Swarm and Evolutionary Computation 11 (2013) 16–30

Contents lists available at SciVerse ScienceDirect

Swarm and Evolutionary Computation

journal homepage: www.elsevier.com/locate/swevo

Regular Paper

Tsallis entropy based optimal multilevel thresholding using cuckoo search algorithm

Sanjay Agrawal [a], Rutuparna Panda [a], Sudipta Bhuyan [a], B.K. Panigrahi [b,*]

[a] Department of Electronics & Telecommunication Engineering, Veer Surendra Sai University of Technology, Burla 768018, India
[b] Department of Electrical Engineering, IIT Delhi-110003, India

ARTICLE INFO

Article history:
Received 21 May 2012
Received in revised form
3 February 2013
Accepted 11 February 2013
Available online 21 March 2013

Keywords:
Image segmentation
Multi-level thresholding
Cuckoo search algorithm
Tsallis entropy

ABSTRACT

In this paper, optimal thresholds for multi-level thresholding in an image are obtained by maximizing the Tsallis entropy using cuckoo search algorithm. The method is considered as a constrained optimization problem. The solution is obtained through the convergence of a meta-heuristic search algorithm. The proposed algorithm is tested on standard set of images. The results are then compared with that of bacteria foraging optimization (BFO), artificial bee colony (ABC) algorithm, particle swarm optimization (PSO) and genetic algorithm (GA). Results are analyzed both qualitatively and quantitatively. It is observed that our results are also encouraging in terms of CPU time and objective function values.

1. Introduction

Image segmentation has the role as a preprocessing step in image processing. Thresholding plays important role in image segmentation. Note that Image segmentation deals with subdividing the image into objects of meaningful information, which is useful for biomedical image processing, biomedical imaging, pattern recognition, remote sensing etc. Over the years many techniques for image segmentation have been developed and proposed in the literature. Thresholding is considered the most preferred technique out of all the existing techniques used for image segmentation. Reason may be due to the fact that it is very simple and efficient. Various techniques for global thresholding are available in the literature to segment images and extract meaningful patterns of interest [1–6]. Bi-level global thresholding is used to divide the image into two regions, foreground and background. The main idea behind using global thresholding as a segmentation technique is that foreground and background areas in an image can be distinguished by observing its histogram with probabilities for each gray level. However, for real life images bi-level thresholding does not give appropriate results. Hence, there is a strong need for multi-level thresholding which divides the histogram of the image into number of classes of homogenous gray levels such that some criterion is optimized. Many such criteria are proposed to achieve multi-level thresholding. Otsu's [7] criteria maximize the sum of between-class variances for separating the classes. According to Kittler and Illingworth [8], the histogram of an image is assumed to follow a mixture of Gaussians and the error between the parametric form and the actual histogram is minimized. Whereas Kapur's criteria [9] maximizes the entropy of each individual class or the sum of entropies based on information theory.

All the above criteria are very effective for bi-level thresholding. However, they fail to identify the optimal thresholds effectively for multi-level thresholding. The computational complexity increases as the number of thresholds increases. Obviously the alternate solution is to think for using evolutionary computational techniques. In this context, various thresholding algorithms are proposed which use different kinds of evolutionary techniques such as GA [10], PSO [11], ABC [12], BFO [13] and hybrid algorithms [6]. Yin [10] proposed a fast scheme using genetic algorithm for determining the optimal thresholds for multilevel thresholding. Yin [11] presented a recursive programming technique to reduce the computation time for computing the minimum cross entropy threshold (MCET) objective function. He then used PSO for obtaining the near-optimal thresholds. Zhang and Wu [12] used ABC algorithm for optimizing the Tsallis entropy. These are used to improvise efficiency of multilevel thresholding algorithms. According to Kamal Hammouche, Moussa Diaf and Patrick Siarry [14] multilevel thresholding has been taken as an optimization problem. They have presented six meta-heuristic algorithms to support their survey. Recently, the authors [15] have shown statistical analysis which addresses the stability and convergence of their method. This has motivated us to introduce a new method

* Corresponding author. Tel./fax: +91 1126591078.
E-mail addresses: bijayaketan.panigrahi@gmail.com, bkpanigrahi@ee.iitd.ac.in (B.K. Panigrahi).

2210-6502/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.swevo.2013.02.001

for finding the optimal thresholds effectively for multi-level thresholding. Here we consider optimal thresholding as a constrained optimization problem. The desired stability yields appropriate constraints for the maximization problem. Interesting and stable solutions are obtained through the convergence of a new meta-heuristic algorithm called cuckoo search.

Tsallis entropy also called non-extensive entropy has been studied for a possible extension of Shannon's entropy to information theory. This study has brought a similarity between the Shannon's entropy and Boltzmann/Gibbs entropy functions. A parameter 'q' called entropic index or Tsallis parameter is also associated with the non-extensivity of the system [1,16–18]. This paper uses the Tsallis entropy based method for image thresholding. The proposed method is used for maximizing the Tsallis entropy. Image thresholding results are presented for a qualitative analysis. Quantitative results defined by PSNR, Standard deviation, SSIM and FSIM are also presented for a comparison.

The paper is organized as follows: Section 2 presents concepts of Tsallis entropy. Section 3 describes concepts of Cuckoo search algorithm. Section 4 outlines the proposed method. Results and discussions are presented in Section 5. Conclusions are drawn in Section 6.

2. Tsallis entropy method

Physically entropy is associated with the measure of disorder in a system. But Shannon extended the concept of entropy to a measure of uncertainty regarding the information content of the system. It is also verified that the Shannon entropy has the extensive property (additivity):

$$S(A+B)=S(A)+S(B) \tag{1}$$

By following the multi-fractal concepts, the Tsallis entropy can be extended to non-extensive system based on a general entropic formula:

$$S_q=\frac{1-\sum_{i=1}^{k}(p_i)^q}{q-1} \tag{2}$$

where k is the total number of possibilities of the system and q is the measure of degree of non-extensivity of the system called Tsallis parameter or entropic index. This entropic form can be extended for a statistical independent system by a pseudo-additive entropic rule:

$$S_q(A+B)=S_q(A)+S_q(B)+(1-q).S_q(A).S_q(B) \tag{3}$$

Such a method is now used for thresholding an image. Let there be G gray levels in a given image and these gray levels are in the range {1, 2… G}. Let $p_i=p_1, p_2 \ldots p_G$ be the probability distribution of the gray levels. From these distributions two probability distributions, one for the foreground (class A) and another for the background (class B), are derived. The probability distributions of the foreground and background classes are given by [12,19]

