J Glob Optim (2007) 37:137–157
DOI 10.1007/s10898-006-9041-0

ORIGINAL PAPER

Application of the cross-entropy method to clustering and vector quantization

Dirk P. Kroese · Reuven Y. Rubinstein · Thomas Taimre

Received: 28 April 2005 / Accepted: 30 April 2006 /
Published online: 10 August 2006

Abstract We apply the cross-entropy (CE) method to problems in clustering and vector quantization. The CE algorithm for clustering involves the following iterative steps: (a) generate random clusters according to a specified parametric probability distribution, (b) update the parameters of this distribution according to the Kullback–Leibler cross-entropy. Through various numerical experiments, we demonstrate the high accuracy of the CE algorithm and show that it can generate near-optimal clusters for fairly large data sets. We compare the CE method with well-known clustering and vector quantization methods such as K-means, fuzzy K-means and linear vector quantization, and apply each method to benchmark and image analysis data.

Keywords Cross-entropy method · Clustering · Vector quantization · Simulation · Global optimization

1 Introduction

Clustering and vector quantization are concerned with the grouping of unlabeled "feature" vectors into clusters, such that samples within a cluster are more similar to each other than samples belonging to different clusters. Usually, it is assumed that the number of clusters is known in advance, but otherwise no prior information is given about the data. Applications of clustering and vector quantization can be found in the areas of communication, data compression and storage, database searching,

D. P. Kroese (✉) · T. Taimre
Department of Mathematics, The University of Queensland, Brisbane 4072, Australia
e-mail: kroese@maths.uq.edu.au

R. Y. Rubinstein
Faculty of Industrial Engineering and Management, Technion, Haifa, Israel
e-mail: ierrr01@ie.technion.ac.il

T. Taimre
e-mail: ttaimre@maths.uq.edu.au

pattern matching, and object recognition. It is out of the scope of this paper to discuss the vast amount of literature on clustering. For details on clustering applications and algorithms we refer to [3, 11, 17, 18, 23, 28] and references therein.

In mathematical terms, the clustering problem reads as follows: given a dataset $\mathcal{Z} = \{\mathbf{z}_1, \ldots, \mathbf{z}_n\}$ of points in some d-dimensional Euclidean space, partition the data into K clusters $R_1, \ldots, R_K$ (with $R_i \cap R_j = \emptyset$, for $i \neq j$, and $\cup_j R_j = \mathcal{Z}$), such that some empirical *loss function* (performance measure) is minimized. A typical loss function is

$$\sum_{j=1}^{K} \sum_{\mathbf{z} \in R_j} ||\mathbf{z} - \mathbf{c}_j||^2, \tag{1}$$

where $\mathbf{c}_j$ represents the cluster center or *centroid* of cluster R_j. The objective is to find a $(K \cdot d)$-dimensional vector of centroids $\mathbf{c} = (\mathbf{c}_1, \ldots, \mathbf{c}_K)$ and the corresponding partition $\{R_1, \ldots, R_K\}$ that minimizes (1). This definition also combines both the encoding and decoding steps in *vector quantization* [28]. Namely, we wish to "quantize" or "encode" the vectors in $\mathcal{Z}$ in such a way that each vector is represented by one of K *source vectors* $\mathbf{c}_1, \ldots, \mathbf{c}_K$, such that the loss (1) of this representation is minimized.

Most well-known clustering and vector quantization methods update the vector of centroids, starting from some initial choice and using iterative (typically gradient-based) procedures. It is important to realize that in this case (1) is seen as a function of the centroids, where each point $\mathbf{z}$ is assigned to the nearest centroid, thus determining the clusters. It is well known that this type of problem—optimization with respect to the centroids—is highly multi-extremal and, depending on the initial clusters, the gradient-based procedures converge to a *local minimum* rather than a global minimum. A standard heuristic is the *K-means* (KM) algorithm [28]; a useful modification is the *fuzzy K-means* (FKM) algorithm [5]. Another well-known method is the *linear vector quantization* (LVQ) algorithm. A detailed description of various types of clustering methods, including those used here, may be found in [11] and the accompanying [27].

An alternative approach to optimizing (1) is to view the loss function as a function of the clusters, rather than the centroids. More precisely, denoting $\mathbf{x} = (x_1, \ldots, x_n)$ the *cluster vector*, with $x_i = j$ if $\mathbf{z}_i \in R_j$, and letting $\mathbf{z}_{ij} = I_{\{x_i = j\}}\, \mathbf{z}_i$ (here I denotes the indicator function), we can write (1) as

$$\sum_{j=1}^{K} \sum_{i=1}^{n} I_{\{x_i=j\}}\, ||\mathbf{z}_{ij} - \mathbf{c}_j||^2 \ , \tag{2}$$

where the centroids are determined as

$$\mathbf{c}_j = \frac{1}{n_j} \sum_{i=1}^{n} \mathbf{z}_{ij} \tag{3}$$

with $n_j = \sum_{i=1}^{n} I_{\{x_i=j\}}$ being the number of points in the jth cluster. The two viewpoints above are mathematically equivalent, in the sense that they yield the same global solution. However, a partition which defines clusters that are spatially overlapping does not correspond to a set of centroids; hence the viewpoints are not identical.

In this paper, we introduce the Cross Entropy (CE) method as an alternative to KM, FKM, and LVQ, for solving the clustering problem. It is not our intention to repeat the principles of the CE method in detail in this paper. For this, we advise the reader to refer to the CE tutorial [9]—which is also available on-line from the CE

Homepage `www.cemethod.org`—and the CE monograph [22]. However, a short summary of the basic concepts, including the main CE algorithm, in given in the Sect. 2. Readers completely new to the CE method may find the example Matlab code (see Appendix) instructive.

In correspondence with the two scenarios discussed above, we present two different settings of the main CE Algorithm. Our first setting is based on reducing the clustering problem to a combinatorial partition problem with n nodes and K partitions, while in the second setting, we view (1) as a continuous multi-extremal optimization problem. That is, in the first setting the decision variable is the cluster vector $\mathbf{x}$ and in the second setting it is the vector of centroids $(\mathbf{c}_1, \ldots, \mathbf{c}_K)$. Both settings are treated in [22].

The purpose of this paper is (a) to show how the CE method can be easily applied to difficult optimization problems such as they occur in clustering analysis; (b) to compare CE with standard clustering algorithms, and show that it can produce more accurate answers. We do not claim that CE is the final answer to all clustering problems: sometimes speed is more important than accuracy, and sometimes the size of the data set can be prohibitively large for CE. However, the strength of the CE algorithm lies in its simplicity, accuracy, adaptability, and ultimately in the fundamental idea that global optimization is about optimizing sampling distributions.

For more references on CE for solving combinatorial and continuous multi-extremal problems (see [19] and [21]). Alternative well-known heuristics, capable of handling the clustering problem, are tabu search [13], genetic algorithms [14], nested partitioning [25] and the Ant Colony Optimization (ACO) meta-heuristic [10]. The recent monograph [26] gives a good overview of stochastic search and optimization. A recent (non-stochastic) global optimization approach to the clustering problem is [24]; a recent fuzzy clustering algorithm is [29]; a recent meta-heuristic is Variable Neighborhood Decomposition Search [16], and a recent local optimization technique for the problem is J-means [15].

A fundamentally different approach to clustering analysis is to assume that the data comes from a mixture of (usually Gaussian) distributions; and the objective is to estimate the parameters of this mixture by maximizing the likelihood function. In [6] a CE approach to global likelihood optimization for such mixture models is given, with good results when compared with the EM algorithm [20].

The rest of the paper is organized as follows. In Sect. 2, we recapitulate, from [22], the main CE Algorithm for solving combinatorial and continuous multi-extremal optimization problems. In Sects. 3 and 4, we present the clustering problem as a combinatorial partition and a continuous multi-extremal problem, respectively. In Sect. 5, numerical results are given for several benchmark problems and for a real problem concerning texture images. Here, we also compare the accuracy of the CE method with the well-known KM, FKM, and LVQ algorithms, and show how CE outperforms these standard methods with regard to finding the global optimum. In Sect. 6, we present the conclusions and give directions for future research.

