Artificial Intelligence Review (2023) 56:13953–14034
https://doi.org/10.1007/s10462-023-10406-6

Forty years of color quantization: a modern, algorithmic survey

M. Emre Celebi[1]

Published online: 27 April 2023

Abstract
Color quantization (CQ), the reduction of the number of distinct colors in a given image with minimal distortion, is a common image processing operation with various applications in computer graphics, image processing/analysis, and computer vision. The first CQ algorithm, median-cut, was proposed over 40 years ago. Since then, many clustering algorithms have been applied to the CQ problem. In this paper, we present a comprehensive overview of the CQ algorithms proposed in the literature. We first examine various aspects of CQ, including the number of distinguishable colors, CQ artifacts, types of CQ, applications of CQ, data structures, data reduction, color spaces and color difference equations, and color image fidelity assessment. We then provide an overview of image-independent CQ algorithms, followed by a detailed survey of image-dependent ones. After presenting a brief discussion of pixel mapping, we conclude our survey with an outline of the open problems in CQ.

Keywords Color quantization · Color reduction · Vector quantization · Data clustering · *k*-means

1 Introduction

24-bit **true-color** images have become ubiquitous over the past two decades (Sharma et al. 1998; Ramanath et al. 2005). A typical true-color image may contain hundreds of thousands of distinct colors, which complicates its display, storage, transmission, processing, and analysis. **Color quantization**[1] (CQ) is a common image processing operation that takes as input a color image and outputs a reproduction of it with the same spatial dimensions but significantly fewer colors. In this paper, we refer to the colors in the true-color input image as **input color**s and those in the **reduced-color**[2] output image as **output color**s or

[1] Also known as **color reduction** (Bragg 1992; Kruger 1992).
[2] Also known as **color-mapped**, **color-quantized**, or **palletized**.

M. Emre Celebi
ecelebi@uca.edu

[1] Department of Computer Science and Engineering, University of Central Arkansas, 201 Donaghey Ave., Conway, AR 72035, USA

representative colors (or **representatives**, for short). Figure 1 shows the Peppers image (USC-SIPI Image Database[3], 512 × 512 pixels) and its quantized versions with 4, 16, 64, and 256 colors. It can be seen that the reproduction is quite good with *only* 64 colors and indistinguishable from its original with 256 colors.

CQ is composed of two phases (Orchard and Bouman 1991): **color palette[4] design** (the selection of a small set of colors that represents the input colors) and **pixel mapping** (the assignment of each pixel in the input image to one of the representatives). The primary objective of CQ is to reduce the number of distinct colors in a given image to a much smaller number with minimal distortion. Since natural images often contain a large number of colors, faithful reproduction of such images with a small color palette is a challenging problem. In fact, as we will see in Sect. 5, CQ can be characterized as a large-scale combinatorial optimization problem.

The term **color quantization** was coined in a little-known paper by Jain and Pratt (1972). However, most researchers consider Heckbert (1980, 1982) the inventor of the first true CQ algorithm. Heckbert described his celebrated median-cut CQ algorithm first in his Bachelor's thesis (Heckbert 1980) and then, with slight modifications, in a journal paper (Heckbert 1982). In addition to being the first of its kind, Heckbert's seminal work introduced much of the terminology used in the CQ literature to this day, described the first divisive CQ algorithm, proposed bit-cutting as a preprocessing step and *k*-means as a postprocessing step, developed the first accelerated pixel mapping algorithm, and suggested the use of dithering to minimize false contours in the output image. It is thus safe to say that Heckbert's work established CQ as a subfield of image processing.

CQ can be considered an instance of **vector quantization** (VQ) (Wu 1992a). In a truecolor **red-green-blue** (RGB) image, each pixel is represented with 24 bits total (8 bits per color component). Assuming a maximum palette size of $K = 256$ and disregarding the space requirements of the palette itself, CQ allows us to represent each pixel in the output image with *only* 8 bits, leading to a modest compression ratio of 3:1. However, the purpose of CQ is *not* lossily compressing the input image, but representing it with a significantly smaller palette. CQ should also *not* be confused with **color (image) segmentation**, which refers to the partitioning of a given image into disjoint regions that are homogeneous in terms of color. Segmentation is a higher level image processing operation, which may or may *not* have CQ as a preprocessing step (see Sect. 1.5). In addition, segmentation almost invariably entails the use of spatial information, whereas CQ generally does *not* (see Sect. 1.3).

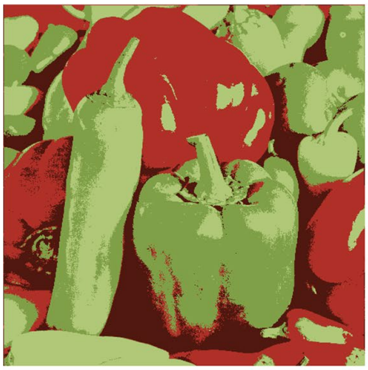
1.1 How many colors can humans distinguish?

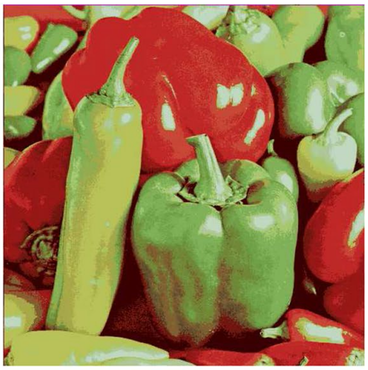
The question of how many colors can be distinguished by the human eye has a long history dating back to the late 1700s (Kuehni 2016). Estimates on the number of **perceptually distinguishable colors** range from 10,000 to 400,000,000 (Masaoka et al. 2013; Kuehni 2016). Factors contributing to the variation in these estimates include the color appearance model, color space, illuminant, just-noticeable difference, and the counting method used (Masaoka et al. 2013). Nevertheless, more recent computational studies (Pointer and Attridge 1998; Martínez-Verdú et al. 2007; Linhares et al. 2008; Morovic et al. 2012;

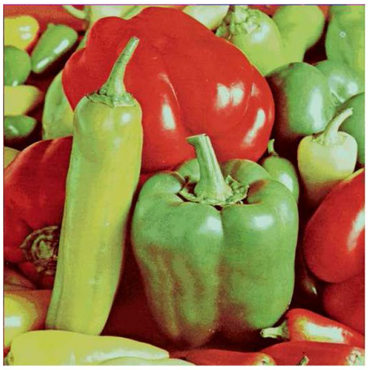
[3] Available at https://sipi.usc.edu/database/database.php?volume=misc.

[4] Also known as a **color map** or **color (lookup) table**.

(a) Peppers (183, 525 colors)


(b) 4 colors


(c) 16 colors


(d) 64 colors

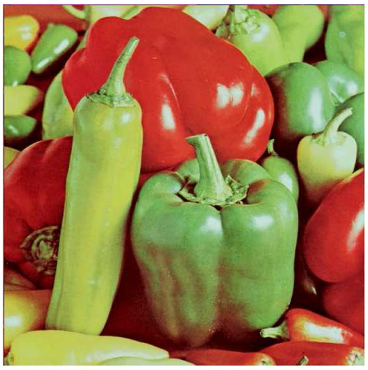

(e) 256 colors

Fig. 1 Peppers and its various quantized versions

Flinkman et al. 2012; Masaoka et al. 2013) reported estimates in the relatively narrow range of 1,700,000–2,500,000.

A typical natural image usually contains far fewer than 2,000,000 distinct colors. In addition, pairwise correlations between the red, green, and blue components of natural images are often very high (Pratt 1970), meaning that colors in such images are nonuniformly distributed within the RGB space. Hence, natural images can usually be represented with a relatively small palette. Figure 2 shows the Baboon (USC-SIPI Image Database, 512×512 pixels) and Peppers images along with three-dimensional (3D) visualizations[5] of their colors in the RGB space. Observe that, in both images, the colors are concentrated primarily around the main diagonal of the RGB cube, indicating the presence of significant spectral correlations.

1.2 Color quantization artifacts

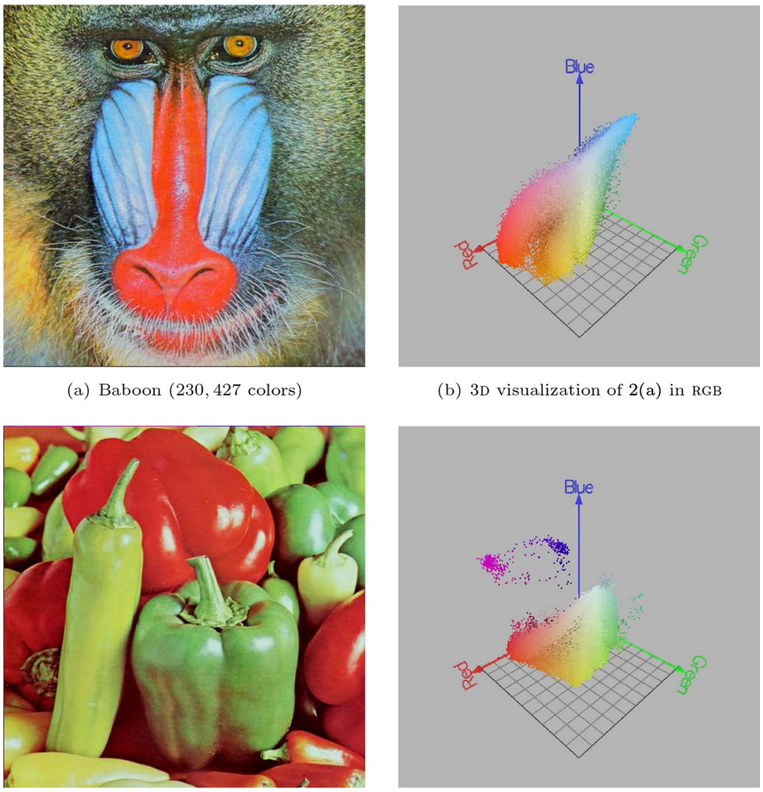
Due to its lossy (or irreversible) nature, CQ inevitably leads to loss of color and fine details. **Color shifts** occur when input colors *not* in the palette are represented with their nearest representatives in the output image. Color shifts can cause rare input colors to disappear, especially if the palette is small. Color loss may also cause **false contours**, which can be highly objectionable to the viewer. False contours tend to appear in large, almost uniformly colored (or slowly-varying) regions in natural images depicting sky, water, skin, etc. Representing such regions with a small palette causes them to split into bands of uniform colors, and visually disturbing contours appear between those bands. As mentioned earlier, in addition to color loss, CQ causes loss of fine details such as texture, leading to the emergence of **flat regions**. However, flat regions may also be generated intentionally for artistic effect, which is referred to as **posterization** (Chao et al. 2021).

Figure 3 shows the Parrots image (Kodak Lossless True Color Image Suite[6], 768×512 pixels) quantized to 16 and 256 colors using the median-cut algorithm. For each quantization, we display the true-color input image, reduced-color output image, and a grayscale **error image** that allows us to visualize the differences between the input and output. The error image is obtained by amplifying the pixelwise normalized Euclidean (ℓ_2) differences between the input and output by a factor of four and then negating them for better visualization. Hence, the cleaner/lighter the error image, the better the reproduction of the input image. In the 16-color case, we observe color loss (the teal color in both parrots), false contours (in the background), and texture loss (feathers in both parrots). By contrast, the 256-color output is a significantly better reproduction of the input, exhibiting *only* subtle contouring (in the background).

It is difficult to recover the fine image details lost due to CQ (Daly and Feng 2004). To this date, *only* a handful of studies (Schmitz and Stevenson 1995; Chan and Fung 2005; Keysers et al. 2006; Fung and Chan 2004, 2006a, b; Kim et al. 2007; Wang et al. 2019) explored the restoration of reduced-color images, an image processing operation referred to as **color dequantization**. False contours resulting from color loss can be addressed using various approaches. There are two main strategies: prevention of the appearance of false contours and removal of the existing false contours. The primary means of prevention is **dithering** (Hains et al. 2003; Baqai et al. 2005; Monga et al. 2006), an image processing operation that exploits the spatial integration property of the **human visual system** (HVS) to create the illusion of more colors. Unfortunately, while dithering usually mitigates false contours, it does *not* always eliminate them. In addition, dithering often introduces its own artifacts (visible noise, false textures, blurred edges, color impulses, etc.) (Akarun et al. 1997). False contour removal appears to be a more promising direction, and is an

[5] These visualizations were rendered using `Color Space 1.1.1` (by Philippe Colantoni).

[6] Available at http://r0k.us/graphics/kodak/.



(a) Baboon (230, 427 colors)

(b) 3D visualization of 2(a) in RGB

(c) Peppers (183, 525 colors)

(d) 3D visualization of 2(c) in RGB

Fig. 2 Baboon and Peppers and their 3D color visualizations

active area of research. However, nearly all existing approaches are designed for grayscale images, compressed videos, or high dynamic range images. For an overview of recent approaches, refer to Bhagavathy et al. (2009), Luzardo et al. (2017), Huang et al. (2018), Song et al. (2020), Tu et al. (2020), and the references therein.

1.3 Classification of color quantization algorithms

There are several ways to classify CQ algorithms:

- **Image-independent** vs. **image-dependent** (Gentile et al. 1990): Image-independent algorithms design a universal (or fixed) palette without regard to any particular input image, whereas image-dependent ones design a custom (or adaptive) palette based on the distribution of the colors in a given input image.
- **Uniform** vs. **nonuniform** (Heckbert 1982): Uniform algorithms place the representatives uniformly throughout the color space, whereas nonuniform algorithms position the representatives nonuniformly based on the distribution of the input colors.

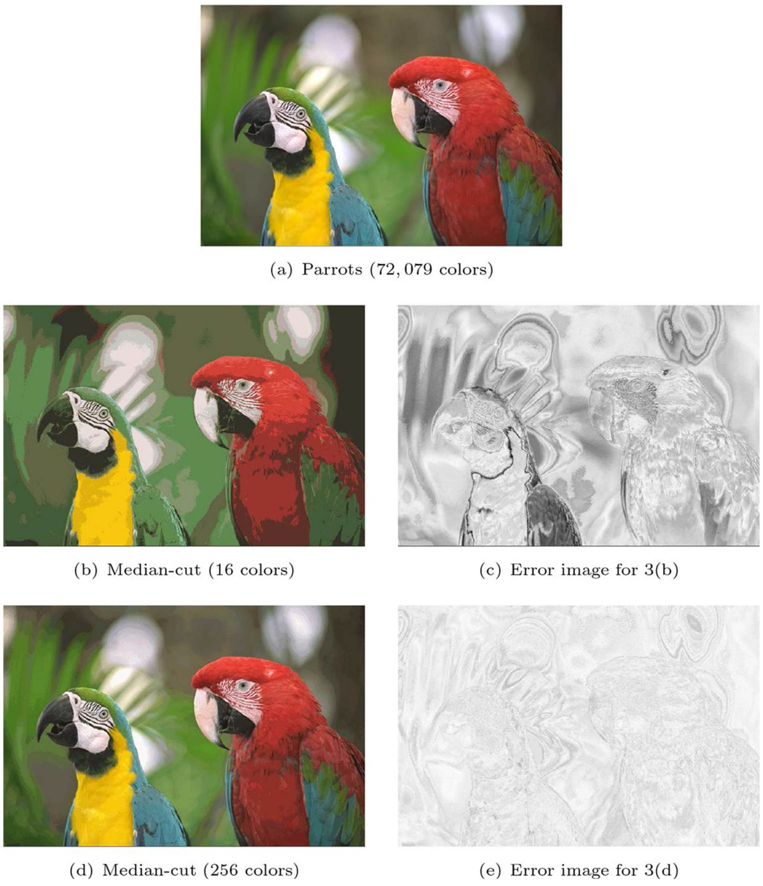

(a) Parrots (72, 079 colors)
(b) Median-cut (16 colors)
(c) Error image for 3(b)
(d) Median-cut (256 colors)
(e) Error image for 3(d)

Fig. 3 Parrots image quantized using the median-cut algorithm

- **Scalar** vs. **vector** (Wu 1992a): Scalar (or componentwise) algorithms treat the input image either as a scalar[7] image (e.g., by traversing the 3D color space using a space-filling curve[8] (Stevens et al. 1983) or projecting the 3D color data onto its first **principal component**, PC1) or a vector image with uncorrelated components (Balasubramanian et al. 1994b; Pei and Cheng 1995) (each of which can be treated as a scalar image). Vector algorithms, on the other hand, treat the input image as a true vector image by taking into account the spectral correlations.
- **Pre-clustering** vs. **post-clustering** (Dekker 1994): Pre-clustering algorithms first divide the input color space into K regions and then compute a representative for each region. Post-clustering algorithms, on the other hand, first select K representatives and then cluster the input colors around these representatives. Pre-clustering algorithms

[7] Buades et al. (2011) demonstrate that the distribution of colors in most natural images can be modeled accurately by a 2D manifold rather than a 1D curve. In other words, reducing the color space dimensionality (D) from three to one often leads to a severe loss of information.

[8] For another application of space-filling curves in CQ, see Subsection 1.7.

compute the palette *only* once, whereas post-clustering algorithms compute an initial palette and then iteratively improve it. Early CQ algorithms (1980–2000) tend to be of the pre-clustering kind, whereas more recent algorithms are generally of the post-clustering kind.

- **Hierarchical** vs. **partitional** (Jain et al. 1999): Hierarchical algorithms recursively find nested clusters in a top-down (or divisive) or bottom-up (or agglomerative) fashion. In contrast, partitional algorithms find all the clusters simultaneously as a partition of the data without imposing a hierarchical structure on the data. There are also hybrid algorithms (e.g., a hierarchical algorithm followed by a partitional one). Pre-clustering algorithms are generally hierarchical, whereas post-clustering algorithms are generally partitional.
- **Context-free** vs. **contextual** (Wu 1992a): Context-free (or non-spatial) algorithms consider *only* the chromatic characteristics of the input image, whereas contextual (or spatial) algorithms consider both chromatic and spatial information (Joy and Xiang 1996; Puzicha et al. 2000; Özdemir and Akarun 2002; Yu and Lo 2003; Papamarkos et al. 2002; Huang et al. 2016). While including contextual information in the CQ process may increase the separation between the color clusters (Papamarkos et al. 2002) and mitigate artifacts such as false contours (Xiang 2018), these advantages often come at the expense of a significantly higher computational cost.
- **Static** vs. **dynamic**: Static algorithms assume that the palette size (K) is a fixed integer to be specified by the user in advance, whereas dynamic algorithms (Atsalakis et al. 2002; Kim and Kehtarnavaz 2005; Atsalakis and Papamarkos 2006; Delon et al. 2007; Nikolaou and Papamarkos 2009; Ramella and di Baja 2013; Palomo and Domínguez 2014; Nieves et al. 2020) can determine the value of K automatically at run-time. Unfortunately, the automatic determination of the number of clusters in a given data set is an ill-defined problem (Baarsch and Celebi 2012). In addition, dynamic algorithms often require additional user-defined parameters, some of which are unintuitive or difficult to tune. Finally, dynamic algorithms tend to require a lot more computation than the static ones.

A vast majority of CQ algorithms proposed to date are image-dependent, nonuniform, vector, context-free, and static. Therefore, in this survey, we focus primarily on such algorithms.

1.4 Characteristics of an ideal color quantization algorithm

The following are the characteristics of an ideal CQ algorithm:

- **Effective**: Produces minimal distortion.
- **Efficient**: Requires a minimal amount of computational resources (CPU time, memory, etc.)
- **Simple**: Is easy to understand and implement[9].

[9] The popularity of a CQ algorithm appears to correlate well with the availability of its open-source implementations. However, practitioners should be aware of the fact that open-source implementations of popular CQ algorithms such as median-cut (Heckbert 1982) and octree (Gervautz and Purgathofer 1988) have varying degrees of quality and faithfulness to the original algorithms.

- **Convenient**: Requires a minimal number of user-defined parameters. If it has such parameters, they are intuitive and easy to determine. In addition, its performance is *not* overly sensitive to the values of these parameters.
- **Deterministic**: For a given input image and parameter values, produces the same output image in every single run.

Unfortunately, many of the above characteristics are in conflict with one another. For example, simple and efficient algorithms are *not* always effective and often randomized (or nondeterministic). In addition, some of these characteristics may be undesirable in specific applications. For instance, if the performance of an algorithm is inadequate in a given application, there may *not* be much to do if the algorithm is deterministic and has *no* user-defined parameters. By contrast, a randomized algorithm or an algorithm with user-defined parameters can be run multiple times (with different random seeds or parameter values, respectively) to obtain better results at the cost of more computation.

1.5 Applications of color quantization

In the past, CQ was necessary due to the limitations of the display hardware, many of which could *not* handle the number of colors in true-color images. Although 24-bit display hardware have become more common, CQ still maintains its practical value. Modern applications of CQ in computer graphics, image processing/analysis, and computer vision include non-photorealistic rendering (Chao et al. 2021), image matting (Chuang et al. 2001), image dehazing (Berman et al. 2016), image compression (Cheng and Bouman 2001), color-to-grayscale conversion (Kuhn et al. 2008), image watermarking/steganography (Tseng and Ding 2012), image segmentation (Mignotte 2008), content-based image retrieval (Mojsilović et al. 2002), color analysis (Celebi et al. 2012b), color-texture analysis (Serrano et al. 2022), saliency detection (Cheng et al. 2015), and skin detection (Phung et al. 2005).

In addition to the applications mentioned above, CQ is an important problem in its own right because of its close connection to the more general VQ and data clustering problems. The landmark median-cut CQ (Heckbert 1980) and Linde–Buzo–Gray VQ (Linde et al. 1980) algorithms were both published in 1980. Since then, the fields of CQ and VQ have followed somewhat parallel developmental paths with some researchers working in both. Many VQ algorithms have their roots in general-purpose data clustering algorithms, and the same is true for many CQ algorithms. Conversely, the CQ literature has also exerted some influence on the data clustering literature (Wan et al. 1988; Schreiber 1991; Inaba et al. 1994; Yager and Filev 1994; Su and Dy 2007; Yang et al. 2008; Celebi and Kingravi 2012, 2015). Consequently, advances in CQ have led and can lead to advances in VQ and data clustering.

1.6 Data structures

In terms of storage, a context-free CQ algorithm can be implemented in two *equivalent* ways (Wan et al. 1988): **spatial-storage** and **histogram-storage**. In the spatial-storage scheme, the input color data is represented as an $N_p \times 3$ array, where $N_p = HW$ denotes the number of pixels in the $H \times W$ input image. This scheme is easy to implement, but wasteful as it stores repeated colors multiple times. On the other hand, in the histogram-storage scheme, each of the N_c input colors is stored *only* once along with its frequency (or count).

This scheme can be more difficult to implement (see below) but often requires less memory and results in a faster CQ, depending on the values of N_p and N_c. Note that, in the remainder of this paper, the symbol N is used to denote the size of the input data set, N_p or N_c, whenever the context does *not* require differentiation.

A **color histogram** can be implemented in various ways:

- 3D **representations**: A color histogram can be implemented straightforwardly using a 3D array whose dimensions correspond to the three color components. Each bin of such a histogram then stores the frequency of a particular input color. This data structure is trivial to implement and requires $\mathcal{O}(1)$ time to access *any* bin. However, it requires a considerable amount of memory, which is generally wasted because colors in natural images often occupy a small part of the color space (see Fig. 2). For example, the histogram of a 24-bit RGB image contains 256^3 elements, which require 64MBs of memory (assuming a 32-bit data type). For this reason, 24-bit color histograms were considered impractical in the early days of CQ, prompting researchers to resort to bit-cutting (see Sect. 3.1). Reitan (1998) investigates adaptive data structures to capture the nonuniform distribution of the input colors in three dimensions, including k-d trees and octrees (see Sect. 4.2). These recursive space-partitioning trees achieve much better space utilization than a 3D histogram, but this comes at the price of simplicity and time efficiency.
- 2D **representations**: To reduce memory requirements, some CQ researchers proposed the use of a 2D array whose elements are pointers to a 1D data structure, e.g., a binary search tree (Balasubramanian and Allebach 1991a; Balasubramanian et al. 1994a; Reitan 1998) or a linked list (Balasubramanian et al. 1994b; Xiang and Joy 1994). For an RGB image, the 2D array can be indexed by the red and green components of the input colors, while the 1D data structure stores the remaining color component (blue). Such a design improves the efficiency of the insertion and search operations by increasing the space utilization of the array. Compared to a 3D histogram, which requires $\mathcal{O}(256^3)$ memory, the resulting hybrid data structure requires *only* $\mathcal{O}(256^2 + N_c)$ memory.
- 1D **representations**: A color histogram can also be represented using a 1D data structure (Balasubramanian and Allebach 1991b; Reitan 1998). For example, we can use a binary search tree that stores the red, green, and blue components of each (r, g, b) color in packed format, e.g., $256^2\, r + 256\, g + b$ or, equivalently, $(r \ll 16) \mid (g \ll 8) \mid b$, where $\ll$ and $\mid$ denote the bitwise left shift and bitwise or operators, respectively. Alternatively, we can use a hash table[10] that uses chaining for collision resolution and a universal hash function of the form (Celebi 2011; Celebi et al. 2015) $h_{\mathbf{a}}(r, g, b) = \left(a_r\, r + a_g\, g + a_b\, b\right) \bmod N_s$, where N_s is a prime number and the elements of sequence $\mathbf{a} = (a_r, a_g, a_b)$ are selected randomly from the set $\{0, 1, \ldots, N_s - 1\}$. The binary search tree and hash table require $\mathcal{O}(N_c)$ and $\mathcal{O}(N_s + N_c)$ memory, respectively.

1.7 Data reduction

We mentioned earlier that CQ can be considered a large-scale combinatorial optimization problem. CQ researchers have employed various techniques to reduce the scale of this problem. Early techniques include taking a uniform (pseudo-)random sample from the input image (e.g., sampling 1,024 pixels from a 512×512 input image (Dixit 1991)) and

[10] Heckbert (1980) was the first to use hashing in CQ.

reducing the spatial dimensions of the input image (e.g., $2:1$ sampling in the horizontal and vertical directions (Goldberg 1991), which reduces N_p by a factor of 4 by eliminating every other column and row).

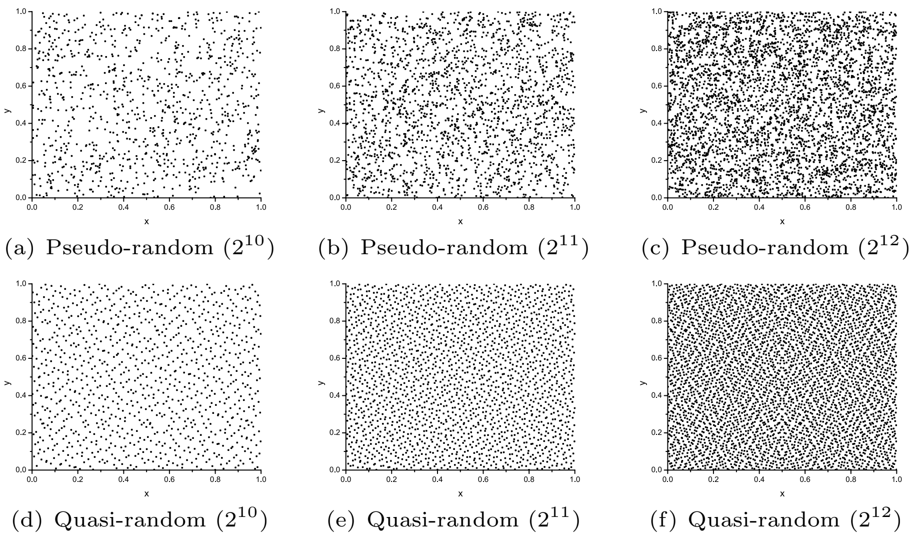
Sampling a 2D image using a pseudo-random sequence is a straightforward operation. However, such a sampling is not only non-deterministic, but also somewhat nonuniform in its coverage of the input pixels, which may bias the CQ algorithm (Thompson et al. 2020). To alleviate these problems, we can use a **quasi-random** sequence (Press et al. 2007, p. 404) instead, which is a deterministic sequence of D-dimensional points that fill $\mathbb{R}^D$ more uniformly than uncorrelated D-dimensional pseudo-random points. This is illustrated in Fig. 4. Here, the top row shows pseudo-random sequences with increasing length from left to right generated using the MT19937 variant of the Mersenne Twister algorithm (Matsumoto and Nishimura 1998), while the bottom row shows quasi-random sequences with corresponding lengths generated using a Sobol' sequence (Sobol' et al. 2011).

There are many quasi-random sequences, including those due to Korobov, Halton, Sobol', Faure, Niederreiter, and Niederreiter and Xing. Among these, Sobol' sequences are often preferred in practice, especially in low dimensions, due to their favorable uniformity properties and the availability of efficient generation algorithms. In the context of CQ, Celebi et al. (2014); Thompson et al. (2020) demonstrate that, in addition to being deterministic, quasi-random sampling gives comparable results to pseudo-random sampling.

Some CQ algorithms are **order-dependent**, meaning that they are sensitive to the order in which the input colors are processed. Such algorithms will thus be affected by the order in which the input image is scanned. A color image with N_p pixels can be scanned in $N_p!$ different ways. The standard scan order is termed a **raster scan** (or **row-major order scan**), where the image is scanned row by row from top to bottom, and from left to right in each row. There are many alternative image scan orders, including serpentine, zigzag, diagonal, and spiral. Among these, scan orders defined by **space-filling curve**s are popular due to their clustering (or neighborhood-preserving) properties. If two points are close on such a curve, they are also close in the image plane; conversely, if two points are close in the image plane, they are *likely* to be close on the curve (Bartholdi III and Platzman 1988).

There are many space-filling curves (Bader 2013), including the Peano curve, Hilbert curve, Morton curve, and Sierpiński curve. Among these, the Hilbert curve is generally preferred in image processing due to its superior clustering properties (Gotsman and Lindenbaum 1996; Moon et al. 2001) and the availability of efficient generation algorithms. Papamarkos et al. (2002); Atsalakis et al. (2002); Atsalakis and Papamarkos (2006) use the Hilbert curve to scan the input image prior to CQ. It should be mentioned that while the clustering properties of space-filling curves can be beneficial in applications such as image compression (Alexandrov et al. 1984; Lempel and Ziv 1986), it is unclear if the same is true in CQ. An alternative technique is to sample the input image following the raster order but with a non-unit step size[11] that decreases in each pass, e.g., $1009, 757, 499, 421, 307, 239, 197, \ldots$ (Verevka and Buchanan 1995). Observe that these step sizes are taken to be prime numbers to minimize the overlap among the subsets of pixels sampled in each pass and that the image is sampled progressively more finely. Finally, some CQ researchers (Fletcher 1991; Pei and Lo 1998; Chang et al. 2005; Wang et al. 2007)

[11] Scanning an image of size N_p pixels in raster order with a step size of P yields a sample of size $\lceil N_p/P \rceil$ pixels.


(a) Pseudo-random (2^{10}) (b) Pseudo-random (2^{11}) (c) Pseudo-random (2^{12})

(d) Quasi-random (2^{10}) (e) Quasi-random (2^{11}) (f) Quasi-random (2^{12})

Fig. 4 Comparison of pseudo-random and quasi-random sampling (number of points shown in parentheses)

sample the input image according to the elaborate **butterfly order** used in fast Fourier transform (FFT) algorithms (Oppenheim and Weinstein 1972).

The sampling techniques discussed above are oblivious to the objective of CQ, and thus none of them come with any theoretical performance guarantees. We can develop a theoretically sound image sampling technique based on the idea of a **coreset**, which is a problem-dependent data summarization technique (Feldman 2020). More specifically, given a data set and a machine learning problem, a coreset is a small subset of the data set that captures its essential features with respect to the problem. In the context of CQ, Valenzuela et al. (2018) explore an image sampling technique based on a recent coreset construction algorithm (Lucic et al. 2016) designed for the *k*-means problem (see Subsection 5.3).

1.8 Notation

Tables 1 and 2 respectively list the acronyms and pseudocode operators used throughout the paper.

1.9 Outline of the survey

In this paper, we provide a comprehensive survey of the CQ research conducted since 1980. Our survey differs from the earlier ones (Domański and Bartkowiak 1998; Brun and Trémeau 2003; Xiang 2018) in three significant ways. First, we cover a time period that is nearly twice as long (1980–2022 as opposed to 1982–1997 (Domański and Bartkowiak 1998), 1982–2000 (Brun and Trémeau 2003), and 1982–2005 (Xiang 2018)). Second, our coverage of the field is more algorithmic than mathematical in nature. For example, in addition to providing a textual description of each algorithm, we employ pseudocodes to illustrate some of the key ones. Third, while the earlier surveys focus primarily on hierarchical algorithms, we focus more on partitional ones, which can

Table 1 Table of acronyms

Acronym	Meaning	Acronym	Meaning
1D/2D/3D	1/2/3-dimensional	OKM	Online KM
ADU	Adaptive distributing units	PC	Principal component
BIRCH	Bal. Iter. Red. Clust. Hier.	PIM	Partition index maxim.
BKM	Batch KM	PM	Pixel mapping
CD	Color difference	PNN	Pairwise nearest neighbor
CF	Clustering feature	PSNR	Peak SNR
CIE	Int. Comm. on illumination	RGB	Red-Green-Blue
CIEDE	CIE difference equation	S-CIELAB	Spatial CIELAB
CL	Competitive learning	SBKM	Stability-based KM
CQ	Color quantization	SNR	Signal-to-noise ratio
FCM	Fuzzy *C*-means	SOM	Self-organizing map
FSKM	Finite-state KM	SQ	Scalar quantization
HVS	Human visual system	SRGB	Standard RGB
IOKM	Incremental OKM	SSE	Sum of squared errors
IUSQ	Independent USQ	TIE	Triangle inequality Elim.
KM	*K*-means	UCS	Uniform color space
MAE	Mean absolute error	USQ	Uniform SQ
MOS	Mean-ordered search	VQ	Vector quantization
MSE	Mean squared error		

Table 2 Table of pseudocode operators

Operator	Name	Effect
$x \leftarrow v$	Assignment	x equals v
$(x, y) \leftarrow (v_1, v_2)$	Multiple assignment	x and y equal v_1 and v_2, respectively
$x \leftarrow y \leftarrow v$	Simultaneous assignment	x and y both equal v
$p \vee q$	Logical disjunction (or)	True if p or q is true, false otherwise
$p \wedge q$	Logical conjunction (and)	True if p and q are true, false otherwise
$p = q$	Equality	True if p and q are equal, false otherwise

attain better results. We should mention that, to keep the paper reasonably concise, we do *not* provide a detailed comparative study of the CQ algorithms covered, but instead give figures illustrating the behavior of a few prominent algorithms. For quantitative comparisons of CQ algorithms, refer to Pérez-Delgado and Gallego (2019); Pérez-Delgado (2020); Pérez-Delgado and Gallego (2020); Pérez-Delgado (2021) and Abernathy and Celebi (2022).