$$p_A=\frac{p_1}{P^A},\frac{p_2}{P^A},\ldots,\frac{p_t}{P^A} \text{ and } p_B=\frac{p_{t+1}}{P^B},\frac{p_{t+2}}{P^B},\ldots,\frac{p_G}{P^B} \tag{4}$$

where $P^A=\sum_{i=1}^{t}p_i$ and $P^B=\sum_{i=t+1}^{G}p_i$

Now the Tsallis entropy for each class can be defined as:

$$S_q^A(t)=\frac{1-\sum_{i=1}^{t}(p_i/P^A)^q}{q-1} \tag{5}$$

$$S_q^B(t)=\frac{1-\sum_{i=t+1}^{G}(p_i/P^B)^q}{q-1} \tag{6}$$

The information measure between two classes (foreground and background) is maximized and the corresponding gray level for which this happens is considered to be the optimum threshold value.

```
Begin
Objective function f(x), x = (x_1,...,x_d)^T;
Initialize a population of n host nests, x_i ( i=1,2,...,n );
While (t < MaxIterations) or ( f_min > tol);
        Get a cuckoo (say k) randomly by Levy flights;
        Evaluate its quality/fitness F_k;
        Choose a nest among n (say j) randomly;
            if (F_k > F_j),
            Replace j by the new solution;
            end
        Abandon a fraction (p_a) of worst nests
        [and build new ones at new locations via Levy flights];
        Keep the best solutions (or nests with quality solutions);
        Rank the solutions and find the current best;
end while
end
```

Fig. 1. Pseudo Code for Cuckoo Search Algorithm.

Table 1
Parameters used for GA.

Parameter	Value
Population size	20
No. of iterations	100
Crossover probability	0.9
Mutation probability	0.1
Selection operator	Roulette wheel selection

Table 2
Parameters used for PSO.

Parameter	Value
Swam size	20
No. of iterations	100
W_{max}, W_{min}	0.4, 0.1
C_1, C_2	2

Table 3
Parameters used for BFO.

Parameter	Value
Number of bacterium (s)	20
Number of chemotatic steps (N_c)	10
Swimming length (N_s)	10
Number of reproduction steps (N_{re})	4
Number of elimination of dispersal events (N_{ed})	2
Depth of attractant ($d_{attract}$)	0.1
Width of attract ($x_{attract}$)	0.2
Height of repellent ($h_{repellent}$)	0.1
Width of repellent ($x_{repellent}$)	10
Probability of elimination and dispersal (P_{ed})	0.02

Table 4
Parameters used for CS.

Parameter	Value
Number of nests	40
No. of iterations	20
Mutation probability value (p_a)	0.25
Scale factor (β)	1.5
Tsallis parameter (q)	4

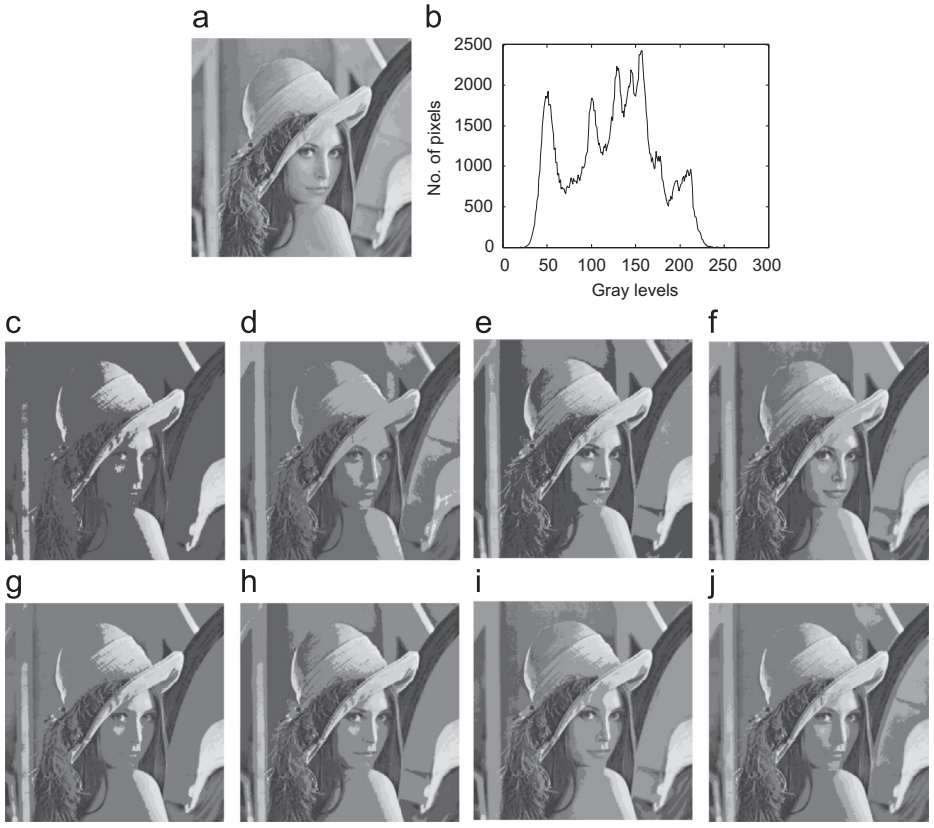

Fig. 2. Results of Lena Image using BFO and CS. a. Original Lena Image, b. Histogram of Lena Image, c. 2-level thresholding using BFO, d. 3-level thresholding using BFO, e. 4-level thresholding using BFO, f. 5-level thresholding using BFO, g. 2-level thresholding using CS, h. 3-level thresholding using CS, i. 4-level thresholding using CS, j. 5-level thresholding using CS.