2 The CE method

The main idea of the CE method for optimization can be stated as follows: Suppose, we wish to maximize some performance function $S(\mathbf{x})$ over all states $\mathbf{x}$ in some set $\mathcal{X}$. Let us denote the maximum by γ^*, thus

$$\gamma^* = \max_{\mathbf{x}\in\mathcal{X}} S(\mathbf{x}) \,. \tag{4}$$

First, we randomize our deterministic problem by defining a family of *sampling pdfs* $\{f(\cdot;\mathbf{v}), \mathbf{v} \in \mathcal{V}\}$ on the set $\mathcal{X}$. We assume that this family includes the degenerate density at $\mathbf{x}^*$, say $f(\cdot;\mathbf{v}^*)$. Next, we associate with (4) estimation problems of the form

$$\ell(\gamma) = \mathbb{P}_{\mathbf{u}}(S(\mathbf{X}) \geqslant \gamma) = \mathbb{E}_{\mathbf{u}} I_{\{S(\mathbf{X})\geqslant\gamma\}}. \tag{5}$$

Here, $\mathbf{X}$ is a random vector with pdf $f(\cdot;\mathbf{u})$, for some $\mathbf{u} \in \mathcal{V}$ and $\gamma \in \mathbb{R}$. This is called the *associated stochastic problem* (ASP).

Having defined an ASP, the goal of the CE algorithm is to generate a sequence of tuples $\{(\gamma_t, \mathbf{v}_t)\}$, that converge quickly to a small neighborhood of the optimal tuple $(\gamma^*, \mathbf{v}^*)$. More specifically, we choose some initial $\mathbf{v}_0$ and a not too small ϱ, say $\varrho = 10^{-2}$, and proceed as follows:

1. *Adaptive updating of* γ_t. For a fixed $\mathbf{v}_{t-1}$, let γ_t be the $(1-\varrho)$-quantile of $S(\mathbf{X})$ under $\mathbf{v}_{t-1}$. A simple estimator $\widehat{\gamma}_t$ of γ_t can be obtained by drawing a random sample $\mathbf{X}_1,\ldots,\mathbf{X}_N$ from $f(\cdot;\mathbf{v}_{t-1})$ and evaluating the sample $(1-\varrho)$-quantile of the performances:
$$\widehat{\gamma}_t = S_{(\lceil(1-\varrho)N\rceil)}, \tag{6}$$
where $S_{(k)}$ denotes the kth order statistic of $\{S(\mathbf{X}_i)\}$.
2. *Adaptive updating of* $\mathbf{v}_t$. For fixed γ_t and $\mathbf{v}_{t-1}$, derive $\mathbf{v}_t$ from the solution of the program
$$\max_{\mathbf{v}} D(\mathbf{v}) = \max_{\mathbf{v}} \mathbb{E}_{\mathbf{v}_{t-1}} I_{\{S(\mathbf{X})\geqslant\gamma_t\}} \ln f(\mathbf{X};\mathbf{v}) \,. \tag{7}$$
The stochastic counterpart of (7) is as follows: for fixed $\widehat{\gamma}_t$ and $\widehat{\mathbf{v}}_{t-1}$, derive $\widehat{\mathbf{v}}_t$ from the following program
$$\max_{\mathbf{v}} \widehat{D}(\mathbf{v}) = \max_{\mathbf{v}} \frac{1}{N}\sum_{i=1}^{N} I_{\{S(\mathbf{X}_i)\geqslant\widehat{\gamma}_t\}} \quad \ln f(\mathbf{X}_i;\mathbf{v}) \,. \tag{8}$$

Remark 2.1 (*Smoothed Updating*) Instead of updating the parameter vector $\mathbf{v}$ directly via the solution of (8) we use the following *smoothed* version

$$\widehat{\mathbf{v}}_t = \alpha\tilde{\mathbf{v}}_t + (1-\alpha)\widehat{\mathbf{v}}_{t-1}, \tag{9}$$

where $\tilde{\mathbf{v}}_t$ is the parameter vector obtained from the solution of (8), and α is called the *smoothing parameter*, with (typically) $0.7 < \alpha \leq 1$. Clearly, for $\alpha = 1$, we have our original updating rule. The reason for using the smoothed (9) instead of the original updating rule is twofold: (a) to smooth out the values of $\widehat{\mathbf{v}}_t$, (b) to reduce the probability that some component $\widehat{v}_{t,i}$ of $\widehat{\mathbf{v}}_t$ will be zero or one at the first few iterations. This is particularly important when $\widehat{\mathbf{v}}_t$ is a vector or matrix of *probabilities*. Note that for $0 < \alpha < 1$ we always have that $\widehat{v}_{t,i} > 0$, while for $\alpha = 1$ one might have (even at the first iterations) that either $\widehat{v}_{t,i} = 0$ or $\widehat{v}_{t,i} = 1$ for some indices i. As result, the algorithm could converge to a wrong solution.

Thus, the main CE optimization algorithm, which includes smoothed updating of parameter vector $\mathbf{v}$ can be summarized as follows:

Algorithm 2.1 (*Main CE Algorithm for Optimization*)

1. Choose some $\widehat{\mathbf{v}}_0$. Set $t = 1$ (level counter).
2. Generate a sample $\mathbf{X}_1, \ldots, \mathbf{X}_N$ from the density $f(\cdot; \mathbf{v}_{t-1})$ and compute the sample $(1 - \varrho)$-quantile $\widehat{\gamma}_t$ of the performances according to (6).
3. Use the **same** sample $\mathbf{X}_1, \ldots, \mathbf{X}_N$ and solve the stochastic program (8). Denote the solution by $\widetilde{\mathbf{v}}_t$.
4. Apply (9) to smooth out the vector $\widetilde{\mathbf{v}}_t$.
5. If for some $t \geq d$, say $d = 5$,

$$\widehat{\gamma}_t = \widehat{\gamma}_{t-1} = \cdots = \widehat{\gamma}_{t-d}, \tag{10}$$

then **stop** (let T denote the final iteration); otherwise set $t = t + 1$ and reiterate from step 2.

Note that the parameter $\mathbf{v}$ is updated in (8) only on the basis of the $(1 - \varrho)\%$ best samples. These are called the *elite samples*. The main ingredients of any CE algorithm are as follows.

- *Trajectory generation*: Generate random samples in $\mathcal{X}$ according to $f(\cdot; \mathbf{v}_{t-1})$.
- *Parameter updating*: Update $\mathbf{v}$ on the basis of the elite samples, in order to produce better performing samples in the next iteration.

Remark 2.2 (*Maximum Likelihood Estimate*) The updating rule for $\mathbf{v}$ follows from cross-entropy minimization and often has a simple form. In particular, it is given by the *maximum likelihood* estimate of $\mathbf{v}$ based on the elite samples.

Remark 2.3 (*Minimization*) Note that for a minimization program, we take the ϱ-quantile of the best (smallest) performances. Also, the $\geqslant$ sign in (7) and (8) is replaced by a $\leqslant$ sign.

3 The clustering problem as a partition problem

In this section, we view the optimization of (1) as a combinatorial partition problem with K partitions. The idea is to partition the points into K disjoint clusters such that the cost of this partition (the loss function) is minimized. In particular, the clusters $R_1, \ldots, R_K$ are represented through a cluster vector $\mathbf{x} \in \mathcal{X} = \{1, \ldots, n\}^K$ as in (2). Thus, $x_i = j$ means that point $\mathbf{z}_i$ belongs to the jth cluster. In this case, the "trajectory generation" (sampling from $\mathcal{X}$) of the CE Algorithm 2.1 consists of drawing random cluster vectors $\mathbf{X} \in \mathcal{X}$ according to an n-dimensional discrete distribution with independent marginals, such that $\mathbb{P}(X_i = j) = p_{ij}$, $i = 1, \ldots, n$, $j = 1, \ldots, K$. Thus, in Algorithm 2.1, the parameter vector $\mathbf{v}$ consists of the probabilities $\{p_{ij}\}$. For $K = 2$ we may, alternatively, let $\mathbf{X}$ be a binary vector with independent components, the ith component being 1 with probability p_i and 0 with probability $1 - p_i$.