The remainder of this survey is organized as follows. Section 2 gives an overview of common color spaces and color difference equations, and discusses color image fidelity assessment. Sections 3 through 6 present palette design algorithms. More specifically, Sect. 3 describes image-independent algorithms, whereas the following three sections present image-dependent algorithms, namely hierarchical algorithms (Sect. 4), partitional algorithms (Sect. 5), and metaheuristic algorithms (Sect. 6). Section 7 examines the pixel mapping phase, and presents an overview of accelerated pixel mapping algorithms. Finally, Sect. 8 concludes the survey, and suggests future research directions.

2 Color spaces, color difference equations, and color image fidelity assessment

In this section, we provide the necessary background information on color science and color image processing. We first give a brief overview of common color spaces and color difference equations. We then discuss the important but often neglected issues of objective and subjective color image fidelity assessment.

2.1 Color spaces

An important consideration in any color image processing operation is the **color space** in which the operation is conducted. The default RGB space[12] is a device-dependent color space in which the color components often have high pairwise correlations. In applications where true multivariate processing is desired, the input RGB image can be used directly or transformed into an **alternative color space** for various reasons, including (i) increasing cluster separation, (ii) reducing the number of color components (D), (iii) decoupling luminance and chromaticity information, (iv) obtaining (approximate) perceptual uniformity, and (v) achieving invariance to various imaging conditions, such as viewing direction, illumination intensity, and highlights. Among these, (perceptual) uniformity is a common reason to perform color space transformations in CQ applications. A (perceptually) **uniform color space** (UCS) is one in which equal *geometric* distances correspond to equal *perceived* **color difference**s (CDs). It is well-known that the RGB space, especially in its linear (i.e., non-gamma-corrected) form, is perceptually nonuniform. Despite nearly a century of colorimetry research, a perfectly uniform color space has *not* been discovered yet[13]. However, various approximately uniform color spaces have been designed (Kuehni 2003), including the 1960 CIE-UCS, 1964 CIE-UCS, OSA, CIELAB/CIELUV[14], CIECAM97, CAM02-UCS, and CAM16-UCS spaces. Despite the fact that they are *not* perfectly uniform, these spaces are often referred to as UCSs.

UCSs should *not* be confused with **intuitive color space**s such as Hue-Saturation-Value (HSV), Hue-Saturation-Lightness (HSL), and Hue-Saturation-Intensity (HSI). These are cylindrical coordinate spaces (Levkowitz and Herman 1993; Hanbury 2008) used for color specification (e.g., designing intuitive color picker tools). The components of these color spaces do *not* accurately model the perceptual attributes of hue, saturation, and lightness,

[12] In this paper, we assume that all color image data is encoded in the **standard** RGB (SRGB) space (Anderson et al. 1996), which was standardized by the International Electrotechnical Commission (IEC) in 1999. This means that the red, green, and blue components of the images are nonlinearly coded (i.e., gamma corrected). It is customary to denote such nonlinear components with primes (Poynton and Funt 2014) (i.e., r', g', and b' as opposed to r, g, and b, respectively). However, we omit the primes throughout the paper to avoid clutter.

[13] There is evidence that a UCS is *not* Euclidean, at least *not* in three dimensions (Urban et al. 2007).

[14] In 1976, the International Commission on Illumination (CIE) recommended two approximately uniform color spaces, namely CIELAB and CIELUV. Nearly half a century later, these spaces are still the CIE recommendations, although CIELUV has fallen out of favor (Fairchild and Johnson 2004). In fact, since its standardization, colorimetric research has mostly revolved around the CIELAB space and its CD equation (Luo 2002).

and their CD equations do *not* uniformly represent perceived CDs (Robertson 1988). In addition, hue is a circular quantity (Crevier 1993; Hanbury 2003), which should be described using circular statistics (e.g., hue values should be averaged using the circular mean formula rather than the arithmetic mean one.)

2.2 Color difference equations

Once we have a color space at hand, we need a CD equation to quantify the geometric distance between two colors in this space. Assuming that we are working in the RGB space, we can use the (unweighted) ℓ_2 distance. Let $\mathbf{p} = (r_1, g_1, b_1)$ and $\mathbf{q} = (r_2, g_2, b_2)$ be two colors in the RGB space. The difference between these colors is given by

$$d(\mathbf{p}, \mathbf{q}) = \sqrt{(r_1 - r_2)^2 + (g_1 - g_2)^2 + (b_1 - b_2)^2}. \tag{1}$$

Similarly, unweighted or weighted ℓ_2 distances are commonly used to quantify CDs in UCSs. The best-known UCS and CD equation pair is (CIELAB, ℓ_2). Let $\mathbf{p} = (L_1, a_1, b_1)$ and $\mathbf{q} = (L_2, a_2, b_2)$ be two colors in the CIELAB space. The difference between these colors is given by the CIE76 CD equation

$$d(\mathbf{p}, \mathbf{q}) = \sqrt{(L_1 - L_2)^2 + (a_1 - a_2)^2 + (b_1 - b_2)^2}. \tag{2}$$

To address the nonuniformities in the CIELAB space, the CIE standardized progressively more advanced equations, namely the CIE94 and CIEDE2000 equations. Despite its superior accuracy, due to its mathematical discontinuities (Sharma et al. 2005), CIEDE2000 is significantly more complex and difficult to implement than CIE94 and CIE76. Therefore, among the three CD equations mentioned above, CIE94 appears to offer the best balance between accuracy and complexity (Melgosa and Huertas 2004). Nevertheless, CIE76 is still prevalent in the color image processing literature.

Given a $H \times W$ color image I and its $H \times W$ reproduction $\tilde{I}$, a simple way to measure the fidelity of $\tilde{I}$ is to apply a CIE CD equation (CIE76, CIE94, or CIEDE2000), or one of their many alternatives (Luo 2002), to the two images in a pixelwise manner, that is, by computing the CD between each pixel in I and the corresponding pixel in $\tilde{I}$. The result is a **difference image**, which is often reduced to a single number by averaging, that is,

$$e_d(I, \tilde{I}) = \frac{1}{N_p} \sum_{r=1}^{H} \sum_{c=1}^{W} d\big(I(r,c), \tilde{I}(r,c)\big), \tag{3}$$

where $N_p = HW$ and $d(\cdot, \cdot)$ is a distance function.

Examples of pixelwise **image fidelity metric**s in the form of Eq. (3) include the **mean absolute error** (MAE) and **mean squared error** (MSE) metrics corresponding to $d = \ell_1$ and $d = \ell_2^2$, respectively. To facilitate comparisons, Eq. (3) can be normalized as follows

$$e_{[d]}(I, \tilde{I}) = \frac{N_p \, e_d(I, \tilde{I})}{\sum_{r=1}^{H} \sum_{c=1}^{W} d(I(r,c), \mathbf{0})}, \tag{4}$$

where $\mathbf{0} = (0, 0, 0)$ is the origin of the color space. By substituting $d = \ell_1$ and $d = \ell_2^2$ in Eq. (4), we obtain the **normalized mean absolute error** (NMAE) and **normalized mean squared error** (NMSE) metrics, respectively.

Another popular image fidelity metric is the **peak signal-to-noise ratio** (PSNR) given by

$$e_{\text{PSNR}}(I, \tilde{I}) = 10 \log_{10} \frac{M^2}{e_{\ell_2^2}(I, \tilde{I})/3}, \tag{5}$$

where M denotes the maximum possible color component value. For an image with B bits per component, we have $M = 2^B - 1$ (e.g., $M = 255$ for $B = 8$).

MSE/PSNR (Wang and Bovik 2009), and to some extent MAE, are widely used image fidelity metrics owing to their conceptual and computational simplicity. However, since the early days of image processing (Huang et al. 1971), researchers have criticized these metrics for disregarding the characteristics of the HVS. The drawbacks of these simplistic metrics have prompted the development of perceptually-based metrics in the 1970s (Sakrison and Algazi 1971; Stockham Jr. 1972; Budrikis 1972; Mannos and Sakrison 1974). However, a vast majority of the metrics proposed before the 1990s were designed for grayscale images.

2.3 Objective color image fidelity assessment

The aforementioned CIE CD equations were developed based on psychophysical experiments involving uniform color patches (placed against uniform gray backgrounds) viewed in controlled conditions. Therefore, such equations are appropriate for assessing *color differences* rather than *color image differences*. To assess the latter, various color image difference metrics include a filtering step to simulate the spatial blurring performed by the HVS. Examples of such spatiochromatic metrics include the spatial CIELAB (S-CIELAB) metric (Zhang and Wandell 1997, 1998; Johnson and Fairchild 2003) and its extensions such as the iCAM metric (Fairchild and Johnson 2004; Johnson et al. 2010) and the more recent FLIP metric (Andersson et al. 2020). These metrics include similar image processing steps to approximate the behavior of the HVS (Johnson et al. 2010). The input images are first transformed into a device-independent color space such as CIEXYZ or approximate human cone responses (LMS), and then into an opponent color space. In the opponent space, the images are filtered with approximations of human contrast sensitivity functions to remove high-frequency information that is imperceptible to the HVS. The filtered images are then transformed into a UCS such as CIELAB wherein pixelwise CDS are computed (e.g., using one of the CIE CD equations described in Subsection 2.2).

Color image fidelity assessment is a complex, interdisciplinary topic that has been the subject of active research since the 1990s (Ortiz-Jaramillo et al. 2019). Dozens of color image fidelity metrics, some with spatial filtering and some without, have been proposed in the literature. Comparative studies (Sheikh et al. 2006; Hardeberg et al. 2008; Pedersen and Hardeberg 2012; Pedersen 2015; Ponomarenko et al. 2015; Ortiz-Jaramillo et al. 2019) reveal that different metrics capture different aspects of image fidelity, and thus there is *no* universally applicable metric. Nevertheless, certain metrics may be more appropriate than others in a given application.

Objective assessment remains to be one of the least explored aspects of CQ. A vast majority of CQ studies employ pixelwise (non-spatial[15]) image fidelity metrics such as

[15] Exceptions include a few CQ studies (Sudha et al. 2003; Lo et al. 2003; Yu and Lo 2003; Nolle and Schaefer 2007; Schaefer 2014; Schaefer and Nolle 2015) that employ the S-CIELAB metric.

the MSE/PSNR, MAE, and their variants, computed mostly in the RGB space. CQ researchers have generally avoided UCSS such as CIELAB for several reasons: (i) the additional computational cost of the transformation between RGB and the alternative color space (Orchard and Bouman 1991; Braquelaire and Brun 1997)[16]; (ii) the higher computational cost of CQ in the alternative color space[17]; (iii) the CIE CD equations used with CIELAB are developed for measuring color differences rather than color image differences (Balasubramanian et al. 1994a, b); iv) performing CQ in the nonlinear (gamma-corrected) RGB space is *not* entirely inappropriate as this space is more uniform than the linear (non-gamma-corrected) RGB space (Zhang and Wandell 1998; Poynton and Funt 2014; Avanaki et al. 2014); and (v) anecdotal evidence that performing CQ in an alternative color space does *not* necessarily give better results (Braquelaire and Brun 1997).

Despite the numerous studies proposing new CQ algorithms, *only* a few studies (Ortiz-Jaramillo et al. 2019; Ramella 2021) focus on the assessment of CQ. Ortiz-Jaramillo et al. (2019) and Ramella (2021) both use the TID2013 database (Ponomarenko et al. 2015), which contains 25 reference color images and their distorted versions (each reference image is subjected to 24 types of distortions, including CQ, applied at 5 levels.) *Only* two fidelity metrics are common to both studies, MSE/PSNR[18] and structural similarity (SSIM) (Wang et al. 2004), which is perhaps the most prevalent metric after MSE/PSNR. Ortiz-Jaramillo *et al.* investigate 25 metrics, and conclude that the two MSE metrics based on the CIE76 and CIEDE2000 equations and the S-CIELAB metric are among the best. On the other hand, Ramella investigates nine metrics and concludes that weighted SNR (WSNR)[19], multiscale SSIM (MSSSIM)[20], and visual SNR (VSNR)[21] perform the best[22]. MSE/PSNR, the simplest metrics tested, outperform SSIM in both studies. Perhaps this is *not* surprising given that SSIM can be derived as a function of PSNR, and thus the values of the two metrics are often highly correlated (Hore and Ziou 2010; Dosselmann and Yang 2011).

2.4 Subjective color image fidelity assessment

So far, we have focused on the objective assessment of CQ. The complementary assessment methodology is known as **subjective assessment**. Broadly speaking, there are two kinds of subjective assessment methodologies (Pérez-Ortiz et al. 2019): **rating** and **ranking**. In a rating experiment, human observers rate the quality of the presented images based on a categorical (e.g., 5: excellent, 4: good, 3: fair, 2: poor, and 1: bad) or continuous interval (e.g., [0, 100]) scale. On the other hand, in a ranking experiment, observers compare

[16] There are, however, accelerated color space transformations with negligible loss of accuracy (Celebi et al. 2010).

[17] For example, CIELAB components are often represented in floating-point format to avoid loss of precision. Such a data representation, however, leads to slower computations than an integer representation.

[18] Ortiz-Jaramillo et al. (2019) compute two separate MSE metrics in the CIELAB space, one using the CIE76 equation and the other using the CIEDE2000 one. By contrast, Ramella (2021) computes PSNR, and by extension MSE, in the RGB space.

[19] WSNR is computed in the Fourier domain after filtering the two images with approximations of the human contrast sensitivity functions (Mannos and Sakrison 1974; Mitsa and Varkur 1993).

[20] MSSSIM is a variant of SSIM computed over a range of scales (Wang et al. 2003).

[21] VSNR is an HVS-based fidelity metric computed in the wavelet domain (Chandler and Hemami 2007).

[22] All three metrics are originally defined for grayscale images.

the presented images. The simplest ranking method is **pairwise comparison**, where two images are compared, typically on a binary scale (e.g., better or worse).

Rating-based assessment is more appropriate when the images are easily distinguishable from one another. Once the subjective experiment is completed, we can easily compute mean opinion scores (Streijl et al. 2016) from the observer responses and then order the images based on their perceived quality. However, rating experiments require careful training of the observers to establish a common understanding of the rating scale. On the other hand, ranking-based assessment is more appropriate when the images are relatively difficult to distinguish. As mentioned earlier, a common type of ranking is pairwise comparisons, which are simpler to perform because it is easier for humans to compare pairs of images than rate them individually. Thus, such ranking experiments require little training compared to rating experiments. The pairwise comparison method has two major drawbacks. First, for N_i images, a complete and balanced design requires $\binom{N_i}{2}$ trials per observer[23]. Second, it is nontrivial to obtain a quality ordering of the images from the comparison matrix. The former drawback can be alleviated using an efficient comparison-based sorting algorithm (Silverstein and Farrell 2001; Maystre and Grossglauser 2017) or an active sampling algorithm (Jamieson and Nowak 2011; Xu et al. 2011; Ye and Doermann 2014; Fan et al. 2017; Li et al. 2018; Heckel et al. 2019; Mikhailiuk et al. 2021), which reduces the number of pairwise comparisons from $\mathcal{O}(N_i^2)$ to often $\mathcal{O}(N_i \log N_i)$ with negligible loss of accuracy in the final ordering of the images. As for the latter drawback, there are a variety methods (Cattelan 2012; Liu et al. 2019) and software[24] (Wickelmaier and Schmid 2004; Turner and Firth 2012; Hatzinger and Dittrich 2012) to analyze pairwise comparison data.

Subjective assessment has found limited application in CQ for several reasons. First, it is nontrivial to design subjective experiments because of the number and diversity of factors involved (experiment method and its parameters, experimental environment, observer selection and training, etc.) Second, it is costly to conduct subjective experiments in terms of time and effort. Third, the statistical analysis of subjective experimental data may be challenging (Tsukida and Gupta 2011; Pérez-Ortiz and Mantiuk 2017). Fourth, subjective experiments may be influenced by observer biases as well as inter- and intra-observer variability. CQ studies employing subjective assessment to date have involved *only* ranking experiments. Specifically, Hadizadeh et al. (2011) and Huang et al. (2016) employ pairwise comparisons, whereas Montagne et al. (2006) employ setwise comparisons, where observers order multiple images from the best to the worst.

It is important to emphasize that assessment, both in its objective and subjective forms, is useful not only for validating CQ algorithms, but also for optimizing them (i.e., determining optimal values for their user-defined parameters). This optimization aspect of assessment can be crucial for CQ algorithms with multiple user-defined parameters.

[23] By contrast, a rating experiment requires *only* N_i trials per observer (Mantiuk et al. 2012).

[24] There are also software for conducting pairwise comparison experiments (Vuong et al. 2018).

3 Image-independent algorithms

In Sect. 1, we mentioned that CQ algorithms can be broadly classified into two categories: image-independent algorithms that design a **universal palette** without regard to any particular input image and image-dependent ones that design a **custom palette** based on the distribution of the colors in a given input image. In this section, we discuss the image-independent algorithms.

3.1 Uniform quantization

In general, image-independent algorithms place the representatives uniformly throughout the color space because of their need to achieve reasonable image quality for a wide variety of input images (Gentile et al. 1990). A small universal palette (e.g., with $K < 16$ colors) can be designed manually based on geometric considerations (Paeth 1990, 1991). The simplest way to design a large universal palette is through **independent uniform scalar quantization** (IUSQ), which involves applying **uniform scalar quantization** (USQ) to each color component *independently* (Jain and Pratt 1972; Gentile et al. 1990). Note that IUSQ is a special case of **independent scalar quantization** (ISQ) (Balasubramanian et al. 1994b), which itself is a primitive form of product (code) vector quantization (Sabin and Gray 1984).

IUSQ can be implemented most efficiently using **bit-cutting** (Joy and Xiang 1993), that is, truncating the least significant bits of each color component. For simplicity of discussion, let us assume that we are working in the RGB space with B bits per component, and we cut the same number C of bits from each component, reducing the number of bits per component to $\hat{B} = B - C$. This means that our independent uniform scalar quantizer has $L = 2^{\hat{B}}$ **(output) levels** per component and the **width** of each **quantization interval** is $W = 2^C$. Consequently, the ith ($i \in \{0, \ldots, L-1\}$) interval is given by $[iW, iW + W - 1]$. The above quantizer then divides the RGB space into $L \times L \times L$ subcubes of identical dimensions ($W \times W \times W$), reducing the number of distinct colors in the space from 2^{3B} to $L^3 = 2^{3\hat{B}}$. In other words, IUSQ *cannot* be used to quantize a given image to an arbitrary number of colors. Partly because of this, IUSQ is generally used as a preprocessor for image-dependent CQ algorithms (see below) rather than a standalone CQ algorithm.

Thanks to the separable nature of the above quantizer, we can quantize a given image very efficiently without resorting to computationally costly nearest neighbor searches in three dimensions. In order to map a given input color (r, g, b), all we have to do is to perform three simple univariate mapping operations, one for each component. If $v \in \{r, g, b\}$ falls in the ith quantization interval on its corresponding color axis, then v maps to the left endpoint[25] of that interval, that is, iW. Computationally, we can determine the output level corresponding to v in one of two equivalent ways: (i) $W\lfloor v/W \rfloor$, and (ii) $(v \gg C) \ll C$. The latter is usually more efficient because it involves *only* bitwise right shift ($\gg$) and left shift ($\ll$) operators.

Let us now examine a common IUSQ configuration, namely cutting $C = 3$ bits from each $B = 8$-bit color component of the RGB space. We have $L = 2^{8-3} = 32$ output levels per component and the width of each quantization interval is $W = 2^3 = 8$. This quantizer then divides the RGB space into $32 \times 32 \times 32$ subcubes of dimensions $8 \times 8 \times 8$, reducing the

[25] This convention is adopted for computational simplicity. In the SQ literature, it is more common to use the interval's midpoint so as to minimize MSE (Gersho and Gray 1992, p. 151).

number of distinct colors in the space from $2^{24} = 16,777,216$ to $2^{15} = 32,768$. The quantization intervals on each color axis (from left to right) are [0, 7], [8, 15], ⋯, [240, 247], and [248, 255]. A given a component value v on a particular color axis maps to the output level $8\lfloor v/8 \rfloor$ or, equivalently, $(v \gg 3) \ll 3$.

Figure 5 shows the Peppers image and its uniformly quantized versions where $C \in \{1, 2, \ldots, 7\}$ bits are cut from each color component. For each case, we also show the corresponding error image. Observe that with each bit cut, the number of distinct colors drops significantly even though the overall visual quality of the image is preserved with as few as 5 bits per component.

Heckbert (1982) was the first to use IUSQ as a preprocessing step for an image-dependent CQ algorithm. He cut $C = 3$ bits from each $B = 8$-bit color component both to reduce the memory requirements of his 3D histogram and to accelerate the subsequent divisive clustering process. Following Heckbert, high-resolution IUSQ has been used as a preprocessor for many hierarchical CQ algorithms. Note that while IUSQ is generally effective at reducing the amount of color data to be clustered (see Fig. 5), it is known to cause false contours, especially in large, almost uniformly colored regions (Balasubramanian et al. 1994a; Xiang and Joy 1994). Therefore, unless time or memory efficiency is of prime importance, IUSQ should be applied in moderation (e.g., by cutting 1 bit (Necaise 1998) or, at most, 2 bits (Shufelt 1997) from each 8-bit color component.) In fact, Stokes et al. (1992) empirically determine that 7.4 bits per component is required in the RGB space to avoid perceptible quantization artifacts.

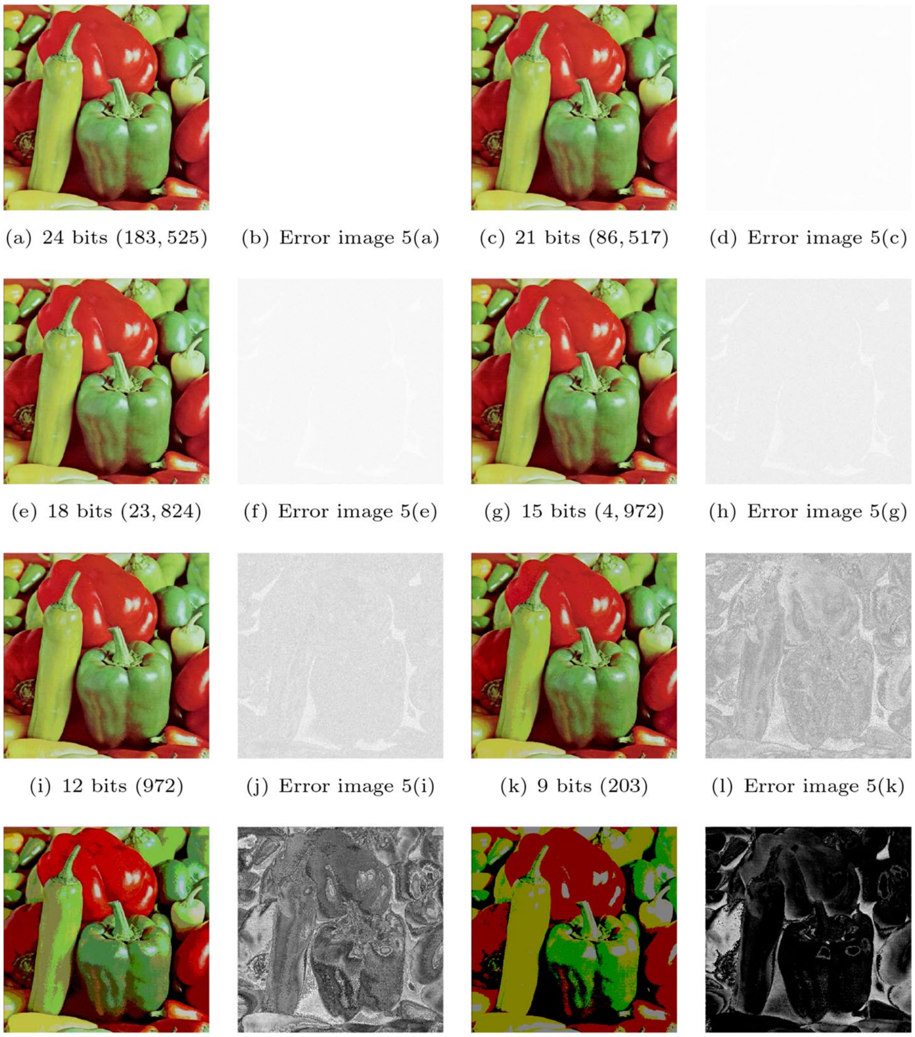
So far, we have examined the case where we cut the same number of bits from each component of the RGB space. Recall that Heckbert, and many subsequent CQ researchers, performed 3-3-3 bit-cutting in the RGB space, which divides the space into $32 \times 32 \times 32$ subcubes. Joy and Xiang (1993), however, perform 3-2-4 bit-cutting[26] to partially compensate for the perceptual nonuniformity of the RGB space. Such an uneven bit-cutting scheme divides the RGB space into $32 \times 64 \times 16$ rectangular boxes. Both 3-3-3 and 3-2-4 schemes require the same amount of memory. However, in accordance with the sensitivity of the HVS to each primary color component, the 3-2-4 scheme represents the green component at a higher resolution at the expense of a lower resolution in the blue component. Similarly, Balasubramanian et al. (1994a, 1994b) propose 1-0-2 bit-cutting.

The computational simplicity of IUSQ stems from its separable and uniform nature. However, implementing such a quantizer is straightforward and efficient *only* in an orthotope-shaped color space such as RGB, wherein quantization cells are rectangular. Several studies (Jain and Pratt 1972; Gentile et al. 1990; Yu and Chen 2006) demonstrate that IUSQ is not only more difficult to implement, but also less effective in alternative color spaces (e.g., YC$_b$C$_r$, YIQ, and I1I2I3 (Ohta et al. 1980)), even if they are linear transformations of RGB.

3.2 Nonuniform quantization

In the previous section, we discussed IUSQ, which involves applying uniform SQ to each color component independently. While uniform SQ is optimal for a uniform distribution, such is rarely the case for the red, green, and blue components of natural images (Wan and Kuo 1998). A more effective solution is to perform ISQ *nonuniformly* using a scalar quantizer such as the Lloyd–Max quantizer (Lloyd 1982; Max 1960), which is equivalent to the

[26] This scheme is inspired by the ITU-R BT.601 luminance equation: $y = 0.299r + 0.587g + 0.114b$.



(a) 24 bits (183, 525) (b) Error image 5(a) (c) 21 bits (86, 517) (d) Error image 5(c)

(e) 18 bits (23, 824) (f) Error image 5(e) (g) 15 bits (4, 972) (h) Error image 5(g)

(i) 12 bits (972) (j) Error image 5(i) (k) 9 bits (203) (l) Error image 5(k)

(m) 6 bits (52) (n) Error image 5(m) (o) 3 bits (8) (p) Error image 5(o)

Fig. 5 Peppers and its various uniformly quantized versions (subfigure captions indicate the number of bits per input color and the number of distinct colors in the reduced-color image in parentheses)

k-means algorithm in one dimension. For a given resolution, a nonuniform scalar quantizer tends to produce less distortion than a uniform scalar quantizer. In addition, given a *B*-bit color component, the number of output levels (L) in a nonuniform quantizer can be arbitrary ($L \in \{1, \ldots, 2^B\}$), whereas L *must* be a power of two in a uniform quantizer. It is important to note that nonuniform SQ need *not* be suboptimal or inefficient. In fact, optimal nonuniform SQ can be performed in linear time using dynamic programming (Wu 1991b; Wu and Zhang 1993).

A nonuniform scalar quantizer captures the distribution of a given color component better than a uniform scalar quantizer, but the independent application of a scalar quantizer to each color component disregards the spectral correlations. Colors in natural images are seldom distributed uniformly in the RGB space. Thus, an independent scalar quantizer tends to allocate representatives to regions where few input colors reside, leading to wasted representatives. In fact, regardless of how it is performed, an independent scalar quantizer divides the 3D RGB space into rectangular boxes (or parallelepipeds) using axis-parallel planes. A vector quantizer, on the other hand, divides the space into convex polyhedra using oblique planes, leading to a more effective tessellation[27] of the space. In fact, VQ can be superior to ISQ even in an uncorrelated color space (Gersho and Gray 1992, pp. 345–349). Finally, as mentioned earlier, an independent scalar quantizer is rigid in terms of the number of quantization cells it can generate. By contrast, a vector quantizer can generate an arbitrary number of quantization cells. The *only* advantages of ISQ over VQ are then its simplicity and efficiency.

In the above discussion, we assumed that the number of bits per color component is a user-defined integer. From a distortion minimization perspective, it makes sense to allocate bits to the components *adaptively* according to their variances. Let $\mathbf{X} = \left(X_1, \ldots, X_D\right)$ be a D-dimensional random vector. Given a bit budget B_{sum}, the **bit allocation problem** is to determine the optimal bit allocation vector $\mathbf{b} = (b_1, \ldots, b_D)$, where b_d is the number of bits for X_d, that minimizes the sum of MSEs of the resulting quantizers subject to $\sum_{d=1}^{D} b_d \leq B_{\text{sum}}$ (Gersho and Gray 1992, pp. 225–257). Assuming that each component X_d is identically distributed with zero mean and variance σ_d^2, Huang and Schultheiss (1963) prove that the dth component of the optimal solution is given by

$$b_d = \frac{B_{\text{sum}}}{D} + \frac{1}{2} \log_2 \frac{\sigma_d^2}{\left(\prod_{d=1}^{D} \sigma_d^2\right)^{1/D}}. \tag{6}$$

It is important to note that the optimal bit allocation vector given by the above equation may have non-integer or even negative components. A greedy algorithm due to Fox (1966) generates non-negative integer allocations, albeit suboptimally. Starting from a zero bit allocation vector, Fox's algorithm iterates B_{sum} times, allocating one bit in each iteration to the quantizer that needs it the most (the neediness of a quantizer is measured by its MSE given its current bit allocation.) In the high-resolution regime, Fox's algorithm simplifies as follows (Ramstad 1982): for each quantizer, take the standard deviation of the corresponding component as the initial demand and each time a bit is allocated to a quantizer, reduce its demand by a factor of two.

The bit allocation problem is NP-hard (Mohr 2002). Various approximation algorithms have been proposed for this problem since the 1960s. For representative algorithms, refer to Farber and Zeger (2006), Hatam and Masnadi-Shirazi (2015), and the references therein.

We have discussed universal palette design using independent uniform/nonuniform SQ with fixed/adaptive bit allocation. A more systematic, albeit computationally more expensive, way to design a large universal palette is to apply VQ to a synthetic image that contains all possible colors in equal numbers[28]. For example, Gentile et al. (1990) design a universal palette of a given size using the k-means algorithm.

[27] In an asymptotically optimal quantization of a uniform 3D distribution, each quantization cell has the shape of a truncated octahedron rather than a cube (Barnes and Sloane 1983).

[28] This could be a 4096×4096 RGB image with 2^{24} distinct colors or the same image uniformly quantized using bit-cutting.

Regardless of the design algorithm used, for a given input image and palette size, a universal palette almost always produces inferior results compared to a custom palette, and the more nonuniform the distribution of the input colors, the worse the universal palette. Consequently, CQ researchers turned to image-dependent algorithms in the early 1980s, and research on image-independent algorithms declined rapidly. Before we close this section, we should mention two interesting image-independent approaches in passing: sampling in the CIELAB space based on a Fibonacci lattice (Mojsilović and Soljanin 2001; Soljanin 2002) and trellis-coded quantization (Cheng et al. 2002).

4 Hierarchical algorithms

We now turn to image-dependent CQ algorithms that design a custom palette for a given input image. In this section, we describe hierarchical algorithms in detail.

4.1 Divisive hierarchical algorithms

As mentioned in Sect. 1, hierarchical algorithms can be divided into two categories: divisive and agglomerative. Since agglomerative algorithms typically have at least $\mathcal{O}(N^2)$ time complexity, most hierarchical CQ algorithms are divisive. Therefore, we first examine those algorithms.

A divisive algorithm partitions the 3D color space of the input image into K nonoverlapping regions using $(K-1)$ planes each of which is uniquely defined by a normal vector and a point. Each of these regions corresponds to a cluster, which is represented by the centroid (or mean) of the input colors that fall into that region. The set of these K centroids is then taken as the palette. Note that, in general, the resulting partition is *not* a **nearest neighbor partition** (aka **Voronoi partition**). In other words, the nearest centroid to a given input color is *not* necessarily the centroid of the region that contains the color.

The four main heuristics used by divisive algorithms are described below (Sproull 1991; Reitan 1999; Celebi et al. 2015).

Selection of a splitting strategy: Following tree-structured vector quantizers (Buzo et al. 1980), most divisive algorithms employ **binary splitting**[29], which is accomplished in one of two ways. One strategy, named the **blind recursion strategy**, is to split every cluster in each iteration (Buzo et al. 1980), which results in K clusters after $\lg K$ iterations. A more common strategy, named the **iterative strategy**, is to split a selected cluster in each iteration, obtaining K clusters after $(K-1)$ iterations. Geometrically, the color space of the input image is partitioned into K subspaces by a sequence of $(K-1)$ split operations. Note that the number of binary splits that can be performed to obtain K subspaces equals the number of full binary trees having exactly K leaves, $\frac{1}{K}\binom{2K-2}{K-1}$, which is too large to permit exhaustive enumeration even for relatively small values of K. For example, the number of all possible splits equals 9, 694, 845 and 14, 544, 636, 039, 226, 909 for $K = 16$ and $K = 32$, respectively.

[29] We can generalize divisive algorithms to generate trees with greater branching factors than two (Makhoul et al. 1985) or even with variable branching factors (Schmidl et al. 1993).

The blind recursion strategy leads to a height-balanced tree[30], and is easier to implement. On the other hand, the iterative strategy is more flexible, and often results in a more effective distortion minimization (Wu 1992b). In the remainder of this discussion, we assume that the iterative strategy is used.

Selection of the next cluster to be split: We have seen that a complete enumeration of all possible binary splits is computationally intractable. Consequently, a heuristic divisive algorithm must have a mechanism to determine the order of binary splits. In other words, the algorithm should select an appropriate cluster in each iteration, and then split it into two subclusters[31]. Possible choices for the next cluster to be split include:

size criterion: the cluster with the greatest size (or cardinality),
range criterion: the cluster with the greatest range on any color axis,
eigenvalue criterion: the cluster with the greatest dominant eigenvalue,
eigenvalue times size criterion: the cluster with the greatest product of its dominant eigenvalue and size,
SSE **criterion**: the cluster with the greatest **sum of squared errors** (SSE)[32], and
look-ahead SSE **criterion**: the cluster whose split would lead to the greatest reduction in the SSE.

Among these criteria, the first two are computationally the simplest ones. However, while the range criterion is sensitive to noise, the size criterion favors large (i.e., populated) clusters without considering their homogeneity. In other words, based on the size criterion, a large yet homogeneous cluster (i.e., one with a relatively small SSE) may be split, while a small yet heterogenous cluster (i.e., one with a relatively large SSE) may *not* be.

The eigenvalue criterion selects the cluster with the greatest dispersion along the PC1 of its covariance matrix. Computing the covariance matrix of a cluster requires linear time in the size of the cluster. The PC1 of this matrix can then be computed efficiently using the power method. The SSE criterion selects the least homogeneous cluster, that is, the one that contributes most to the total SSE. It can be shown that the eigenvalue and eigenvalue times size criteria are approximations of the SSE criterion (Brun and Trémeau 2003).