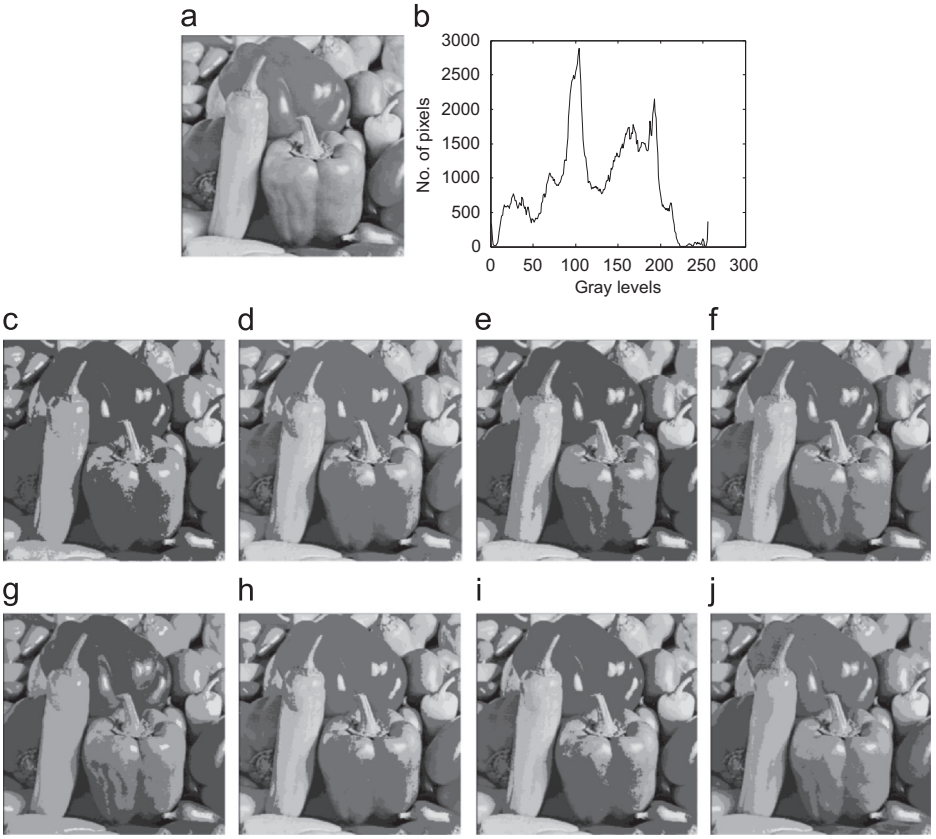

Fig. 3. Results of Pepper Image using BFO and CS. a. Original Pepper Image, b. Histogram of Pepper Image, c. 2-level thresholding using BFO, d. 3-level thresholding using BFO, e. 4-level thresholding using BFO, j. 5-level thresholding using BFO, g. 2-level thresholding using CS, h. 3-level thresholding using CS, i. 4-level thresholding using CS and j. 5-level thresholding using CS.

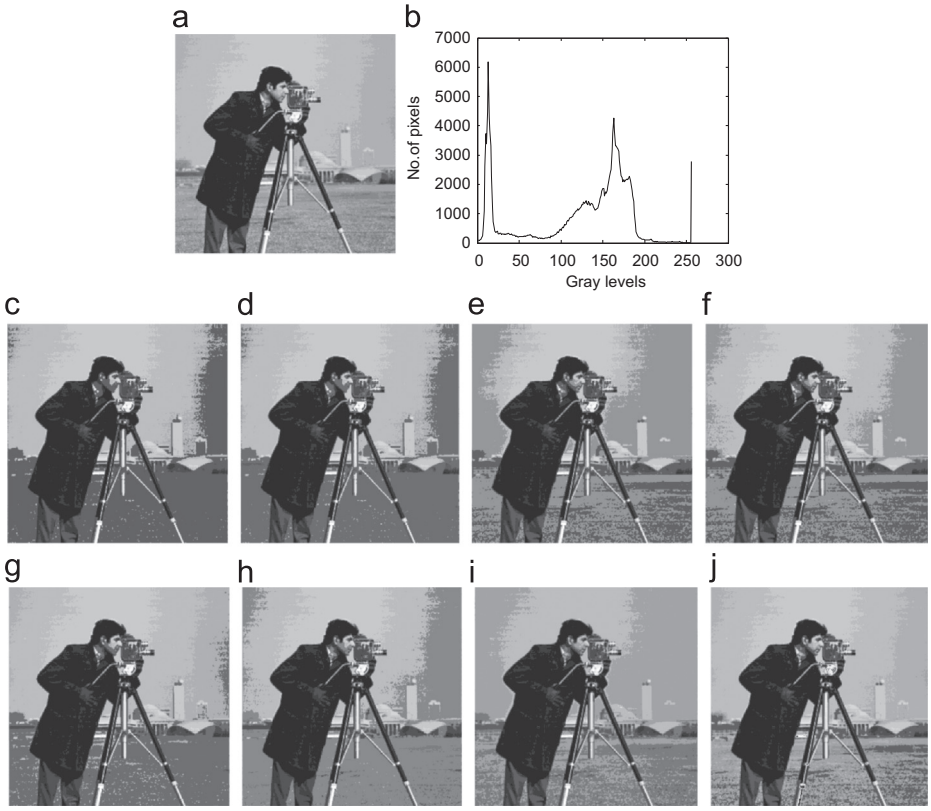

Fig. 4. Results of Cameraman Image using BFO a nd CS. a. Original Cameraman Image, b. Histogram of Cameraman Image, c. 2-level thresholding using BFO, d. 3-level thresholding using BFO, e. 4-level thresholding using BFO, f. 5-level thresholding using BFO, g. 2-level thresholding using CS, h. 3-level thresholding using CS, i. 4-level thresholding using CS, j. 5-level thresholding using CS.