With the performance $S(\mathbf{x})$ given in (2), and the centroids defined via (3), the updating rule for p_{ij} is

$$\widehat{p}_{t,ij} = \frac{\sum_{k=1}^{N} I_{\{S(\mathbf{X}_k) \leqslant \widehat{\gamma}_t\}} I_{\{X_{ki} = j\}}}{\sum_{k=1}^{N} I_{\{S(\mathbf{X}_k) \leqslant \widehat{\gamma}_t\}}} \,. \tag{11}$$

This has a very simple interpretation: we update the probability that $X_i = j$ by taking the fraction of times that $X_i = j$ for the elite samples.

Example 3.1 Let $n = 5$ and $K = 2$. To generate a feasible cluster we draw $\mathbf{X}$ from a 5-dimensional Bernoulli distribution $\mathsf{Ber}(\mathbf{p})$ with independent marginals. Assume that a particular outcome of $\mathbf{X}$ is $\mathbf{x} = (1,0,0,1,0)$. The associated partition is, clearly, $\{R_1, R_2\} = \{\{1,4\},\{2,3,5\}\}$. In this case the loss and the centroids can be explicitly calculated, provided the points $z_1, z_2,\ z_3,\ z_4,\ z_5$ are given. Namely, the loss is

$$\begin{aligned}&||z_1 - \mathbf{c}_1||^2 + ||z_4 - \mathbf{c}_1||^2\\&\quad + ||z_2 - \mathbf{c}_2||^2 + ||z_3 - \mathbf{c}_2||^2 + ||z_5 - \mathbf{c}_2||^2\end{aligned} \tag{12}$$

and the centroids are

$$\mathbf{c}_1 = \tfrac{1}{2}(z_1 + z_4), \qquad \mathbf{c}_2 = \tfrac{1}{3}(z_2 + z_3 + z_5). \tag{13}$$

4 The clustering problem as a continuous multi-extremal problem

We next present a CE approach for solving the clustering problem by viewing it as a continuous multi-extremal optimization problem where, as in the KM method, the centroids $\mathbf{c}_1, \ldots, \mathbf{c}_K$ are the decision variables. In short, we consider the program

$$\min_{\mathbf{c}_1,\ldots,\mathbf{c}_K} S(\mathbf{c}_1, \ldots, \mathbf{c}_K) = \min_{\mathbf{c}_1,\ldots,\mathbf{c}_K} \sum_{j=1}^{K} \sum_{\mathbf{z} \in R_j} ||\mathbf{z} - \mathbf{c}_j||^2, \tag{14}$$

where $R_j = \{\mathbf{z} : ||\mathbf{z} - \mathbf{c}_j|| < ||\mathbf{z} - \mathbf{c}_k||,\ k \neq j\}$. That is, R_j is the set of data points that are closer to $\mathbf{c}_j$ than to any other centroid. Note that the performance (or objective) function S to minimize is a real-valued function on $\mathcal{X} = \mathbb{R}^{dK}$. In order to apply the CE method, we need to specify (a) a parametric family of sampling distributions on $\mathbb{R}^{dK}$ and (b) the updating rules for the parameters of the sampling distribution. The latter involves the Kullback–Leibler or CE distance, and results typically in updating the parameters via the maximum likelihood estimates of the elite (i.e., best) samples. It is important to realise that the choice of the sampling distribution is quite arbitrary and has nothing to do with the (suspected) distribution of the cluster data. We choose the sampling distribution to be dK-dimensional Gaussian of a specific form (to be explained next). The reason for choosing a Gaussian distribution is that the updating formulas become particularly simple.

For better insight and easy reference we consider the program (14) with $K = 2$ clusters and dimension $d = 2$. The sampling distribution is such that, we independently sample random centroids $\mathbf{C}_1$ and $\mathbf{C}_2$ according to 2-dimensional normal distributions $\mathsf{N}(\boldsymbol{\mu}_1, \Sigma_1)$ and $\mathsf{N}(\boldsymbol{\mu}_2, \Sigma_2)$, respectively. The parameter vector $\mathbf{v}$ in Algorithm 2.1 now consists of the mean vectors and covariance matrices $\{\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \Sigma_1, \Sigma_2\}$. As in a typical CE application for continuous multi-extremal optimization, we set the initial matrices Σ_1 and Σ_2 to be diagonal (with quite large variances on the diagonals) and then, we proceed according to Algorithm 2.1 as follows:

1. Choose, deterministically or randomly, the initial mean vectors and covariance matrices $\boldsymbol{\mu}_i$, Σ_i, $i = 1, 2$.
2. Generate $K = 2$ sequences of centroids (for clusters 1 and 2, respectively)

   $$\mathbf{Y}_{11}, \ldots, \mathbf{Y}_{1N} \quad \text{and} \quad \mathbf{Y}_{21}, \ldots, \mathbf{Y}_{2N},$$

   according to $\mathbf{Y}_{jk} \sim \mathsf{N}(\boldsymbol{\mu}_j, \Sigma_j)$, $j = 1, 2$, independently. For each $k = 1, \ldots, N$ calculate the objective function as in (14), with $\mathbf{c}_j$ replaced by $\mathbf{Y}_{jk}$, $j = 1, 2$.

3. Complete Steps 2,3 and 4 of Algorithm 2.1. In particular, update the parameters $(\boldsymbol{\mu}_1, \boldsymbol{\mu}_2)$ and (Σ_1, Σ_2), according to the Kullback–Leibler cross-entropy (see Remark 4.2).
4. If the stopping criterion (see Remark 4.1) is met, then stop and accept the resulting parameter vector $(\boldsymbol{\mu}_{1T}, \boldsymbol{\mu}_{2T})$ (at the final Tth iteration) as the estimate of the true optimal solution $(\mathbf{c}_1^*, \mathbf{c}_2^*)$ to the program (14); otherwise reiterate from Step 2.

Remark 4.1 (*Stopping criterion*) There are many possible variations to the the standard stopping criterion in Step 5 of Algorithm 2.1. In particular, for continuous optimization, criterion (10) may not be appropriate, as the $\widehat{\gamma}_t$ may, sometimes, never be equal, thus preventing the algorithm from stopping. An alternative, for the present problem, is to stop when the maximum of the diagonal elements in Σ_{1T} and Σ_{2T} is less than some η, say 10^{-4}.

Remark 4.2 (*Parameter updating*) Using Remark 2.2, we see that the means and variances for each centroid are updated simply as the corresponding sample mean and sample variance of the $N^{\text{elite}} = \lceil \varrho N \rceil$ elite samples. Specifically, if $\mathbf{X}_1, \ldots, \mathbf{X}_{N^{\text{elite}}}$ are the elite samples corresponding to a specific centroid (for cluster 1 or 2), then the related $\boldsymbol{\mu}$ and Σ are updated as

$$\widehat{\boldsymbol{\mu}} = \frac{1}{N^{\text{elite}}} \sum_{i=1}^{N^{\text{elite}}} \mathbf{X}_i$$

and

$$\widehat{\Sigma} = \frac{1}{N^{\text{elite}}} \sum_{i=1}^{N^{\text{elite}}} (\mathbf{X}_i - \widehat{\boldsymbol{\mu}})(\mathbf{X}_i - \widehat{\boldsymbol{\mu}})^T .$$

Note that, we have not assumed independent components for each centroid distribution $\mathsf{N}(\boldsymbol{\mu}, \Sigma)$. Therefore, in the 2-dimensional case, we need to update five parameters for each centroid. For a d-dimensional normal distribution the number of parameters is $d + (d+1)d/2$. However, if we use *independent* components for each $\mathsf{N}(\boldsymbol{\mu}, \Sigma)$ centroid distribution, then the number of distributional parameters is $2\,d$, because only the means and variances need to be updated; the off-diagonal elements of Σ are equal 0. It follows that for K clusters the total number of decision variables is $2\,d\,K$, when using independent components. Henceforth, we will only consider the case with independent components.

