfData—a web app for visualizing data together with their summaries," *PLoS Biol.* **17**(3), e3000202 (2019).

Harrison Bounds received his BS degree in computer science from the University of Central Arkansas, United States. He is currently pursuing his MS degree in robotics at Northwestern University, United States.

M. Emre Celebi received his PhD in computer science and engineering from the University of Texas at Arlington, United States. He is currently a professor and the chair of the Department of Computer Science and Engineering at the University of Central Arkansas, United States. He has published nearly 200 articles on image processing/analysis and data mining. According to Google Scholar, his research has received over 18,000 citations so far. He is a senior member of the IEEE and a fellow of the SPIE.

Jordan Maxwell: Biography is not available.