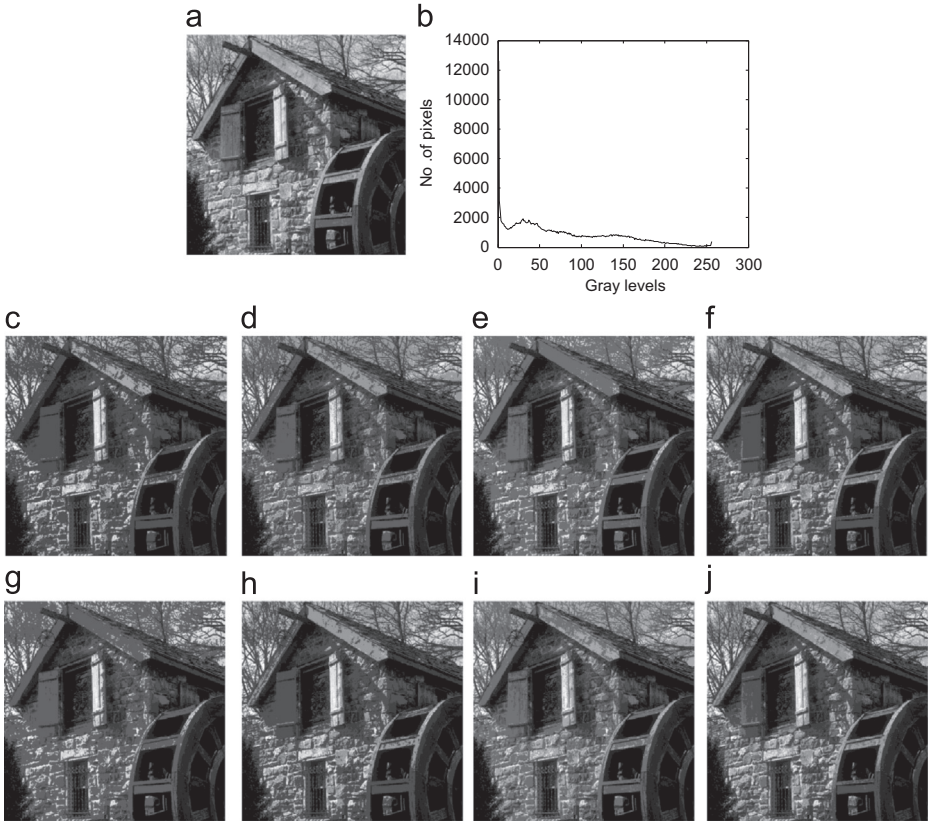

Fig. 5. Results of House Image using BFO and CS. a. Original House Image, b. Histogram of House Image, c. 2-level thresholding using BFO, d. 3-level thresholding using BFO, e. 4-level thresholding using BFO, f. 5-level thresholding using BFO, g. 2-level thresholding using CS, h. 3-level thresholding using CS, i. 4-level thresholding using CS and j. 5-level thresholding using CS.

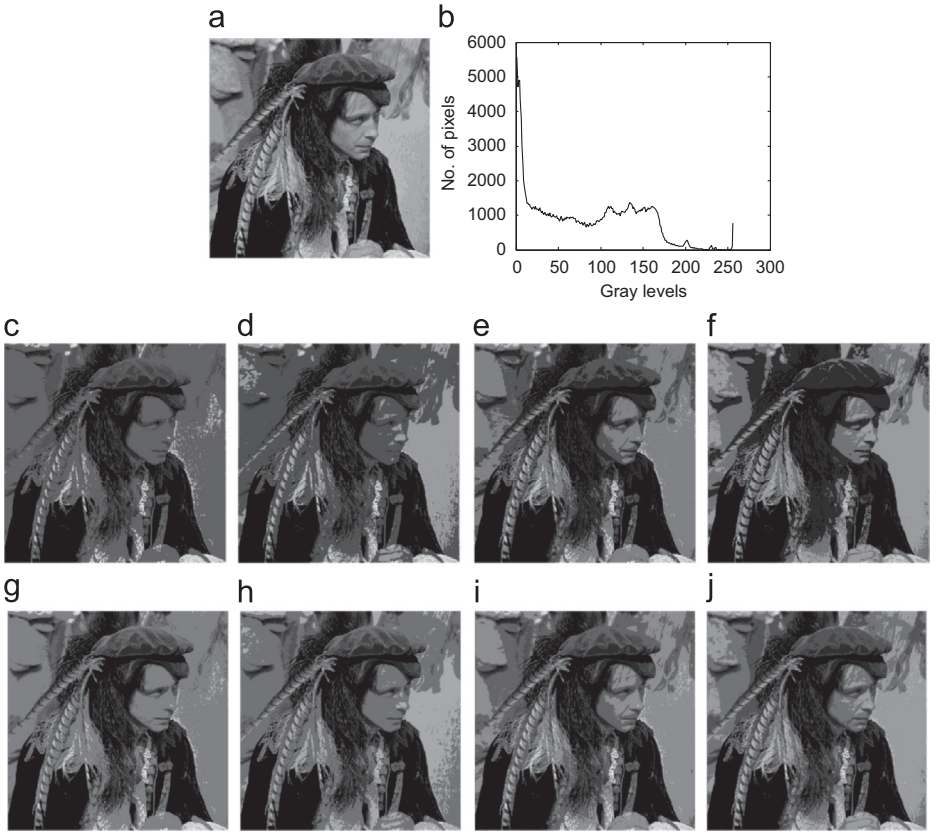

Fig. 6. Results of Hunter Image using BFO and CS. a. Original Hunter Image, b. Histogram of Hunter image, c. 2-level thresholding using BFO, d. 3-level thresholding using BFO, e. 4-level thresholding using BFO, f. 5-level thresholding using BFO, g. 2-level thresholding using CS, h. 3-level thresholding using CS, i. 4-level thresholding using CS and j. 5-level thresholding using CS.

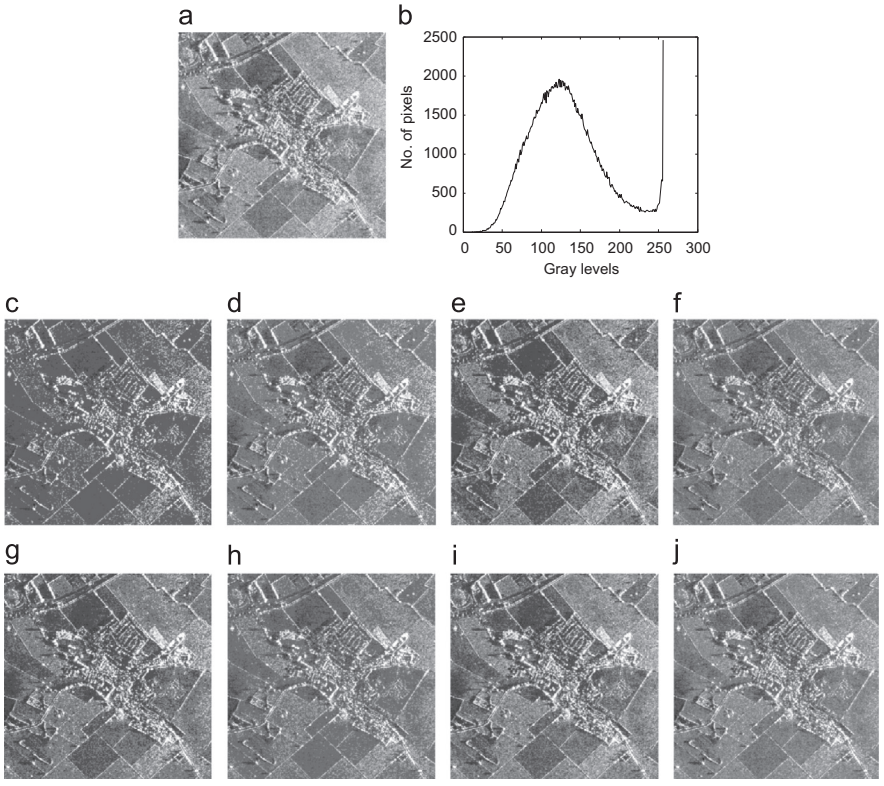

Fig. 7. Results of Map Image using BFO and CS. a. Original Map Image, b. Histogram of Map image, c. 2-level thresholding using BFO, d. 3-level thresholding using BFO, e. 4-level thresholding using BFO, f. 5-level thresholding using BFO, g. 2-level thresholding using CS, h. 3-level thresholding using CS, i. 4-level thresholding using CS, j. 5-level thresholding using CS.