Remark 4.3 (*Starting positions*) One advantage of the CE method is that, as a global optimization method, it is very robust with respect to the initial positions of the centroids. Provided that the initial standard deviations are chosen large enough, the initial means have little or no effect on the accuracy and convergence speed of the algorithm. In general, we choose the initial means and standard deviations such that the initial sampling distribution is fairly "uniform" over the smallest rectangle that contains the data points. Practically, this means that the initial standard deviations should not be too small, say equal to the width or height of this "bounding box."

For the KM method, however, a correct choice of starting positions is essential. A well-known data-dependent initialization method is to generate the starting positions independently, drawing each centroid from a d-dimensional Gaussian distribution $\mathsf{N}(\boldsymbol{\mu}, \Sigma)$, where $\boldsymbol{\mu}$ is the sample mean of the data and Σ the sample covariance matrix of the data.

Remark 4.4 (*Modifications*) Various modifications to the standard CE algorithm can be found in [22]. In our numerical experiments the following two modifications proved useful as follows:

1. *Injection.* The idea behind the injection modification, first proposed in [6], is to inject extra variance into the sampling distribution in order to avoid premature "shrinkage" of the distribution. More precisely, let S_t^* denote the best performance found at the tth iteration, and σ_t^* denote the largest standard deviation at the tth iteration. If σ_t^* is sufficiently small, and $|S_t^* - S_{t-1}^*|$ is also small, then add $B = c\,|S_t^* - S_{t-1}^*|$ to each standard deviation, for some fixed c. Appropriate values for c and δ vary; however, for the texture image data in Sect. 5, $c = 50$ and $\delta = 0.05$ were used. When using CE with injection, a possible stopping criterion (see also Remark 4.1) is to stop once a certain number of injections, $\text{Inj}_{\max}$ say, is reached.
   Note that if B is very small ($\leq \delta$, say), we can just add some constant value, say δ, to avoid having injections that are too small.
2. *Component-wise updating.* The idea behind component-wise updating is, as the name suggests, to update the parameter vectors component–by–component. In particular, only a *single* component of one of the mean vectors is updated at each iteration. The order in which the components are updated can be either according to some fixed permutation, such as updating μ_{11} first, then μ_{12}, and so on, or according to some other (possibly random) permutation. In our numerical experiments we chose the second option. Component-wise updating is often used in simulated annealing (see e.g., [12]). However, in the context of CE this is, to our knowledge, the first application of its kind.

5 Numerical experiments

In this section, we present numerical experiments using the CE Algorithm 2.1 as well as the three well-known clustering heuristics KM, FKM, and LVQ. The Matlab code for these last three algorithms was taken from the Matlab *classification toolbox* [27] (see also [11]). We apply the methods first to two benchmark examples and then to a practical application in image analysis. We found that, in the examples, the continuous optimization approach of Sect. 4, is more accurate than the discrete optimization approach of Sect. 3. Hence, we only present our numerical results for the former.

5.1 Benchmark problems

Two well-known types of 2-dimensional data sets were used from [27]:

(a) Banana data: Points are scattered around a segment of a circle.
(b) 3-Gaussian mixture data: Points are generated from a mixture of three 2-dimensional Gaussian distributions.

Generation of these data sets is straightforward. For convenience a banana data generation algorithm is included in Sect. 7.2.

Tables 1–3, present a comparative study of CE Algorithm 2.1 and the traditional clustering ones for $n = 200$ and various cluster sizes K, on the data sets (a) and (b). In all experiments, we use $\alpha = 0.7$ and $\varrho = 0.025$. In all cases, a sample size of $N = 800$ is taken, so that the number of elite samples is $N^{\text{elite}} = \varrho N = 20$. All initial standard deviations are 14 for the banana data and six for the 3-Gaussian data, corresponding to the width/height of the bounding box for the data. The initial means are chosen uniformly over this bounding box. The starting positions for the other algorithms are chosen according to the standard initialization procedure for the KM algorithm discussed in Remark 4.3. We stop the CE algorithm when the performance no longer changes in two decimal places for ten iterations, or if the largest standard deviation is less than $\eta = 10^{-4}$.

Each method was repeated ten times for both data sets (a) and (b). We use the following notations in the tables: T denotes the average total number of iterations; $\bar{\gamma}_T$ denotes the averaged solution over ten runs; γ^* is the best known solution; $\bar{\varepsilon}$ denotes the average relative experimental error (based on ten runs) with respect to the best known solution γ^*. That is,

$$\bar{\varepsilon} = \frac{\bar{\gamma}_T - \gamma^*}{\gamma^*}. \tag{15}$$

Similarly, ε_* and ε^* denote the largest and smallest relative experimental errors. Finally, CPU denotes the average CPU time in seconds on a 3 GHz PC.

In order to accurately identify the global minimum, we repeated our experiments many times, using different ϱ, N, and smoothing parameter α (see Remark 2.1). The smallest value found from these experiments is given by γ^* for each case. It was found that the smallest CE performance of the ten runs gives a reliable estimate of the true global minimum.

We note that CE is quite robust with respect to the parameters N, ϱ, and α. That is, similar good results were obtained for parameter choices in the ranges $N = 400-4000$, $\alpha = 0.2 - 0.8$, and $\varrho = 0.01 - 0.2$.

We see that the CE algorithm, although significantly slower, is more accurate and consistent than the other algorithms. Amongst the faster algorithms, FKM is by far the best for these data. Observe also from Tables 1–3 that, as K increases, the efficiency (in terms of $\bar{\varepsilon}$, ε_*, ε^*) of CE increases relative to its counterparts for KM, FKM, and LVQ. In general, we found this to be the case. This can explained by arguing as follows:

1. The number of minima of the objective function in (14) increases with K.
2. The CE method, which presents a global optimization method typically avoids the local minima, and as a result settles down at the global minimum.

Table 1 Performance of the four different methods for the data sets (a) and (b), with $n = 200, K = 5$, $N = 800, N^{\text{elite}} = 20$, and $\alpha = 0.7$

Approach	T	$\bar{\gamma}_T$	γ^*	$\bar{\varepsilon}$	ε_*	ε^*	CPU
(a) Banana data set							
CE	46.2	517.597	515.613	0.0038474	5.90906e-010	0.0222069	8.4844
KM	12.5	536.72	515.613	0.0409353	0.00120497	0.254114	0.0155
FKM	74.9	521.298	515.613	0.0110255	0.0110232	0.0110265	0.0313
LVQ	17.6	541.542	515.613	0.0502883	0.000675106	0.345366	0.021375
(b) Three Gaussian mixture							
CE	37.8	37.5173	37.3444	0.00462838	5.39914e-009	0.0462837	7.0049
KM	7.9	55.5058	37.3444	0.486322	0	2.31036	0.0235
FKM	54.9	46.8244	37.3444	0.253852	0.0199335	0.519954	0.0234
LVQ	7.2	52.8893	37.3444	0.416257	0.00562148	2.30022	0.0156667

Table 2 Performance of the four different methods for the data sets (a) and (b), with $n = 200$, $K = 10$, $N = 800$, $N^{\text{elite}} = 20$, and $\alpha = 0.7$

Approach	T	$\bar{\gamma}_T$	γ^*	$\bar{\varepsilon}$	ε_*	ε^*	CPU
(a) Banana data set							
CE	89.3	259.958	251.563	0.0333721	0.0081804	0.0977549	32.78
KM	10.1	304.103	251.563	0.208856	0.0563821	0.472844	0.0548
FKM	106.4	262.803	251.563	0.0446827	0.0327853	0.0741326	0.0766
LVQ	15.2	301.617	251.563	0.198975	0.0385996	0.399317	0.0281
(b) Three Gaussian mixture							
CE	68.5	16.6375	15.3991	0.0804214	8.80297e-005	0.20986	25.441
KM	11.2	24.7502	15.3991	0.607251	0.276936	1.53184	0.113
FKM	80	20.4847	15.3991	0.330259	0.26562	0.595413	0.0578
LVQ	7.6	22.599	15.3991	0.467559	0.288255	0.769473	0.0155556