For a given splitting plane orientation and position (see below), the look-ahead SSE criterion selects the cluster whose split would lead to the greatest reduction in the SSE. Let $\mathcal{P}_i$ be the cluster to be split in a given iteration and $\mathcal{P}_i^l$ and $\mathcal{P}_i^r$ be the resulting subclusters. The SSE reduction achieved by splitting P_i is given by $\text{SSE}(\mathcal{P}_i) - \left(\text{SSE}(\mathcal{P}_i^l) + \text{SSE}(\mathcal{P}_i^r)\right)$. This shows that the SSE criterion is, in fact, an approximation of the look-ahead SSE criterion, where the influence of the second term, that is, the sum of the SSEs of the subclusters, is disregarded. To determine the (stepwise) optimal cluster, the look-ahead criterion requires that two clusters be tentatively split in each iteration (except the first one), resulting in a total of $2(K-2)$ tentative splits. Therefore, depending on the other decisions and the implementation strategy, this criterion may be computationally expensive.

[30] As Wu (1992a) notes, in a search tree, keeping the tree balanced is necessary to achieve logarithmic query time in the worst-case. However, in the palette design phase, distortion minimization is more important than achieving a balanced palette, while in the pixel mapping phase, we are concerned with amortized, rather than worst-case, time complexity.

[31] In the first iteration, we split the *only* cluster that contains the entire data set.

[32] For a formal definition of SSE, see Eq. (12) in Subsection 5.3. Some CQ researchers use the terms **variance** and SSE synonymously even though, strictly speaking, the variance of a given cluster equals its *mean squared error*, that is, its *sum of squared errors* divided by its size.

Among the six criteria discussed above, the SSE criterion strikes a good balance between effectiveness and efficiency. Note that for any of these criteria, in each iteration, *only* the statistics (size, range, dominant eigenvalue, etc.) of the two most recently formed clusters need to be computed since the statistics of the remaining clusters have been computed in earlier iterations.

Selection of the splitting plane orientation: Once the algorithm decides which cluster to split (say, $\mathcal{P}_i$), it needs to determine the splitting plane orientation. An optimal plane is one that minimizes $\text{SSE}(\mathcal{P}_i^l) + \text{SSE}(\mathcal{P}_i^r)$. Assuming that $\mathcal{P}_i$ is sufficiently large and its colors are in general position, there are $\mathcal{O}(n_i^3)$ ways to split $\mathcal{P}_i$ into two subclusters using a plane (Harding 1967), where n_i is the size of $\mathcal{P}_i$. Currently, the best algorithm for optimal binary splitting in 3D space requires $\mathcal{O}(n_i^3)$ time (Hasegawa et al. 1993), which is impractical for large data sets such as color image data. Therefore, we need an efficient heuristic to determine a suboptimal splitting plane. Intuitively, we need to orient the plane orthogonal to the direction in which the data dispersion is the greatest. Thus, the plane may be orthogonal to the color axis with the greatest range, the color axis with the greatest variance, the PC1 of the cluster, or another carefully selected axis. Among these choices, the PC1 is the most sensible one, as this is the direction along which the data spread is the greatest. In fact, it can be shown that PC1 is the optimal choice for multivariate Gaussian data (Dasgupta and Freund 2009). A computationally cheaper alternative is to use the color axis with the greatest variance. Note that when the splitting axis is one of the coordinate axes, the color space is partitioned into rectangular boxes using axis-parallel cuts, as in conventional decision trees (Safavian and Landgrebe 1991). By contrast, when the splitting axis is the PC1, which is a linear combination of the coordinate axes, the cuts are oblique and the resulting subspaces are convex polyhedra, as in oblique decision trees (Murthy et al. 1994).

Selection of the splitting plane position: The final decision concerns the selection of the splitting plane position, that is, the point where the plane intersects the splitting axis. The splitting plane may pass through the mean, the median, the radius-weighted mean (see below), or another carefully selected point on the splitting axis. The rationale behind the choice of the median point is that the resulting subclusters will contain approximately the same number of colors. This makes sense when building multidimensional search trees, e.g., *k*-d trees (Bentley 1975), but for the purpose of partitioning, there is *no* sound justification to require that each cluster contain nearly the same number of colors without considering the distribution of these colors (Wan et al. 1988). In fact, it can be shown that the mean point is the optimal choice for multivariate Gaussian data (Dasgupta and Freund 2009), and also that the mean is closer to the optimal splitting point than the median for *any* non-symmetric continuous univariate distribution (Wu and Witten 1985).

Figure 6 illustrates a divisive clustering algorithm on a 2D toy data set with four compact and well-separated clusters (Ruspini 1970). This algorithm employs an iterative binary splitting strategy, where, in each iteration, the cluster with the greatest SSE is split along the coordinate axis with the greatest variance at the mean point. After three splits, the algorithm obtains four clusters whose centroids are denoted by stars in Fig. 6d.

Now that we have discussed the issues surrounding the design of heuristic divisive CQ algorithms, we describe several representative algorithms that fit into this framework.

As mentioned in Sect. 1, the first divisive CQ algorithm was the **median-cut algorithm**, which was proposed by Heckbert in his 1980 Bachelor's thesis (Heckbert 1980), and later published as a journal paper (Heckbert 1982). In his thesis, the author proposed median-cut as an iterative algorithm that splits the cluster with the greatest range on any color axis, along the same axis at the median point. On the other hand, in his paper, the author described median-cut as a (blind) recursive algorithm that splits every cluster in each iteration along the color axis with the greatest range at the median point. Historically, some CQ researchers considered median-cut a recursive algorithm (Watanabe 1988; Wan et al. 1988; Wu 1992b, a), while others considered it an iterative algorithm (Gervautz and Purgathofer 1988; Orchard and Bouman 1991; Joy and Xiang 1993). In the modern literature, however, median-cut is often regarded as an iterative algorithm that splits the largest cluster in each iteration (Reitan 1999).

Wan *et al.*'s **marginal variance minimization algorithm** (Wan et al. 1988, 1990) splits the cluster with the greatest SSE along the color axis whose split would lead to the greatest reduction in marginal variance (more accurately, marginal SSE). For a given color axis, both the optimal splitting point and the resulting reduction in marginal variance can be computed in linear time using Otsu's thresholding algorithm[33] (Otsu 1979), which is equivalent to optimal 2-means clustering on histogrammed data. Note that this algorithm is *not* a true (multivariate) variance-based algorithm, as the variances are computed along the individual color axes, hence the qualifier *marginal*.

Orchard and Bouman's **oblique-cut algorithm** (Orchard and Bouman 1991) splits the cluster with the greatest eigenvalue along its PC1 at the mean point. This is the first divisive CQ algorithm that features oblique cuts.

Wu's **variance minimization algorithm** (Wu 1991a) improves upon Wan *et al.*'s algorithm by splitting the cluster with the greatest SSE along the color axis whose split would lead to the greatest reduction in true variance (more accurately, SSE). For a given color axis, the algorithm considers each component value (in order), and computes the reduction in the SSE that would be attained if this point were taken as the splitting point. This is the first divisive CQ algorithm based on true variance minimization. A less efficient version of the same algorithm was proposed earlier by Milvang (1987).

In a later study, Wu (1992b) improves upon his variance minimization algorithm by splitting the selected cluster in each iteration along its PC1. The optimal splitting point is found using the same linear-time plane sweeping approach proposed earlier by the author (Wu 1991a). The oblique splitting plane is then adjusted iteratively using the 2-means algorithm[34]. This, of course, amounts to performing a local optimization over the two most recently formed clusters. The author experiments with two different criteria to select the next cluster to be split: the SSE and the look-ahead SSE.

Regardless of the way it is performed, binary splitting is at best a greedy (or stepwise-optimal) approach that makes each splitting decision regardless of its impact on subsequent splits (Wu 1992a). Wu's **optimal principal quantization algorithm** (Wu 1992a) improves upon his earlier work based on the observation that the PC1s of the subclusters resulting from the first few splits remain approximately the same as the PC1 of the image. Hence, the first four to eight splits are performed simultaneously over the PC1 of the image using a linear-time dynamic programming algorithm. The algorithm then obtains K clusters by splitting this initial partition iteratively in a binary manner using the algorithm proposed earlier by the same author (Wu 1992b).

Joy and Xiang's **center-cut algorithm** (Joy and Xiang 1993) is similar to the iterative median-cut algorithm described in Heckbert's thesis (Heckbert 1980). The *only* difference is that, in each iteration, the former algorithm splits the selected cluster along its color axis with the greatest range at the center (i.e., mid-range) rather than the median. Using the

[33] Other thresholding algorithms (Yang and Tsai 1998) can be used as well.

[34] A more elaborate approach would be to adjust all t ($t \in \{1, \dots, K-1\}$) splitting planes simultaneously after split t using $(t+1)$-means clustering (Howard and Harris 1966).

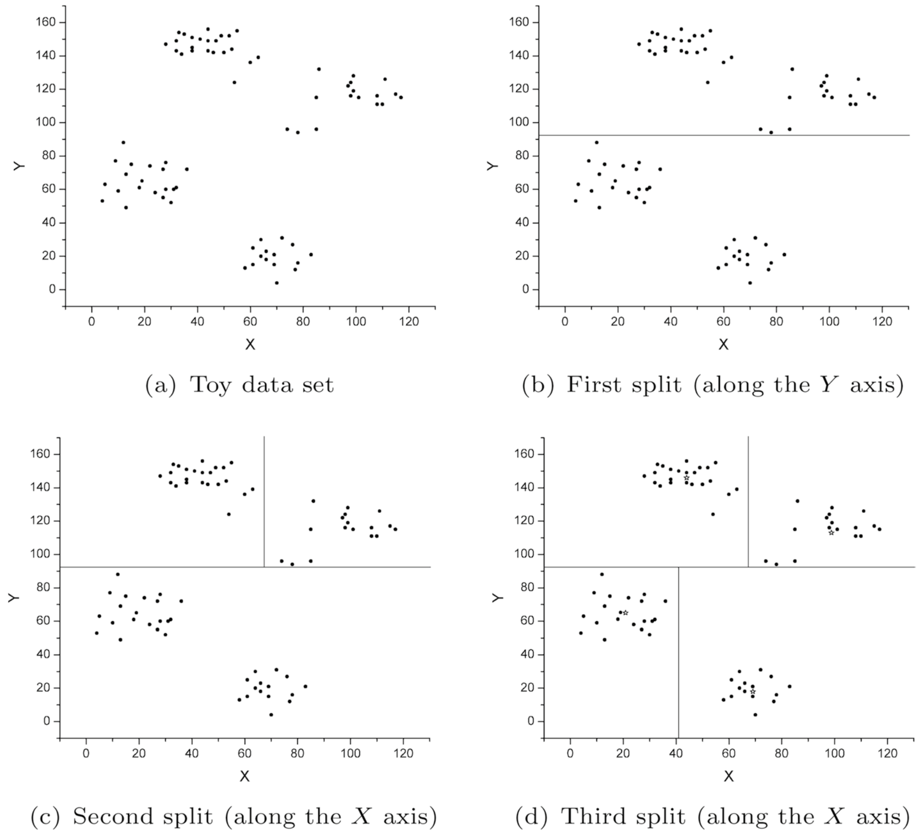


Fig. 6 Illustration of a divisive clustering algorithm on a toy data set ($K = 4$)

center as the splitting point is not only computationally cheaper, but also reduces the cluster volume by half in an attempt to limit the maximum distortion.

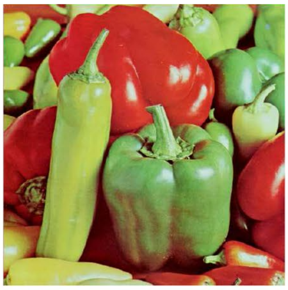
Liu and Chang's algorithm (Liu and Chang 1995) first finds the *best split axis* for each candidate cluster by tentatively splitting the cluster along each of the color axes at the mean point, and computing the corresponding reduction in the SSE. The algorithm then splits the cluster whose best split axis would lead to the greatest reduction in the SSE.

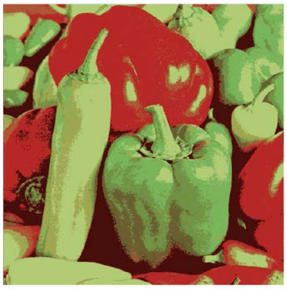
Yang and Lin's **radius-weighted mean-cut algorithm** (Yang and Lin 1996) splits the cluster with the greatest SSE along the line passing through the **radius-weighted mean** and the cluster centroid at the radius-weighted mean point[35]. The radius-weighted mean of a cluster is the weighted mean of its colors, where the weight of a color is given by its ℓ_2 distance to the cluster centroid.

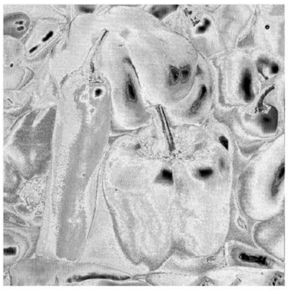
Braquelaire and Brun's algorithm (Braquelaire and Brun 1997) splits the cluster with the greatest SSE along the color axis with the greatest variance. The optimal splitting point

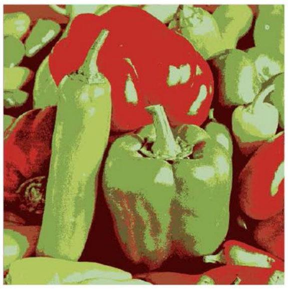
35 If the radius-weighted mean and centroid coincide, the splitting plane is taken orthogonal to the color axis with the greatest variance.

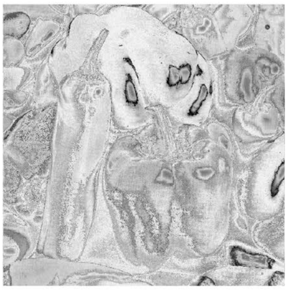
Fig. 7 Comparison of divisive CQ algorithms (number of distinct colors shown in parentheses)

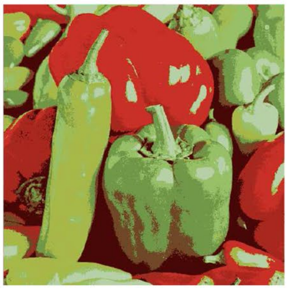

(a) Peppers (183, 525)


(b) Median-cut (8)


(c) Error image for 7(b)


(d) Oblique-cut (8)


(e) Error image for 7(d)


(f) Variance-cut (8)

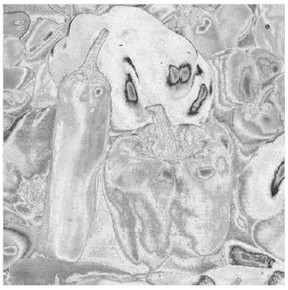

(g) Error image for 7(f)

Table 3 Comparison of divisive CQ algorithms (btw: between, dom. eig: : dominant eigenvalue, mar: marginal, max: maximum, min: minimization, mod: modified, prod: product, rad: radius, red: reduction, var: variance, w: with, wtd: weighted)

Algorithm	References	Cluster	Plane orientation	Plane position
median-cut	Heckbert (1982)	max. size	axis w. max. range	Median
mar. var. min.	Wan et al. (1990)	max. SSE	axis w. max. mar. SSE red.	Otsu threshold
oblique-cut	Orchard and Bouman (1991)	max. dom. eig.	PC1	Mean
var. min.	Wu (1991a)	max. SSE	axis w. max. SSE red.	Exhaustive search
mod. var. min.	Wu (1992a)	max. SSE	PC1	Exhaustive search
optimal principal quantization	Wu (1992b)	first 4–8 cuts simultaneous and then max. SSE	PC1	Exhaustive search
center-cut	Joy and Xiang (1993)	max. range on any axis	axis w. max. range	mid-range
–	Liu and Chang (1995)	max. SSE red.	axis w. max. SSE red.	mean
rad.-wtd. mean-cut	Yang and Lin (1996)	max. SSE	line btw. rad.-wtd. mean and centroid	rad.-wtd. mean
–	Braquelaire and Brun (1997)	max. SSE	axis w. max. var.	exhaustive search
var.-cut	Celebi et al. (2015)	max. SSE	axis w. max. var.	mean
–	Ueda et al. (2017)	max. dom. eig. $\times$ size	PC1	Otsu threshold

is determined using a faster and more robust variant of Wu's linear-time plane sweeping approach (Wu 1991a).

Celebi et al.'s **variance-cut algorithm** (Celebi et al. 2015) splits the cluster with the greatest SSE along the color axis with the greatest variance at the mean point. This axis-parallel splitting plane is then adjusted iteratively using an accelerated 2-means algorithm.

Ueda et al.'s algorithm (Ueda et al. 2017) splits the cluster with the greatest product of its dominant eigenvalue and size. The splitting point is determined using linear discriminant analysis, which happens to optimize the same criterion as Otsu's thresholding algorithm.

Figure 7 compares three divisive CQ algorithms, namely median-cut, oblique-cut, and variance-cut, on the Peppers image. It can be seen that variance-cut, which uses the SSE criterion to select the next cluster to be split, outperforms oblique-cut, which uses the less accurate eigenvalue criterion. Not surprisingly, median-cut, which uses the size criterion, performs the worst.

Table 3 compares the divisive CQ algorithms described in this section based on the heuristics used in their design.

4.2 Agglomerative hierarchical algorithms

An agglomerative clustering algorithm starts with N singleton clusters, each containing a single color. These clusters are repeatedly merged (pairwise) until K of them remain. Various criteria can be used to measure the distance (or similarity) between a pair of clusters. The following **merging criteria** are prevalent: single linkage, complete linkage, unweighted average linkage, weighted average linkage, centroid linkage, median linkage, and minimum variance.

A naive agglomerative clustering algorithm generates K clusters by iterating $(N - K)$ times, merging the nearest (or most similar) pair of clusters in each iteration. Like most greedy algorithms, this naive algorithm is *not* guaranteed to find the optimal partition[36]. Clearly, the time complexity of the algorithm is $\mathcal{O}(N^3)$. While time-optimal $\mathcal{O}(N^2)$ formulations for the aforementioned merging criteria exist (Müllner 2013), some of these algorithms are slow in practice, and many of them are difficult to implement. This is in stark contrast to the divisive clustering algorithms that can be implemented in $\mathcal{O}(N)$ time.

Among the merging criteria mentioned earlier, Ward's minimum variance criterion (and its associated algorithm) (Ward 1963) is one of the most popular (Murtagh and Legendre 2014). Starting from N singleton clusters with a total SSE of zero, in each iteration, **Ward's algorithm** merges the pair of clusters with the least **merger cost**, that is, the pair whose union would lead to the least increase in the SSE. Let $\mathcal{P}_i$ (with size n_i and centroid $\mathbf{m}_i$) and $\mathcal{P}_{\hat{i}}$ (with size $n_{\hat{i}}$ and centroid $\mathbf{m}_{\hat{i}}$) be a pair of clusters. The following are straightforward to prove:

- The merger cost for this pair is $\left(n_i n_{\hat{i}}/(n_i + n_{\hat{i}})\right)\|\mathbf{m}_i - \mathbf{m}_{\hat{i}}\|_2^2$, the weighted ℓ_2^2 distance between $\mathbf{m}_i$ and $\mathbf{m}_{\hat{i}}$;
- The size of the combined cluster is $(n_i + n_{\hat{i}})$; and

[36] Global optimality can be guaranteed *only* in the case of single linkage clustering. However, despite its theoretically appealing properties (Fisher and Van Ness 1971; Van Ness 1973; Ackerman et al. 2010; Carlsson and Memoli 2010), single linkage is generally *not* preferred due to its tendency to generate elongated clusters.

- The centroid of the combined cluster is $(n_i\mathbf{m}_i + n_{\hat{i}}\mathbf{m}_{\hat{i}})/(n_i + n_{\hat{i}})$.

About 20 years after the publication of Ward's seminal paper, Equitz (1984, 1989) rediscovered Ward's algorithm under the name **pairwise nearest neighbor algorithm** (PNN), and applied it to the VQ problem. Later, Velho et al. (1997) rediscovered the same algorithm under the name **pairwise clustering algorithm**, and applied it to the CQ problem. Finally, more than half a century after its introduction, Großwendt et al. (2019) discovered two important theoretical properties of Ward's algorithm. First, it gives a 2-approximation for the SSE objective if the clusters in the optimal K-partition[37] are well separated. Second, it recovers the optimal K-partition if the clusters are well separated and satisfy a (size) balance condition.

Now that we have introduced conventional agglomerative clustering and described Ward's algorithm in some depth, we proceed with a discussion of the three most prominent agglomerative algorithms in the CQ literature, namely the octree, PNN, and BIRCH algorithms.

The first agglomerative CQ algorithm was the **octree algorithm** proposed by Gervautz and Purgathofer (1988). The algorithm is based on the octree (Jackins and Tanimoto 1980), a tree data structure in which each internal node has eight children. The octree is a generalization of the quadtree to three dimensions. Thus, an octree can be used to partition a 3D space recursively. The root of such an octree represents the entire space; the children of the root correspond to suboctants of the space; the grandchildren of the root correspond to sub-suboctants; and so on. An octree of depth eight can thus represent all possible colors in the 24-bit RGB space. In this color octree, the depth of a node is proportional to its color homogeneity, with the leaf nodes initially representing the input colors.

Given an input image, the octree algorithm makes one pass over the image, attempting to insert the input colors successively into an initially empty octree. For each pixel, the algorithm descends the tree to determine if the pixel's color is already present in the tree. If the color has been inserted previously, the statistics of the corresponding leaf (*e.g.*, the number of distinct colors represented by the leaf, and the sums of the red, green, and blue components of those colors) are updated. On the other hand, a previously unseen color results in one of two actions. If there are fewer than K leaves, the new color is inserted into the tree as a leaf node. Otherwise, the tree is reduced in a bottom-up manner so that the number of leaves remains equal to K. Once the algorithm scans through the entire image in the above manner, the set of mean colors of the leaf nodes is taken as the palette.

The reduction operation involves identifying a suitable internal node at the maximum depth, assigning the mean color of its children to this node, and then deleting the children. In other words, the selected internal node is transformed into a leaf node whose color represents the mean color of its former children. Note that the height of the tree decreases over time due to reductions.

The octree algorithm stores at most K leaves and $(K-1)$ internal nodes at any given time. Hence, the algorithm requires *only* $\mathcal{O}(K)$ memory. This memory-efficient formulation sets the algorithm apart from its divisive rivals that require $\mathcal{O}(N)$ memory.

The octree algorithm has made a significant impact on the CQ literature. Another influential agglomerative approach is based on the PNN algorithm, which, as mentioned earlier, is identical to Ward's algorithm. In fact, several agglomerative CQ algorithms (Balasubramanian and Allebach 1991a; Dixit 1991; Xiang and Joy 1994; Velho et al. 1997; Brun and

[37] For a formal definition of K-partition, see Eq. (7) in Sect. 5.

Mokhtari 2000) are variants of the PNN algorithm. The popularity of the PNN algorithm in CQ can be attributed to the algorithm's least-squares formulation, which links it to k-means and explains its tendency to generate roughly spherical clusters.

Figure 8 shows the Peppers image quantized to 8 colors using the octree algorithm. It can be seen that, compared to divisive algorithms, octree spreads the distortion more evenly throughout the output image at the expense of increased mean distortion (Xiang and Joy 1994). In Subsection 5.2, we will describe another algorithm, maximin, that minimizes the maximum distortion in a more systematic manner.

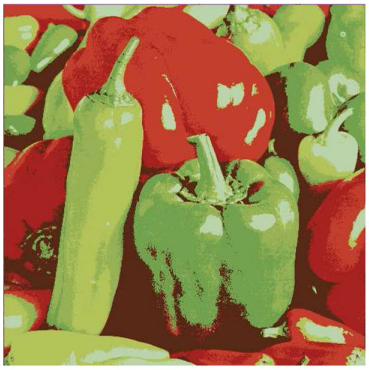
Following Equitz (1989), Balasubramanian and Allebach (1991a) propose an accelerated PNN algorithm based on the k **-d tree** (Bentley 1975), a multivariate generalization of the binary search tree that facilitates a wide range of associative queries, including nearest neighbor queries. A k-d tree can be used to partition a multidimensional space recursively. Each internal node of a conventional k-d tree partitions the data that it represents into two subsets using a hyperplane orthogonal to one of the coordinate axes and passing through the median point. Leaf nodes (or buckets), on the other hand, store the data points. The tree is built recursively by splitting the node under examination provided that it contains more than a predefined number of data points (this threshold is referred to as the *bucket size*). Observe the similarity of the k-d tree construction method to the divisive median-cut algorithm. In their implementation, Balasubramanian and Allebach split each node along the color axis with the greatest variance at the mean, rather than the median, point. Note also that, instead of an axis-parallel k-d tree, a more adaptive divisive algorithm (e.g., one that splits the node with the greatest SSE in each iteration along its PC1) could be used to obtain a better initial partition.

Recall that a naive implementation of the PNN algorithm starts with N singleton clusters and ends with K clusters after performing $(N-K)$ mergers. In each iteration, the algorithm identifies a pair of clusters with the least merger cost. Clearly, searching for an optimal pair of clusters requires $\mathcal{O}(N^2)$ time, resulting in an overall time complexity of $\mathcal{O}(N^3)$. On the other hand, the accelerated PNN algorithm considers *only* the pairs of clusters whose centroids reside in the same k-d tree bucket. In each iteration, the algorithm visits each bucket to determine an optimal pair of clusters to merge. *Only* a predefined fraction of these pairs are merged because some buckets may *not* have any close pairs. The mergers are performed in ascending order of their costs. Once these mergers are completed, the tree is balanced to account for the centroids lost/gained as a result of the mergers. Balancing is accomplished by merging small buckets with their neighbors and splitting large buckets. This three-step process (optimal pair search, mergers, and tree balancing) continues until K clusters remain. The set of centroids of these clusters is then taken as the palette.

The accelerated algorithm described above has a time complexity of $\mathcal{O}(N \log N)$ (Equitz 1989). This is a significant improvement over the $\mathcal{O}(N^3)$ complexity of the naive algorithm. However, while the naive algorithm is stepwise optimal, the accelerated algorithm is clearly *not*. Unfortunately, we are *not* aware of any CQ studies that compare the effectiveness of the two PNN algorithms thoroughly. In addition to its lack of stepwise optimality, the accelerated algorithm is more difficult to implement than its naive counterpart.

Another way to accelerate the PNN algorithm for CQ is to reduce the number of input colors prior to clustering. For example, Dixit (1991) employs uniform random sampling for data reduction, while both Velho et al. (1997) and Brun and Mokhtari (2000) employ uniform quantization.

(a) Peppers (183, 525 colors)


(b) Octree (8 colors)

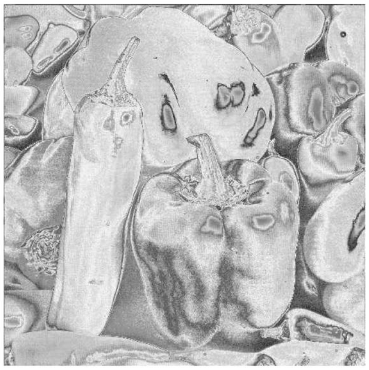

(c) Error image for 8(b)

Fig. 8 Peppers image quantized using the octree algorithm

Dixit (1991) presents an approximate agglomerative algorithm that resembles PNN. The algorithm starts by sampling the input image uniformly at random[38]. The colors of the sampled pixels are stored in a table of size T as initial clusters, in ascending order of their size ($T \leq S$ due to duplicates). This table is traversed from top to bottom, merging pairs of clusters. Let $\mathcal{P}_i$ be the cluster under examination at a given time. An optimal match for $\mathcal{P}_i$ from the remaining set $\{\mathcal{P}_{i+1}, \ldots, \mathcal{P}_T\}$ of clusters based on the PNN merger cost formula is determined. Let $\mathcal{P}_{\hat{\imath}}$ be the optimal match for $\mathcal{P}_i$. The pair $(\mathcal{P}_i, \mathcal{P}_{\hat{\imath}})$ is merged, and either one of them is replaced with the combined cluster (i.e., its size and centroid) in the table. The matched pair is excluded from the current iteration of the optimal matching process, and the algorithm proceeds with finding a match for the next unmatched cluster. Once $T/2$ pairs of clusters are merged, the next iteration starts. Hence, each iteration halves the size of the table (i.e., the number of clusters), and the algorithm continues to iterate until K clusters remain. Assuming an initial table size of $T = \mathcal{O}(K \log K)$, the time complexity of this algorithm is $\mathcal{O}(K^2 \log K)$.

[38] The author reports that a sample of size $S = 1,024$ pixels suffices for a 512×512 input image.

The BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies) algorithm (Zhang et al. 1997) provides a scalable approach to clustering data sets that are too large to fit in memory. The algorithm first generates a compact summary of the data set through a single pass. This summary is represented by an in-memory, height-balanced tree called **clustering feature-tree** (CF-tree). Given a set $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\} \subset \mathbb{R}^D$ of N data points in D dimensions, the CF representing $\mathcal{X}$ is the triplet $\text{CF}(\mathcal{X}) = (N, \mathbf{s}, ss)$ where $\mathbf{s} = \sum_{i=1}^{N} \mathbf{x}_i$ is the sum of the data points and $ss = \sum_{i=1}^{N} \|\mathbf{x}_i\|^2$ is the sum of their ℓ_2^2 norms. Essentially, $\text{CF}(\mathcal{X})$ is a three-number summary of $\mathcal{X}$ based on its first three moments. Various summary statistics for $\mathcal{X}$ (*e.g*, centroid and radius—see below) can be computed from $\text{CF}(\mathcal{X})$ without having access to the individual data points. In addition, CFs are additive in that given two disjoint subsets $\mathcal{X}_1$ and $\mathcal{X}_2$ with their corresponding summaries $\text{CF}(\mathcal{X}_1) = (N_1, \mathbf{s}_1, ss_1)$ and $\text{CF}(\mathcal{X}_2) = (N_2, \mathbf{s}_2, ss_2)$, the CF of their union is simply $\text{CF}(\mathcal{X}_1 \cup \mathcal{X}_2) = (N_1 + N_2, \mathbf{s}_1 + \mathbf{s}_2, ss_1 + ss_2)$.

Each leaf node in a CF-tree stores at most N_l CFs, each representing a summary of a subset of data points. Each internal node, on the other hand, stores at most N_i CFs, one for each of its children. Thus, both types of nodes represent summaries of subsets of data points. The CFs in a leaf node, however, must satisfy a compactness requirement: the radius[39] of a CF must be less than a predefined threshold T. Because of this compactness requirement (Sheikholeslami et al. 1998) and its particular CF representation, BIRCH tends to generate spherical clusters. The size of the CF-tree is primarily controlled by the user-defined parameters N_l, N_i, and T.

After obtaining a hierarchical summary of $\mathcal{X}$ in the form of a CF-tree, BIRCH clusters the leaves of the tree using another clustering algorithm such as Ward's algorithm. This process is fast since there are far fewer leaves in $\text{CF}(\mathcal{X})$ than data points in $\mathcal{X}$.

The advantages of BIRCH over conventional agglomerative clustering algorithms include its time[40] and memory efficiency and its resistance to outliers. On the other hand, BIRCH is difficult to implement, and its flexible formulation comes at the cost of about a dozen user-defined parameters.

Bing et al. (2004) adapt the BIRCH algorithm to the CQ problem. The authors first obtain an initial palette of size $2K$ by building a CF-tree from the input colors. The most frequent color in the initial palette is selected as the *base color* of an intermediate palette (of size K). The remaining colors in the initial palette are ordered according to their weighted distances to this base color. The weighted distance between the base color and any other color is computed as the product of the frequency of the latter color and the ℓ_2 distance between the two colors (the frequency and distance terms are raised to the powers of user-defined parameters W_f and W_d, respectively.) The $(K - 1)$ colors in the initial palette with the greatest weighted distances are added to the intermediate palette. The final palette is obtained by merging the non-selected colors in the initial palette with their nearest colors in the intermediate palette. The time complexity of the resulting CQ algorithm is $\mathcal{O}(N)$. In addition to the three BIRCH-related parameters (i.e., N_l, N_i, and T), the algorithm has two other user-defined parameters (W_f and W_d). The authors experiment with $(W_f, W_d) = (1, 1)$ and $(W_f, W_d) = (1, 2)$. Note that the part of the algorithm that selects K colors from the initial palette aims to select colors that are both frequent and dissimilar to the base color.

[39] The radius r of $\mathcal{X}$ is given by $r = \sqrt{\sum_{i=1}^{N} \|\mathbf{x}_i - \bar{\mathbf{x}}\|^2 / N}$, where $\bar{\mathbf{x}} = \mathbf{s}/N$ is the **centroid** of $\mathcal{X}$. Given $\text{CF}(\mathcal{X})$, the radius can be computed as $r = \sqrt{ss/N - \|\bar{\mathbf{x}}\|^2}$.

[40] The tree building phase requires *only* $\mathcal{O}(N)$ time.

However, the algorithm can easily select multiple frequent colors that are each dissimilar to the base color, yet similar to each other.

The main drawback of conventional hierarchical clustering algorithms is that their splitting/merging decisions are irreversible. Once a divisive algorithm splits a cluster, it *cannot* recover the same cluster later. Similarly, once an agglomerative algorithm merges a pair of clusters, it *cannot* split them later. In general, the cumulative effect of erroneous decisions is more significant in agglomerative algorithms because such algorithms execute $(N - K)$ steps, starting from singleton clusters that contain very little information (Williams 1971). By contrast, divisive algorithms execute *only* $(K - 1)$ steps, starting from a cluster that contains the *entire* data set. Another drawback of conventional hierarchical clustering algorithms is that many of them do *not* optimize a particular objective function (Dasgupta 2016; Cohen-Addad et al. 2019). Without an explicit objective, it is difficult not only to compare such algorithms but also to analyze their theoretical properties. Nevertheless, divisive hierarchical algorithms are efficient (requiring linear time in N and linear or logarithmic time in K), and most of them are deterministic and relatively easy to implement. Therefore, such algorithms are often used to initialize partitional algorithms, which are discussed in the next section.

5 Partitional algorithms

Due to limited computational resources, early CQ algorithms (1980–2000) were mostly hierarchical. In the early 2000s, researchers turned to partitional CQ algorithms, which can achieve better results if they are initialized properly.

Given a data set $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\} \subset \mathbb{R}^D$ and an integer K ($K \in \{1, \dots, N\}$), a partitional algorithm divides $\mathcal{X}$ into a collection $\mathscr{P} = \{\mathcal{P}_1, \dots, \mathcal{P}_K\}$ of K exhaustive, mutually exclusive, and nonempty subsets, that is,

$$\bigcup_{i=1}^{K} \mathcal{P}_i = \mathcal{X} \text{(exhaustive)} \tag{7a}$$

$$\mathcal{P}_i \cap \mathcal{P}_{\hat{\imath}} = \varnothing, \quad i, \hat{\imath} \in \{1, \dots, K\},\ i \neq \hat{\imath} \text{(mutually exclusive)} \tag{7b}$$

$$n_i = |\mathcal{P}_i| > 0, \quad i \in \{1, \dots, K\} \text{ (nonempty)} \tag{7c}$$

The family $\mathscr{P}$ of sets is called a K-**partition** and the subsets $\mathcal{P}_1, \dots, \mathcal{P}_K$ are termed its **cluster**s. Clearly, the above conditions imply that $n_1 + \dots + n_K = N$.