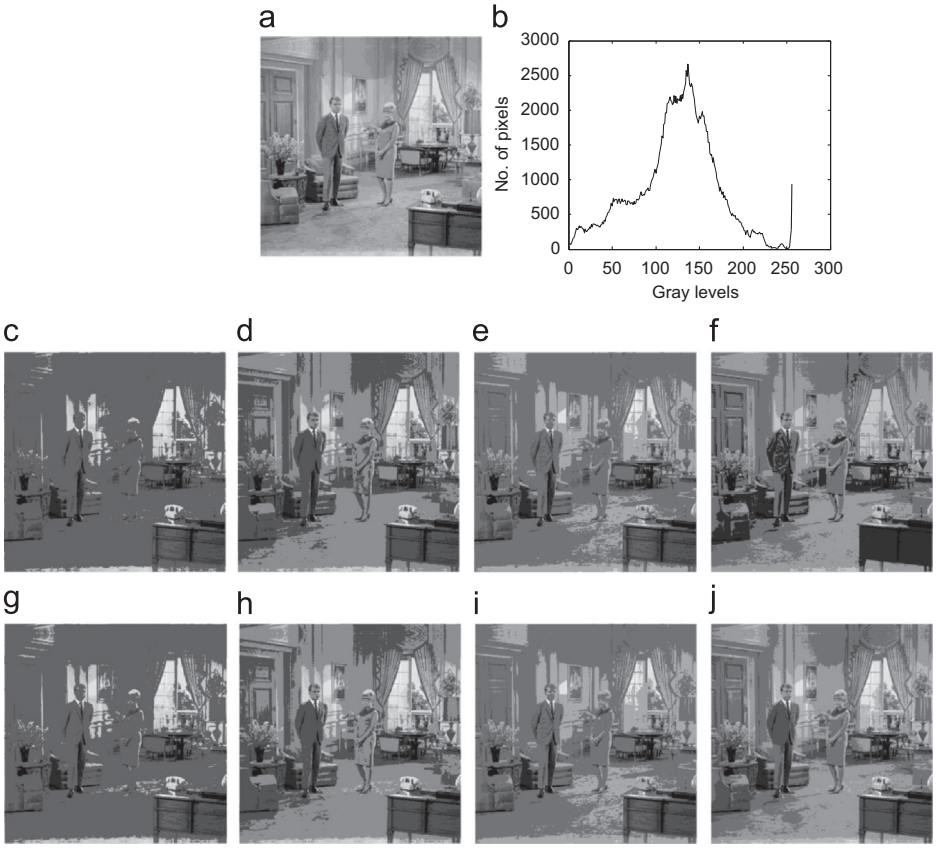

Fig. 8. Results of Livingroom Image using BFO and CS. a. Original Livingroom Image, b. Histogram of Livingroom image, c. 2-level thresholding using BFO, d. 3-level thresholding using BFO, e. 4-level thresholding using BFO, f. 5-level thresholding using BFO, g. 2-level thresholding using CS, h. 3-level thresholding using CS, i. 4-level thresholding using CS, j. 5-level thresholding using CS.

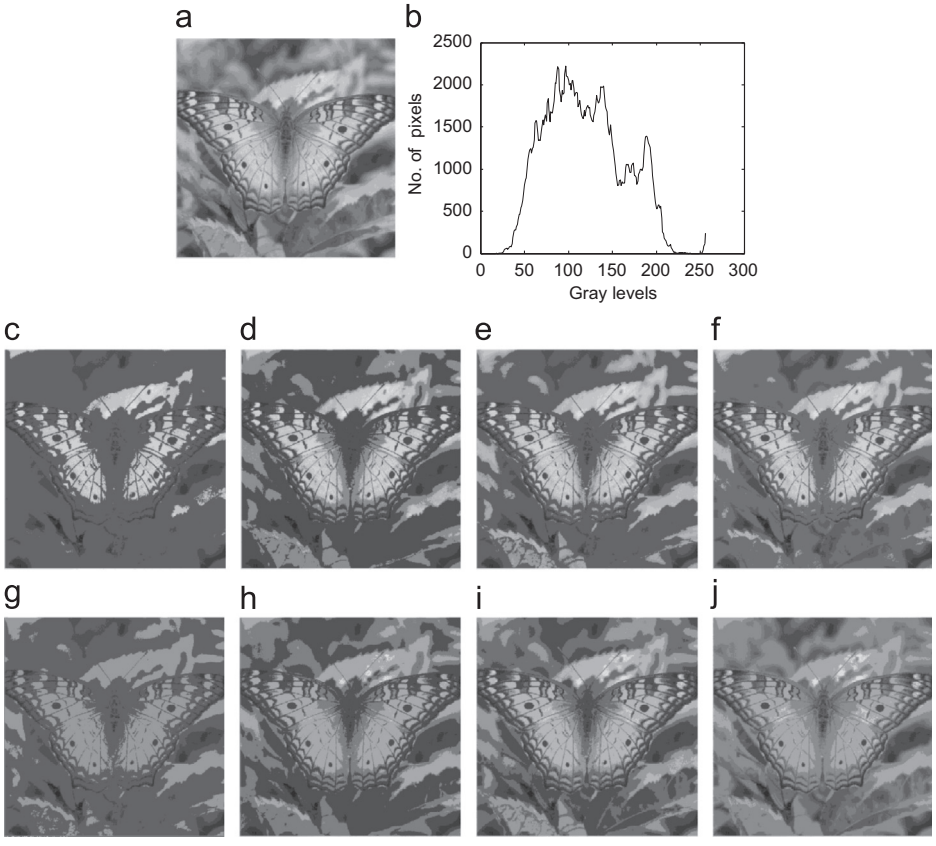

Fig. 9. Results of Butterfly Image using BFO and CS. a. Original Butterfly Image, b. Histogram of Butterfly image, c. 2-level thresholding using BFO, d. 3-level thresholding using BFO, e. 4-level thresholding using BFO, f. 5-level thresholding using BFO, g. 2-level thresholding using CS, h. 3-level thresholding using CS, i. 4-level thresholding using CS, j. 5-level thresholding using CS.