Table 3 Performance of the four different methods for the data sets (a) and (b), with $n = 200$, $K = 20$, $N = 800$, $N^{\text{elite}} = 20$, and $\alpha = 0.7$

Approach	T	$\bar{\gamma}_T$	γ^*	$\bar{\varepsilon}$	ε_*	ε^*	CPU
(a) Banana data set							
CE	122.6	131.466	126.716	0.0374862	0	0.0818811	89.9206
KM	9.1	177.689	126.716	0.402263	0.178212	0.706801	0.2901
FKM	252.4	148.842	126.716	0.174614	0.114661	0.23969	0.3259
LVQ	13.4	161.707	126.716	0.27614	0.173362	0.394308	0.0421
(b) Three Gaussian mixture							
CE	122	7.03842	6.61466	0.0640637	0.0151182	0.130905	89.2449
KM	9.6	14.0762	6.61466	1.12803	0.691861	1.8025	0.2624
FKM	138.6	9.48018	6.61466	0.433208	0.256727	1.08898	0.1859
LVQ	12.3	12.9741	6.61466	0.961423	0.509683	1.68469	0.0344

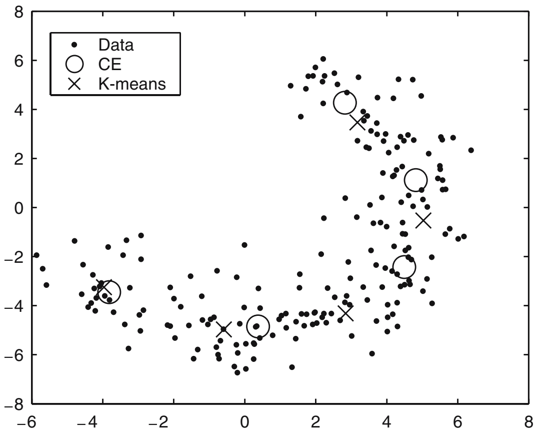

Fig. 1 The CE results for vector quantization of the 2-D *banana* data set. Circles designate the final cluster centers (centroids) produced by CE. Crosses are cluster centers produced by the KM algorithm

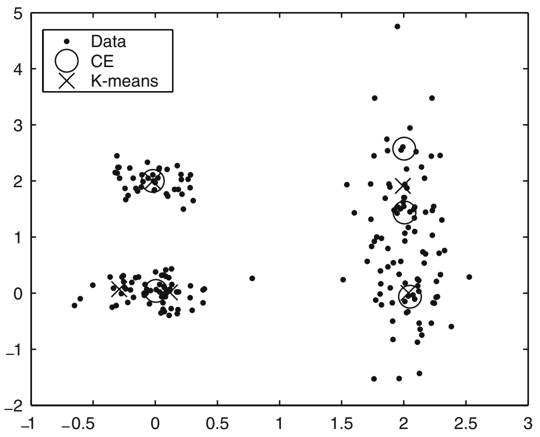

Fig. 2 The CE results for vector quantization of the 2-D 3-Gaussian data set. Circles designate the final cluster centers (centroids) produced by CE. Crosses are cluster centers produced by the KM algorithm

Table 4 Evolution of Algorithm 2.1 for the banana problem with $n = 200$, $d = 2$, $K = 5, N = 800, N^{\text{elite}} = 20$ and $\alpha = 0.7$, and with $\sigma_1^* = 3$

t	$\widehat{\gamma}_t$	S_t^*	σ_t^*
1	1786.40	1386.39	3.00000
2	1360.80	862.073	3.06570
3	1042.16	770.373	3.48781
4	921.520	708.242	3.49147
5	810.130	616.364	3.57854
6	699.275	625.932	2.69131
7	597.351	565.497	1.41700
8	554.466	534.791	0.79520
9	534.290	522.097	0.55452
10	524.233	518.937	0.35925
11	519.332	516.814	0.20460
12	517.210	516.146	0.13030
13	516.389	516.028	0.08783
14	515.974	515.716	0.06146
15	515.783	515.679	0.03990
16	515.676	515.645	0.02653
17	515.641	515.628	0.01627
18	515.625	515.618	0.01039
19	515.618	515.615	0.00685
20	515.615	515.614	0.00489
21	515.614	515.613	0.00342
22	515.613	515.613	0.00208
23	515.613	515.613	0.00130
24	515.613	515.613	0.00085

3. The alternatives, KM, FKM and LVQ, which present local optimization methods are typically trapped by these local minima.

It is clear that the "classical" KM method with an average relative experimental error of 10–100% compares poorly to the CE method, which is slower but yields a vastly superior relative error of less than 1%.

Figures 1 and 2 shows the difference in the placement of the centroids for CE (circles) and KM (crosses) for the banana and 3-Gaussian data, respectively. Note that for the 3-Gaussian data the KM algorithm has (wrongly) placed *two* centroids in the lower left-hand cluster.

Finally, Table 4 presents the evolution of Algorithm 2.1 for the banana problem with $n = 200$, and $K = 5$. Here S_t^* denotes the best (smallest) performance of the elite samples, $\widehat{\gamma}_t$ the worst performance of the elite samples, and σ_t^* the largest standard deviation (all at iteration t).

5.2 A fairer comparison

Because CE is much slower than the other three algorithms, one could object and say that the previous results are not fair on the other algorithms. Namely, one could run the other algorithms multiple times for each run of the CE algorithm. To assess if CE still outperforms the other algorithms in accuracy when, we allow multiple runs, we conducted similar experiments as before, but now using the same amount of time for each algorithm.

All results that follow use CE with the following parameters: $N = 800$, $N^{\text{elite}} = 20$ and $\alpha = 0.7$. The stopping criterion used for these experiments was the same as that used in the previous numerical experiments. As before, the initial cluster means are random on the spread of the data in each dimension, and the initial standard deviations in each dimension are equal to the maximum spread of the data over all dimensions. In the Tables 5–14 below, min, max and mean are the minimum, maximum, and mean of the replications (ten in the case of CE; many more for the other algorithms). The CPU gives the *total* CPU time in seconds. The last column lists the average number of iterations required for each replication. The 3-Gaussian and Banana data sets are the same as before. We have added a Spiral data set and the 'XOR' data set, both from [27], the PCB3038 dataset from [2] and a 5-Gaussian data set. The 5-Gaussian data set was generated by drawing 300 points from a mixture of five Gaussian distributions with the following weights, means, and covariances:

cluster	mean vector	covariance matrix	weight
1	$\begin{pmatrix}0.6\\6\end{pmatrix}$	$\begin{pmatrix}1 & 0.3\\0.3 & 1\end{pmatrix}$	0.1
2	$\begin{pmatrix}10\\-10\end{pmatrix}$	$\begin{pmatrix}1 & -0.2\\-0.2 & 1\end{pmatrix}$	0.2
3	$\begin{pmatrix}3\\-1\end{pmatrix}$	$\begin{pmatrix}2 & 0\\0 & 2\end{pmatrix}$	0.1
4	$\begin{pmatrix}0\\10\end{pmatrix}$	$\begin{pmatrix}3 & 0\\0 & 3\end{pmatrix}$	0.3
5	$\begin{pmatrix}10\\-30\end{pmatrix}$	$\begin{pmatrix}1 & 0.7\\0.7 & 1\end{pmatrix}$	0.3

As noted before, the 3-Gaussian and banana data sets were taken from the *Classification Toolbox*.