The number of ways in which a set of N objects can be divided into K nonempty subsets is given by $\frac{1}{K!}\sum_{i=0}^{K}(-1)^{K-i}\binom{K}{i} i^N$, which can be approximated by $K^N/K!$ for fixed K as $N \to \infty$. For example, even for a tiny set of size 100, there are $0.2755 \cdot 10^{94}$ ways to partition this set into 10 nonempty subsets. Therefore, exhaustive search is computationally infeasible, except in special cases.

Virtually all partitional clustering algorithms applied to the CQ problem to date are center-based algorithms. Like hierarchical CQ algorithms, these center-based partitional CQ algorithms represent each cluster $\mathcal{P}_i$ with a center $\mathbf{c}_i$. Once the clustering process is completed, the set $\mathcal{C} = \{\mathbf{c}_1, \dots, \mathbf{c}_K\}$ of centers is then taken as the palette. Consequently, we use the terms *center* and *representative* interchangeably in this section.

5.1 The popularity algorithm

The **popularity algorithm**[41] (Heckbert 1982) is conceptually the simplest CQ algorithm. It first builds a low-resolution (or coarse) color histogram of the input image using bit-cutting[42]. The set of K most frequent colors in this histogram is then taken as the palette.

Let N_b be the number of bins in the histogram. The simplest way to determine the K largest bins is to sort the histogram in $\mathcal{O}(N_b \log N_b)$ time using a comparison-based algorithm such as **quicksort**. A more efficient approach is to first find the Kth largest element in $\mathcal{O}(N_b)$ time using a partition-based selection algorithm such as **quickselect** (Hoare 1971), and then take the K elements that are greater than or equal to it.

The popularity algorithm may produce acceptable results on synthetic images with large, uniformly colored regions, but it generally performs poorly on natural images with a wide range of colors. The algorithm tends to overrepresent the background colors, which are usually of limited interest. In addition, it is sensitive to the resolution of the histogram. If the resolution is too high, it may fail to locate K prominent peaks, or it may select frequent yet similar colors. On the other hand, if the resolution is too low, significant color shifts may occur.

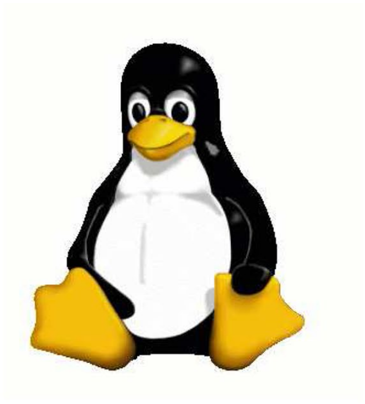
The popularity algorithm considers *only* the frequency of the histogram colors, and disregards their diversity. Braudaway (1987) proposes a **modified popularity algorithm** that considers both frequency and diversity. The algorithm starts by building a low-resolution color histogram of the input image, and then selecting the most frequent color as the first center ($\mathbf{c}_1$). The remaining $(K-1)$ centers are selected successively as follows. Suppose that the algorithm is at the beginning of iteration i ($i \in \{2, \ldots, K\}$), that is, it has just selected $\mathbf{c}_{i-1}$. Before selecting $\mathbf{c}_i$, to avoid selecting a color that is frequent yet similar to $\mathbf{c}_{i-1}$, the algorithm reduces the frequency of each histogram color $\mathbf{c}$ by a factor of $\left(1 - e^{-C\|\mathbf{c}-\mathbf{c}_{i-1}\|^2}\right)^{-1}$, where C is a user-defined positive number. This function reduces the frequency of $\mathbf{c}_{i-1}$ to zero so that it *cannot* be selected again. The frequencies of $\mathbf{c}_{i-1}$'s nearest neighbors are reduced severely so that they are unlikely to be selected. Finally, the frequencies of $\mathbf{c}_{i-1}$'s sufficiently distant neighbors are left unchanged. With a proper C value, the modified popularity algorithm can significantly outperform the standard one. Celebi et al. (2014, 2015) report good results with $C = 0.25$. Note that Yager and Filev (1994) later proposed a similar clustering algorithm under the name **mountain algorithm**.

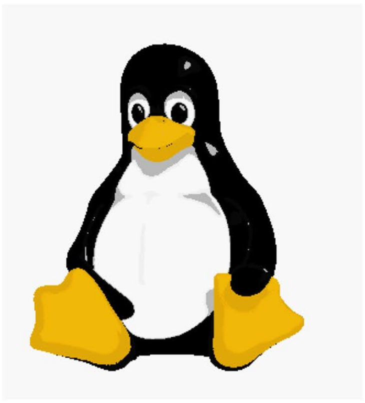
Figures 9 and 10 compare the original and modified popularity algorithms on the Tux (credits[43], 327×360 pixels) and Peppers images, respectively. In each case, the input image is preprocessed using 3-3-3 bit-cutting (cutting fewer bits makes little difference for Tux, but leads to worse results for Peppers). As expected, both algorithms perform fairly well on the synthetic image (with the modified algorithm producing slightly better results). On the other hand, the original algorithm performs very poorly on the natural image because it simply selects the most frequent 8 colors, which all happen to be shades of red, without considering their separation.

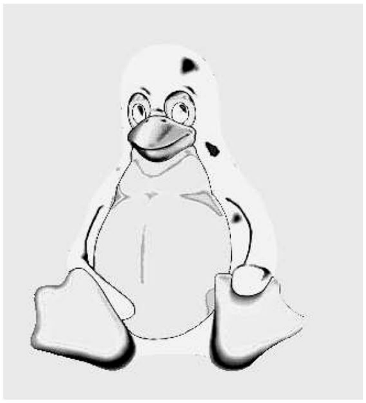
[41] Less commonly known as the **populosity algorithm** (Velho et al. 1997).

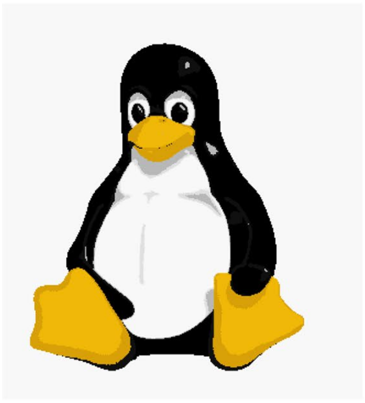
[42] In the context of divisive CQ algorithms, the primary purpose of bit-cutting is to reduce time and memory requirements. For the popularity algorithm and its variants, however, bit-cutting is *necessary* to detect the dominant peaks of the color histogram effectively.

[43] Courtesy of Larry Ewing (lewing@isc.tamu.edu) and the GIMP


(a) Linux 2.0 Penguin (193 colors)


(b) Popularity (8 colors)


(c) Error image for 9(b)


(d) Modified popularity (8 colors)

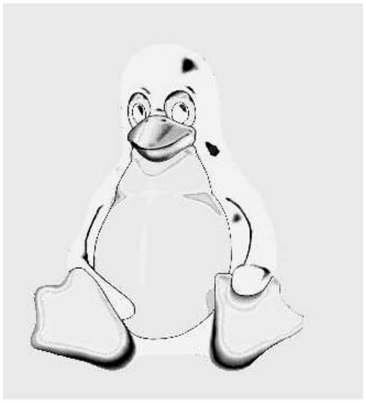
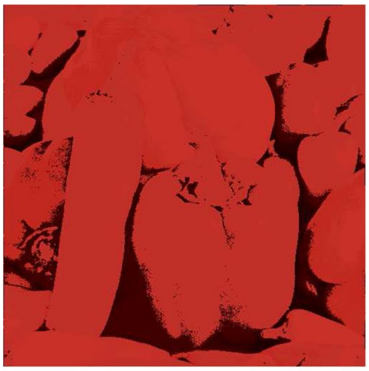
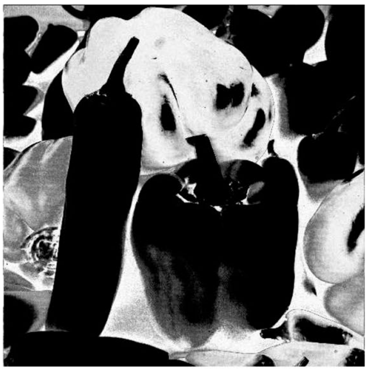
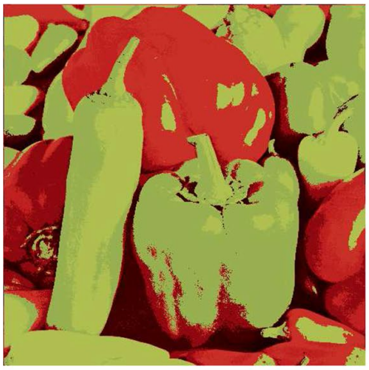

(e) Error image for 9(d)

Fig. 9 Comparison of popularity-based CQ algorithms on a synthetic image

(a) Peppers (183, 525 colors)


(b) Popularity (8 colors)


(c) Error image for 10(b)


(d) Modified popularity (8 colors)

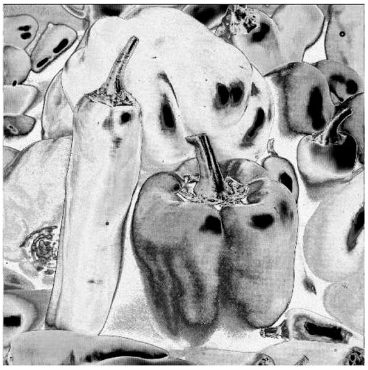

(e) Error image for 10(d)

Fig. 10 Comparison of popularity-based CQ algorithms on a natural image

5.2 The maximin algorithm

The **maximin algorithm** (Gonzalez 1985) selects $\mathbf{c}_1$ arbitrarily from $\mathcal{X}$ and the remaining $(K-1)$ centers are selected successively as follows. In iteration i ($i \in \{2, \ldots, K\}$), $\mathbf{c}_i$ is selected to be the data point with the greatest minimum-distance to the previously selected

$(i-1)$ centers, that is, $\mathcal{C}^{(i-1)} = \{\mathbf{c}_1, \dots, \mathbf{c}_{i-1}\}$. In other words, $\mathbf{c}_i$ is selected to be $\mathbf{x}_{j^*} \in \mathcal{X}$ with index

$$j^* = \operatorname*{arg\,max}_{j \in \{1,\dots,N\}} \delta\left(\mathbf{x}_j, \mathcal{C}^{(i-1)}\right), \tag{8}$$

where $\delta\left(\mathbf{x}_j, \mathcal{C}^{(i-1)}\right)$ is the distance between $\mathbf{x}_j$ and its nearest center in $\mathcal{C}^{(i-1)}$, that is,

$$\delta\left(\mathbf{x}_j, \mathcal{C}^{(i-1)}\right) = \min_{\hat{\imath} \in \{1,\dots,i-1\}} d(\mathbf{x}_j, \mathbf{c}_{\hat{\imath}}), \tag{9}$$

where $d(\cdot,\cdot)$ is a metric.

If there are multiple candidates for $\mathbf{x}_{j^*}$, a common tie-breaking rule is to select the data point with the smallest index. Using the K centers given by this algorithm, we can construct a Voronoi partition of $\mathcal{X}$ by assigning each data point to its nearest center.

We mentioned earlier that $\mathbf{c}_1$ can be selected arbitrarily[44]. A simple strategy is to select a data point uniformly at random from $\mathcal{X}$, but this would render the algorithm nondeterministic. A convenient and deterministic alternative is to select the data point nearest to the centroid of $\mathcal{X}$ given by $\bar{\mathbf{x}} = (1/N)\sum_{j=1}^{N} \mathbf{x}_j$. Now that $\mathbf{c}_1$ is selected deterministically, maximin becomes completely deterministic.

An interesting feature of maximin is that the algorithm selects *exactly* one center from each of the K clusters if $\mathcal{X}$ is composed of compact and separated clusters, that is, if each of the possible intra-cluster distances is less than each of the possible inter-cluster ones (Hathaway et al. 2006). This can be expressed mathematically as follows

$$\frac{\displaystyle\min_{\substack{i,\hat{\imath} \in \{1,\dots,K\},\\ i \neq \hat{\imath}}} \min_{\mathbf{x} \in \mathcal{C}_i, \mathbf{x}' \in \mathcal{C}_{\hat{\imath}}} d(\mathbf{x}, \mathbf{x}')}{\displaystyle\max_{i \in \{1,\dots,K\}} \max_{\mathbf{x},\mathbf{x}' \in \mathcal{C}_i} d(\mathbf{x}, \mathbf{x}')} > 1. \tag{10}$$

This greedy algorithm was originally developed as a 2-approximation for the **discrete** k **-center problem**, which is defined as follows. Given a set $\mathcal{X}$ of N data points in a metric space, find K representative data points (or centers) such that the maximum distance between any data point and its nearest center is minimized. If D or K is *not* fixed, the problem with $d = \ell_2$ is NP-hard even on the plane (Fowler et al. 1981; Masuyama et al. 1981; Megiddo and Supowit 1984). In fact, it is NP-hard to approximate this problem to within a factor of less than 2 (Hsu and Nemhauser 1979). Hence, maximin is the best possible polynomial-time approximation algorithm for this problem.

The pseudocode for maximin is given in Algorithm 1[45]. It is important to note that while the algorithm works with *any* distance, its approximation guarantee holds *only* for metric distances. However, for the case of $d = \ell_2$, there is *no* need to compute the square root because the algorithm requires *only* relative distances.

Clearly, a naive implementation[46] of Algorithm 1 requires $\mathcal{O}(NK)$ time. Feder and Greene (1988) propose an elaborate 2-approximation algorithm for the k-center problem

[44] For color image data, one option is the most frequent color (Houle and Dubois 1986).

[45] For conciseness, we omit the handling of exceptional cases such as data points equidistant to multiple centers.

[46] For faster implementations of maximin, refer to Geraci et al. (2006), Zhao et al. (2013), and Borgelt and Yarikova (2020).

(with *any* ℓ_p metric) with $\mathcal{O}(N \log K)$ time complexity, which is optimal under the algebraic computation tree model.

```
input : $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\} \subset \mathbb{R}^D$, $K$
output: $\mathcal{C} = \{\mathbf{c}_1, \ldots, \mathbf{c}_K\} \subset \mathbb{R}^D$
1  Select $\mathbf{c}_1$ arbitrarily
2  for $j \leftarrow 1$ to $N$ do
3      $d_j \leftarrow \infty$    ▷ $\mathbf{x}_j$'s distance to nearest center
4  end
5  Select the remaining $(K-1)$ centers
6  for $i \leftarrow 2$ to $K$ do
7      $d_{\max} \leftarrow -\infty$
8      for $j \leftarrow 1$ to $N$ do
9          dist $\leftarrow d(\mathbf{x}_j, \mathbf{c}_{i-1})$    ▷ $\mathbf{c}_{i-1}$ : most recently selected
            center
10         if $dist < d_j$ then $d_j \leftarrow$ dist    ▷ $\mathbf{c}_{i-1}$ is nearest to $\mathbf{x}_j$
11         if $d_{max} < d_j$ then
12             $(d_{\max}, j^*) \leftarrow (d_j, j)$    ▷ $\mathbf{x}_j$ is farthest to $\mathcal{C}^{(i-1)}$
13         end
14     end
15     $\mathbf{c}_i \leftarrow \mathbf{x}_{j^*}$    ▷ $\mathbf{x}_{j^*}$ is the new center
16 end
```

Algorithm 1: The Maximin Algorithm

Maximin can be extended to deal with weighted data $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\} \subset \mathbb{R}^D$, where each data point $\mathbf{x}_j$ has a nonnegative weight w_j associated with it. These weights might reflect the importance of individual data points or relative frequencies of groups of data points. For example, in a CQ application, the weighted algorithm may be computationally advantageous if we already have a color histogram of the input image at hand.

Without loss of generality, we assume that the weights add up to one, that is, $\sum_{j=1}^{N} w_j = 1$. The objective function for the **weighted** k **-center problem** is identical to that for the unweighted problem, with the exception that the distances are weighted.

The **weighted maximin algorithm** (Dyer and Frieze 1985) selects a data point with the greatest weight as $\mathbf{c}_1$. In iteration $i \in \{2, \ldots, K\}$, $\mathbf{c}_i$ is selected to be the data point with the greatest minimum-weighted-distance to the previously selected $(i-1)$ centers. In other words, $\mathbf{c}_i$ is selected to be $\mathbf{x}_{j^*} \in \mathcal{X}$ with index

$$j^* = \underset{j \in \{1, \ldots, N\}}{\arg\max}\, w_j \delta\left(\mathbf{x}_j, \mathcal{C}^{(i-1)}\right). \tag{11}$$

As in the unweighted algorithm, if there are multiple candidates for $\mathbf{x}_{j^*}$, we employ a tie-breaking rule. Using the K centers given by this algorithm, we can construct a Voronoi partition of $\mathcal{X}$ by assigning each data point to its nearest center.

Maximin performs well on well-separated data sets provided that there are *no* outliers. Turnbull and Elkan (2005) suggest applying maximin to a small random subset of $\mathcal{X}$ to reduce the possibility of selecting outliers as centers. Maximin also runs faster on such a subset. Assuming that the clusters are equal in size, it can be shown that a random subset of size $\hat{N} = \lceil CK \ln K \rceil$ for some constant $C > 1$ contains at least one data point from each of the K clusters with probability greater than $\left(1 - K^{1-C}\right)$. For example, for $K = 32$ and $C = 2$, we need a subset of size $\hat{N} = 222$ and the probability of selecting at least one data point from each cluster exceeds 0.96. Observe that the size $\hat{N}$ of the subset is independent of the size N of the full data set. Celebi (2009, 2011) reports good results with $C = 2$.

Maximin was applied to the CQ problem first by Houle and Dubois (1986) and later by Goldberg (1991) and Xiang (1997). Xiang's implementation differs from the earlier ones by its use of the cluster centroids rather than the centers selected by the standard algorithm. As mentioned earlier, the cluster centroids can be obtained by first assigning each data point to its nearest center, and then averaging the data points in each cluster.

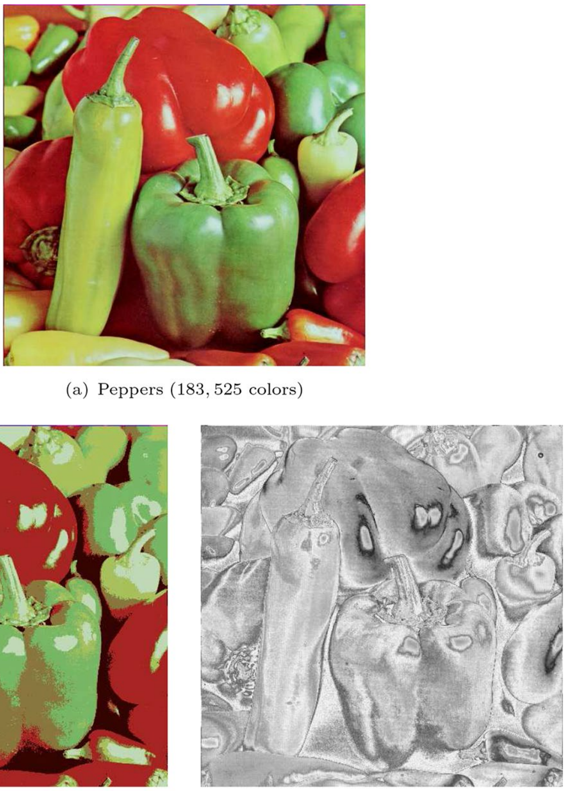
Figure 11 shows the Peppers image quantized to 8 colors using Xiang's variant of the maximin algorithm. The almost uniformly dark error image confirms that the algorithm spreads the distortion more or less evenly throughout the image in an attempt to minimize the maximum distortion (i.e., objectionable color shifts) (Xiang and Joy 1994; Celebi 2011; Celebi et al. 2015). In fact, if the goal is to minimize the mean distortion, maximin may *not* be appropriate because it gives an N-approximation for the k-means problem in the worst case. Due to its tendency to sacrifice mean distortion, several studies (Reitan 1999; Celebi 2009, 2011; Celebi et al. 2014, 2015; Thompson et al. 2020) conclude that maximin is inferior to most other CQ algorithms. Nevertheless, maximin can be used as an effective and deterministic initializer for more elaborate partitional clustering algorithms such as batch k-means and online k-means, as demonstrated recently by Thompson et al. (2020).

The maximin algorithm aims to maximize the dispersion of the representatives in the color space or, equivalently, their diversity. Hsieh and Fan (2000) propose a **modified maximin algorithm** that considers both the frequency and diversity of colors. The algorithm first builds a low-resolution color histogram of the input image, and then traverses this histogram in descending order of frequency. The most frequent color is taken as $\mathbf{c}_1$. The second most frequent color is taken as $\mathbf{c}_2$ provided that its ℓ_2 distance to the nearest previously selected center is greater than a user-defined threshold T; otherwise, the next most frequent color is considered. The remaining $(K - 2)$ centers are selected similarly.

Hsieh and Fan (2000) recommend setting $T = \sqrt[3]{255^3/K}$ initially. If the algorithm fails to select K centers after going through the histogram, T is reduced by a small amount, and the clustering process is restarted. The set of K centers selected is taken as the initial palette. Clusters are formed around the initial representatives by assigning each histogram color to its nearest representative. The set of weighted centroids of these clusters is then taken as the final palette. This algorithm is sensitive to the value of T. If T is too small, frequent yet similar colors may be included in the initial palette. On the other hand, if T is too large, rare colors may be selected, or the histogram may be exhausted before K colors can be selected.

More recently, Huang (2021) proposed another modified maximin algorithm[47]. The algorithm first builds a builds a low-resolution color histogram of the input image. The most frequent color is taken as $\mathbf{c}_1$. In iteration i ($i \in \{2, \ldots, K\}$), $\mathbf{c}_i$ is selected to be the data point with the greatest product of the square root of its frequency and its minimum-distance to the previously selected $(i - 1)$ centers. Compared to Hsieh and Fan's algorithm, Huang's algorithm has the advantage of *not* requiring a threshold value. However, ℓ_2^2 distance and square root of frequency are incommensurable quantities. For example, the distance term may dominate in small images, whereas the frequency term may dominate in large images. Hence, both terms should be normalized by their observed maximums in the input image.

[47] Yuan and Goldberg (1988) proposed a similar algorithm in the context of VQ.


(a) Peppers (183, 525 colors)

(b) Maximin (8 colors)

(c) Error image for 11(b)

Fig. 11 Peppers image quantized using Xiang's maximin algorithm

5.3 The *K*-means algorithm

A popular clustering objective function is the **sum of errors** (SE) given by

$$J_{\text{SE}} = \sum_{\mathbf{x} \in \mathcal{X}} \delta(\mathbf{x}, \mathcal{C}), \tag{12}$$

where $\delta(\mathbf{x}, \mathcal{C})$ denotes the Bregman divergence (Bregman 1967) between $\mathbf{x}$ and its nearest center in $\mathcal{C} = \{\mathbf{c}_1, \ldots, \mathbf{c}_K\}$, that is,

$$\delta(\mathbf{x}, \mathcal{C}) = \min_{i \in \{1,\ldots,K\}} d_B(\mathbf{x}, \mathbf{c}_i), \tag{13}$$

where $d_B(\mathbf{x}, \mathbf{c}_i)$ denotes the Bregman divergence between $\mathbf{x}$ and $\mathbf{c}_i$. It is important to note that, unlike the discrete k-center objective, the set $\mathcal{C}$ of centers in objective (12) is *not* constrained to be a subset of $\mathcal{X}$.

Bregman divergences are a family of nonmetric distance functions that includes the squared Euclidean distance (ℓ_2^2), squared Mahalanobis distance, Kullback–Leibler divergence, and Itakura–Saito divergence. In practice, the most popular Bregman divergence is ℓ_2^2, in which case Eq. (12) is referred to as the **sum of squared errors** (SSE).

For *any* Bregman divergence, it can be shown that the optimal center $\mathbf{c}_i$ for cluster $\mathcal{P}_i$ is given by the **centroid** of the cluster (Banerjee et al. 2005), that is,

$$\mathbf{c}_i = \frac{1}{n_i} \sum_{\mathbf{x} \in \mathcal{P}_i} \mathbf{x}, \tag{14}$$

where n_i denotes the size of $\mathcal{P}_i$.

For $K > 1$, the k **-means problem**[48] (KM), that is, minimizing Eq. (12) with $d = \ell_2^2$, is a nonsmooth and nonconvex optimization problem with numerous local minima. If D or K is *not* fixed, the problem is NP-hard[49] even for $K = 2$ ($D \geq 2$) (Aloise et al. 2009) or $D = 2$ ($K \geq 2$) (Vattani 2009; Mahajan et al. 2009). Recently, Awasthi et al. (2015) proved that there exists a constant $\varepsilon > 0$ such that it is NP-hard to approximate this problem to within a factor of $(1 + \varepsilon)$. Later, Lee et al. (2017) established that $\varepsilon \geq 0.0013$.

Among the heuristics for the KM problem, the **(batch)** k **-means algorithm**[50] (BKM) is the simplest and most popular one (Celebi 2015). The algorithm is based on the idea that it is easy to determine the optimal clusters when the centers are fixed and, similarly, it is easy to determine the optimal centers when the clusters are fixed. The algorithm starts with K arbitrary centers, typically selected uniformly at random from $\mathcal{X}$. It then alternates between an assignment step and an update step until a predefined termination criterion is met. In the assignment step, each data point is assigned to its nearest center using Eq. (13), whereas, in the update step, each center is recomputed as the mean of all data points assigned to it using Eq. (14). This is an **alternating minimization** procedure because in the assignment step, the centers are held constant and the partition is optimized, whereas in the update step, the partition is held constant and the centers are optimized.

The pseudocode for BKM is given in Algorithm 2[51]. In each iteration, we first assign each data point to its nearest center, and then update the size and vector sum of the corresponding cluster. After computing the Voronoi partition of $\mathcal{X}$ in this manner, we recompute each center as the ratio between its vector sum and size. These iterations continue until the relative change in the centers between two consecutive iterations drops below a threshold, that is,

$$\frac{1}{K} \sum_{i=1}^{K} \frac{\left\| \mathbf{c}_i^{(t-1)} - \mathbf{c}_i^{(t)} \right\|_2}{\left\| \mathbf{c}_i^{(t-1)} \right\|_2 + \left\| \mathbf{c}_i^{(t)} \right\|_2} \leq \varepsilon, \tag{15}$$

where $\mathbf{c}_i^{(t)}$ is the center of $\mathcal{P}_i$ at the end of iteration t ($t = 1, 2, \ldots$) and $\varepsilon \geq 0$ is a user-defined threshold. The cases $\varepsilon = 0$ and $\varepsilon > 0$ correspond to **termination upon convergence** and **early termination**, respectively.

Another common termination criterion is to stop the iterations whenever the relative change in the SSE between two consecutive iterations drops below a threshold (Linde et al.

[48] In the clustering literature, k **-means** may refer to an objective function to be minimized or the best-known algorithm for minimizing this objective.

[49] In the early CQ literature, the hardness of the KM problem was incorrectly attributed to various authors, including Hyafil and Rivest (1976), Brucker (1978), and Garey et al. (1982).

[50] Also known as the Lloyd's algorithm (Lloyd 1982), generalized Lloyd algorithm (GLA) (Gray and Karnin 1982), or Linde–Buzo–Gray algorithm (LBG) (Linde et al. 1980).

[51] As before, we omit the handling of exceptional cases such as empty clusters or data points equidistant to multiple centers.

1980), that is, $\left(\mathrm{SSE}^{(t-1)} - \mathrm{SSE}^{(t)}\right)/\mathrm{SSE}^{(t-1)} \leq \varepsilon$, where $\mathrm{SSE}^{(t)}$ denotes the SSE at the end of iteration t ($t = 1, 2, \ldots$) with $\mathrm{SSE}^{(0)} = \infty$, and $\varepsilon \geq 0$ is a user-defined threshold ($\varepsilon = 0.001$ and $\varepsilon = 0.0001$ are commonly used in the literature.)

```
input  : $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\} \subset \mathbb{R}^D$, $K$
output: $\mathcal{C} = \{\mathbf{c}_1, \ldots, \mathbf{c}_K\} \subset \mathbb{R}^D$
1  while termination criterion is not met do
2      for $i \leftarrow 1$ to $K$ do
3          $n_i \leftarrow 0$    ▷ size of $\mathcal{P}_i$
4          $\hat{\mathbf{c}}_i \leftarrow \mathbf{0}$    ▷ vector sum of $\mathcal{P}_i$
5      end
6      Assign each data point to the nearest center
7      for $j \leftarrow 1$ to $N$ do
8          $i \leftarrow \underset{\hat{\imath} \in \{1,\ldots,K\}}{\arg\min} \|\mathbf{x}_j - \mathbf{c}_{\hat{\imath}}\|_2^2$    ▷ $\mathbf{c}_i$ is nearest to $\mathbf{x}_j$
9          $n_i \leftarrow n_i + 1$
10         $\hat{\mathbf{c}}_i \leftarrow \hat{\mathbf{c}}_i + \mathbf{x}_j$
11     end
12     for $i \leftarrow 1$ to $K$ do
13         $\mathbf{c}_i \leftarrow \hat{\mathbf{c}}_i / n_i$    ▷ recompute $\mathbf{c}_i$
14     end
15 end
```

Algorithm 2: The BKM Algorithm

BKM can be extended to deal with weighted data $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\} \subset \mathbb{R}^D$, where each data point $\mathbf{x}_j$ has a nonnegative weight w_j associated with it. Without loss of generality, we assume that the weights add up to one, that is, $\sum_{j=1}^{N} w_j = 1$. The objective function for the **weighted** k **-means problem** is identical to that for the unweighted problem, with the exception that the distances are weighted. The optimal center $\mathbf{c}_i$ for $\mathcal{P}_i$ is given by the weighted centroid of the cluster, that is,

$$\mathbf{c}_i = \frac{1}{\sum_{\mathbf{x}_j \in \mathcal{P}_i} w_j} \sum_{\mathbf{x}_j \in \mathcal{P}_i} w_j \mathbf{x}_j. \tag{16}$$

The pseudocode for the **weighted (batch)** k **-means algorithm** can be obtained by making two simple modifications to Algorithm 2: (1) $n_i \leftarrow n_i + 1$ should be $n_i \leftarrow n_i + w_j$, and (2) $\hat{\mathbf{c}}_i \leftarrow \hat{\mathbf{c}}_i + \mathbf{x}_j$ should be $\hat{\mathbf{c}}_i \leftarrow \hat{\mathbf{c}}_i + w_j \mathbf{x}_j$.

BKM is undoubtedly the most popular hard partitional clustering algorithm (Wu et al. 2008). Its popularity can be attributed to several reasons:

- It is conceptually simple and easy to implement.
- It has a time complexity linear in N, D, and K, that is, $\mathcal{O}(NDK)$ per iteration. For this reason, it can be used to initialize more expensive clustering algorithms. Furthermore, numerous sequential and parallel acceleration techniques are available in the literature.
- It is guaranteed to converge to a local minimum of its objective in a finite number of iterations.
- It is insensitive to the order in which the data points are processed.

On the other hand, BKM has several significant disadvantages:

- It is designed to discover spherical clusters that are well separated. In other words, it is unsuitable for discovering arbitrary shaped or overlapping clusters.
- It tends to discover clusters of approximately equal size.
- Because of the nonconvexity of its objective and its descent formulation, it converges to a local minimum, which can be arbitrarily far from a global minimum. For the same reasons, it is sensitive to the selection of the initial cluster centers.

It is important to note that these drawbacks are shared by many other partitional clustering algorithms (Celebi 2015).

Heckbert (1982) was the first to suggest using BKM[52] to improve the palette generated by a divisive CQ algorithm such as median-cut. However, Heckbert and several other early CQ researchers (Wu and Witten 1985; Wan et al. 1988, 1990; Orchard and Bouman 1991; Wu 1992b) deemed BKM to be computationally impractical[53] due to its iterative nature. This negative perception of BKM prompted the development of various efficient divisive CQ algorithms between the early 1980s and the early 2000s.

Adaptations of BKM to the CQ problem primarily focus on two issues: (i) obtaining better local minima through adaptive initialization, and (ii) accelerating the algorithm. The former concerns the algorithm's effectiveness, whereas the latter concerns its efficiency. However, these issues are *not* independent, as initialization also affects convergence.

We first examine the issue of **initialization**. Recall that the KM objective is a nonsmooth and nonconvex function with numerous local minima. Hence, a local search algorithm such as BKM is expected to converge to a local minimum nearest to the initial configuration. Unfortunately, such a local minimum can be far from a global minimum, especially in a large data set. Therefore, initialization is crucial for BKM and its variants (Celebi and Kingravi 2012; Celebi et al. 2013; Celebi and Kingravi 2015). Adverse effects of improper initialization include empty clusters, slower convergence, and a higher chance of getting stuck in a poor local minimum.

The most popular initialization method is **random selection**, where K data points are selected uniformly at random from $\mathcal{X}$. This method is likely to select data points from dense regions, but it has *no* mechanism to avoid selecting nearby data points. In a CQ application, this method is likely to select frequent colors without considering their diversity. Another popular initialization method is the maximin algorithm. Recall that, unlike random selection, maximin considers diversity but *not* frequency. There are dozens of other initialization methods (Celebi et al. 2013) of which k **-means++** (Arthur and Vassilvitskii 2007) is the most well-known. In this paper, we refer to these as **generic initialization method**s because they are *not* specifically designed for color image data.

As mentioned earlier, we can also use a hierarchical CQ algorithm (e.g., median-cut) as an initializer. We refer to these as **image-specific initialization method**s. Celebi (2011) compares seven generic and seven image-specific methods as initializers for BKM. Not surprisingly, the author finds that image-specific methods are generally more effective and efficient than the generic ones because the former are specifically designed for color image data. Celebi also demonstrates that BKM reduces the MSE produced by various hierarchical

[52] Various CQ researchers incorrectly cite Shafer and Kanade (1987) or Celenk (1990) in the context of BKM.

[53] For example, Wu and Witten (1985) report execution times ranging from 12 to 34 hours for 256×256 images ($K = 256$).

CQ algorithms by an average of 18–50% and, in general, the more ineffective the hierarchical algorithm by itself, the more BKM improves upon it. For example, BKM initialized using median-cut gives, on average, 47% lower MSE than median-cut alone. This comprehensive experiment refutes the claim made by some of the early CQ researchers (Heckbert 1982; Wan et al. 1988, 1990; Orchard and Bouman 1991) that BKM offers *only* a marginal improvement over hierarchical CQ algorithms.