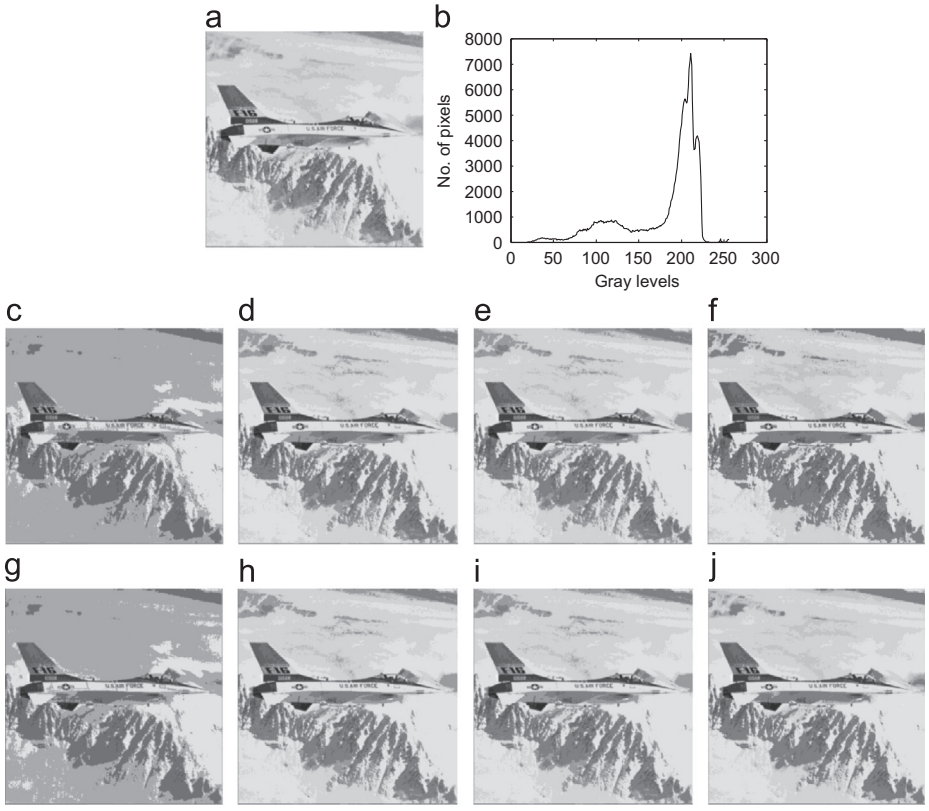

Fig. 10. Results of Airplane Image using BFO and CS. a. Original Airplane Image, b. Histogram of Airplane image, c. 2-level thresholding using BFO, d. 3-level thresholding using BFO, e. 4-level thresholding using BFO, f. 5-level thresholding using BFO, g. 2-level thresholding using CS, h. 3-level thresholding using CS, i. 4-level thresholding using CS, j. 5-level thresholding using CS.

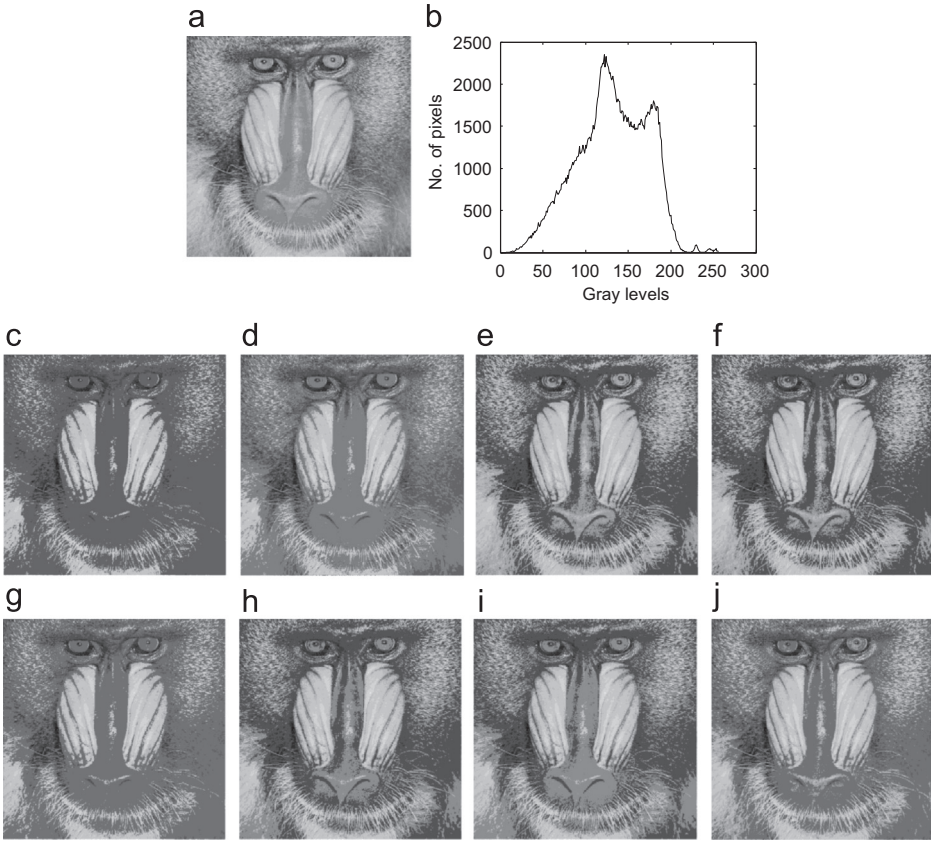

Fig. 11. Results of Baboon Image using BFO and CS. a. Original Baboon Image, b. Histogram of Baboon image, c. 2-level thresholding using BFO, d. 3-level thresholding using BFO, e. 4-level thresholding using BFO, f. 5-level thresholding using BFO, g. 2-level thresholding using CS, h. 3-level thresholding using CS, i. 4-level thresholding using CS, j. 5-level thresholding using CS.