Table 5 5-Gaussian data set, with $K = 5$ clusters

	min	max	mean	trials	CPU	av its
CE	946.164	1379.72	1203.8	10	119.126	43.8
KM	946.164	20249.8	1730.94	4,713	119.499	8.1725
FKM	953.088	3588.99	1219.08	6,835	119.258	30.4739
LVQ	946.165	20246.6	1597.97	10,083	119.224	9.62967

Table 6 5-Gaussian data set, with $K = 20$ clusters

	min	max	mean	trials	CPU	av its
CE	254.7	281.532	267.332	10	1819.77	168.1
KM	335.185	1356.75	565.062	5,797	1821.16	11.4426
FKM	271.257	483.349	338.759	7,081	1821.26	131.123
LVQ	290.518	1381.91	552.814	40,590	1820.07	14.8994

Table 7 3-Gaussian data set, with $K = 5$ clusters

	min	max	mean	trials	CPU	av its
CE	37.3444	37.4955	37.3595	10	73.749	39.7
KM	37.3444	237.819	51.8926	5,944	73.958	7.63089
FKM	38.0888	56.7619	42.5827	3,402	73.855	51.0188
LVQ	37.3446	239.401	50.198	10,072	73.823	6.51728

Table 8 3-Gaussian data set, with $K = 20$ clusters

	min	max	mean	trials	CPU	av its
CE	6.93033	8.48561	7.40947	10	961.607	131.2
KM	8.10137	36.5534	13.8324	4,494	962.457	10.8369
FKM	7.0981	16.0369	9.26119	3,973	962.561	190.337
LVQ	7.94878	35.6519	13.1938	35,781	961.774	9.48928

Table 9 Banana data set, with $K = 5$ clusters

	min	max	mean	trials	CPU	av its
CE	515.613	538.943	520.219	10	108.67	59
KM	515.613	1870.87	557.903	8,836	108.863	10.8833
FKM	521.297	521.299	521.298	3,480	108.774	75.4983
LVQ	515.614	1031.04	552.229	6,984	108.85	14.9954

Table 10 Banana data set, with $K = 20$ clusters

	min	max	mean	trials	CPU	av its
CE	125.607	134.363	129.072	10	885.54	120.8
KM	134.968	300.128	181.277	7,456	886.07	9.45333
FKM	139.327	167.908	149.067	2,892	888.232	241.989
LVQ	129.716	302.239	168.528	21,689	885.715	13.3571

Table 11 XOR data set (1,000 points), with $K = 5$ clusters

	min	max	mean	trials	CPU	av its
CE	8867.88	8868.41	8868.02	10	440.013	48.3
KM	8867.88	14014.8	9008.92	8,954	441.383	28.7916
FKM	9013.31	9013.32	9013.31	637	444.316	483.356
LVQ	8868.05	12023.3	9187.76	6,680	440.49	27.291

Table 12 XOR data set (1,000 points), with $K = 10$ clusters

	min	max	mean	trials	CPU	av its
CE	4824.88	4956.16	4853.05	10	1706.47	94.2
KM	4825.09	6382.55	4992.18	14,273	1710.83	26.8819
FKM	5165.29	5216.86	5193.27	1,491	1714.22	440.872
LVQ	4825.7	6876.7	5042.75	14,840	1707.17	29.1883

Table 13 Spiral data set (200 points), with $K = 5$ clusters

	min	max	mean	trials	CPU	av its
CE	27.2287	27.4268	27.2947	10	68.9489	36.6
KM	27.2287	50.954	28.1902	2,573	69.6973	13.7101
FKM	28.4778	28.478	28.4779	2,559	69.1008	68.0008
LVQ	27.2291	52.2351	29.9398	8,789	68.9809	8.09182

Table 14 PCB3038 data set, with $K = 10$ clusters

	min	max	mean	trials	CPU	av its
CE	$5.60251 \cdot 10^8$	$5.84443 \cdot 10^8$	$5.6527 \cdot 10^8$	10	9654.39	139.3
KM	$5.60251 \cdot 10^8$	$8.70012 \cdot 10^8$	$5.88778 \cdot 10^8$	22,976	9661.27	34.9701
FKM	$5.66532 \cdot 10^8$	$5.66532 \cdot 10^8$	$5.66532 \cdot 10^8$	4,883	9661.78	232.149
LVQ	$5.60251 \cdot 10^8$	$8.39822 \cdot 10^8$	$5.90523 \cdot 10^8$	9,333	9657.99	119.684

5.3 Image Texture Data

In this section, we apply the clustering procedure to texture images. Texture data usually exhibits a complex spatial–frequency pattern, which is hard to describe. Some examples of real-life textures are depicted in Fig. 3, where a number of different texture patterns are easily distinguished by human eye. However, such a task is not trivial using clustering algorithms. There are well-known techniques and algorithm for analyzing and classifying texture patterns. Some of them are based on statistical autocorrelation properties [7], wavelet transform [8] and Markov random fields [4].

To proceed, note that the gray scale value of each pixel in a texture image is meaningless without its neighbors. A common way to characterize a texture is to examine the entire neighborhood of each pixel. In our examples we use 5×5 pixel neighborhood as a feature patch. It is equivalent to a 25-dimensional data vector $\mathbf{z}_i$, where i corresponds to a specific pixel location in the image. The entire texture image produces a large data set of texture vectors $\{\mathbf{z}_i\}$. The high self-similarity of these texture patterns results in a large number of vectors clustered around the centroids $\{\mathbf{c}_j\}$. We

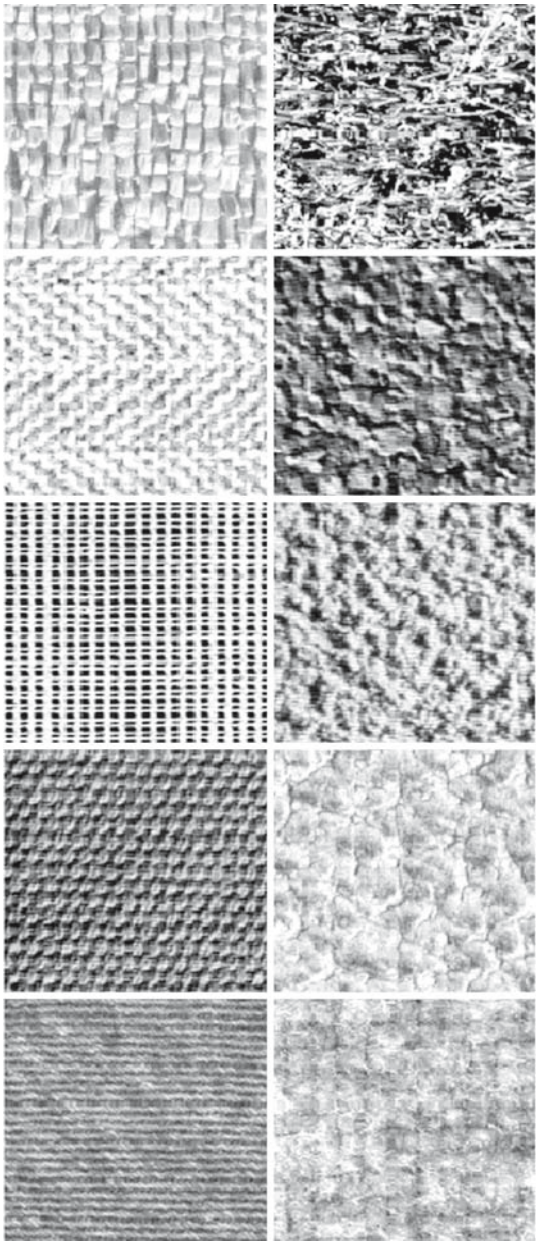

Fig. 3 Different image texture patterns

assume that $K = 5$ clusters and a total of $n = 256$ points are sufficient to describe the textures of Fig. 3 with low distortion. Table 15 presents the comparative study of the CE, KM, FKM and LVQ algorithms for a "raffia" test image. As with the "fairer comparison" tables, the CE approach was run (repeated) 10 times, and the others were allotted the same amount of time to be continually repeated. Figure 4 shows the original raffia texture image, taken from the *USC-SIPI Image Database* [1].