We now turn to the issue of **acceleration**. A naive implementation of Algorithm 2 can take a long time to cluster a large data set. This is because the algorithm computes $NK\ \ell_2^2$ distances per iteration, each of which requires $(2D - 1)$ additions and D multiplications. Hence, we can accelerate BKM by reducing one or more of the following quantities[54]:

- Number of data points (N): In most applications, N is the dominant factor in the execution time, as $N \gg D$ and $N \gg K$. Sampling and weighting are two common ways to reduce N. Note that most algorithms that employ sampling are approximate.
- Number of clusters (K): Given a data point, many accelerated algorithms try to determine its nearest center without computing the distance between the data point and all K centers. In other words, these algorithms aim to compute fewer than NK distances per iteration. Representative algorithms of this kind are described below.
- Cost of distance computations: **Dimensionality reduction** may be used to reduce the number of attributes per data point (D), thereby accelerating the distance computations. Dimensionality reduction may also accelerate convergence and facilitate clustering by eliminating irrelevant, redundant, or noisy attributes. Other ways to accelerate distance computations include numerical approximations, geometric identities, and partial distance elimination (see below). Note that algorithms that employ dimensionality reduction or numerical approximations are usually approximate.
- Number of iterations: We can accelerate convergence by reducing N, D, or K, increasing the convergence threshold (ε), or using a better initialization method. However, *none* of these modifications is guaranteed to accelerate convergence, and each one is likely to affect the final centers generated by BKM.

Numerous accelerated BKM algorithms have been proposed since the 1980s. We first describe two approximate algorithms briefly, and then discuss three exact algorithms in detail.

Huang and Chang (2004) propose the **finite-state k -means algorithm** (FSKM)[55]. The first iteration of FSKM is the same as that of BKM. In each of the subsequent iterations, when searching for the nearest center to a given data point, the algorithm considers *only* the K_f nearest neighbors of the center to which the data point was assigned in the previous iteration. This strategy is faster since the nearest center search is performed in a smaller set of K_f centers rather than the entire set of K centers. Here, K_f is a user-defined number in

[54] In this paper, an **exact accelerated algorithm** refers to an accelerated algorithm that generates the same output as the corresponding naive algorithm when started from identical initial conditions. On the other hand, an **approximate accelerated algorithm** refers to an accelerated algorithm that generates approximately the same output as the corresponding naive algorithm when started from identical initial conditions. We restrict ourselves to sequential acceleration techniques, as opposed to parallel ones, as the former techniques are more appropriate for and common in CQ applications. Note that a sequential acceleration algorithm does *not* necessarily have a lower time complexity than its naive counterpart.

[55] This algorithm was proposed for VQ earlier by Chang et al. (1992).

the range $[1, K]$. If $K_f = K$, FSKM reduces to BKM. Otherwise, the former approximates the latter.

In each iteration (except the first one), FSKM first computes the pairwise ℓ_2^2 distances between the K centers, and then determines the K_f nearest neighbors of each center. If the latter is accomplished using a partition-based selection algorithm such as quickselect, the assignment step of FSKM requires $\mathcal{O}(K^2D + K^2 \min\{K_f, \log K\} + NDK_f)$ time. Comparing this to the $\mathcal{O}(NDK)$ complexity of the assignment step of BKM, we see that FSKM can be faster than BKM *only if* $K_f \ll K$. Huang and Chang report that, for $K = 256$, FSKM with $K_f = 8$ is 16 to 28 times faster than BKM, while obtaining similar results. Unfortunately, the authors do *not* suggest a systematic way to set K_f.

Hu and Lee (2007) propose another approximate BKM algorithm, which we refer to as the **stability-based *k* -means algorithm** (SBKM)[56]. The first T iterations of SBKM are the same as those of BKM. In the subsequent iterations, the clustering process is accelerated based on the concepts of center stability and point activity. More specifically, if a center does *not* move by more than Δ units (as measured by the ℓ_2^2 distance) in two successive iterations, it is classified as stable. Furthermore, data points previously assigned to stable centers are classified as inactive.

In each iteration, *only* unstable centers and active points participate in the clustering process. During the first T iterations, all K centers are considered unstable and all N data points are considered active. As the iterations progress, the number of unstable centers, and thus the number of active data points, decreases rapidly, resulting in increasingly faster iterations. The authors report that SBKM with $T = 10$ and $\Delta = 1$ is 4 to 21 times faster than BKM, while obtaining similar results.

It is important to stress that, unlike their predecessor (BKM), FSKM and SBKM are *not* guaranteed to converge. Therefore, these algorithms should be terminated early, as discussed earlier.

We now turn to exact approaches for accelerating BKM. The simplest way to accelerate BKM is the **partial distance elimination** technique (Cheng et al. 1984; Bei and Gray 1985). Let $\mathbf{x}$ be a data point, $\mathbf{c}$ be a candidate nearest center, and the minimum distance between $\mathbf{x}$ and the previously examined centers be $d_{\min}$ (initialized to ∞). It is easy to see that we can abort the computation of $\|\mathbf{x} - \mathbf{c}\|_2^2$ as soon as the (partial) distance between the first $\hat{d}$ attributes of $\mathbf{x}$ and $\mathbf{c}$ exceeds the current minimum distance, that is, whenever $\sum_{d=1}^{\hat{d}} (x_d - c_d)^2 > d_{\min}$, where $\hat{d} \in \{1, \ldots, D\}$.

Searching for the nearest center to a data point using the standard ℓ_2^2 distance requires $(2D - 1)$ additions and D multiplications, whereas the above elimination technique requires $2\hat{d}$ additions, $\hat{d}$ multiplications, and $\hat{d}$ comparisons. Therefore, this technique is advantageous *only if* $\hat{d} \ll D$, which is often the case when D is reasonably large. Otherwise, the time saved by the early termination of the full distance computation is offset by the additional time spent on comparisons. In our experience, for low-dimensional data such as color image data, the cost of additional comparisons often renders this technique slower than direct computation.

The partial distance elimination technique aims to accelerate BKM by reducing the *cost* of distance computations. By contrast, most accelerated BKM algorithms aim to reduce the *number* of distance computations. Consider a data point $\mathbf{x}$, two centers $\mathbf{c}_i$ and $\mathbf{c}_{\hat{\imath}}$, and a distance metric $d(\cdot, \cdot)$. By the **triangle inequality**, we have $d(\mathbf{c}_i, \mathbf{c}_{\hat{\imath}}) \leq d(\mathbf{x}, \mathbf{c}_i) + d(\mathbf{x}, \mathbf{c}_{\hat{\imath}})$.

[56] An exact version of this algorithm was proposed earlier by Kaukoranta et al. (2000).

Therefore, if we know that $2d(\mathbf{x}, \mathbf{c}_i) \leq d(\mathbf{c}_i, \mathbf{c}_{\hat{\imath}})$, we can conclude that $d(\mathbf{x}, \mathbf{c}_i) \leq d(\mathbf{x}, \mathbf{c}_{\hat{\imath}})$ without having to compute $d(\mathbf{x}, \mathbf{c}_{\hat{\imath}})$. Note that for $d = \ell_2^2$, the (relaxed) triangle inequality test becomes $4\|\mathbf{x} - \mathbf{c}_i\|_2^2 \leq \|\mathbf{c}_i - \mathbf{c}_{\hat{\imath}}\|_2^2$. In an efficient implementation, we can avoid repeated multiplications by 4 by premultiplying the pairwise ℓ_2^2 distances between the centers by 1/4.

The **triangle inequality elimination algorithm** (TIE) (Chen and Pan 1989) precomputes the pairwise ℓ_2^2 distances between the centers at the beginning of each iteration. The $(K-1)$ distances associated with each center are then sorted in ascending order. Now, consider a data point $\mathbf{x}$ that was assigned to cluster $\mathcal{P}_k$ in the previous iteration[57]. We compare $\mathbf{x}$ against the centers in ascending order of their distances from center $\mathbf{c}_k$ using the triangle inequality test. If the test fails, we compute the ℓ_2^2 distance between $\mathbf{x}$ and the current center under consideration; otherwise, we abort the search because we know that the test will succeed for the remaining centers in the sorted list.

The pseudocode for TIE is given in Algorithm 3. In the pseudocode, $\mathbf{d}_i$ is an array of distances associated with center $\mathbf{c}_i$ (e.g., upon sorting, $d_{i\hat{\imath}}$ denotes $\mathbf{c}_i$'s distance to its $\hat{\imath}$th nearest center.) On line 15, the function `sort_index(`$\mathbf{d}_i$`)` sorts the input array $\mathbf{d}_i$ in ascending order, and returns an array $\mathbf{p}_i$ that contains the indices of the sorted values in the unsorted input array, that is, p_{i1} is the index of the smallest value in $\mathbf{d}_i$, p_{i2} is the index of the second smallest value in $\mathbf{d}_i$, and so on.

On lines 28–30 of Algorithm 3, we declare the candidate center $\mathbf{c}_{\hat{\imath}}$ to be the nearest center to $\mathbf{x}_j$ in one of two cases: (i) $\mathbf{c}_{\hat{\imath}}$ is closer to $\mathbf{x}_j$ than its current nearest center; or (ii) $\mathbf{c}_{\hat{\imath}}$ has the same distance to $\mathbf{x}_j$ as its current nearest center, but the index of the former center is less than that of the latter one. Strictly speaking, the second test is *not* necessary, but, as discussed earlier in this section, if there are multiple centers equidistant to the query data point, the center with the smallest index is conventionally taken as the nearest center. Therefore, without the second test, the TIE algorithm may *not* give identical results to a standard BKM implementation.

Another popular accelerated BKM algorithm is the **mean-ordered search algorithm** (MOS) (Ra and Kim 1993), which is based on the idea that two data points can be matched approximately based on their means. Let $\mathbf{x}$ be a data point and $\mathbf{c}$ be a candidate nearest center. The MOS algorithm uses the following inequality

$$D\left(m_{\mathbf{x}} - m_{\mathbf{c}}\right)^2 \leq \|\mathbf{x} - \mathbf{c}\|_2^2, \tag{17}$$

where $m_{\mathbf{x}} = \frac{1}{D}\sum_{d=1}^{D} x_d$ and $m_{\mathbf{c}} = \frac{1}{D}\sum_{d=1}^{D} c_d$ are the means of $\mathbf{x}$ and $\mathbf{c}$, respectively. Consequently, if $\mathbf{c}$ satisfies

$$d_{\min} < D\left(m_{\mathbf{x}} - m_{\mathbf{c}}\right)^2, \tag{18}$$

where $d_{\min}$ is the ℓ_2^2 distance between $\mathbf{x}$ and its current nearest center, then $\mathbf{c}$ can be eliminated from consideration without having to compute $\|\mathbf{x} - \mathbf{c}\|_2^2$. This is because inequalities (17) and (18) imply $d_{\min} < \|\mathbf{x} - \mathbf{c}\|_2^2$, which means that $\mathbf{c}$ *cannot* possibly be closer to $\mathbf{x}$ than its current nearest center. The computation of the right side of (18) is cheaper than that of $\|\mathbf{x} - \mathbf{c}\|_2^2$ provided that the means are precomputed; the former requires one addition and two multiplications, whereas the latter requires $(2D - 1)$ additions and D multiplications.

[57] We can arbitrarily assign all data points to $\mathcal{P}_1$ before the first iteration.

The means of the data points can be computed before clustering, whereas those of the centers can be computed at the beginning of each iteration.

Inequality (17) alone is insufficient to design an efficient search algorithm. We also need to start the search from a center that is likely to be the nearest center to $\mathbf{x}$ (this way, we can ensure that the lower bound $d_{\min}$ in (18) is small, which allows us to eliminate a large number of candidate centers). Fortunately, there is a simple solution: we can sort the centers based on their means (or, equivalently, their sums), and locate the center that has the nearest mean to $\mathbf{x}$'s using binary search. Let $\mathbf{c}_i$ be this tentative match. Starting from $\mathbf{c}_i$, we can then traverse the centers in "up" and "down" fashion until we locate the nearest center. An additional benefit of this search scheme is that once we encounter a center that satisfies (18), we can terminate the search in the current direction since the centers further toward the current extreme end *cannot* possibly be closer to $\mathbf{x}$ than its current nearest center. In other words, if we are going in the up direction and $\mathbf{c}_{i-1}$ satisfies (18), we can eliminate centers $\mathbf{c}_{i-1}, \dots, \mathbf{c}_1$. Similarly, if we are going in the down direction and $\mathbf{c}_{i+1}$ satisfies (18), we can eliminate centers $\mathbf{c}_{i+1}, \dots, \mathbf{c}_K$.

The pseudocode for MOS is given in Algorithm 4. For efficiency reasons, we represent each data point/center by its sum, rather than its mean (the sums of $\mathbf{x}_j$ and $\mathbf{c}_i$ are denoted by s_j^p and s_i^c, respectively.) The function `bsearch`$(\{s_1^c, \dots, s_K^c\}, s_j^p)$ finds the index of the center with the nearest sum to $\mathbf{x}_j$'s using binary search. The purpose of the `sort_index` function was explained earlier in the description of the TIE algorithm. Let $\mathbf{c}_i$ be the tentative match for $\mathbf{x}_j$. We traverse the sorted set of centers in the following order: c_{i+1}, c_{i-1}, c_{i+2}, c_{i-2}, and so on.

It should be noted that more elaborate approaches for accelerating BKM have been proposed in the literature (Hamerly and Drake 2015; Wang et al. 2020). These include algorithms based on k-d trees (Kanungo et al. 2002; Lai and Liaw 2008) and more sophisticated uses of the triangle inequality (Elkan 2003). Some of these algorithms (Elkan 2003) are *not* suitable for low-dimensional data such as color image data, as they incur significant overhead to create and update auxiliary data structures. Others (Kanungo et al. 2002; Lai and Liaw 2008) provide acceleration comparable to TIE and MOS, but are difficult to understand or implement. In contrast, TIE and MOS are conceptually simple, relatively easy to implement, and incur negligible computational overhead, making them ideal candidates for CQ.

Several exact accelerated BKM algorithms for CQ have been proposed in the literature. Kasuga et al. (2000) first use a divisive clustering algorithm to obtain K initial centers, and then form K_m macroclusters around these centers. When searching for the nearest center to a data point $\mathbf{x}$, the algorithm computes distances from $\mathbf{x}$ to *only* these K_m macroclusters and to certain centers in specific macroclusters, which are determined using the triangle inequality. For $K = 256$, the authors recommend $K_m = 2\sqrt{K}$, and estimate the theoretical speed-up of their algorithm to be roughly $\sqrt{K}$. However, they report a modest 4-fold speed-up.

Hu and Su (2008a); Hu et al. (2009) propose an algorithm that combines TIE with MOS. The authors report 2.5- to 14-fold speed-up over BKM. Celebi (2009, 2011) presents a weighted TIE algorithm. The author reports a maximum of 392-fold speed-up over BKM (with an average of 62-fold). More recently, Huang (2021) proposed an algorithm based on a variant of MOS.

Numerous elimination conditions such as those used in TIE and MOS have been proposed in the literature (Xie et al. 2016; Wang et al. 2018). In general, the more conditions used in an accelerated BKM algorithm, the larger the reduction in the number of distance computations. However, using an excessive number of conditions to minimize the search space aggressively may take more time than computing ℓ_2^2 distances directly, especially in low dimensions.

```
input  : $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\} \subset \mathbb{R}^D$, $K$
output: $\mathcal{C} = \{\mathbf{c}_1, \ldots, \mathbf{c}_K\} \subset \mathbb{R}^D$
1  for $j \leftarrow 1$ to $N$ do
2      $l_j \leftarrow 1$   ▷ index of $\mathbf{x}_j$'s nearest center
3  end
4  while termination criterion is not met do
5      for $i \leftarrow 1$ to $K$ do
6          $n_i \leftarrow 0$   ▷ size of $\mathcal{P}_i$
7          $\hat{\mathbf{c}}_i \leftarrow \mathbf{0}$   ▷ vector sum of $\mathcal{P}_i$
8      end
9      Compute pairwise $\ell_2^2$ distances between the centers
10     for $i \leftarrow 1$ to $K$ do
11         $d_{ii} \leftarrow 0$
12         for $\hat{\imath} \leftarrow i + 1$ to $K$ do
13             $d_{i\hat{\imath}} \leftarrow d_{\hat{\imath}i} \leftarrow (1/4)\,\|\mathbf{c}_i - \mathbf{c}_{\hat{\imath}}\|_2^2$
14         end
15         $\mathbf{p}_i \leftarrow$ sort_index($\mathbf{d}_i$)
16     end
17     Assign each data point to the nearest center
18     for $j \leftarrow 1$ to $N$ do
19         $k \leftarrow l_j$   ▷ $\mathbf{x}_j$'s current nearest center
20         $d_{\min} \leftarrow \|\mathbf{x}_j - \mathbf{c}_k\|_2^2$   ▷ distance to nearest center
21         Update nearest center, if necessary
22         for $i \leftarrow 2$ to $K$ do
23             if $d_{min} < d_{ki}$ then break   ▷ abort search
24             $\hat{\imath} \leftarrow p_{ki}$   ▷ $\mathbf{c}_{\hat{\imath}}$ may be closer to $\mathbf{x}_j$ than $\mathbf{c}_k$
25             $d_{\text{tmp}} \leftarrow \|\mathbf{x}_j - \mathbf{c}_{\hat{\imath}}\|_2^2$
26             if $(d_{tmp} < d_{min}) \vee ((d_{tmp} = d_{min}) \wedge (\hat{\imath} < k))$ then
27                 $\mathbf{c}_{\hat{\imath}}$ is closer to $\mathbf{x}_j$ than $\mathbf{c}_k$
28                 $d_{\min} \leftarrow d_{\text{tmp}}$   ▷ distance to nearest center
29                 $k \leftarrow \hat{\imath}$   ▷ index of nearest center
30                 $i \leftarrow 1$   ▷ reset search
31             end
32         end
33         $l_j \leftarrow k$   ▷ original index of nearest center
34         $n_k \leftarrow n_k + 1$
35         $\hat{\mathbf{c}}_k \leftarrow \hat{\mathbf{c}}_k + \mathbf{x}_j$
36     end
37     for $i \leftarrow 1$ to $K$ do
38         $\mathbf{c}_i \leftarrow \hat{\mathbf{c}}_i / n_i$   ▷ recompute $\mathbf{c}_i$
39     end
40 end
```

Algorithm 3: The TIE Algorithm

```
input  : $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\} \subset \mathbb{R}^D$, $K$
output: $\mathcal{C} = \{\mathbf{c}_1, \dots, \mathbf{c}_K\} \subset \mathbb{R}^D$
for $j \leftarrow 1$ to $N$ do
    $s_j^p \leftarrow \sum_{d=1}^{D} x_{jd}$    ▷ sum of $\mathbf{x}_j$
end
while termination criterion is not met do
    for $i \leftarrow 1$ to $K$ do
        $n_i \leftarrow 0$    ▷ size of $\mathcal{P}_i$
        $\hat{\mathbf{c}}_i \leftarrow \mathbf{0}$    ▷ vector sum of $\mathcal{P}_i$
        $s_i^c \leftarrow \sum_{d=1}^{D} c_{id}$    ▷ sum of $\mathbf{c}_i$
    end
    $\mathbf{p} \leftarrow$ sort_index($\{s_1^c, \dots, s_K^c\}$)
    Assign each data point to the nearest center
    for $j \leftarrow 1$ to $N$ do
        $d \leftarrow u \leftarrow i \leftarrow$ bsearch($\{s_1^c, \dots, s_K^c\}, s_j^p$)    ▷ tentative match
        $(d_{\min}, \text{down}, \text{up}) \leftarrow \left(\|\mathbf{x}_j - \mathbf{c}_i\|_2^2, \text{true}, \text{true}\right)$
        repeat
            if down then
                $d \leftarrow d + 1$    ▷ down index
                if $(d = K + 1) \vee \left((s_j^p - s_d^c)^2 > D d_{min}\right)$ then
                  down $\leftarrow$ false
                else
                    $d_{\text{tmp}} \leftarrow \|\mathbf{x}_j - \mathbf{c}_d\|_2^2$
                    if $(d_{tmp} < d_{min}) \vee ((d_{tmp} = d_{min}) \wedge (p_d < p_i))$ then
                      $(d_{\min}, i) \leftarrow (d_{\text{tmp}}, d)$
                end
            end
            if up then
                $u \leftarrow u - 1$    ▷ up index
                if $(u = 0) \vee \left((s_j^p - s_u^c)^2 > D d_{min}\right)$ then  up $\leftarrow$ false
                else
                    $d_{\text{tmp}} \leftarrow \|\mathbf{x}_j - \mathbf{c}_u\|_2^2$
                    if $(d_{tmp} < d_{min}) \vee ((d_{tmp} = d_{min}) \wedge (p_u < p_i))$ then
                      $(d_{\min}, i) \leftarrow (d_{\text{tmp}}, u)$
                end
            end
        until down $\vee$ up
        $k \leftarrow p_i$    ▷ original index of nearest center
        $n_k \leftarrow n_k + 1$
        $\hat{\mathbf{c}}_k \leftarrow \hat{\mathbf{c}}_k + \mathbf{x}_j$
    end
    for $i \leftarrow 1$ to $K$ do
        $\mathbf{c}_i \leftarrow \hat{\mathbf{c}}_i / n_i$    ▷ recompute $\mathbf{c}_i$
    end
end
```

Algorithm 4: The MOS Algorithm

Before moving on to other KM variants, we make a few remarks on the BKM algorithm and its accelerated variants:

- BKM does *not* require an unbounded number of iterations to converge (Equitz 1989). It can be shown that for a 24-bit color image with N pixels, BKM requires at most $3 \cdot 256^2 \cdot N^5$ iterations (Har-Peled and Sadri 2005). However, BKM converges much faster in practice, requiring *only* a few hundred iterations for typical color images[58] (Thompson et al. 2020). Furthermore, since most of the progress is made in earlier iterations, the algorithm can be terminated early.
- Starting BKM with a more effective initializer does *not* guarantee a better SSE or faster convergence. However, there are initialization methods that generally outperform others in both respects (Celebi et al. 2013).
- It can be misleading to compare the efficiency of initialization methods based on the number of BKM iterations they lead to. For example, we can use an elaborate and computationally expensive initializer to achieve faster convergence (Celebi 2011).
- An approximate BKM algorithm does *not* necessarily converge or, if it does, it does *not* do so faster than standard BKM (Huang and Chang 2004).
- An approximate BKM algorithm does *not* necessarily perform worse than standard BKM in terms of SSE (Huang and Chang 2004).
- Most acceleration techniques for BKM involve geometric inequalities that can lead to numerical instabilities when implemented using finite-precision arithmetic.
- Sampling the input image does *not* guarantee a faster convergence because while each iteration will be faster, there may be more of them.
- An accelerated BKM algorithm does *not* necessarily run faster than a naive one. Most accelerated algorithms pay off *only if* K is sufficiently large.
- It can be misleading to compare the efficiency of accelerated BKM algorithms based on the number of distance computations they require. This is because an accelerated algorithm may perform multiple tests to reduce the number of distance computations drastically at the cost of significant computational overhead.

MacQueen (1967) proposes an online formulation of the BKM algorithm. The two KM algorithms differ in when and how they update the cluster centers. The **online** k **-means algorithm** (OKM) updates the *nearest* center immediately after the presentation of *each* data point, whereas BKM recomputes *all* centers after the presentation of the *entire* set of data points.

The pseudocode for OKM is given in Algorithm 5. Here, $\texttt{rand}(l, u)$ is a uniform random number generator that returns an integer between l and u (inclusive), n_i denotes the number of data points assigned to $\mathbf{c}_i$ (or, equivalently, the number of times $\mathbf{c}_i$ has been updated), and $P \geq 0$ denotes the user-defined exponent of the learning rate, which is explained below. With MacQueen's choice of $P = 1$, the combination of statements $n_i \leftarrow n_i + 1$ and $\mathbf{c}_i \leftarrow \mathbf{c}_i + n_i^{-P}(\mathbf{x}_r - \mathbf{c}_i)$ reduces to $\mathbf{c}_i \leftarrow (n_i\mathbf{c}_i + \mathbf{x}_r)/(n_i + 1)$, which ensures that $\mathbf{c}_i$ represents the mean of all data points assigned to it.

Assuming a single pass over $\mathcal{X}$, that is, N presentations, the time complexity of OKM is dominated by the nearest center search operations, which take $\mathcal{O}(NDK)$ time. This complexity is identical to that of BKM. However, in terms of the actual number of arithmetic

[58] In general, the exact number of iterations depends on the number, dimensionality, and distribution of the data points, the number of clusters sought, and the initial centers.

operations, an OKM pass costs slightly more than a BKM pass due to the additional random number generation and learning rate computation operations. It is important to notice that the *total* computational requirements of the two algorithms are often substantially different. This is because, in practice, the online algorithm is often terminated after one pass or, at most, a few passes, whereas the batch algorithm converges after many passes[59]. Therefore, the online algorithm can be significantly faster than the batch one (Thompson et al. 2020; Abernathy and Celebi 2022).

```
input  : $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\} \subset \mathbb{R}^D$, $K$, $P$
output: $\mathcal{C} = \{\mathbf{c}_1, \ldots, \mathbf{c}_K\} \subset \mathbb{R}^D$
1  for $i \leftarrow 1$ to $K$ do
2      $n_i \leftarrow 0$    ▷ # data points assigned to $\mathbf{c}_i$
3  end
4  while termination criterion is not met do
5      for $j \leftarrow 1$ to $N$ do
6          $r \leftarrow \texttt{rand}(1, N)$    ▷ select a random data point from $\mathcal{X}$
7          $i \leftarrow \underset{\hat{\imath} \in \{1,\ldots,K\}}{\arg\min} \|\mathbf{x}_r - \mathbf{c}_{\hat{\imath}}\|_2^2$    ▷ $\mathbf{c}_i$ is nearest to $\mathbf{x}_r$
8          $n_i \leftarrow n_i + 1$
9          $\mathbf{c}_i \leftarrow \mathbf{c}_i + n_i^{-P}(\mathbf{x}_r - \mathbf{c}_i)$    ▷ update $\mathbf{c}_i$
10     end
11 end
```
Algorithm 5: The OKM Algorithm

Unlike its batch counterpart, OKM traverses the data points randomly. This random presentation reduces the sensitivity of the algorithm to the order in which the data points are processed. In this regard, random presentation has been shown to be superior to cyclic presentation, that is, $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N, \mathbf{x}_1, \mathbf{x}_2, \ldots$ (Mulier and Cherkassky 1995; Bermejo and Cabestany 2002). In fact, cyclic presentation is likely to introduce bias into the learning procedure, especially in redundant data such as color image data.

The OKM algorithm can also be viewed as an instance of the **competitive learning** (CL) paradigm (Rumelhart and Zipser 1985; Grossberg 1987), which is a form of **unsupervised learning** (Celebi and Aydin 2016) closely related to neural networks (Ahalt et al. 1990). In a basic CL algorithm, we have a randomly distributed set of (neural) units that compete for the right to respond to a given subset of inputs (Rumelhart and Zipser 1985). Whenever an input is presented, the most closely matching unit (typically in the ℓ_2 sense) is declared the winner, and moved toward the input. Since *only* the winner is adapted, this learning paradigm is termed **hard competitive learning** (or **winner-take-all learning**)[60].

Let $\mathbf{x}(t)$ be the input at time t ($t = 1, 2, \ldots$) and $\mathbf{c}(t)$ be the winner (with respect to the ℓ_2 distance). The adaptation equation for $\mathbf{c}(t)$ is given by

$$\mathbf{c}(t) = \mathbf{c}(t-1) + \eta(t)(\mathbf{x}(t) - \mathbf{c}(t-1)), \tag{19}$$

where $\eta \in [0, 1]$ is the **learning rate**, which is typically a monotonically decreasing function of time. The larger the η value, the more emphasis is given to the new input, and hence

[59] Thompson et al. (2020) show that a single-pass variant of BKM is faster than OKM, but performs very poorly.

[60] For a discussion of the alternative soft competitive learning (or winner-take-most learning) paradigm, see Subsection 5.5.

the faster the learning. However, very large values of η may prevent the algorithm from converging. In general, η is selected to satisfy the Robbins–Monro conditions (Robbins and Monro 1951) given by

$$\lim_{t \to \infty} \eta(t) = 0, \tag{20a}$$

$$\sum_{t=1}^{\infty} \eta(t) = \infty, \tag{20b}$$

$$\sum_{t=1}^{\infty} \eta(t)^2 < \infty. \tag{20c}$$

These conditions ensure that the learning rate decreases fast enough to suppress the noise, but not too fast to avoid premature convergence. Under mild regularity conditions, this algorithm converges almost surely to a local minimum (Bottou 1998).

Rearranging the terms in Eq. (19), we obtain

$$\mathbf{c}(t) = \eta(t)\mathbf{x}(t) + (1 - \eta(t))\mathbf{c}(t - 1), \tag{21}$$

which indicates that the new center $\mathbf{c}(t)$ is a convex combination of the old center $\mathbf{c}(t-1)$ and the input $\mathbf{x}(t)$. In other words, values of η in [0, 1] move $\mathbf{c}$ along the line segment joining $\mathbf{c}$ and $\mathbf{x}$.

The choice $\eta(t) = t^{-P}$ with $P \in (0.5, 1]$ guarantees converge. The value $P = 1$ gives us the most popular rate in the stochastic approximation literature (i.e., the **harmonic rate**), $\eta(t) = 1/t$, which coincides with MacQueen's choice. This rate leads to the fastest asymptotic convergence for $K = 1$ (Yair et al. 1992). In practice, data sets are finite ($N < \infty$), and we seek multiple clusters ($K > 1$). In addition, the clustering problem is nonstationary because the subsets of data that fall into each cluster change through the iterations (Darken and Moody 1990). Therefore, the asymptotically optimal harmonic rate is often suboptimal in practice. More specifically, $\eta(t) = 1/t$ decays too rapidly for $K > 1$. Multiple studies (Darken and Moody 1990; Wu and Yang 2006; Thompson et al. 2020) recommend the so-called **square root rate**, $\eta(t) = 1/\sqrt{t}$, instead.

It is interesting to note that, while there are numerous exact accelerated BKM algorithms (Hamerly and Drake 2015; Wang et al. 2020), to the best of our knowledge, there is *no* exact accelerated OKM algorithm. This is because it is difficult to accelerate OKM in an exact manner due to the frequent center updates. In fact, online algorithms such as OKM are often accelerated (approximately) through better learning rate schedules. Fortunately, with an appropriate schedule, a one-pass OKM implementation can be orders of magnitude faster than its many-pass batch counterpart (Thompson et al. 2020; Abernathy and Celebi 2022). Thus, there is little need to accelerate OKM algorithmically.

Thompson et al. (2020) apply the OKM algorithm (with maximin initialization and square root learning rate) to the CQ problem. The authors demonstrate that, compared to the batch algorithm (with maximin initialization), the (one-pass) online algorithm is easier to implement, 41 to 300 times faster, and gives very similar results.

Abernathy and Celebi (2022) improve upon Thompson et al. (2020)'s OKM algorithm by integrating the initialization method into the clustering algorithm. Their **incremental online** k **-means clustering algorithm** (IOKM) begins by setting $\mathbf{c}_0$ to the centroid $\bar{\mathbf{x}}$ of $\mathcal{X}$ (for $K = 1$, this choice of $\mathbf{c}_0$ is clearly optimal). It then adds 2^{t+1} new centers in iteration t

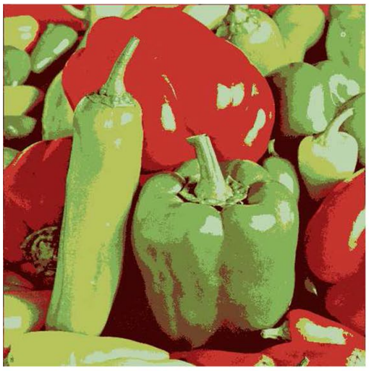
($t \in \{0, \ldots, \log_2 K - 1\}$) by splitting each of the existing centers into two. When a center $\mathbf{c}_k$ is split, the left child inherits its parent's attributes (i.e., $\mathbf{c}_{2k+1} = \mathbf{c}_k$), whereas the right child becomes a slightly perturbed version of its parent (i.e., $\mathbf{c}_{2k+2} = \mathbf{c}_k + \boldsymbol{\varepsilon}$ where $\boldsymbol{\varepsilon}$ is an arbitrary vector of small positive ℓ_2 norm). The set of final K centers is then taken as the palette. Note that in the description above, we assumed that K is a power of two for simplicity. If this is *not* the case, we perform $\lfloor \log_2 K \rfloor$ iterations as described above, and then perform one last iteration in which we split *only* $K - 2^{\lfloor \log_2 K \rfloor}$ of the centers from the previous iteration. The authors demonstrate that their initialization-free IOKM algorithm (with $\boldsymbol{\varepsilon} = \mathbf{0}$ and square root learning rate) is easier to implement and more effective than its predecessor, OKM.

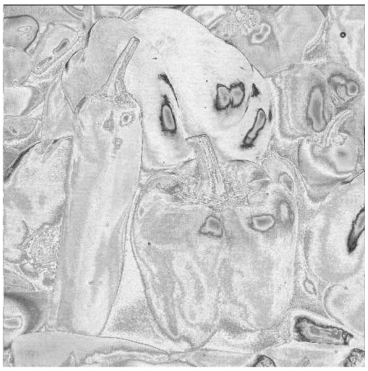
The simple CL algorithm described above sometimes encounters the so-called **dead unit problem**, where certain units never win the competition and hence never learn, typically because of poor initialization. From a clustering perspective, such units represent empty clusters that do *not* contribute to the clustering quality. Uchiyama and Arbib (1994a, 1994b) propose the **adaptive distributing units algorithm** (ADU), a modified CL algorithm that alleviates the dead unit problem. ADU starts with a single unit, whose center is given by the centroid $\bar{\mathbf{x}}$ of $\mathcal{X}$. In each iteration, a data point is selected uniformly at random from $\mathcal{X}$, and the nearest unit (with respect to the ℓ_2 distance) is declared the winner. This unit is then updated by moving its center closer to the data point, and incrementing its win count. New units are added by splitting existing units that win a user-specified number of times, until the number of units reaches K. This splitting policy prevents the formation of dead units by preventing certain units from monopolizing the competition. Note that when a new unit is generated by duplicating an existing unit, the two units are temporarily colocated. Whenever a data point to which these two units are the nearest is presented, one of these units becomes the winner and gets updated, while the other remains unchanged. Hence, the problem of two identical units moving together *cannot* occur.

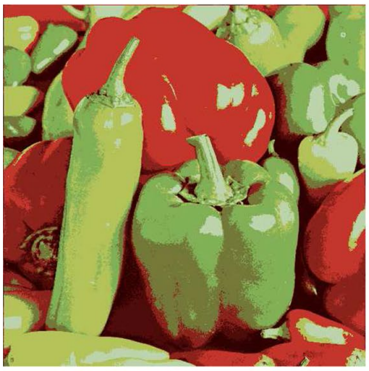
The pseudocode for ADU is given in Algorithm 6. The symbols $T_{\max}$ and $N_{\max}$ denote the maximum number of iterations (or presentations) and the win count threshold, respectively. It can be shown that $T_{\max} \geq (2K - 3)N_{\max}$ guarantees that K units are generated upon termination (Uchiyama and Arbib 1994b). Unlike most partitional algorithms described in this section, ADU does *not* require initialization, which makes the algorithm even easier to implement than OKM. However, like OKM, ADU is somewhat sensitive to the order in which the data points are processed.