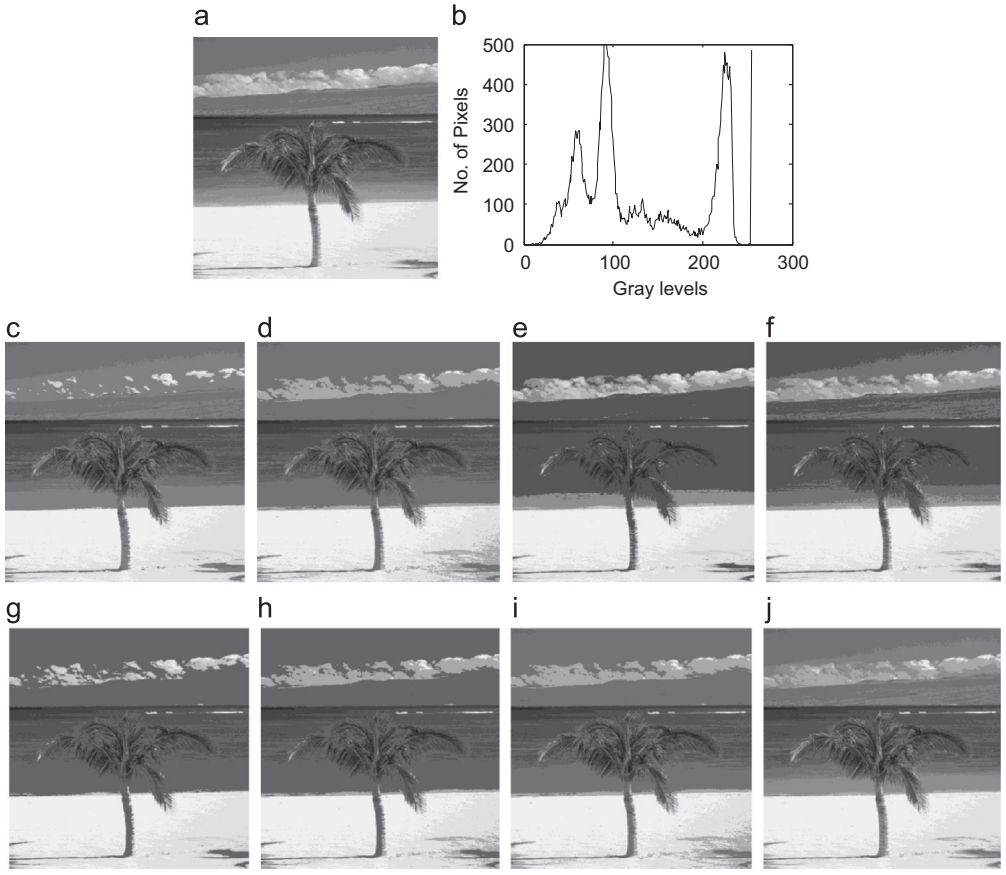

Fig. 12. Results of 46076 Image using BFO and CS. a. Original 46076 Image, b. Histogram of 46076 image, c. 2-level thresholding using BFO, d. 3-level thresholding using BFO, e. 4-level thresholding using BFO, f. 5-level thresholding using BFO, g. 2-level thresholding using CS, h. 3-level thresholding using CS, i. 4-level thresholding using CS, j. 5-level thresholding using CS.

This is obtained by maximizing the objective function for bi-level thresholding:

$$T_{opt} = \arg\max[S_q^A(t)+S_q^B(t)+(1-q)\cdot S_q^A(t)\cdot S_q^B(t)] \quad (7)$$

Subject to the following constraint:

$$|P^A+P^B|-1 < S < 1-|P^A-P^B|$$

where

$$S(t) = S = [S_q^A(t)+S_q^B(t)+(1-q)\cdot S_q^A(t)\cdot S_q^B(t)] \quad (8)$$

It is noteworthy to mention here that the constraint equation (8) is verified inside the proposed algorithm in order to ensure stability. Here the optimal threshold value 'T' is the gray level that maximizes Eq. (7) subject to the constraint defined in Eq. (8). Note that the solution is obtained by solving this constrained optimization problem.

This method can also be easily extended to multi-level thresholding. The optimal multilevel thresholding problem is configured as an m-dimensional optimization problem. Interestingly, 'm' optimal thresholds $[T_1, T_2,\ldots,T_m]$ for a given image can be determined by maximizing the objective function:

$$[T_1,T_2,\ldots,T_m] = \arg\max[S_q^1(t)+S_q^2(t)+\ldots+S_q^M(t) + (1-q)\cdot S_q^1(t)\cdot S_q^2(t)\cdots S_q^M(t)] \quad (9)$$

where

$$S_q^1(t) = \frac{1-\sum_{i=1}^{t_1}(p_i/P^1)^q}{q-1}$$

$$S_q^2(t) = \frac{1-\sum_{i=t_1+1}^{t_2}(p_i/P^2)^q}{q-1}$$

and

$$S_q^M(t) = \frac{1-\sum_{i=t_{m+1}}^{G}(p_i/P^M)^q}{q-1}, \quad M = m+1$$

Subject to the set of constraints defined below:

$$|P^1+P^2|-1 < S^1 < 1-|P^1-P^2|$$
$$|P^2+P^3|-1 < S^2 < 1-|P^2-P^3| \quad (10)$$
$$|P^m+P^{m+1}|-1 < S^M < 1-|P^m-P^{m+1}|$$

These equations are important for stability point of view. Here, P^1, P^2,..., P^{m+1} corresponding to S^1, S^2, ..., S^M are computed with T_1, T_2,..., T_m, respectively. To be specific, here the aim is to optimize the objective function defined by Eq. (9) subject to the constraints given in the set of equations Eq. (10) using Cuckoo search algorithm. These constraints are checked inside the proposed algorithm. Thus, stability of the proposed algorithm is ensured for different types of images.

Note that 'M' refers to 'class M' and is equal to $m+1$. Here Cuckoo search algorithm is used to optimize the objective function defined by Eq. (9) subject to the constraints equation (10). The validity of the theory is proved with the help of numerical illustrations.

3. Cuckoo search algorithm

Recently, the Cuckoo Search (CS) algorithm is proposed by Yang and Deb [20,21] and Rajabioun [22]. It is important to mention here that CS is also a population based stochastic global search algorithm. We can use CS for finding a global optimal solution. CS algorithm is

inspired by the obligate brood parasitism of some cuckoo species by laying their eggs in the nests of other host birds of other species found in different places. While considering the multi-dimensional space where the optimal solution is sought, the authors in [20–22] proved that the CS algorithm provide us efficient means for maximizing the objective function. Note that the quality or fitness of a solution can simply be proportional to the value of the objective function as it is the case with other search algorithms. CS is different from other population based search algorithms in the sense that it is a meta-heuristic search algorithm. Since the CS utilizes the coded discrete information, it can easily be applied to ill-structured discrete optimization problems as well as to continuous optimization problems. It is capable in finding optimal solutions to complex problems without any exhaustive search. Therefore CS may be useful for nonlinear problems and multi-objective optimizations. Here in a CS algorithm, a pattern corresponds to a nest and each individual attribute of the pattern corresponds to a Cuckoo-egg.

The underlying principles of CS algorithm are:

1. Interestingly, each cuckoo bird lays one egg at a time, and dumps its egg in a randomly chosen nest of another bird from other species.
2. Usually the best nests containing high quality eggs are carried over to the next generations.
3. The number of available host nests is fixed. And the egg laid by a cuckoo bird is discovered by the host bird with a probability $p_a \in [0, 1]$. Note that the worst nests are discovered and dropped from further calculations.

Based on these three principles, the basic steps of the CS algorithm are summarized below (presented as a Pseudo Code in Fig. 1).