Table 15 Raffia Image Data Set, with $K = 5$ clusters

	min	mean	max	$\bar{\varepsilon}$	ε_*	ε^*	trials	CPU	av its
CE	29.67	29.97	30.53	0.0101	0.0001	0.0291	10	1948.96	13.4
KM	29.67	30.03	38.54	0.0123	0	0.2992	72,958	1949.51	11.71
FKM	35.99	35.99	35.99	0.2131	0.2131	0.2132	7,563	1950.52	174.25
LVQ	29.67	30.15	38.57	0.0162	0.0001	0.3003	71,237	1949.49	10.12

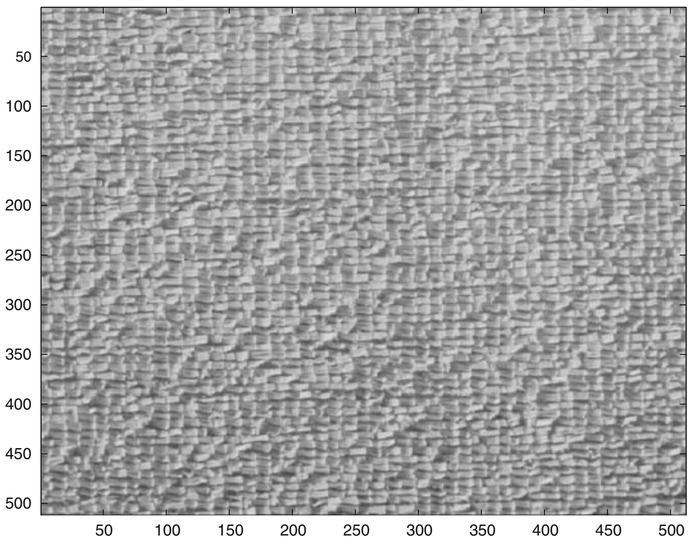

Fig. 4 Raffia image

Figure 5 depicts our actual test image, which is a 20×20 sub-image of the original raffia texture, chosen to represent the whole image.

Figure 6 depicts all of the 256 sub-images of size 5×5 generated from the 20×20 raffia test image.

It is important to note the following:

- In contrast to many of the test cases, the KM and LVQ algorithms perform significantly better than FKM here.
- When applying the standard CE algorithm (without modifications), our results (not presented here) indicate that both KM and LVQ perform better, both in accuracy and speed, than CE.
- However, when applying the component-wise updating and injection modifications discussed in Sect. 4, CE outperforms all other methods.

All results that follow use CE with the component-wise updating, and injection, with the following parameters: $N = 100$, $N^{\text{elite}} = 10$, and $\alpha = 0.9$, $\text{Inj}_{\max} = 100$. The initial means are random on the spread of the data in each dimension, and the initial standard deviations in each dimension are equal to the maximum spread of the data over all dimensions. We stop when the number of injections that have occurred so far

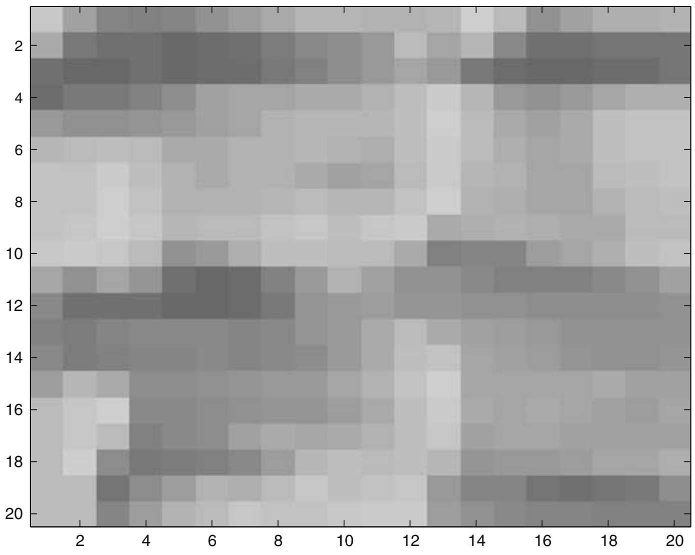

Fig. 5 Raffia test image

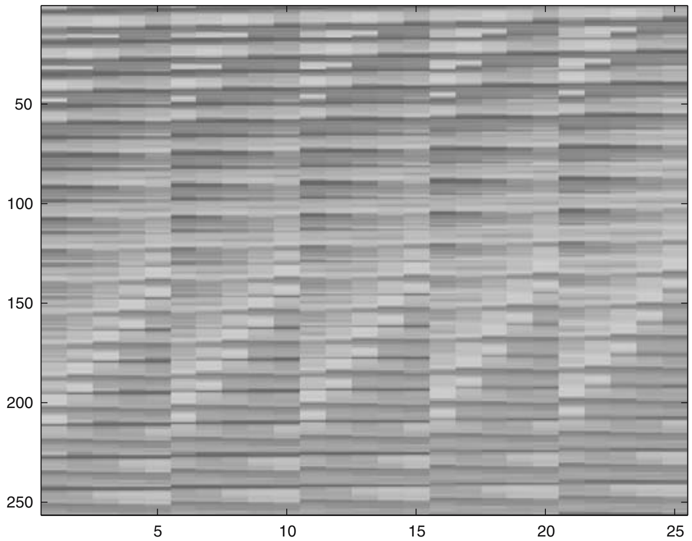

Fig. 6 Collection of 256 sub-images of size 5 × 5 of the raffia test image

is equal to the maximum number of injections allowed (100 here). In all figures, the image grey levels take values between 0 and 1.

Figure 7 depicts the images corresponding to the optimal five cluster centers (five vectors of length 25) found by CE.

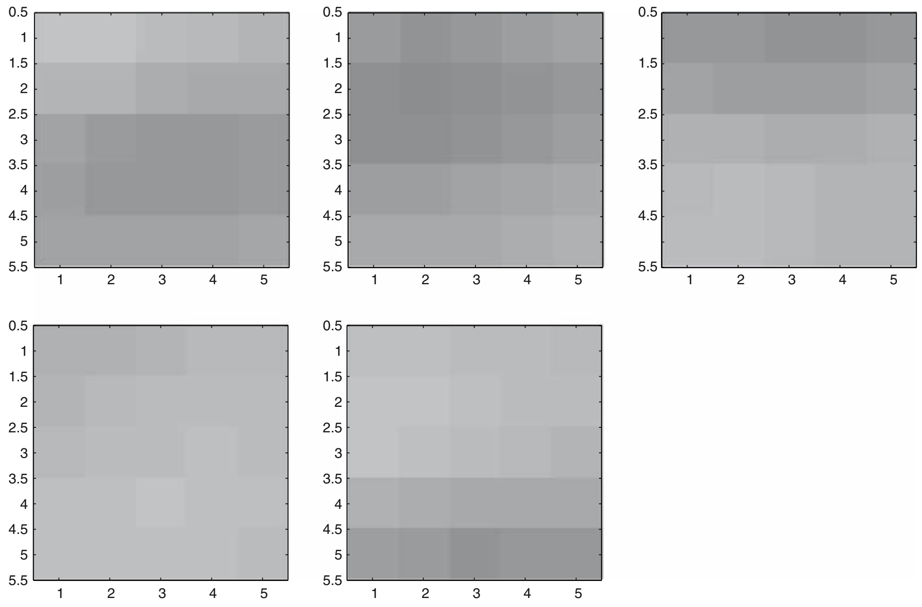

Fig. 7 The optimal five cluster centers found by CE

6 Conclusions and directions for future research

This paper presents, an application of the cross-entropy method to the clustering and vector quantization problems.

The proposed algorithm involves the generation of random clusters using either independent k-point distributions (Sect. 3) or independent Gaussian distributions (Sect. 4), followed by an updating of the parameters of the associated distributions using cross-entropy minimization. Our numerical studies suggest that the proposed algorithm is reliable, in the sense that in approximately 99% of the cases the relative error ε does not exceed 1%. Our main conclusion is that the CE offers an easily implementable and accurate alternative to the standard clustering methods like KM, FKM, and LVQ method, if one is interested in obtaining globally optimal solutions.

Although method presented here is more computationally intensive per run than the standard algorithms, it is typically able to outperform them even when the standard algorithms are allowed as much CPU time as the CE method. We note that, for applications where time is critical or computational resources are scarce, or with extremely large data sets and large number of dimensions, the method presented here may consume an unacceptable amount of computational resources.

Further topics for investigation include (a) establishing convergence of Algorithm 2.1 for finite sampling (i.e., $N < \infty$) with emphasis on the complexity and the speed of convergence under the suggested stopping rules; (b) establishing confidence intervals (regions) for the optimal solution; and (c) application of parallel optimization techniques to the proposed methodology.