Celebi et al. (2014) adapt ADU to the CQ problem. The authors report competitive results with BKM using the following parameter values: $N_{\max} = 400\sqrt{K}$, $T_{\max} = (2K - 3)(K + 7)N_{\max}$, and $\eta = 0.015$. Note that with a constant learning rate $\eta = C$, a CL algorithm converges to a neighborhood of a solution whose size is proportional to C. We can reduce the size of this neighborhood, and thus the variance of the solutions, by reducing C. This is why η should be a small positive number.

(a) Peppers (183, 525 colors)


(b) Weighted TIE (8 colors)


(c) Error image for 12(b)


(d) ADU (8 colors)

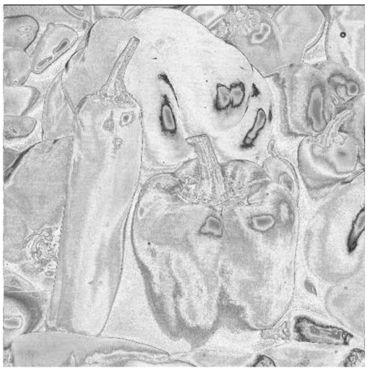

(e) Error image for 12(d)

Fig. 12 Comparison of KM-based CQ algorithms

```
input  : $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\} \subset \mathbb{R}^D$, $K$, $N_{\max}$, $T_{\max}$, $\eta$
output: $\mathcal{C} = \{\mathbf{c}_1, \dots, \mathbf{c}_K\} \subset \mathbb{R}^D$
$k \leftarrow 1$   ▷ # units
$n_1 \leftarrow 0$   ▷ first unit's win count
$\mathbf{c}_1 \leftarrow \bar{\mathbf{x}}$   ▷ first unit's center
for $t \leftarrow 1$ to $T_{max}$ do
    $r \leftarrow \mathtt{rand}(1, N)$   ▷ select a random data point from $\mathcal{X}$
    $i \leftarrow \underset{\hat{\imath} \in \{1,\dots,K\}}{\arg\min} \|\mathbf{x}_r - \mathbf{c}_{\hat{\imath}}\|_2^2$   ▷ $\mathbf{c}_i$ is nearest to $\mathbf{x}_r$
    $n_i \leftarrow n_i + 1$   ▷ update $\mathbf{c}_i$'s win count
    $\mathbf{c}_i \leftarrow \mathbf{c}_i + \eta(\mathbf{x}_r - \mathbf{c}_i)$   ▷ update $\mathbf{c}_i$
    if $(n_i = N_{max}) \wedge (k < K)$ then
        Split unit $\mathbf{c}_i$
        $k \leftarrow k + 1$
        $(n_k, n_i) \leftarrow (0, 0)$
        $\mathbf{c}_k \leftarrow \mathbf{c}_i$
    end
end
```

Algorithm 6: The ADU Algorithm

Figure 12 compares two KM variants, weighted TIE (Celebi 2011) and ADU (Celebi et al. 2014), on the Peppers image. It can be seen that the error image for weighted TIE is slightly cleaner than that for ADU, and both algorithms produce significantly better results than the non-KM-based CQ algorithms described so far.

5.4 The fuzzy *C*-means algorithm

Given a data set $\mathcal{X} = \left\{\mathbf{x}_1, \dots, \mathbf{x}_N\right\} \subset \mathbb{R}^D$, a real matrix $\mathscr{U} = [u_{ij}]_{K \times N}$ represents a **hard** K **-partition** of $\mathcal{X}$ if its elements satisfy three conditions (Bezdek 1981)

$$u_{ij} \in \{0, 1\} \quad i \in \{1, \dots, K\}, j \in \{1, \dots, N\}, \tag{22a}$$

$$\sum_{i=1}^{K} u_{ij} = 1 \quad j \in \{1, \dots, N\}, \tag{22b}$$

$$0 < \sum_{j=1}^{N} u_{ij} < N \quad i \in \{1, \dots, K\}. \tag{22c}$$

The ith row of $\mathscr{U}$, i.e., $\mathcal{U}_i = \left(u_{i1}, \dots, u_{iN}\right)$, represents the characteristic function of cluster $\mathcal{P}_i$: $u_{ij} = 1$ if $\mathbf{x}_j \in \mathcal{P}_i$, and 0 otherwise; condition (22b) means that each $\mathbf{x}_j$ belongs to *exactly* one of the K clusters; condition (22c) means that *no* cluster is empty and *no* cluster is all of $\mathcal{X}$, that is, $K \in \{2, \dots, N\}$. For obvious reasons, $\mathscr{U}$ is called a **partition matrix** (or **membership matrix**). Note that in the fuzzy clustering literature, the symbol C is often used to denote the number of clusters. However, in the following discussion, we continue to use the symbol K for consistency.

The concept of hard K-partition can be generalized by relaxing condition (22a) as $u_{ij} \in [0, 1]$, in which case the partition matrix $\mathscr{U}$ is said to represent a **fuzzy K -partition** of $\mathcal{X}$. In such a partition matrix, the total membership of each $\mathbf{x}_j$ is still 1, but since $u_{ij} \in [0, 1]$ for all i, $\mathbf{x}_j$ may have an arbitrary distribution of membership among the K fuzzy clusters.

The **fuzzy c -means algorithm** (FCM) is a generalization of BKM in which data points can belong to multiple clusters (Bezdek 1981). The algorithm aims to generate an optimal fuzzy K-partition of $\mathcal{X}$ by minimizing

$$J_{\text{FCM}} = \sum_{j=1}^{N} \sum_{i=1}^{K} u_{ij}^{M} \left\| \mathbf{x}_j - \mathbf{c}_i \right\|_2^2, \tag{23}$$

where $M \in [1, \infty)$ is a user-defined **weighting exponent** that controls the degree of membership sharing among the fuzzy clusters. In many applications, this weighting exponent is taken in the range [1.1, 5], with $M = 2$ an overwhelming favorite (Bezdek et al. 1999, p. 34). Note that $M = 2$ is computationally advantageous, as this is the *only* value that leads to an integer exponent in the partition matrix update equation given next.

Like its hard counterpart, FCM is based on alternating minimization. In each iteration, we first update the cluster memberships as follows

$$u_{ij} = \left[\sum_{\hat{\imath}=1}^{K} \left(\frac{\left\| \mathbf{x}_j - \mathbf{c}_i \right\|_2^2}{\left\| \mathbf{x}_j - \mathbf{c}_{\hat{\imath}} \right\|_2^2} \right)^{1/(M-1)} \right]^{-1}, \tag{24}$$

for all i and j.

Based on the updated memberships, we then update center $\mathbf{c}_i$ (for all i) as follows

$$\mathbf{c}_i = \frac{1}{\sum_{j=1}^{N} u_{ij}^{M}} \sum_{j=1}^{N} u_{ij}^{M} \mathbf{x}_j, \tag{25}$$

which is the weighted centroid of $\mathcal{X}$. These iterations continue until termination criterion (15) is satisfied.

The larger the M value, the fuzzier the membership assignments. As $M \overset{+}{\to} 1$, FCM converges to a BKM solution. Conversely, as $M \to \infty$ it can be shown that $u_{ij} \to 1/K$ for all i and j, so $\mathbf{c}_i \to \bar{\mathbf{x}}$ (for all i), the centroid of $\mathcal{X}$.

The pseudocode for FCM is given in Algorithm 7[61]. A naive implementation of FCM requires $\mathcal{O}(NDK^2)$ time per iteration, which is quadratic in K. The pseudocode gives an $\mathcal{O}(NDK)$ formulation due to Kolen and Hutcheson (2002).

FCM can be extended to deal with weighted data $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\} \subset \mathbb{R}^D$, where each data point $\mathbf{x}_j$ has a nonnegative weight w_j associated with it. Without loss of generality, we assume that the weights add up to one, that is, $\sum_{j=1}^{N} w_j = 1$. The objective function for the **weighted fuzzy c -means problem** is identical to that for the unweighted problem, with the exception that the distances are weighted. The optimal center $\mathbf{c}_i$ for $\mathcal{P}_i$ is given by the weighted centroid of $\mathcal{X}$, that is,

[61] For conciseness, we omit the handling of exceptional cases such as empty clusters or data points coincident with centers.

$$\mathbf{c}_i = \frac{1}{\sum_{j=1}^{N} w_j u_{ij}^M} \sum_{j=1}^{N} w_j u_{ij}^M \mathbf{x}_j. \tag{26}$$

The pseudocode for the **weighted fuzzy** c **-means algorithm** can be obtained by making two simple modifications to Algorithm 7: (1) $n_i \leftarrow n_i + m$ should be $n_i \leftarrow n_i + w_j m$, and (2) $\hat{\mathbf{c}}_i \leftarrow \hat{\mathbf{c}}_i + m\mathbf{x}_j$ should be $\hat{\mathbf{c}}_i \leftarrow \hat{\mathbf{c}}_i + w_j m\mathbf{x}_j$.

Despite its linear time complexity, FCM can be slow when dealing with large data sets. Various accelerated FCM algorithms have been proposed since the 1980s. Nearly all of these algorithms are inexact, as they involve numerical approximations (Cannon et al. 1986; Höppner 2002) or sampling (followed by optional weighting) (Cheng et al. 1998; Pal and Bezdek 2002; Eschrich et al. 2003; Hathaway and Bezdek 2006; Hathaway and Hu 2009; Parker and Hall 2014). Furthermore, most of these algorithms attain *only* modest (e.g., 2- to 6-fold) speed-ups. It is important to emphasize that FCM is more difficult to accelerate than BKM because the former algorithm involves soft (or partial) memberships, whereas the latter one involves hard (or binary) memberships.

```
input : $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\} \subset \mathbb{R}^D$, $K$, $M$
output: $\mathcal{C} = \{\mathbf{c}_1, \ldots, \mathbf{c}_K\} \subset \mathbb{R}^D$
1  $p \leftarrow 1/(1 - M)$
2  while termination criterion is not met do
3      for $i \leftarrow 1$ to $K$ do
4          $n_i \leftarrow 0$   ▷ weight of $\mathcal{P}_i$
5          $\hat{\mathbf{c}}_i \leftarrow \mathbf{0}$   ▷ vector sum of $\mathcal{P}_i$
6      end
7      for $j \leftarrow 1$ to $N$ do
8          $d_{\text{sum}} \leftarrow 0$   ▷ $\mathbf{x}_j$'s total distance to all centers
9          for $i \leftarrow 1$ to $K$ do
10             $d_i \leftarrow \left(\|\mathbf{x}_j - \mathbf{c}_i\|_2^2\right)^p$   ▷ $\mathbf{x}_j$'s distance to $\mathbf{c}_i$
11             $d_{\text{sum}} \leftarrow d_{\text{sum}} + d_i$
12         end
13         for $i \leftarrow 1$ to $K$ do
14             $m \leftarrow (d_i/d_{\text{sum}})^M$   ▷ $\mathbf{x}_j$'s membership to $\mathcal{P}_i$
15             $n_i \leftarrow n_i + m$
16             $\hat{\mathbf{c}}_i \leftarrow \hat{\mathbf{c}}_i + m\mathbf{x}_j$
17         end
18     end
19     for $i \leftarrow 1$ to $K$ do
20         $\mathbf{c}_i \leftarrow \hat{\mathbf{c}}_i/n_i$   ▷ recompute $\mathbf{c}_i$
21     end
22 end
```

Algorithm 7: The FCM Algorithm

The earliest applications of FCM to the CQ problem were described by Kok et al. (1993) and Çak et al. (1998). The former study does *not* report any numerical results. In addition, both studies omit crucial details such as the value of the weighting exponent used in the experiments.

The first rigorous study of fuzzy clustering in CQ was conducted by Özdemir and Akarun (2002). The authors modify the FCM objective to maximize the so-called **partition index** defined for data point $\mathbf{x}_j$ as $p_j = \sum_{i=1}^{K} u_{ij}^{M}$, which quantifies how well $\mathbf{x}_j$ has been classified. If the classification is relatively unambiguous ($u_{ij} \approx 1$ for some i, that is, $\mathbf{x}_j$ is much closer to a particular center than it is to other centers), p_j approaches its maximum value of one. If, on the other hand, the classification is relatively ambiguous ($u_{ij} \approx 1/K$ for all i, that is, $\mathbf{x}_j$ is nearly equidistant to all K centers), p_j approaches its minimum value of K^{1-M}.

The FCM objective (23) can be modified to incorporate the sum of partition indices for all data points as follows

$$J_{\text{PIM}} = \sum_{j=1}^{N} \sum_{i=1}^{K} u_{ij}^{M} \left\| \mathbf{x}_j - \mathbf{c}_i \right\|_2^2 - R \sum_{j=1}^{N} \sum_{i=1}^{K} u_{ij}^{M}. \tag{27}$$

The FCM-like alternating optimization algorithm to minimize this modified objective is called the **partition index maximization algorithm** (PIM). The center update equation for PIM is identical to that for FCM. On the other hand, the membership update equation for PIM is given by

$$u_{ij} = \left[\sum_{\hat{\imath}=1}^{K} \left(\frac{\left\| \mathbf{x}_j - \mathbf{c}_i \right\|_2 - R}{\left\| \mathbf{x}_j - \mathbf{c}_{\hat{\imath}} \right\|_2 - R} \right)^{2/(M-1)} \right]^{-1}, \tag{28}$$

for all i and j, which reduces to the membership update equation for FCM for $R = 0$. To prevent undefined values in the above equation, we set $u_{ij} = 1$ if $\left\| \mathbf{x}_j - \mathbf{c}_i \right\|_2 \leq R$. This means that we classify $\mathbf{x}_j$ in a *hard* manner if it falls in the spherical region of radius R centered at $\mathbf{c}_i$, and in a *soft* manner based on Eq. (28) otherwise. The greater the radius R, the more data points are classified in a hard manner, and hence the faster the PIM algorithm (compared to FCM)[62]. However, R *cannot* be too large because otherwise, PIM ceases to be a fuzzy clustering algorithm. Özdemir and Akarun recommend setting R to a user-defined fraction C of the ℓ_2^2 distance between the nearest two centers, that is, $R = C \min_{i \neq \hat{\imath}} \left\| \mathbf{c}_i - \mathbf{c}_{\hat{\imath}} \right\|_2^2$, where $C \in [0, 0.5)$. The authors report good results with $M = 1.3$ and $C = 0.49$. Yang et al. (2008) suggest $C \in [0, 0.25)$ because $C \geq 0.25$ might violate condition (22b).

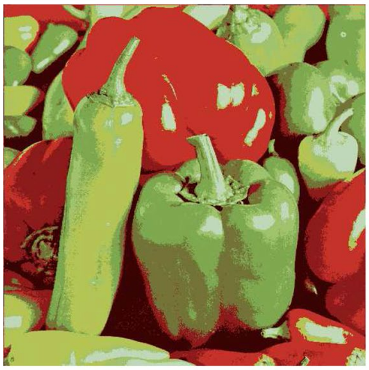
Wen and Celebi (2011) implement accelerated variants of BKM and FCM, and compare the resulting CQ algorithms. Both algorithms feature sampling of the input image (2 : 1 sampling in the horizontal and vertical directions) as well as weighting. BKM is further accelerated using the TIE algorithm described in Subsection 5.3. The authors investigate four hierarchical CQ algorithms to initialize BKM and FCM: median-cut, octree, marginal variance minimization, and variance minimization. Four FCM variants[63] are tested: FCM$_{1.25}$ FCM$_{1.5}$, FCM$_{1.75}$, and FCM$_{2.0}$, where the subscripts indicate the values of the weighting exponent. In terms of MAE, all BKM and FCM variants perform similarly. In general, as M is increased, the MSE attained by FCM increases, with FCM$_{1.25}$ performing slightly better than BKM. With respect to efficiency, BKM vastly outperforms FCM. In particular, BKM is 92 times faster than FCM, on average.

Schaefer and Zhou (2009); Schaefer (2014) apply several fuzzy/rough clustering algorithms to the CQ problem. These algorithms include standard FCM, multistage random

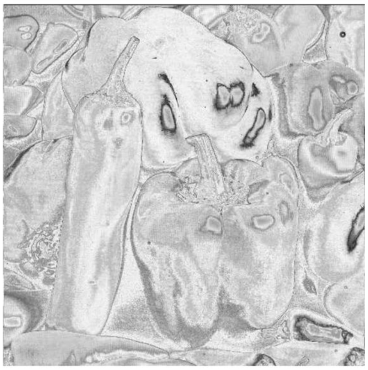
[62] In practice (Celebi 2009), PIM is *only* slightly faster than FCM because the former is based on ℓ_2 distances, whereas the latter is based on ℓ_2^2 distances.

[63] Recall that FCM with $M = 1$ is identical to BKM.

(a) Peppers (183, 525 colors)



(b) FCM (8 colors)



(c) Error image for 13(b)

Fig. 13 Peppers image quantized using Wen and Celebi's FCM algorithm

sampling FCM (Cheng et al. 1998), enhanced FCM (Szilágyi et al. 2003), anisotropic mean-shift-based FCM (Zhou et al. 2009), rough c-means (Schaefer et al. 2011), and fuzzy rough c-means (Schaefer et al. 2012). Their experiments demonstrate that, when initialized with randomly selected centers, these algorithms perform similarly on average for $K = 16$. While some of these algorithms may be faster than naive FCM, they also have more user-defined parameters, and many of them are more difficult to implement.

Due to its fuzzy nature, FCM is expected to be less sensitive to noise, cluster overlap, and initialization than its hard counterpart. However, in the context of CQ, these potential advantages of FCM have been demonstrated *only* to a limited extent (Szilágyi et al. 2014, 2016). In addition, despite significant efforts from fuzzy clustering researchers (Chen and Wang 1999; Dembélé and Kastner 2003; Yu et al. 2004; Yu and Yang 2005; Ozkan and Turksen 2007; Schwämmle and Jensen 2010; Huang et al. 2012; Wu 2012; Ren et al. 2019; Zhou and Yang 2019), there is still *no* universally accepted method to set the weighting exponent.

Figure 13 shows the Peppers image quantized to 8 colors using Wen and Celebi's variant of the FCM algorithm with $M = 2$. It can be seen that the error image for FCM is darker in most parts than that for TIE (shown in Fig. 12c), indicating that FCM produces more distortion than its hard counterpart.

5.5 The self-organizing map algorithm

In Subsection 5.3, we discussed the dead unit (or empty cluster) problem that plagues hard CL algorithms such as OKM, especially when they are initialized poorly. We then presented a modified CL algorithm that alleviates the dead unit problem. Recall that hard CL algorithms adapt *only* the winner unit in each iteration. Therefore, units initialized near high-density regions of the data space may monopolize the competition, whereas those initialized near low-density regions may become perpetual losers. In this subsection, we describe another approach to address the dead unit problem, the so-called **soft competitive learning** (or **winner-take-most learning**) paradigm.

Soft CL algorithms adapt not only the winner, but also some of the losers so that they are *not* completely excluded from the competition. Among these algorithms, Kohonen's **self-organizing map algorithm** (SOM) (Kohonen 1982) is one of the most prominent. The SOM algorithm performs a nonlinear, topology-preserving mapping from a high-dimensional data manifold to a low-dimensional (typically 1D or 2D) lattice (Van Hulle 2012). Here, topology (or neighborhood) preservation means topologically close (or neighboring) units respond to similar inputs.

The competitive stage of SOM is identical to that of OKM: in each iteration, a data point selected uniformly at random from $\mathcal{X}$ is presented, which induces a competition for activation among the K units. The two algorithms differ in their cooperative stage: while OKM adapts *only* the winner unit using Eq. (19), SOM adapts both the winner unit i^* and its lattice neighbors as follows

$$\mathbf{c}_i(t) = \mathbf{c}_i(t-1) + h(i, i^*, t)\big(\mathbf{x}(t) - \mathbf{c}_i(t-1)\big), \tag{29}$$

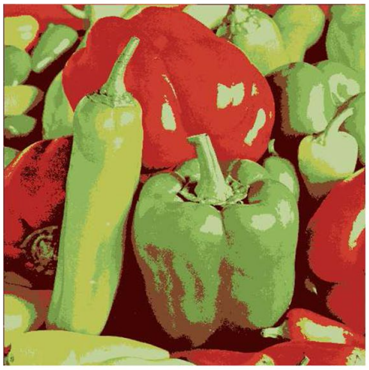
where h is the **neighborhood function**, a scalar-valued function of the lattice coordinates $\mathbf{r}_i$ and $\mathbf{r}_{i^*}$ of units i and i^*, respectively. The function h is often taken as a Gaussian, that is,

$$h(i, i^*, t) = \eta(t) \exp\left(-\frac{\|\mathbf{r}_i - \mathbf{r}_{i^*}\|_2^2}{2\sigma^2(t)}\right), \tag{30}$$

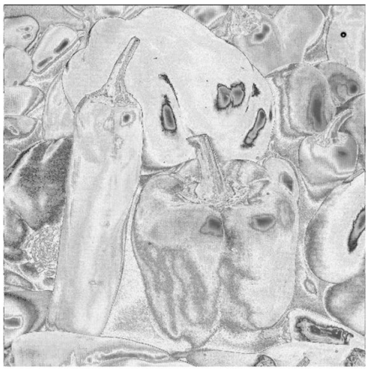
where $\eta(t)$ is the learning rate and $\sigma(t)$ is the standard deviation of the Gaussian, which corresponds to the **neighborhood range**. Both $\eta(t)$ and $\sigma(t)$ are usually taken to be monotonically decreasing functions of time. For example, $\eta(t)$ may decay according to an **exponential schedule** (Ritter and Schulten 1988) given by $\eta(t) = \eta_i(\eta_f/\eta_i)^{t/T_{\max}}$, where $T_{\max}$ is the number of iterations (i.e., $t \in \{0, \ldots, T_{\max}\}$), and $\eta_i = \eta(0)$ and $\eta_f = \eta(T_{\max})$ are respectively the initial and final values of $\eta(t)$ with $\eta_i \geq \eta_f > 0$. The initial learning rate η_i is typically taken as one, in which case the exponential schedule reduces to $\eta(t) = \eta_f^{t/T_{\max}}$. Other popular schedules in the SOM literature (Kohonen 2013) include the **linear schedule** and **harmonic schedule** given by $\eta(t) = \eta_i(1 - t/T_{\max})$ and $\eta(t) = \eta_i C/(C + t)$ with $C > 0$, respectively. Similarly, $\sigma(t)$ may decay according to an exponential, linear, or harmonic schedule.

In Subsection 5.3, we discussed the role of the learning rate $\eta(t)$ in an online learning algorithm. As for the neighborhood range $\sigma(t)$, we start with a relatively large value to promote topology preservation in the early iterations, and then decrease it monotonically with time. Observe that when $\sigma(t)$ vanishes, the adaptation equation for SOM, Eq. (29), reduces to that for simple CL, Eq. (19), meaning that SOM reduces to OKM.

(a) Peppers (183, 525 colors)



(b) SOM (8 colors)



(c) Error image for 14(b)

Fig. 14 Peppers image quantized using Dekker's SOM algorithm

The Gaussian neighborhood function given above ensures that each neighbor of the winner is moved toward the input at a rate depending on its topological closeness to the winner. A computationally simpler neighborhood function is given by

$$h(i, i^*, t) = \begin{cases} \eta(t) & \|\mathbf{r}_i - \mathbf{r}_{i^*}\|_2 < r(t), \\ & 0 \text{ otherwise}, \end{cases} \tag{31}$$

where $r(t)$ is the radius of the neighborhood, which is also a monotonically decreasing function of time. This simplified neighborhood function moves the winner and its neighbors within a radius toward the input at the same rate.

There are two common ways to initialize a SOM. Random initialization involves using K data points selected uniformly at random from $\mathcal{X}$ as initial centers. PC initialization (Kohonen 2013) involves selecting the K initial centers from the subspace spanned by the first D' PCs of the data, where D' is the dimensionality of the map ($D' \in \{1, 2, 3\}$). Not surprisingly, the effectiveness of an initialization method appears to be data dependent. In a recent experimental study on 1D SOMs (Akinduko et al. 2016), random initialization outperformed PC initialization on nonlinear data sets, whereas the situation was reversed on quasilinear data sets.

Table 4 Comparison of Partitional CQ Algorithms (diff: difficult, effi: efficient, effec: effective(ness), freq: frequency, hist. res.: histogram resolution, impl: implement, init: initialization, mod: modified, moder: moderately, param: parameter, pres: presentation, sens: sensitive, some: somewhat, sub: substantially, sqrt: square root)

Algorithm	References	Advantages	Disadvantages
Popularity	Heckbert (1982)	effi.; easy to impl.	Disregards diversity of colors; hist. res. param. diff. to set
Mod. popularity	Braudaway (1987)	effi.; easy to impl.	C param. diff. to set
Maximin	Houle and Dubois (1986)	effec. on well-separated data; effi.; easy to impl.	Disregards freq. of colors; sens. to outliers; spreads distortion uniformly
Mod. maximin	Hsieh and Fan (2000)	effi.; easy to impl.	T param. diff. to set
Mod. maximin	Huang (2021)	effi.; easy to impl.	ℓ_2^2 distance and sqrt. of freq. are incommensurable
FSKM	Huang and Chang (2004)	moder. faster than BKM; some. easy to impl.	K_f param. diff. to set; sens. to init.; *no* convergence guarantee
SBKM	Hu and Lee (2007)	moder. faster than BKM; some. easy to impl.	sens. to init.; *no* convergence guarantee
–	Kasuga et al. (2000)	Slightly faster than BKM	diff. to impl.
TIE + MOS	Hu and Su (2008a)	moder. faster than BKM	diff. to impl.; sens. to init.
weighted TIE	Celebi (2011)	sub. faster than BKM	some. diff. to impl.; sens. to init.
MOS	Huang (2021)	moder. faster than BKM	some. diff. to impl.; sens. to init.
OKM	Thompson et al. (2020)	sub. faster than BKM; easy to impl.	sens. to init.
IOKM	Abernathy and Celebi (2022)	sub. faster than BKM; easy to impl.; init.-free	effec. *cannot* be improved using a better init.
ADU	Celebi et al. (2014)	effi.; easy to impl.; init.-free	scales poorly with K; some. sens. to pres. order
PIM	Özdemir and Akarun (2002)	less sens. to noise, cluster overlap, and init. than BKM	Slower than FCM; M and C param. diff. to set
FCM	Wen and Celebi (2011)	less sens. to noise, cluster overlap, and init. than BKM	sub. slower than BKM; M param. diff. to set
SOM	Dekker (1994)	effi.; less sens. to init. than BKM; generates a some. contiguous palette	diff. to impl.; many param.; some. sens. to pres. order

In Subsection 5.3, we discussed the difficulty of accelerating OKM. As OKM is a special case of SOM, where the winner has *no* neighbors, the acceleration of SOM appears to be at least as difficult as that of OKM. For this reason, most accelerated SOM variants are either approximations (Koikkalainen and Oja 1990; Lampinen and Oja 1990; Su and Chang 2000; Bernard et al. 2020) or parallel algorithms (Ienne et al. 1997; Lawrence et al. 1999; Xiao et al. 2015; Wittek et al. 2017; Liu et al. 2018), which are *not* based on the online SOM formulation given above, but rather a batch SOM formulation (Kohonen 1993).

The first rigorous application of the SOM algorithm to the CQ problem was proposed by Dekker (1994). The author employs a 1D SOM[64], initialized with centers spread evenly along the main diagonal of the RGB cube, that is, $\mathbf{c}_i = (r_i, g_i, b_i)$, where $r_i = g_i = b_i = 255\lfloor (i-1)/(K-1) \rfloor$ for $i \in \{1, \dots, K\}$. The algorithm is essentially a one-pass algorithm, but, for implementation reasons, the scan is divided into 100 cycles. Thus, $N/100$ pixels are presented in each cycle. The learning rate schedule during cycle t is taken as $\eta(t) = \exp(-0.03t)$. Hence, the learning rate starts at 1 (at cycle 0), and decreases exponentially until reaching its final value of about 0.0513 (at cycle 99). The neighborhood function is taken as $h(i, i^*, t) = \eta(t)\left(1 - (|i - i^*| / \lfloor r(t) \rfloor)^2\right)$, where the radius is given by $r(t) = 32\exp(-0.0325t)$. Thus, the radius starts at 32 (at cycle 0), and decreases exponentially until reaching about 2.02 at cycle 85, after which point it drops below 2. This means that in the last 14 cycles, *only* the winner is updated.

Dekker uses the **conscience mechanism** of DeSieno (1988) to prevent certain units from monopolizing the competition. Let $\mathbf{x}$ be the current input and $\mathbf{c}_i$ be one of the units. The conscience mechanism computes the distance[65] between the two as $\|\mathbf{x} - \mathbf{c}_i\|_1 - b_i$, where b_i is a bias factor that increases as unit i loses, thereby favoring the selection of less frequently winning units. The bias is defined as $b_i = \alpha(1/K - f_i)$, where α is a constant, and f_i estimates the frequency at which unit i is nearest to the input. Initially, we have $f_i = 1/K$, and hence $b_i = 0$. Let i^* be the nearest unit to $\mathbf{x}$ with respect to the ℓ_1 metric, that is, $i^* = \underset{i \in \{1,\dots,K\}}{\arg\min} \|\mathbf{x} - \mathbf{c}_i\|_1$. The frequency of unit i^* is updated as $f_{i^*} \leftarrow f_{i^*} - \beta f_{i^*} + \beta$, whereas that of unit $i \neq i^*$ is updated as $f_i \leftarrow f_i - \beta f_i$. Note that the bias of unit i^* can then be updated directly as $b_{i^*} \leftarrow b_{i^*} + \alpha\beta f_{i^*} - \alpha\beta$, whereas that of unit $i \neq i^*$ can updated as $b_i \leftarrow b_i + \alpha\beta f_i$. Dekker recommends $\alpha = 1024$ and $\beta = 1/1024$ so that $\alpha\beta = 1$, which simplifies the above update equations. It is important to note that the conscience mechanism aims to achieve clusters of approximately equal size, which is rarely optimal from a distortion minimization perspective.

Figure 14 shows the Peppers image quantized to 8 colors using Dekker's SOM algorithm. It can be seen that while the algorithm performs better than early hierarchical algorithms (e.g., median-cut and octree), it is *not* competitive with modern hierarchical algorithms or KM-based partitional algorithms.

From a CQ perspective, the advantages of SOM are (i) its biologically-inspired formulation is appealing to researchers; (ii) thanks to its soft-competitive design, the algorithm is less sensitive to initialization and the dead unit problem; and (iii) the final palette is somewhat contiguous (i.e., adjacent colors are similar), which can be exploited for post-quantization image processing operations such as edge detection (Mojsilović and Soljanin 2001) and compression (Dekker 1994; Pei and Lo 1998; Mojsilović and Soljanin 2001; Chang et al. 2005; Pei et al. 2006).

[64] According to Pei and Lo (1998), in the context of CQ, a 2D SOM performs slightly worse than a 1D one.

[65] For computational efficiency, Dekker adopts the ℓ_1 metric rather than the ℓ_2^2 distance.

Dekker's algorithm is the best-known application of SOM to CQ. However, his seminal paper illustrates a major challenge researchers face when designing SOM-based CQ algorithms: the large number of user-defined parameters required for controlling their behavior. In fact, Dekker's algorithm, as well as its successors (Pei and Lo 1998; Chang et al. 2005; Wang et al. 2007; Chung et al. 2012), each have a half-dozen to dozen parameters. Furthermore, the performance of these algorithms is known to be sensitive to the values of *at least* some of their parameters (Ienne et al. 1997; Su and Chang 2000; Chang et al. 2005; Wang et al. 2007). Other disadvantages of SOM include (i) convergence is formally proved *only* for the simplest case (both the input space and lattice are 1D) (Cottrell et al. 2016; ii) it is not only slow, but also difficult to accelerate (Chung et al. 2012; iii) it is somewhat sensitive to the order in which the data points are processed; and (iv) it is more difficult to implement (Kohonen 2013) than its KM-based alternatives.

Table 4 compares the partitional CQ algorithms described in this section.

6 Metaheuristic algorithms

Many recent CQ algorithms are based on **metaheuristics** (Blum and Roli 2003). These algorithms formulate CQ as a *global*[66] optimization problem, and then attempt to solve it using various nature-inspired metaheuristics. Metaheuristics applied to CQ to date include single-solution-based metaheuristics such as simulated annealing (Fiume and Ouellette 1989; Nolle and Schaefer 2007; Schaefer and Nolle 2015) and variable neighborhood search (Hansen et al. 2007) as well as population-based metaheuristics such as evolutionary algorithms (genetic algorithms (Freisleben and Schrader 1997; Scheunders 1997; Taşdizen et al. 1998), evolution strategies (González et al. 2000), etc.) and swarm intelligence algorithms (particle swarm optimization (Omran et al. 2005), ant colony optimization (Ghanbarian et al. 2007; Pérez-Delgado 2015, 2021), etc.) There are also CQ algorithms that combine a conventional (hierarchical or partitional) clustering algorithm with a metaheuristic (Pérez-Delgado and Gallego 2019, 2020) or multiple metaheuristics (Pérez-Delgado 2018, 2019, 2020).

Compared to conventional clustering algorithms, metaheuristic algorithms can optimize more complex, e.g., perceptually-based (Schaefer and Nolle 2015), objective functions or even multiple objective functions simultaneously. For example, it is generally easier to formulate an objective function with a variable number of clusters in a metaheuristic framework. Unfortunately, these black-box algorithms have several major disadvantages over conventional ones. First they often involve a large number of user-defined parameters (initial/final temperature, cooling schedule, population size, crossover/mutation probability, etc.) some of which are difficult to tune or control (Eiben et al. 1999). Second, they are generally randomized to escape from local minima. Third, due to the vast search space in clustering problems, they usually require many iterations, which renders them computationally expensive (they can be orders of magnitude slower than conventional clustering algorithms). Fourth, they are typically more difficult to implement. We should, however, mention that some of the recent metaheuristic-based CQ algorithms (Pérez-Delgado and Gallego 2020, 2019) have *only* a few user-defined parameters, and are reasonably efficient.

[66] By contrast, conventional partitional algorithms discussed in Sect. 5 are all *local* optimization algorithms.

7 Pixel mapping

Recall that CQ consists of two main phases: color palette design and pixel mapping[67] (PM). In general, the former is a computationally hard problem, whereas the latter can be solved exactly using a trivial linear-time algorithm, as explained below. Therefore, most of the CQ literature deals with the former phase, which is also the main focus of this paper. In this section, we briefly discuss the latter phase for completeness.