We note that in addition to its simplicity, the advantage of using the CE method is that it does not require direct estimation of the gradients, as many other algorithms do (for example, the stochastic approximation, steepest ascent or conjugate gradient method). Moreover, as a global optimization procedure the CE method is quite robust with respect to starting conditions and sampling errors, in contrast to some other heuristics, such as simulated annealing or guided local search.

Although in the present examples the discrete (partition) CE approach did not compete well with the continuous CE approach, the former may be useful when the performance is a complicated function of the data. For example, the data could represent a collection of proteins, each of which has a list of characteristics. In this case the performance function is not merely the sum of the Euclidean distances but some complicated function of these characteristics.

Acknowledgements We are most grateful to Uri Dubin for his contributions to an early version of this paper. This work was supported by the Australian Research Council (grant No. DP0558957) and the Israel Science Foundation (grant No. 191-565). Thomas Taimre acknowledges the financial support of the ARC Centre of Excellence for Mathematics and Statistics of Complex Systems.

Appendix

7.1 Example matlab code

In this paper, we used the CE method to optimize a highly multi-modal function in (dK)-dimensional space, by sampling the potential solutions from a family of (dK)-dimensional Gaussian distributions, updating the dK means and variances via the sample mean and sample variances of the elite samples. Thus, the sampling distribution comes closer at each iteration to the "degenerate" distribution, which is the optimal sampling distribution for the problem.

The easiest way to explain the general idea is to provide a concrete 1-dimensional (1-D) toy example. Suppose, for example, we wish to optimize the 1-D, bi-modal function

$$S(x) = \mathrm{e}^{-(x-2)^2} + 0.8\,\mathrm{e}^{-(x+2)^2} .$$

The following Matlab code should speak for itself.

```
S = inline('exp(-(x-2).^2) + 0.8*exp(-(x+2).^2)');
mu = -10; sigma = 10; rho = 0.1; N = 100; eps = 1E-3;
t=0;  % iteration counter
while sigma > eps
  t = t+1;
  x = mu + sigma*randn(N,1);
  SX = S(x); % Compute the performance.
  sortSX = sortrows([x SX],2);
  mu = mean(sortSX((1-rho)*N:N,1));
  sigma = std(sortSX((1-rho)*N:N,1));
  fprintf('%g  %6.9f  %6.9f  %6.9f \n', t, S(mu),mu, sigma)
end
```

7.2 Generating banana data

Banana shaped data sets occur frequently in clustering and classification test problems. Below is a simple Matlab function that generates such data.

```
\begin{verbatim}
function b = banana(n,sigma,radius,theta1,theta2)
% Generate a Banana-shaped data set
% n      - number of data points (default: 200)
% sigma  - move points according to a normal dist. with this
%          standard deviation in both x and y directions
%          (default: 1)
% radius - the radius of the circle (of which the "banana" is
%          an arc (default: 5)
% theta1 - starting angle of the arc (default: 9*pi/8)
% theta2 - ending angle of the arc (default: 19*pi/8)
if nargin<4,theta1=pi + pi/8;end
if nargin<5,theta2=(5*pi/2 - pi/8) - theta1;end
if nargin<3,radius=5;end
if nargin<2,sigma=1;end
if nargin<1,n=200;end
randn('seed', 123456789); %optional
rand('seed', 987654321);  %optional
angles=theta1 + rand(n,1)*theta2; % angles between pi/8 and 11*pi/8
b=radius.*[cos(angles),sin(angles)]; % transform these to random points
                                      % on an arc of a circle
b=b + sigma*randn(n,2); % shift these points off the arc ind.
                        % normally in x and y directions
```

References

1. The USC-SIPI Image Database.: http://sipi.usc.edu/services/database/Database.html
2. TSPLIB : A Traveling Salesman Problem Library. http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/
3. Ahalt, S.C., Krishnamurthy, A.K., Chen, P., Melton, D.E.: Competitive learning algorithms for vector quantization. Neural Net. **3**, 277–290 (1990)
4. Betke, M., Makris, N.: Fast object recognition in noisy images using simulated annealing. In: Proceedings of the Fifth International Conference on Computer Vision, pp. 523–530 (1995)
5. Bezdek, J.: Pattern Recognition with Fuzzy Objective Function Algorithms. Plenum Press, New York (1981)
6. Botev, Z., Kroese, D.P.: Global likelihood optimization via the cross-entropy method, with an application to mixture models. In: Ingalls, R.G., Rossetti, M.D., Smith, J.S., Peters, B.A. (eds.) Proceedings of the 2004 Winter Simulation Conference, IEEE, Washington, DC, December 2004
7. Brown, L.: A survey of image registration techniques. Technical report, Department of Computer Science, Columbia University (1992)
8. Chen, J., Kundu, A.: Unsupervised texture segmentation using multichannel decomposition and hidden Markov models. IEEE Trans. Image Process. **4**, 603–619 (1995)
9. de Boer, P.T., Kroese, D.P., Mannor, S., Rubinstein, R.Y.: A tutorial on the cross-entropy method. Ann Oper. Res. Vol. 134, pp.19–67. Springer-Verlag (2005)
10. Dorigo, M., Di Caro, G.: The ant colony optimization meta-heuristic. In: Corne, D., Dorigo, M., Glover, F. (eds.) New Ideas in optimization, pp. 11–32. McGraw-Hill (1999)
11. Duda, R.O., Hart, P.E., Stork, D.G.: Pattern Classification. Wiley, New York (2001)
12. Geman, S., Geman, D.: Stochastic relaxation, Gibbs distribution and the Bayesian restoration of images. IEEE Trans. PAMI **6**, 721–741 (1984)
13. Glover, F., Laguna, M.L.: Modern Heuristic Techniques for Combinatorial Optimization, Chapter 3: Tabu Search. Blackwell Scientific Publications, Oxford (1993)

14. Goldberg, D.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison Wesley, Reading, MA (1989)
15. Hansen, P., Mladenović, N.: J-means: a new local search heuristic for minimum sum of squares clustering. Pattern Recogn. **34**, 405–413 (2001)
16. Hansen, P., Mladenović, N., Perez-Brito, D.: Variable neighborhood decomposition search. J Heuristics **7**, 335–350, 2001.
17. Jain, A.K., Murty, M.N., Flynn, P.J.: Data clustering: a review. ACM Comput. Surv. **31**, 264–323 (1999)
18. Kaufman, L., Rousseeuw, P.: Finding Groups in Data, and Introduction to Cluster Analysis. Wiley, New York (1990)
19. Keith, J., Kroese, D.P.: Sequence alignment by rare event simulation. In: Proceedings of the 2002 Winter Simulation Conference, pp. 320–327. San Diego (2002)
20. McLachlan, G., Krishnan, T.: The EM Algorithm and Extensions. Wiley, New York (1997)
21. Rubinstein, R.Y.: The cross-entropy method for combinatorial and continuous optimization. Methodol Comp. Appl. Prob. **2**, 127–190 (1999)
22. Rubinstein, R.Y., Kroese, D.P.: The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation and Machine Learning. Springer-Verlag, New York (2004)
23. Salomon, D.: Data Compression: The Complete Reference. Springer-Verlag, New York (2000)
24. Sherali, H.D., Desai, J.: A global optimization rlt-based approach for solving the hard clustering problem. J Global Optim. **32**, 281–306 (2005)
25. Shi, L., Olafsson, S.: Nested partitioning method for global optimization. Oper. Res. **48**(3), 390–407 (2000)
26. Spall, J.C.: Introduction to Stochastic Search and Optimization. John Wiley, New York (2003)
27. Stork, D.G., Yom-Tov, E.: Computer Manual to Accompany Pattern Classification. Wiley, New York (2004)
28. Webb, A.: Statistical Pattern Recognition. Arnold, London (1999)
29. Yang, M.-S., Wu, K.-L.: Unsupervised possibilistic clustering. Pattern Recog. **39**, 5–21 (2006)