Given an input image, the palette design phase produces a small palette that represents the input colors. To minimize distortion, each pixel in the input image (or **query pixel** in the nearest neighbor search terminology) should be mapped to its nearest representative in the palette. The result of the PM phase is thus a reduced-color image, wherein *only* colors from the palette can appear.

A key aspect of PM is the distance function $d(\cdot, \cdot)$ used to quantify the dissimilarity between a query pixel and a representative. Let I and $\tilde{I}$ be the $H \times W$ input and output images, respectively, and $\mathcal{C} = \{\mathbf{c}_1, \dots, \mathbf{c}_K\}$ be the palette. The mapping operation is then given by

$$\tilde{I}(r, c) = \underset{\mathbf{c}_i \in \mathcal{C}}{\arg\min}\, d(I(r, c), \mathbf{c}_i), \tag{32}$$

for $r \in \{1, \dots, H\}$ and $c \in \{1, \dots, W\}$.

Since MSE is the most popular fidelity metric in CQ applications, it is customary to adopt the $d = \ell_2^2$ distance in the PM phase as well. Nevertheless, some studies use alternative distance functions. For example, Dekker (1994) uses the ℓ_1 metric to accelerate PM. Unfortunately, the ℓ_1 norm is a poor approximation for the ℓ_2 norm. More specifically, the former can overestimate the latter by as much as $100(\sqrt{D} - 1)\%$ in D dimensions ($\approx 73.2\%$ for $D = 3$) (Celebi et al. 2011). In addition, ℓ_1 is unlikely to provide an appreciable computational advantage over ℓ_2^2 on a modern CPU. Verevka and Buchanan (1995) use an ℓ_2 approximation due to Chaudhuri et al. (1992) as a substitute for ℓ_2^2. In fact, there are several such ℓ_2 norm approximations in the literature (Celebi et al. 2011, 2012a). These approximations are computationally cheaper than ℓ_2, as they eliminate the expensive square root operation (in some cases, also reduce the number of multiplications) at the cost of increasing the numbers of comparison and absolute value operations. Hence, using an ℓ_2 approximation is unlikely to yield a significant speed-up over the computationally simpler ℓ_2^2, which does *not* involve any square roots.

Another issue is whether or not the PM phase is coupled with the palette design phase. A prime example of an uncoupled PM algorithm is the **trivial algorithm** that computes the ℓ_2^2 distance between a given query pixel and each of the K representatives in $\mathcal{O}(K)$ time, leading to an $\mathcal{O}(N_p K)$ exact PM algorithm. Coupled PM algorithms, on the other hand, are typically found in divisive CQ algorithms. Many of these CQ algorithms build a binary tree approximating the 3D Voronoi diagram of the input color space, which can then be used for approximate PM in $\mathcal{O}(N_p \log K)$ time (Brun and Secroun 1998). These algorithms achieve faster PM at the cost of greater distortion by mapping each query pixel to the representative of its cluster rather than its nearest representative (Wu 1992b, a). Note that an optimal PM algorithm has a time complexity of $\mathcal{O}(N_p)$.

[67] Sometimes referred to as **inverse color map computation** (Heckbert 1982)

For each query pixel, the trivial algorithm performs an exhaustive search in the palette. This is inefficient because, for any query pixel, many of the representatives will likely be too far to be its nearest representative. We now briefly describe the accelerated PM algorithms proposed in the CQ literature. The first such algorithm, named **locally sorted search**, was proposed by Heckbert (1982). The algorithm starts by performing bit-cutting, that is, by dividing the RGB cube into $L \times L \times L$ equal-sized subcubes, where L is a user-defined integer (the author uses $L = 8$ for $K = 256$). For each subcube, it maintains a sorted list of representatives that are nearest to some color in that subcube. Given a query pixel, it first determines the subcube that contains the color of that pixel, and then performs an exhaustive search *only* in the list associated with that subcube. This algorithm reduces the number of ℓ_2^2 distance computations required for an exact PM. However, it uniformly divides the RGB space without considering the distribution of the input colors (Necaise 1998). In addition, the optimal L value is difficult to estimate (Brun and Secroun 1998). Necaise (1998) proposes a modified locally sorted search algorithm that divides the RGB space adaptively into rectangular boxes of varying dimensions using a k-d tree.

Thomas (1991) proposes an algorithm that explicitly computes the 3D Voronoi diagram generated by the representatives. The algorithm requires $\mathcal{O}(2^{3B})$ memory, where B is the number of bits per color component. To reduce the memory requirements of the algorithm, the author employs bit-cutting, that is, truncating the least significant C bits of each component (the author uses $C = 3$). Due to this modification, however, the algorithm provides *only* an approximate PM. Brun and Secroun (1998) describe a three-step algorithm that improves upon Thomas's Voronoi-diagram-based approach. The algorithm first projects the representatives onto the plane spanned by the first two PCs of the input image. It then computes an approximate 2D Voronoi diagram generated by the projected representatives. Finally, a correction step is performed to reduce the approximation errors committed in the first two steps. For a sufficiently large palette, the resulting algorithm provides a near-exact PM in roughly $\mathcal{O}(N_p)$ time (independent of K).

Cheng and Yang (2001) propose an exact accelerated PM algorithm based on 1D projections onto a rough approximation for the PC1 of the palette given by the line ℓ passing through the mean representative and the representative farthest from it. The algorithm first projects the representatives onto ℓ, and then sorts them to obtain an ordered set $\mathcal{S}$ of projection values. Given a query pixel, it determines the nearest representative by performing repeated binary searches over $\mathcal{S}$ with the search key taken as the projection of the query pixel on ℓ. Hung and Chang (2002) modify Cheng and Yang's algorithm by taking the PC1 of the palette as the projection axis, and performing a single binary search per query pixel to find an approximate nearest representative, which is then refined by means of linear searches among the immediate left and right neighbors of that representative. Hwang and Chang (2002) extend Hung and Chang's approach by employing 2D projections onto the plane spanned by the first two PCs of the palette. More recently, Hu and Su (2008b) combine TIE with MOS[68]. It should noted that there is a fine balance between the effectiveness of elimination and the efficiency of search. Therefore, overly complicated elimination strategies do *not* necessarily pay off in practice.

[68] Dekker (1994) also employs a MOS-like accelerated PM algorithm.

8 Conclusions and future research directions

In this paper, we presented a comprehensive survey of CQ algorithms proposed since 1980. We first provided background information on CQ, color science, and color image processing. We then presented a detailed survey of color palette design, followed by a brief discussion of accelerated pixel mapping.

Despite the over four decades of research on CQ, many unsolved problems remain:

- The selection of the color space in which to perform CQ and the selection of color image fidelity metrics with which to evaluate the quality of CQ are nontrivial decisions that deserve further exploration.
- In addition to objective methods, subjective methods should be used to assess CQ algorithms.
- A vast majority of CQ algorithms developed to date are context-free. Due to the importance of contextual information, algorithms that incorporate such information in accordance with the characteristics of the HVS should be developed. Metaheuristic algorithms are particularly attractive for optimizing complex HVS-based objective functions.
- Most CQ algorithms are static, requiring the user to specify the palette size in advance. However, in some applications, we are interested in determining the number of dominant colors in a given image (Celebi and Zornberg 2014; Barata et al. 2016). Therefore, the development of dynamic CQ algorithms should be prioritized for such applications.
- Due to the advances in camera technology, true-color images have been steadily getting larger over the years. It may be too slow to quantize such large images even with a linear-time CQ algorithm, especially if the algorithm is a partitional one that requires multiple passes over the input image. Therefore, effective and efficient methods should be developed to sample large images.
- Researchers have developed many real-time CQ algorithms since the 1980s. However, to the best of our knowledge, there have been *no* attempts to develop an interactive CQ algorithm. It is *not* difficult to envision that such an algorithm can be used in various computer graphics applications such as non-photorealistic rendering.
- Most CQ algorithms are designed for software implementation. Among the few existing hardware solutions, most concern the SOM algorithm implemented on field-programmable gate arrays (FPGAS) (Sudha et al. 2003; Atsalakis et al. 2004; Chang et al. 2006; Kurdthongmee 2008, 2011, 2016; Khalifa et al. 2020). While these hardware-based SOM quantizers can achieve real-time processing speeds, they involve certain approximations (e.g., fixed-point or integer representation as opposed to floating-point representation, the ℓ_1 metric instead of the ℓ_2^2 distance, and winner-take-all learning rather than winner-take-most learning) that compromise their effectiveness. More effort should be directed toward designing hardware-friendly and effective CQ algorithms. In addition, implementation of CQ algorithms on graphics processing units (GPUS) should be investigated further (Leung et al. 2010; Bottisti et al. 2012; Trapp et al. 2019).
- There are currently very few supervised CQ algorithms (Hou et al. 2020; Yoo et al. 2020; Park et al. 2022). Convolutional neural networks (CNNS) are a promising avenue of research in this respect.
- Most open-source CQ software implement conventional hierarchical algorithms (e.g., median-cut, marginal variance minimization, octree, oblique-cut, and variance minimization). CQ researchers should release implementations of newer algorithms as open-source software.

- Many CQ studies employ images taken from the USC-SIPI Image Database or the Kodak Lossless True Color Image Suite. Recently, Celebi and Pérez-Delgado published a large and diverse benchmark database named CQ100[69] on which CQ algorithms can be developed, tested, and compared. This database should be used in CQ studies so that its deficiencies can be identified and subsequently addressed.

Funding This material is based upon work supported by the National Science Foundation under Award No. OIA-1946391. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation.

Data availability Data sharing is not applicable to this article as no data sets were generated or analyzed.

References

Abernathy AD, Celebi ME (2022) The incremental online *k*-means clustering algorithm and its application to color quantization. Expert Syst Appl 207(117):927

Ackerman M, Ben-David S, Loker D (2010) Towards property-based classification of clustering paradigms. In: Advances in neural information processing systems, pp 10–18

Ahalt SC, Krishnamurthy AK, Chen P et al (1990) Competitive learning algorithms for vector quantization. Neural Netw 3(3):277–290

Akarun L, Yardimci Y, Cetin AE (1997) Adaptive methods for dithering color images. IEEE Trans Image Process 6(7):950–955

Akinduko AA, Mirkes EM, Gorban AN (2016) SOM: stochastic initialization versus principal components. Inf Sci 364–365:213–221

Alexandrov VV, Gorsky ND, Mysko SN (1984) Recursive pyramids and their use for image coding. Pattern Recogn Lett 2(5):301–310

Aloise D, Deshpande A, Hansen P et al (2009) NP-hardness of euclidean sum-of-squares clustering. Mach Learn 75(2):245–248

Anderson M, Motta R, Chandrasekar S, et al (1996) Proposal for a standard default color space for the internet—sRGB. In: Proceedings of the color and imaging conference, pp 238–245

Andersson P, Nilsson J, Akenine-Möller T et al (2020) FLIP: a difference evaluator for alternating images. Proc ACM Comput Graph Interact Tech 3(2):15-1-15:23

Arthur D, Vassilvitskii S (2007) *K*-means++: the advantages of careful seeding. In: Proceedings of the 18th annual ACM–SIAM symposium on discrete algorithms, pp 1027–1035

Atsalakis A, Papamarkos N (2006) Color reduction and estimation of the number of dominant colors by using a self-growing and self-organized neural gas. Eng Appl Artif Intell 19(7):769–786

Atsalakis A, Papamarkos N, Andreadis I (2002) On estimation of the number of image principal colors and color reduction through self-organized neural networks. Int J Imaging Syst Technol 12(3):117–127

Atsalakis A, Papamarkos N, Kroupis N et al (2004) Colour quantisation technique based on image decomposition and its embedded system implementation. IEE Proc 151(6):511–524

Avanaki A, Espig K, Kimpe T, et al (2014) Perceptual Uniformity of commonly used color spaces. In: Proceedings of the medical imaging 2014: digital pathology conference, pp 90410V-1–90410V-6

Awasthi P, Charikar M, Krishnaswamy R, et al (2015) The hardness of approximation of euclidean *K*-means. In: Proceedings of the 31st international symposium on computational geometry, pp 754–767

Baarsch J, Celebi ME (2012) Investigation of internal validity measures for *K*-means clustering. In: Proceedings of the international multiconference of engineers and computer scientists, pp 471–476

Bader M (2013) Space-filling curves: an introduction with applications in scientific computing. Springer, Berlin

[69] Available at https://data.mendeley.com/datasets/vw5ys9hfxw/2.

Balasubramanian R, Allebach JP (1991a) A new approach to palette selection for color images. J Imaging Technol 17(6):284–290

Balasubramanian R, Allebach JP (1991b) A new approach to palette selection for color images. In: Proceedings of the SPIE electronic imaging symposium, pp 58–69

Balasubramanian R, Allebach JP, Bouman CA (1994a) Color-image quantization with use of a fast binary splitting technique. J Opt Soc Am A 11(11):2777–2786

Balasubramanian R, Bouman CA, Allebach JP (1994b) Sequential scalar quantization of color images. J Electron Imaging 3(1):45–59

Banerjee A, Merugu S, Dhillon IS et al (2005) Clustering with bregman divergences. J Mach Learn Res 6:1705–1749

Baqai FA, Lee JH, Agar AU et al (2005) Digital color halftoning. IEEE Signal Process Mag 22(1):87–96

Barata C, Celebi ME, Marques JS et al (2016) Clinically inspired analysis of dermoscopy images using a generative model. Comput Vis Image Underst 151:124–137

Barnes ES, Sloane NJA (1983) The optimal lattice quantizer in three dimensions. SIAM J Algebr Discret Methods 4(1):30–41

Bartholdi JJ III, Platzman LK (1988) Heuristics based on spacefilling curves for combinatorial problems in euclidean space. Manag Sci 34(3):291–305

Bei CD, Gray RM (1985) An improvement of the minimum distortion encoding algorithm for vector quantization. IEEE Trans Commun 33(10):1132–1133

Bentley JL (1975) Multidimensional binary search trees used for associative searching. Commun ACM 18(9):509–517

Berman D, Treibitz T, Avidan S (2016) Non-local image Dehazing. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 1674–1682

Bermejo S, Cabestany J (2002) The effect of finite sample size on online K-Means. Neurocomputing 48(1–4):511–539

Bernard Y, Hueber N, Girau B (2020) A fast algorithm to find best matching units in self-organizing maps. In: Proceedings of the international conference on artificial neural networks, pp 825–837

Bezdek JC (1981) Pattern recognition with fuzzy objective function algorithms. Springer, Berlin

Bezdek JC, Keller J, Krisnapuram R et al (1999) Fuzzy models and algorithms for pattern recognition and image processing. Kluwer Academic Publishers, Boston

Bhagavathy S, Llach J, Zhai J (2009) Multiscale probabilistic dithering for suppressing contour artifacts in digital images. IEEE Trans Image Process 18(9):1936–1945

Bing Z, Junyi S, Qinke P (2004) An adjustable algorithm for color quantization. Pattern Recogn Lett 25(16):1787–1797

Blum C, Roli A (2003) Metaheuristics in combinatorial optimization: overview and conceptual comparison. ACM Comput Surv 35(3):268–308

Borgelt C, Yarikova O (2020) Initializing K-means clustering. In: Proceedings of the 9th international conference on data science, technology and applications, pp 260–267

Bottisti D, Mendez L, Dechev D (2012) CuNeuQuant: a CUDA implementation of the NeuQuant Image-Quantization algorithm. In: Proceedings of the international conference on image processing, computer vision, and pattern recognition

Bottou L (1998) Online learning and stochastic approximations. In: Saad D (ed) On-line learning in neural networks. Cambridge University Press, Cambridge, pp 9–42

Bragg D (1992) A simple color reduction filter. In: Kirk D (ed) Graphics gems III. Morgan Kaufmann, pp 20–22

Braquelaire JP, Brun L (1997) Comparison and optimization of methods of color image quantization. IEEE Trans Image Process 6(7):1048–1052

Braudaway GW (1987) Procedure for optimum choice of a small number of colors from a large color palette for color imaging. In: Proceedings of the electronic imaging conference, pp 71–75

Bregman LM (1967) The relaxation method of finding the common point of convex sets and its application to the solution of problems in convex programming. USSR Comput Math Math Phys 7(3):200–217

Brucker P (1978) On the complexity of clustering problems. In: Henn R, Korte B, Oettli W (eds) Optimization and operations research. Springer, Berlin, pp 45–54

Brun L, Mokhtari M (2000) Two high speed color quantization algorithms. In: Proceedings of the 1st international conference on color in graphics and image processing, pp 116–121

Brun L, Secroun C (1998) A fast algorithm for inverse colormap computation. Comput Graph Forum 17(4):263–271

Brun L, Trémeau A (2003) Color quantization. In: Sharma G (ed) Digital color imaging handbook. CRC Press, Boca Raton, pp 589–638

Buades A, Lisani JL, Morel JM (2011) Dimensionality of color space in natural images. J Opt Soc Am A 28(2):203–209

Budrikis ZL (1972) Visual fidelity criterion and modeling. Proc IEEE 60(7):771–779

Buzo A, Gray, Jr. RMA. H.and Gray, Markel JD (1980) Speech coding based upon vector quantization. IEEE Trans Acoust Speech Signal Process 28(5):562–574

Cannon RL, Dave JV, Bezdek JC (1986) Efficient implementation of the Fuzzy *C*-means clustering algorithms. IEEE Trans Pattern Anal Mach Intell 8(2):248–255

Carlsson G, Memoli F (2010) Characterization, stability and convergence of hierarchical clustering methods. J Mach Learn Res 11:1425–1470

Cattelan M (2012) Models for paired comparison data: a review with emphasis on dependent data. Stat Sci 27(3):412–433

Çak S, Dizdar EN, Ersak A (1998) A Fuzzy colour quantizer for renderers. Displays 19(2):61–65

Celebi ME (2009) Fast color quantization using weighted sort-means clustering. J Opt Soc Am A 26(11):2434–2443

Celebi ME (2011) Improving the performance of *K*-means for color quantization. Image Vis Comput 29(4):260–271

Celebi ME (ed) (2015) Partitional clustering algorithms. Springer

Celebi ME, Aydin K (eds) (2016) Unsupervised learning algorithms. Springer

Celebi ME, Kingravi H (2012) Deterministic initialization of the *K*-means algorithm using hierarchical clustering. Int J Pattern Recognit Artif Intell 26(7):1250,018

Celebi ME, Kingravi HA (2015) Linear, deterministic, and order-invariant initialization methods for the *K* -means clustering algorithm. In: Celebi ME (ed) Partitional clustering algorithms. Springer, Berlin, pp 79–98

Celebi ME, Zornberg A (2014) Automated quantification of clinically significant colors in dermoscopy images and its application to skin Lesion classification. IEEE Syst J 8(3):980–984

Celebi ME, Kingravi H, Celiker F (2010) Fast colour space transformations using minimax approximations. IET Image Proc 4(2):70–80

Celebi ME, Celiker F, Kingravi HA (2011) On Euclidean norm approximations. Pattern Recogn 44(2):278–283

Celebi ME, Kingravi HA, Celiker F (2012a) Comments on 'on approximating Euclidean metrics by weighted *t*-cost distances in arbitrary dimension'. Pattern Recogn Lett 33(10):1422–1425

Celebi ME, Wen Q, Hwang S, et al (2012b) Color quantization of dermoscopy images using the *K*-means clustering algorithm. In: Celebi ME, Schaefer G (eds) Color medical image analysis. Springer, Berlin, pp 87–107

Celebi ME, Kingravi H, Vela PA (2013) A comparative study of efficient initialization methods for the *K* -means clustering algorithm. Expert Syst Appl 40(1):200–210

Celebi ME, Hwang S, Wen Q (2014) Colour quantisation using the adaptive distributing units algorithm. Imaging Sci J 62(2):80–91

Celebi ME, Wen Q, Hwang S (2015) An effective real-time color quantization method based on divisive hierarchical clustering. J Real-Time Image Proc 10(2):329–344

Celenk M (1990) A color clustering technique for image segmentation. Comput Vis Graph Image Process 52(2):145–170

Chan YH, Fung YH (2005) A regularized constrained iterative restoration algorithm for restoring color-quantized images. Signal Process 85(7):1375–1387

Chandler DM, Hemami SS (2007) VSNR: a wavelet-based visual signal-to-noise ratio for natural images. IEEE Trans Image Process 16(9):2284–2298

Chang RF, Chen WT, Wang JS (1992) A fast finite-state algorithm for vector quantizer design. IEEE Trans Signal Process 40(1):221–225

Chang CH, Xu P, Xiao R et al (2005) New adaptive color quantization method based on self-organizing maps. IEEE Trans Neural Networks 16(1):237–249

Chang CH, Shibu M, Xiao R (2006) Self organizing feature map for color quantization on FPGA. In: Omondi AR, Rajapakse JC (eds) FPGA implementations of neural networks. Springer, Berlin, pp 225–245

Chao CKT, Singh K, Gingold Y (2021) PosterChild: blend-aware artistic posterization. Comput Graph Forum 40(4):87–99

Chaudhuri D, Murthy CA, Chaudhuri BB (1992) A modified metric to compute distance. Pattern Recogn 25(7):667–677

Chen SH, Pan JS (1989) Fast search algorithm for VQ-based recognition of isolated words. IEE Proc I 136(6):391–396

Chen MS, Wang SW (1999) Fuzzy clustering analysis for optimizing Fuzzy membership functions. Fuzzy Sets Syst 103(2):239–254

Cheng H, Bouman CA (2001) Document compression using rate-distortion optimized segmentation. J Electron Imaging 10(2):460–474

Cheng SC, Yang CK (2001) Fast and novel technique for color quantization using reduction of color space dimensionality. Pattern Recogn Lett 22(8):845–856

Cheng DY, Gersho A, Ramamurthi B, et al (1984) Fast search algorithms for vector quantization and pattern matching. In: Proceedings of the IEEE international conference on acoustics, speech, and signal processing, pp 372–375

Cheng TW, Goldgof DB, Hall LO (1998) Fast Fuzzy clustering. Fuzzy Sets Syst 93(1):49–56

Cheng SS, Xiong Z, Wu X (2002) Fast trellis-coded color quantization of images. Real-Time Imaging 8(4):265–275

Cheng MM, Mitra NJ, Huang X et al (2015) Global contrast based salient region detection. IEEE Trans Pattern Anal Mach Intell 37(3):569–582

Chuang YY, Curless B, Salesin DH, et al (2001) A Bayesian approach to digital matting. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 264–271

Chung KL, Huang YH, Wang JP et al (2012) Speedup of color palette indexing in self-organization of Kohonen feature map. Expert Syst Appl 39(3):2427–2432

Cohen-Addad V, Kanade V, Mallmann-Trenn F et al (2019) Hierarchical clustering: objective functions and algorithms. J ACM 66(4):1–42

Cottrell M, Olteanu M, Rossi F, et al (2016) Theoretical and applied aspects of the self-organizing maps. In: Proceedings of the 11th international workshop WSOM 2016. Springer, Berlin, pp 3–26

Crevier D (1993) Computing statistical properties of Hue distributions for color image analysis. In: Proceedings of the SPIE intelligent robots and computer vision XII conference, pp 613–623

Daly SJ, Feng X (2004) Decontouring: prevention and removal of false contour artifacts. In: Proceedings of the SPIE electronic imaging symposium, pp 130–149

Darken C, Moody J (1990) Fast adaptive K-means clustering: some empirical results. In: Proceedings of the 1990 international joint conference on neural networks, pp 233–238

Dasgupta S (2016) A cost function for similarity-based hierarchical clustering. In: Proceedings of the 48th annual ACM symposium on theory of computing, pp 118–127

Dasgupta S, Freund Y (2009) Random projection trees for vector quantization. IEEE Trans Inf Theory 55(7):3229–3242

Dekker A (1994) Kohonen neural networks for optimal colour quantization. Netw Comput Neural Syst 5(3):351–367

Delon J, Desolneux A, Lisani JL et al (2007) Automatic color palette. Inverse Problems and Imaging 1(2):265–287

Dembélé D, Kastner P (2003) Fuzzy C-means method for clustering microarray data. Bioinformatics 19(8):973–980

DeSieno D (1988) Adding a conscience to competitive learning. In: Proceedings of the IEEE 1988 international conference on neural networks, pp 117–124

Dixit SS (1991) Quantization of color images for display/printing on limited color output devices. Comput Graph 15(4):561–567

Domański M, Bartkowiak M (1998) Compression. In: Sangwine SJ, Horne REN (eds) The colour image processing handbook. Chapman & Hall, Boca Raton, pp 242–304

Dosselmann R, Yang XD (2011) A comprehensive assessment of the structural similarity index. SIViP 5(1):81–91

Dyer ME, Frieze AM (1985) A simple heuristic for the P-centre problem. Oper Res Lett 3(6):285–288

Eiben AE, Hinterding R, Michalewicz Z (1999) Parameter control in evolutionary algorithms. IEEE Trans Evolut Comput 3(2):124–141

Elkan C (2003) Using the triangle inequality to accelerate K-Means. In: Proceedings of the 20th international conference on machine learning, pp 147–153

Equitz WH (1984) Fast algorithms for vector quantization picture coding. Master's thesis, Massachusetts Institute of Technology

Equitz WH (1989) A new vector quantization clustering algorithm. IEEE Trans Acoust Speech Signal Process 37(10):1568–1575

Eschrich S, Ke J, Hall LO et al (2003) Fast accurate Fuzzy clustering through data reduction. IEEE Trans Fuzzy Syst 11(2):262–270

Fairchild MD, Johnson GM (2004) iCAM framework for image appearance, differences, and quality. J Electron Imaging 13(1):126–138

Fan Z, Jiang T, Huang T (2017) Active sampling exploiting reliable informativeness for subjective image quality assessment based on pairwise comparison. IEEE Trans Multimedia 19(12):2720–2735

Farber B, Zeger K (2006) Quantization of multiple sources using nonnegative integer bit allocation. IEEE Trans Inf Theory 52(11):4945–4964

Feder T, Greene DH (1988) Optimal algorithms for approximate clustering. In: Proceedings of the 20th annual ACM symposium on theory of computing, pp 434–444

Feldman D (2020) Core-sets: updated survey. In: Ros F, Guillaume S (eds) Sampling techniques for supervised or unsupervised tasks. Springer, Berlin, pp 23–44

Fisher L, Van Ness JW (1971) Admissible clustering procedures. Biometrika 58(1):91–104

Fiume E, Ouellette M (1989) On distributed, probabilistic algorithms for computer graphics. In: Proceedings of the graphics interface '89, pp 211–218

Fletcher P (1991) A SIMD parallel colour quantization algorithm. Comput Graph 15(3):365–373

Flinkman M, Laamanen H, Vahimaa P et al (2012) Number of colors generated by smooth nonfluorescent reflectance spectra. J Opt Soc Am A 29(12):2566–2575

Fowler RJ, Paterson MS, Tanimoto SL (1981) Optimal packing and covering in the plane are NP-complete. Inf Process Lett 12(3):133–137

Fox B (1966) Discrete optimization via marginal analysis. Manag Sci 13(3):210–216

Freisleben B, Schrader A (1997) An evolutionary approach to color image quantization. In: Proceedings of the 1997 IEEE international conference on evolutionary computation, pp 459–464

Fung YH, Chan YH (2004) POCS-based algorithm for restoring colour-quantised images. IEE Proc Vis Image Signal Process 151(2):119–127

Fung YH, Chan YH (2006) A POCS-based restoration algorithm for restoring halftoned color-quantized images. IEEE Trans Image Process 15(7):1985–1992

Fung YH, Chan YH (2006) A simulated annealing restoration algorithm for restoring halftoned color-quantized images. Signal Process 21(4):280–292

Garey MR, Johnson D, Witsenhausen HS (1982) The complexity of the generalized Lloyd-Max problem. IEEE Trans Inf Theory 28(2):255–256

Gentile RS, Allebach JP, Walowit E (1990) Quantization of color images based on uniform color spaces. J Imaging Technol 16(1):11–21

Geraci F, Pellegrini M, Maggini M, et al (2006) Cluster generation and cluster labelling for web snippets. In: Proceedings of the 13th international conference on string processing and information retrieval, pp 25–36

Gersho A, Gray RM (1992) Vector quantization and signal compression. Kluwer Academic Publishers, Boston

Gervautz M, Purgathofer W (1988) A simple method for color quantization: octree quantization. In: Magnenat-Thalmann N, Thalmann D (eds) New trends in computer graphics. Springer, Berlin, pp 219–231

Ghanbarian A, Kabir E, Charkari N (2007) Color reduction based on ant colony. Pattern Recogn Lett 28(12):1383–1390

Goldberg N (1991) Colour image quantization for high resolution graphics display. Image Vis Comput 9(5):303–312

Gonzalez TF (1985) Clustering to minimize the maximum intercluster distance. Theoret Comput Sci 38(2–3):293–306

González AI, Graña M, Albizuri FX et al (2000) A near real-time evolution-based adaptation strategy for dynamic color quantization of image sequences. Inf Sci 122(2–4):161–183

Gotsman C, Lindenbaum M (1996) On the metric properties of discrete space-filling curves. IEEE Trans Image Process 5(5):794–797

Gray RM, Karnin ED (1982) Multiple local optima in vector quantizers. IEEE Trans Inf Theory 28(2):256–261

Großwendt A, Røglin H, Schmidt M (2019) Analysis of Ward's method. In: Proceedings of the 30th annual ACM-SIAM symposium on discrete algorithms, pp 2939–2957

Grossberg S (1987) Competitive learning: from interactive activation to adaptive resonance. Cogn Sci 11(1):23–63

Hadizadeh H, Bajic IV, Saeedi P, et al (2011) Good-looking green images. In: Proceedings of the 18th IEEE international conference on image processing, pp 3177–3180

Hains C, Wang SG, Knox K (2003) Digital color halftones. In: Sharma G (ed) Digital color imaging handbook. CRC Press, Boca Raton, pp 385–490

Hamerly G, Drake J (2015) Accelerating Lloyd's algorithm for *K*-means clustering. In: Celebi ME (ed) Partitional clustering algorithms. Springer, Berlin, pp 41–78

Hanbury A (2003) Circular statistics applied to colour images. Proc Comput Vis Winter Workshop 2003:55–60

Hanbury A (2008) Constructing cylindrical coordinate colour spaces. Pattern Recogn Lett 29(4):494–500

Hansen P, Lazić J, Mladenović N (2007) Variable neighbourhood search for colour image quantization. IMA J Manag Math 18(2):207–221

Hardeberg JY, Bando E, Pedersen M (2008) Evaluating colour image difference metrics for Gamut-mapped images. Color Technol 124(4):243–253

Harding EF (1967) The number of partitions of a set of N points in K dimensions induced by hyperplanes. Proc Edinb Math Soc (Ser 2) 15(4):285–289

Har-Peled S, Sadri B (2005) How fast is the K-means method? Algorithmica 41(3):185–202

Hasegawa S, Imai H, Inaba M et al (1993) Efficient algorithms for variance-based K-clustering. In: Shin SY, Kunii TL (eds) Computer graphics and applications. World Scientific Publishing Co., Singapore, pp 75–88

Hatam M, Masnadi-Shirazi MA (2015) Optimum nonnegative integer bit allocation for wavelet based signal compression and coding. Inf Sci 297:332–344

Hathaway RJ, Bezdek JC (2006) Extending Fuzzy and probabilistic clustering to very large data sets. Comput Stat Data Anal 51(1):215–234

Hathaway RJ, Hu Y (2009) Density-weighted Fuzzy C-means clustering. IEEE Trans Fuzzy Syst 17(1):243–252

Hathaway RJ, Bezdek JC, Huband JM (2006) Maximin initialization for cluster analysis. In: Proceedings of the 11th iberoamerican congress in pattern recognition, pp 14–26

Hatzinger R, Dittrich R (2012) prefmod: an R package for modeling preferences based on paired comparisons, rankings, or ratings. J Stat Softw 48(10):1–31

Heckbert PS (1980) Color image quantization for frame buffer display. Bachelor's thesis, Massachusetts Institute of Technology

Heckbert P (1982) Color image quantization for frame buffer display. ACM SIGGRAPH Comput Graph 16(3):297–307

Heckel R, Shah NB, Ramchandran K et al (2019) Active ranking from pairwise comparisons and when parametric assumptions do not help. Ann Stat 47(6):3099–3126

Hoare CAR (1971) Proof of a program: find. Commun ACM 14(1):39–45

Höppner F (2002) Speeding up Fuzzy C-means: using a hierarchical data organisation to control the precision of membership calculation. Fuzzy Sets Syst 128(3):365–376

Hore A, Ziou D (2010) Image quality metrics: PSNR vs. SSIM. In: Proceedings of the 2010 international conference on pattern recognition, pp 2366–2369

Hou Y, Zheng L, Gould S (2020) Learning to structure an image with few colors. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 10,116–10,125

Houle G, Dubois E (1986) Quantization of color images for display on graphics terminals. In: Proceedings of the IEEE global telecommunications conference, pp 1138–1142

Howard N, Harris B (1966) A hierarchical grouping routine (IBM 360/65 Fortran IV Program). Tech. rep., University of Pennsylvania Computer Center

Hsieh IS, Fan KC (2000) An adaptive clustering algorithm for color quantization. Pattern Recogn Lett 21(4):337–346

Hsu WL, Nemhauser GL (1979) Easy and hard bottleneck location problems. Discret Appl Math 1(3):209–215

Hu YC, Lee MG (2007) K-means based color palette design scheme with the use of stable flags. J Electron Imaging 16(3):033,003

Hu YC, Su BH (2008a) Accelerated K-means clustering algorithm for colour image quantization. Imaging Sci J 56(1):29–40

Hu YC, Su BH (2008b) Accelerated pixel mapping scheme for colour image quantisation. Imaging Sci J 56(2):68–78

Hu YC, Lee MG, Tsai P (2009) Colour palette generation schemes for colour image quantization. Imaging Sci J 57(1):46–59

Huang SC (2021) An efficient palette generation method for color image quantization. Appl Sci 11(3):1043

Huang YL, Chang RF (2004) A fast finite-state algorithm for generating RGB palettes of color quantized images. J Inf Sci Eng 20(4):771–782

Huang J, Schultheiss P (1963) Block quantization of correlated gaussian random variables. IEEE Trans Commun Syst 11(3):289–296

Huang TS, Schreiber WF, Tretiak OJ (1971) Image processing. Proc IEEE 59(11):1586–1609

Huang M, Xia Z, Wang H et al (2012) The range of the value for the Fuzzifier of the Fuzzy C-means algorithm. Pattern Recogn Lett 33(16):2280–2284

Huang HZ, Xu K, Martin RR et al (2016) Efficient, edge-aware, combined color quantization and dithering. IEEE Trans Image Process 25(3):1152–1162

Huang Q, Kim HY, Tsai WJ et al (2018) Understanding and removal of false contour in HEVC compressed images. IEEE Trans Circuits Syst Video Technol 28(2):378–391

Hung KL, Chang CC (2002) An improvement of a technique for color quantization using reduction of color space dimensionality. Informatica 26(1):11–16

Hwang KF, Chang CC (2002) A fast pixel mapping algorithm using principal component analysis. Pattern Recogn Lett 23(14):1747–1753

Hyafil L, Rivest RL (1976) Constructing optimal binary decision trees is NP-complete. Inf Process Lett 5(1):15–17

Ienne P, Thiran P, Vassilas N (1997) Modified self-organizing feature map algorithms for efficient digital hardware implementation. IEEE Trans Neural Networks 8(2):315–330

Inaba M, Katoh N, Imai H (1994) Applications of weighted voronoi diagrams and randomization to variance-based *K*-clustering. In: Proceedings of the 10th annual symposium on computational seometry, pp 332–339

Jackins CL, Tanimoto SL (1980) Oct-trees and their use in representing three-dimensional objects. Comput Graphics Image Process 14(3):249–270

Jain AK, Pratt WK (1972) color image quantization. In: Proceedings of the 1972 national telecommunications conference, p 34

Jain AK, Murty MN, Flynn PJ (1999) Data clustering: a review. ACM Comput Surv 31(3):264–323

Jamieson KG, Nowak RD (2011) Active ranking using pairwise comparisons. In: Proceedings of the 25th annual conference on neural information processing systems, pp 2240–2248

Johnson GM, Fairchild MD (2003) A top down description of S-CIELAB and CIEDE2000. Color Res Appl 28(6):425–435

Johnson GM, Song X, Montag ED et al (2010) Derivation of a color space for image color difference measurement. Color Res Appl 35(6):387–400

Joy G, Xiang Z (1993) Center-cut for color image quantization. Visual Comput 10(1):62–66

Joy G, Xiang Z (1996) Reducing false contours in quantized color images. Comput Graph 20(2):231–242

Kanungo T, Mount D, Netanyahu N et al (2002) An efficient *K*-means clustering algorithm: analysis and implementation. IEEE Trans Pattern Anal Mach Intell 24(7):881–892

Kasuga H, Yamamoto H, Okamoto M (2000) Color quantization using the fast *K*-means algorithm. Syst Comput Japan 31(8):33–40

Kaukoranta T, Fränti P, Nevalainen O (2000) A fast exact GLA based on code vector activity detection. IEEE Trans Image Process 9(8):1337–1342

Keysers D, Lampert CH, Breuel TM (2006) Color image dequantization by constrained diffusion. In: Proceedings of the SPIE/IS &T electronic imaging symposium, pp 605,803–1–605,803–10

Khalifa KB, Blaiech AG, Abadi M et al (2020) New hardware architecture for self-organizing map used for color vector quantization. J Circuits Syst Comput 29(1):2050,002

Kim N, Kehtarnavaz N (2005) DWT-based scene-adaptive color quantization. Real-Time Imaging 11(5–6):443–453

Kim TH, Ahn J, Choi MG (2007) Image dequantization: restoration of quantized colors. Comput Graph Forum 26(3):619–626

Kohonen T (1982) Self-organized formation of topologically correct feature maps. Biol Cybern 43:59–69

Kohonen T (1993) Things you haven't heard about the self-organizing map. In: Proceedings of the IEEE international conference on neural networks, pp 1147–1156

Kohonen T (2013) Essentials of the self-organizing map. Neural Netw 37:52–65

Koikkalainen P, Oja E (1990) Self-organizing hierarchical feature maps. In: Proceedings of the 1990 IJCNN international joint conference on neural networks, pp 279–284

Kok CW, Chan SC, Leung SH (1993) Color quantization by Fuzzy quantizer. In: Proceedings of the IS &T/SPIE's symposium on electronic imaging: science and technology, pp 235–242

Kolen JF, Hutcheson T (2002) Reducing the time complexity of the Fuzzy *C*-means algorithm. IEEE Trans Fuzzy Syst 10(2):263–267

Kruger A (1992) Reduction of computer-generated images. PhD thesis, University of Iowa

Kuehni RG (2003) Color space and its divisions: color order from antiquity to the present. Wiley, New York

Kuehni RG (2016) How many object colors can we distinguish? Color Res Appl 41(5):439–444

Kuhn GR, Oliveira MM, Fernandes LA (2008) An improved contrast enhancing approach for color-to-grayscale mappings. Vis Comput 24(7):505–514

Kurdthongmee W (2008) A novel hardware-oriented Kohonen SOM image compression algorithm and its FPGA implementation. J Syst Architect 54(10):983–994

Kurdthongmee W (2011) Utilization of a rational-based representation to improve the image quality of a hardware-based *K*-SOM quantizer. J Real-Time Image Proc 6(3):199–211

Kurdthongmee W (2016) A hardware centric algorithm for the best matching unit searching stage of the SOM-based quantizer and its FPGA implementation. J Real-Time Image Proc 12(1):71–80

Lai JZC, Liaw YC (2008) Improvement of the *K*-means clustering filtering algorithm. Pattern Recogn 41(12):3677–3681

Lampinen J, Oja E (1990) Fast computation of Kohonen self-organization. In: Soulié FF, Hérault J (eds) Neurocomputing: algorithms, architectures and applications. Springer, p 65–74

Lawrence RD, Almasi GS, Rushmeier HE (1999) A scalable parallel algorithm for self-organizing maps with applications to sparse data mining problems. Data Min Knowl Disc 3(2):171–195

Lee E, Schmidt M, Wright J (2017) Improved and simplified inapproximability for *K*-means. Inf Process Lett 120:40–43

Lempel A, Ziv J (1986) Compression of two-dimensional data. IEEE Trans Inf Theory 32(1):2–8

Leung CS, Ho TY, Xiao Y (2010) GPU color quantization. In: Engel W (ed) GPU Pro: advanced rendering techniques. A K Peters, p 3–13

Levkowitz H, Herman GT (1993) GLHS: a generalized lightness, Hue, and saturation color model. CVGIP 55(4):271–285

Li J, Mantiuk RK, Wang J, et al (2018) Hybrid-MST: a hybrid active sampling strategy for pairwise preference aggregation. In: Proceedings of the 32nd international conference on neural information processing systems, pp 3479–3489

Linde Y, Buzo A, Gray RM (1980) An algorithm for vector quantizer design. IEEE Trans Commun 28(1):84–95

Linhares JMM, Pinto PD et al (2008) The number of discernible colors in natural scenes. J Opt Soc Am A 25(12):2918–2924

Liu TS, Chang LW (1995) Fast color image quantization with error diffusion and morphological operations. Signal Process 43(3):293–303

Liu Q, Crispino M, Scheel I, et al (2019) Model-based learning from preference data. Annu Rev Stat Appl pp 329–354

Liu Y, Sun J, Yao Q, et al (2018) A scalable heterogeneous parallel SOM based on MPI/CUDA. In: Proceedings of the 10th Asian conference on machine learning, pp 264–279

Lloyd S (1982) Least squares quantization in PCM. IEEE Trans Inf Theory 28(2):129–136

Lo KC, Chan YH, Yu M (2003) Colour quantization by three-dimensional frequency diffusion. Pattern Recogn Lett 24(14):2325–2334

Lucic M, Bachem O, Krause A (2016) Strong coresets for hard and soft bregman clustering with applications to exponential family mixtures. In: Proceedings of the 19th international conference on artificial intelligence and statistics, pp 1–9

Luo MR (2002) Development of colour-difference formulae. Rev Prog Color Relat Top 32(1):28–39

Luzardo G, Aelterman J, Luong H, et al (2017) Real-time false-contours removal for inverse tone mapped HDR content. In: Proceedings of the 25th ACM international conference on multimedia, pp 1472–1479

MacQueen J (1967) Some methods for classification and analysis of multivariate observations. In: Proceedings of the 5th Berkeley symposium on mathematical statistics and probability, pp 281–297

Mahajan M, Nimbhorkar P, Varadarajan K (2009) The planar *K*-means problem is NP-hard. In: Proceedings of the 3rd international workshop on algorithms and computation, pp 274–285

Makhoul J, Roucos S, Gish H (1985) Vector quantization in speech coding. Proc IEEE 73(11):1551–1588

Mannos JL, Sakrison DJ (1974) The effects of a visual fidelity criterion of the encoding of images. IEEE Trans Inf Theory 20(4):525–536

Mantiuk RK, Tomaszewska A, Mantiuk R (2012) Comparison of four subjective methods for image quality assessment. Comput Graph Forum 31(8):2478–2491

Martínez-Verdú F, Perales E, Chorro E et al (2007) Computation and visualization of the MacAdam limits for any lightness, Hue angle, and light source. J Opt Soc Am A 24(6):1501–1515

Masaoka K, Berns RS, Fairchild MD et al (2013) Number of discernible object colors is a conundrum. J Opt Soc Am A 30(2):264–277

Masuyama S, Ibaraki T, Hasegawa T (1981) The computational complexity of the *M*-center problems on the plane. Trans IEICE Jpn 64(2):E-57-64

Matsumoto M, Nishimura T (1998) Mersenne twister: a 623-dimensionally equidistributed uniform pseudorandom number generator. ACM Trans Model Comput Simul 8(1):3–30

Max J (1960) Quantizing for minimum distortion. IRE Trans Inf Theory 6(1):7–12

Maystre L, Grossglauser M (2017) Just sort it! A simple and effective approach to active preference learning. In: Proceedings of the 34th international conference on machine learning, pp 2344–2353

Megiddo N, Supowit KJ (1984) On the complexity of some common geometric location problems. SIAM J Comput 13(1):182–196

Melgosa M, Huertas R (2004) Relative significance of the terms in the CIEDE2000 and CIE94 color-difference formulas. J Opt Soc Am A 21(12):2269–2275

Mignotte M (2008) Segmentation by fusion of histogram-based *K*-means clusters in different color spaces. IEEE Trans Image Process 17(5):780–787

Mikhailiuk A, Wilmot C, Pérez-Ortiz M, et al (2021) Active sampling for pairwise comparisons via approximate message passing and information gain maximization. In: Proceedings of the 25th international conference on pattern recognition, pp 2559–2566

Milvang O (1987) An adaptive algorithm for color image quantization. In: Proceedings of the 5th scandinavian conference on image analysis, pp 43–47

Mitsa T, Varkur KL (1993) Evaluation of contrast sensitivity functions for the formulation of quality measures incorporated in halftoning algorithms. In: Proceedings of the 1993 IEEE international conference on acoustics, speech, and signal processing, pp 301–304

Mohr AE (2002) Bit allocation in sub-linear time and the multiple-choice knapsack problem. In: Proceedings of the data compression conference, pp 352–361

Mojsilović A, Soljanin E (2001) Color quantization and processing by fibonacci lattices. IEEE Trans Image Process 10(11):1712–1725

Mojsilović A, Hu J, Soljanin E (2002) Extraction of perceptually important colors and similarity measurement for image matching, retrieval, and analysis. IEEE Trans Image Process 11(11):1238–1248

Monga V, Damera-Venkata N, Evans BL (2006) Color image halftoning. In: Lukac R, Plataniotis KN (eds) Color image processing: methods and applications. CRC Press, Boca Raton, pp 157–183

Montagne C, Lelandais S, Smolarz A et al (2006) Adaptive color quantization using the "Baker's transformation''. J Electron Imaging 15(2):023,015

Moon B, Jagadish HV, Faloutsos C et al (2001) Analysis of the clustering properties of the Hilbert space-filling curve. IEEE Trans Knowl Data Eng 13(1):124–141

Morovic J, Cheung V, Morovic P (2012) Why we don't know how many colors there are? In: Proceedings of the conference on colour in graphics, imaging, and vision, pp 49–53

Mulier FM, Cherkassky VS (1995) Statistical analysis of self-organization. Neural Netw 8(5):717–727

Müllner D (2013) fastcluster: fast hierarchical, agglomerative clustering routines for R and Python. J Stat Softw 53(9):1–18

Murtagh F, Legendre P (2014) Ward's hierarchical agglomerative clustering method: which algorithms implement Ward's criterion? J Classif 31(3):274–295

Murthy SK, Kasif S, Salzberg S (1994) A system for induction of oblique decision trees. J Artif Intell Res 2(1):1–32

Necaise RD (1998) Improvements to the color quantization process. PhD thesis, College of William & Mary

Nieves JL, Gomez-Robledo L, Chen YJ et al (2020) Computing the relevant colors that describe the color palette of paintings. Appl Opt 59(6):1732–1740

Nikolaou N, Papamarkos N (2009) Color reduction for complex document images. Int J Imaging Syst Technol 19(1):14–26

Nolle L, Schaefer G (2007) Colour map design through optimization. Eng Optim 39(3):327–343

Ohta YI, Kanade T, Sakai T (1980) Color information for region segmentation. Comput Graphics Image Process 13(3):222–241

Omran MG, Engelbrecht AP, Salman A (2005) A color image quantization algorithm based on particle swarm optimization. Informatica 29(3):261–269

Oppenheim AV, Weinstein CJ (1972) Effects of finite register length in digital filtering and the fast Fourier transform. Proc IEEE 60(8):957–976

Orchard M, Bouman C (1991) Color quantization of images. IEEE Trans Signal Process 39(12):2677–2690

Ortiz-Jaramillo B, Kumcu A, Platisa L et al (2019) Evaluation of color differences in natural scene color images. Signal Process 71:128–137

Otsu N (1979) A threshold selection method from gray-level histograms. IEEE Trans Syst Man Cybern 9(1):62–66

Özdemir D, Akarun L (2002) Fuzzy algorithm for color quantization of images. Pattern Recogn 35(8):1785–1791

Ozkan I, Turksen IB (2007) Upper and lower values for the level of fuzziness in FCM. Inf Sci 177(23):5143–5152

Paeth AW (1990) Mapping RGB triples onto four bits. In: Glassner A (ed) Graphics gems I. Morgan Kaufmann, pp 233–245

Paeth AW (1991) Mapping RGB triples onto 16 distinct values. In: Arvo J (ed) Graphics gems II. Morgan Kaufmann, pp 143–146

Pal NR, Bezdek JC (2002) Complexity reduction for "large image'' processing. IEEE Trans Syst Man Cybern 32(5):598–611

Palomo EJ, Domínguez E (2014) Hierarchical color quantization based on self-organization. J Math Imaging Vis 49(1):1–19

Papamarkos N, Atsalakis AE, Strouthopoulos CP (2002) Adaptive color reduction. IEEE Trans Syst Man Cybern 32(1):44–56

Park JH, Kim SH, Lee JC, et al (2022) Scalable color quantization for task-centric image compression. ACM Trans Multimed Comput Commun Appl

Parker JK, Hall LO (2014) Accelerating Fuzzy-*C* means using an estimated subsample size. IEEE Trans Fuzzy Syst 22(5):1229–1244

Pedersen M (2015) Evaluation of 60 full-reference image quality metrics on the CID:IQ. In: Proceedings of the 2015 IEEE international conference on image processing, pp 1588–1592

Pedersen M, Hardeberg JY (2012) Full-reference image quality metrics: classification and evaluation. Found Trends Comput Graph Vis 7(1):1–80

Pei SC, Cheng CM (1995) Dependent scalar quantization of color images. IEEE Trans Circuits Syst Video Technol 5(2):124–139

Pei SC, Lo YS (1998) Color image compression and limited display using self-organization Kohonen map. IEEE Trans Circuits Syst Video Technol 8(2):191–205

Pei SC, Chuang YT, Chuang WH (2006) Effective palette indexing for image compression using self-organization of Kohonen feature map. IEEE Trans Image Process 15(9):2493–2498

Pérez-Delgado ML (2015) Colour quantization with ant-tree. Appl Soft Comput 36:656–669

Pérez-Delgado ML (2018) Artificial ants and fireflies can perform colour quantisation. Appl Soft Comput 73:153–177

Pérez-Delgado ML (2019) The color quantization problem solved by swarm-based operations. Appl Intell 49:2482–2514

Pérez-Delgado ML (2020) Color quantization with particle swarm optimization and artificial ants. Soft Comput 24:4545–4573

Pérez-Delgado ML (2021) Revisiting the iterative ant-tree for color quantization algorithm. J Vis Commun Image Represent 78(103):180

Pérez-Delgado ML, Gallego JAR (2019) A hybrid color quantization algorithm that combines the greedy orthogonal Bi-partitioning method with artificial ants. IEEE Access 7:128,714-128,734

Pérez-Delgado ML, Gallego JAR (2020) A two-stage method to improve the quality of quantized images. J Real-Time Image Proc 17(3):581–605

Pérez-Ortiz M, Mantiuk RK (2017) A practical guide and software for analysing pairwise comparison experiments. https://arxiv.org/abs/1712.03686

Pérez-Ortiz M, Mikhailiuk A, Zerman E et al (2019) From pairwise comparisons and rating to a unified quality scale. IEEE Trans Image Process 29:1139–1151

Phung SL, Bouzerdoum A, Chai D (2005) Skin segmentation using color pixel classification: analysis and comparison. IEEE Trans Pattern Anal Mach Intell 27(1):148–154

Pointer MR, Attridge GG (1998) The number of discernible colours. Color Res Appl 23(1):52–54

Ponomarenko N, Jin L, Ieremeiev O et al (2015) Image database TID2013: peculiarities, results and perspectives. Signal Process 30:57–77

Poynton C, Funt B (2014) Perceptual uniformity in digital image representation and display. Color Res Appl 39(1):6–15

Pratt WK (1970) Spatial transform coding of color images. IEEE Trans Commun Technol 19(6):980–992

Press WH, Teukolsky SA, Vetterling WT et al (2007) Numerical recipes: the art of scientific computing, 3rd edn. Cambridge, Cambridge University Press

Puzicha J, Held M, Ketterer J et al (2000) On spatial quantization of color images. IEEE Trans Image Process 9(4):666–682

Ra SW, Kim JK (1993) A fast mean-distance-ordered partial codebook search algorithm for image vector quantization. IEEE Trans Circuits Syst II 40(9):576–579

Ramanath R, Snyder WE, Yoo Y et al (2005) Color image processing pipeline. IEEE Signal Process Mag 22(1):34–43

Ramella G (2021) Evaluation of quality measures for color quantization. Multimed Tools Appl 80:32,975-33,009

Ramella G, di Baja GS (2013) A new technique for color quantization based on histogram analysis and clustering. Int J Pattern Recognit Artif Intell 27(3):1360,006

Ramstad TA (1982) Sub-band coder with a simple adaptive bit-allocation algorithm: a possible candidate for digital mobile telephony? In: Proceedings of the IEEE international conference on acoustics, speech, and signal processing, pp 203–207

Reitan PJ (1998) 3D visualization of color image histograms. Comput Netw ISDN Syst 30(20–21):2025–2035

Reitan PJ (1999) Hybrid approaches to color image quantization. PhD thesis, University of Maryland, Baltimore County

Ren M, Wang Z, Jiang J (2019) A self-adaptive FCM for the optimal Fuzzy weighting exponent. Int J Comput Intell Appl 18(2):1950,008

Ritter HJ, Schulten K (1988) Kohonen's self-organizing maps: exploring their computational capabilities. In: Proceedings of the IEEE 1988 international conference on neural networks, pp 109–116

Robbins H, Monro S (1951) A stochastic approximation method. Ann Math Stat 22(3):400–407

Robertson PK (1988) Visualizing color Gamuts: a user interface for the effective use of perceptual color spaces in data displays. IEEE Comput Graph Appl 8(5):50–64

Rumelhart D, Zipser D (1985) Feature discovery by competitive learning. Cogn Sci 9(1):75–112

Ruspini EH (1970) Numerical methods for Fuzzy clustering. Inf Sci 2(3):319–350

Sabin MJ, Gray RM (1984) Product code vector quantizers for waveform and voice coding. IEEE Trans Acoust Speech Signal Process 32(3):474–488

Safavian SR, Landgrebe D (1991) A survey of decision tree classifier methodology. IEEE Trans Syst Man Cybern 21(3):660–674

Sakrison DJ, Algazi VR (1971) Comparison of line-by-line and two-dimensional encoding of random images. IEEE Trans Inf Theory 17(4):386–398

Schaefer G (2014) Soft computing-based colour quantisation. EURASIP J Image Video Process 2014(1):1–9

Schaefer G, Nolle L (2015) A hybrid color quantization algorithm incorporating a human visual perception model. Comput Intell 31(4):684–698

Schaefer G, Zhou H (2009) Fuzzy clustering for colour reduction in images. Telecommun Syst 40(1):17–25

Schaefer G, Zhou H, Celebi ME et al (2011) Rough colour quantisation. Int J Hybrid Int Syst 8(1):25–30

Schaefer G, Hu Q, Zhou H et al (2012) Rough *C*-means and Fuzzy rough *C*-means for colour quantisation. Fund Inform 119(1):113–120

Scheunders P (1997) A genetic *C*-means clustering algorithm applied to color image quantization. Pattern Recogn 30(6):859–866

Schmidl TM, Cosman PC, Gray RM (1993) Unbalanced non-binary tree-structured vector quantizers. In: Proceedings of the 27th asilomar conference on signals, systems and computers, pp 1519–1523

Schmitz BE, Stevenson RL (1995) Color palette restoration. Graph Models Image Process 57(5):409–419

Schreiber T (1991) A voronoi diagram based adaptive *K*-means-type clustering algorithm for multidimensional weighted data. In: Proceedings of the international workshop on computational geometry, pp 265–275

Schwämmle V, Jensen ON (2010) A simple and fast method to determine the parameters for Fuzzy *C* -means cluster analysis. Bioinformatics 26(22):2841–2848

Serrano C, Lazo M, Serrano A et al (2022) Clinically inspired skin lesion classification through the detection of dermoscopic criteria for basal cell carcinoma. J Imaging 8(7):197

Shafer SA, Kanade T (1987) Color Vision. In: Shapiro SC (ed) Encyclopedia of artificial intelligence, vol 1. Wiley, New York, pp 124–131

Sharma G, Vrhel MJ, Trussell HJ (1998) Color imaging for multimedia. Proc IEEE 86(6):1088–1108

Sharma G, Wu W, Dalal EN (2005) The CIEDE2000 color-difference formula: implementation notes, supplementary test data, and mathematical observations. Color Res Appl 30(1):21–30

Sheikh HR, Sabir MF, Bovik AC (2006) A statistical evaluation of recent full reference image quality assessment algorithms. IEEE Trans Image Process 15(11):3440–3451

Sheikholeslami G, Chatterjee S, Zhang A (1998) WaveCluster: a multi-resolution clustering approach for very large spatial databases. In: Proceedings of the 24th international conference on very large data bases, pp 428–439

Shufelt JA (1997) Texture analysis for enhanced color image quantization. Graph Models Image Process 59(3):149–163

Silverstein DA, Farrell JE (2001) Efficient method for paired comparison. J Electron Imaging 10(2):394–398

Sobol' IM, Asotsky D, Kreinin A et al (2011) Construction and comparison of high-dimensional Sobol' generators. Wilmott 56:64–79

Soljanin E (2002) Writing sequences on the plane. IEEE Trans Inf Theory 48(6):1344–1354

Song Q, Su GM, Cosman PC (2020) Efficient debanding filtering for inverse tone mapped high dynamic range videos. IEEE Trans Circuits Syst Video Technol 30(8):2575–2589

Sproull RF (1991) Refinements to nearest-neighbor searching in *K*-dimensional trees. Algorithmica 6(4):579–589

Stevens RJ, Lehar AF, Preston FH (1983) Manipulation and presentation of multidimensional image data using the Peano scan. IEEE Trans Pattern Anal Mach Intell 5(5):520–526

Stockham TG Jr (1972) Image processing in the context of a visual model. Proc IEEE 60(7):828–842
Stokes M, Fairchild MD, Berns RS (1992) Precision requirements for digital color reproduction. ACM Trans Graph 11(4):406–422
Streijl RC, Winkler S, Hands DS (2016) Mean opinion score (MOS) revisited: methods and applications. Multimed Syst 22(2):213–227
Su MC, Chang HT (2000) Fast self-organizing feature map algorithm. IEEE Trans Neural Netw 11(3):721–733
Su T, Dy JG (2007) In search of deterministic methods for initializing *K*-means and Gaussian mixture clustering. Intell Data Anal 11(4):319–338
Sudha N, Srikanthan T, Mailachalam B (2003) A VLSI architecture for 3-D self-organizing map based color quantization and its FPGA implementation. J Syst Architect 48(11–12):337–352
Szilágyi L, Benyó Z, Szilágyi SM, et al (2003) MR brain image segmentation using an enhanced Fuzzy *C*-means algorithm. In: Proceedings of the 25th annual international conference of the IEEE engineering in medicine and biology society, pp 724–726
Szilágyi L, Dénesi G, Kovács L, et al (2014) Comparison of various improved-partition Fuzzy *C*-means clustering algorithms in fast color reduction. In: Proceedings of the 2014 IEEE 12th international symposium on intelligent systems and informatics, pp 197–202
Szilágyi L, Dénesi G, Enăchescu C (2016) Fast color quantization via Fuzzy clustering. In: Proceedings of the international conference on neural information processing, pp 95–103
Taşdizen T, Akarun L, Ersoy C (1998) Color quantization with genetic algorithms. Signal Process 12(1):49–57
Thomas SW (1991) Efficient inverse color map computation. In: Arvo J (ed) Graphics gems II. Academic Press, Cambridge, pp 116–125
Thompson S, Celebi ME, Buck KH (2020) Fast color quantization using MacQueen's *K*-means algorithm. J Real-Time Image Proc 17(5):1609–1624
Trapp M, Pasewaldt S, Döllner J (2019) Techniques for GPU-based color quantization. In: Proceedings of the 27th international conference in central Europe on computer graphics, visualization and computer vision, pp 81–87
Tseng HW, Ding WB (2012) Reversible data hiding scheme for colour images based on pixel clustering and histogram shifting. Imaging Sci J 60(1):47–53
Tsukida K, Gupta MR (2011) How to analyze paired comparison data. Tech. Rep. UWEETR-2011-0004, University of Washington
Tu Z, Lin J, Wang Y et al (2020) Adaptive debanding filter. IEEE Signal Process Lett 27:1715–1719
Turnbull D, Elkan C (2005) Fast recognition of musical genres using RBF networks. IEEE Trans Knowl Data Eng 17(4):580–584
Turner H, Firth D (2012) Bradley-Terry models in R: the BradleyTerry2 package. J Stat Softw 48(9):1–21
Uchiyama T, Arbib MA (1994a) An algorithm for competitive learning in clustering problems. Pattern Recogn 27(10):1415–1421
Uchiyama T, Arbib MA (1994b) Color image segmentation using competitive learning. IEEE Trans Pattern Anal Mach Intell 16(12):1197–1206
Ueda Y, Koga T, Suetake N et al (2017) Color quantization method based on principal component analysis and linear discriminant analysis for Palette-based image generation. Opt Rev 24(6):741–756
Urban P, Rosen MR, Berns RS et al (2007) Embedding non-euclidean color spaces into euclidean color spaces with minimal isometric disagreement. J Opt Soc Am A 24(6):1516–1528
Valenzuela G, Celebi ME, Schaefer G (2018) Color quantization using coreset sampling. In: Proceedings of the 2018 IEEE international conference on systems, man, and cybernetics, pp 2096–2101
Van Hulle MM (2012) Self-organizing maps. In: Rozenberg G, Bäck T, Kok JN (eds) Handbook of natural computing. Springer, Berlin, pp 585–622
Van Ness JW (1973) Admissible clustering procedures. Biometrika 60(2):422–424
Vattani A (2009) *K*-means requires exponentially many iterations even in the plane. In: Proceedings of the 25th annual symposium on computational geometry, pp 324–332
Velho L, Gomez J, Sobreiro MVR (1997) Color image quantization by pairwise clustering. In: Proceedings of the 10th Brazilian symposium on computer graphics and image processing, pp 203–210
Verevka O, Buchanan JW (1995) Local *K*-means algorithm for colour image quantization. In: Proceedings of the graphics/vision interface conference, pp 128–135
Vuong J, Kaur S, Heinrich J et al (2018) Versus—a tool for evaluating visualizations and image quality using a 2AFC methodology. Visual Inf 2(4):225–234
Wan X, Kuo CCJ (1998) A new approach to image retrieval with hierarchical color clustering. IEEE Trans Circuits Syst Video Technol 8(5):628–643

Wan SJ, Wong SKM, Prusinkiewicz P (1988) An algorithm for multidimensional data clustering. ACM Trans Math Softw 14(2):153–162

Wan SJ, Prusinkiewicz P, Wong SKM (1990) Variance-based color image quantization for frame buffer display. Color Res Appl 15:52–58

Wang Z, Bovik AC (2009) Mean squared error: love it or leave it? A new look at signal fidelity measures. IEEE Signal Process Mag 26(1):98–117

Wang Z, Simoncelli EP, Bovik AC (2003) Multiscale structural similarity for image quality assessment. In: Proceedings of the 37th Asilomar conference on signals, systems and computers, pp 1398–1402

Wang Z, Bovik AC, Sheikh HR et al (2004) Image quality assessment: from error visibility to structural similarity. IEEE Trans Image Process 13(4):600–612

Wang CH, Lee CN, Hsieh CH (2007) Sample-size adaptive self-organization map for color images quantization. Pattern Recogn Lett 28(13):1616–1629

Wang Y, Pan Z, Li R (2018) Performance re-evaluation on "codewords distribution-based optimal combination of equal-average equal-variance equal-norm nearest neighbor fast search algorithm for vector quantization encoding. IEEE Trans Image Process 27(2):718–720

Wang Y, Huang H, Wang C, et al (2019) GIF2Video: color dequantization and temporal interpolation of GIF images. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 1419–1428

Wang S, Sun Y, Bao Z (2020) On the efficiency of K-means clustering: evaluation, optimization, and algorithm selection. Proc VLDB Endow 14(2):163–175

Ward J (1963) Hierarchical grouping to optimize an objective function. J Am Stat Assoc 58(301):236–244

Watanabe T (1988) A fast algorithm for color image quantization using only 256 colors. Syst Comput Jpn 19(3):64–72

Wen Q, Celebi ME (2011) Hard vs. Fuzzy C-means clustering for color quantization. EURASIP J Adv Signal Process 1:118–129

Wickelmaier F, Schmid C (2004) A Matlab function to estimate choice model parameters from paired-comparison data. Behav Res Methods Instrum Comput 36(1):29–40

Williams WT (1971) Principles of clustering. Annu Rev Ecol Syst 2:303–326

Wittek P, Gao SC, Lim IS et al (2017) somoclu: an efficient parallel library for self-organizing maps. J Stat Softw 78(9):1–21

Wu X (1991a) Efficient statistical computations for optimal color quantization. In: Arvo J (ed) Graphics gems II. Academic Press, Cambridge, pp 126–133

Wu X (1991b) Optimal quantization by matrix searching. J Algorithms 12(4):663–673

Wu X (1992a) Color quantization by dynamic programming and principal analysis. ACM Trans Graph 11(4):348–372

Wu X (1992b) Statistical colour quantization for minimum distortion. In: Falcidieno B, Herman I, Pienovi C (eds) Computer graphics and mathematics. Springer, Berlin, pp 189–202

Wu KL (2012) Analysis of parameter selections for Fuzzy C-means. Pattern Recogn 45(1):407–415

Wu X, Witten IH (1985) A fast K-means type clustering algorithm. Tech. Rep. 85/197/10, University of Calgary

Wu KL, Yang MS (2006) Alternative learning vector quantization. Pattern Recogn 39(3):351–362

Wu X, Zhang K (1993) Quantizer monotonicities and globally optimal scalar quantizer design. IEEE Trans Inf Theory 39(3):1049–1053

Wu X, Kumar V, Quinlan JR et al (2008) Top 10 algorithms in data mining. Knowl Inf Syst 14(1):1–37

Xiang Z (1997) Color image quantization by minimizing the maximum intercluster distance. ACM Trans Graph 16(3):260–276

Xiang Z (2018) Color quantization. In: Gonzalez TF (ed) Handbook of approximation algorithms and metaheuristics, 2nd edn. CRC Press, Boca Raton, pp 691–709

Xiang Z, Joy G (1994) Color image quantization by agglomerative clustering. IEEE Comput Graph Appl 14(3):44–48

Xiao Y, Feng RB, Han ZF et al (2015) GPU accelerated self-organizing map for high dimensional data. Neural Process Lett 41(3):341–355

Xie YF, Liu JH, Zhang CF et al (2016) Codewords distribution-based optimal combination of equal-average equal-variance equal-norm nearest neighbor fast search algorithm for vector quantization encoding. IEEE Trans Image Process 25(12):5806–5813

Xu Q, Jiang T, Yao Y, et al (2011) Random partial paired comparison for subjective video quality assessment via HodgeRank. In: Proceedings of the 19th ACM international conference on multimedia, pp 393–402

Yager RR, Filev DP (1994) Approximate clustering via the mountain method. IEEE Trans Syst Man Cybern 24(8):1279–1284

Yair E, Zeger K, Gersho A (1992) Competitive learning and soft competition for vector quantizer design. IEEE Trans Signal Process 40(2):294–309

Yang CY, Lin JC (1996) RWM-Cut for color image quantization. Comput Graph 20(4):577–588

Yang CK, Tsai WH (1998) Color image compression using quantization, thresholding, and edge detection techniques all based on the moment-preserving principle. Pattern Recogn Lett 19(2):205–215

Yang MS, Wu KL, Hsieh JN et al (2008) Alpha-cut implemented Fuzzy clustering algorithms and switching regressions. IEEE Trans Syst Man Cybern Part B (Cybernetics) 38(3):588–603

Ye P, Doermann D (2014) Active sampling for subjective image quality assessment. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 4249–4256

Yoo I, Luo X, Wang Y, et al (2020) GIFnets: differentiable GIF encoding framework. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 14473–14482

Yu CH, Chen SY (2006) Universal colour quantisation for different colour spaces. IEE Proc Vis Image Signal Process 153(4):445–455

Yu MP, Lo KK (2003) Contextual algorithm for color quantization. J Electron Imaging 12(3):442–447

Yu J, Yang MS (2005) Optimality test for generalized FCM and its application to parameter selection. IEEE Trans Fuzzy Syst 13(1):164–176

Yu J, Cheng Q, Huang H (2004) Analysis of the weighting exponent in the FCM. IEEE Trans Syst Man Cybern Part B (Cybernetics) 34(1):634–639

Yuan G, Goldberg M (1988) A sequential initialization technique for vector quantizer design. Pattern Recogn Lett 7(3):157–161

Zhang X, Wandell BA (1997) A spatial extension of CIELAB for digital color-image reproduction. J Soc Inform Display 5(1):61–63

Zhang X, Wandell BA (1998) Color image fidelity metrics evaluated using image distortion maps. Signal Process 70(3):201–214

Zhang T, Ramakrishnan R, Livny M (1997) BIRCH: a new data clustering algorithm and its applications. Data Min Knowl Disc 1(2):141–182

Zhao Y, Sheong FK, Sun J et al (2013) A fast parallel clustering algorithm for molecular simulation trajectories. J Comput Chem 34(2):95–104

Zhou K, Yang S (2019) Fuzzifier selection in Fuzzy *C*-means from cluster size distribution perspective. Informatica 30(3):613–628

Zhou H, Schaefer G, Sadka A et al (2009) Anisotropic mean shift based Fuzzy *C*-means segmentation of dermoscopy images. IEEE J Select Top Signal Proces 3(1):26–34