The choice of control parameters in CS algorithm is simple and required for implementing the algorithm. Note that the control parameters of the CS algorithm are the scale factor (β) and the mutation probability value (p_a) [23]. While generating new

Table 5
Comparison of best objective function values and their corresponding threshold values.

Test images	m	Best objective function values				Optimum threshold values			
		CS	BFO	PSO	GA	CS	BFO	PSO	GA
Lena	2	**0.888976**	0.8889	0.8889	0.8889	98,170	120,164	120,164	120,164
	3	**1.296296**	1.296278	1.296268	1.296247	79,116,169	81,124,178	110,149,187	98,159,181
	4	**1.654320**	1.654271	1.654255	1.654208	73,101,129,185	85,124,161,193	85,118,164,200	86,120,151,205
	5	**1.995884**	1.995787	1.995773	1.995717	56,106,153,175,213	76,108,136,164,193	86,117,142,166,196	95,130,152,173,200
Pepper	2	**0.888983**	0.8889	0.8889	0.8889	51,161	82,154	82,154	82,154
	3	**1.296296**	1.296278	1.296274	1.296262	109,157,218	86,118,190	93,133,179	75,103,182
	4	**1.654319**	1.654264	1.654248	1.654225	47,61,81,147	71,121,161,197	73,121,141,176	73,109,141,193
	5	**1.995874**	1.995771	1.995766	1.995739	51,90,134,171,203	70,109,139,169,197	78,111,141,169,198	78,105,139,168,200
Cameraman	2	**0.888960**	0.8889	0.8889	0.8889	72,155	120,154	120,154	120,154
	3	**1.296294**	1.296189	1.296180	1.296141	36,66,145	78,128,178	78,121,173	81,143,170
	4	**1.654319**	1.654190	1.654183	1.654177	45,86,104,160	91,123,156,211	82,122,154,201	76,116,148,202
	5	**1.995882**	1.995674	1.995669	1.995663	51,94,133,171,202	70,107,134,158,200	78,110,133,159,199	88,118,143,169,205
House	2	**0.888886**	0.888761	0.888761	0.888761	79,173	87,145	87,145	87,145
	3	**1.296292**	1.296092	1.296090	1.296052	56,127,182	88,133,199	90,133,199	82,123,177
	4	**1.654316**	1.653630	1.653586	1.653581	61,93,152,212	67,105,146,189	70,112,152,189	73,111,151,189
	5	**1.995879**	1.994217	1.993744	1.993426	49,92,131,170,203	66,95,121,155,200	70,104,134,160,212	60,99,114,158,198
Hunter	2	**0.888930**	0.8889	0.8889	0.8889	98,170	94,137	94,137	94,137
	3	**1.296295**	1.296270	1.296267	1.296227	70,117,170	82,118,171	83,143,174	87,147,173
	4	**1.654320**	1.654258	1.654255	1.654240	61,91,150,193	71,110,142,182	78,109,143,187	90,119,150,191
	5	**1.995883**	1.995766	1.995720	1.995713	51,91,133,172,203	65,93,123,150,182	70,103,139,174,198	79,114,144,174,198
Map	2	**0.888885**	0.881206	0.881206	0.881206	98,172	114,176	114,176	114,176
	3	**1.296192**	1.273982	1.267481	1.232429	88,147,194	84,142,198	90,131,183	80,145,172
	4	**1.654020**	1.587902	1.585544	1.579716	56,109,152,206	73,113,156,203	78,121,158,189	80,117,157,199
	5	**1.993884**	1.828422	1.818369	1.788800	49,92,131,170,202	75,112,147,174,206	79,113,142,170,191	91,118,144,174,206
Livingroom	2	**0.888888**	0.888881	0.888881	0.888881	75,168	81,144	81,144	81,144
	3	**1.296292**	1.296281	1.296275	1.296255	50,115,169	89,143,197	91,137,198	88,117,178
	4	**1.654320**	1.654263	1.654247	1.654244	61,91,150,213	67,107,145,186	87,126,165,200	90,126,158,199
	5	**1.995864**	1.995743	1.995701	1.995627	51,92,134,171,201	72,111,139,164,199	71,125,150,176,205	69,126,157,182,204
Butterfly	2	**0.888889**	0.888825	0.888825	0.888825	78,150	97,136	97,136	97,136
	3	**1.296290**	1.296202	1.296190	1.296168	84,113,167	99,135,197	100,135,185	89,124,169
	4	**1.654320**	1.653424	1.652617	1.652564	63,92,150,210	95,120,144,189	89,122,143,178	94,121,141,179
	5	**1.995884**	1.994823	1.991453	1.989359	50,90,132,165,198	89,114,141,170,213	70,107,134,162,189	70,119,140,170,214
Airplane	2	**0.888982**	0.8889	0.8889	0.8889	78,171	72,153	72,153	72,153
	3	**1.296293**	1.296223	1.296204	1.296180	71,125,182	99,143,193	98,134,192	89,148,172
	4	**1.654318**	1.654277	1.654262	1.654243	68,107,135,190	68,103,135,182	85,117,153,180	79,111,153,173
	5	**1.995878**	1.995795	1.995784	1.995768	50,89,125,156,182	61,94,121,150,185	75,107,134,157,185	73,98,131,162,192
Baboon	2	**0.888976**	0.8889	0.8889	0.8889	62,126	91,147	91,147	91,147
	3	**1.296296**	1.296284	1.296274	1.296202	60,131,147	111,148,188	108,155,181	111,136,193
	4	**1.654320**	1.654266	1.654262	1.654241	59,99,126,186	75,114,146,175	62,115,144,174	94,125,152,177
	5	**1.995884**	1.995744	1.995737	1.995708	57,69,123,156,198	78,106,136,157,179	84,110,132,153,175	90,116,139,159,180
46076	2	**0.888888**	0.888868	0.8888	0.8888	68,186	78,165	70,164	60,164
	3	**1.296295**	1.296249	1.296228	1.296217	71,152,199	51,116,158	40,149,187	30,159,181
	4	**1.654320**	1.654057	1.654055	1.654008	43,134,164,212	62,92,153,212	55,118,164,200	46,120,151,205
	5	**1.995883**	1.995717	1.995703	1.995700	39,40,89,152,206	51,93,134,171,200	56,117,142,166,196	45,130,152,173,200

solutions $x^{(t+1)}$, for a cuckoo i, a Levy flight is performed:

$$x_i^{(t+1)} = x_i^{(t)} + \alpha \oplus \text{Levy}(\lambda), \quad (11)$$

where $\alpha > 0$ is the step size. Here we choose $\alpha = 1$. Levy flights provide a random walk while their random steps are drawn from a Levy distribution for large steps defined by:

$$\text{Levy} \backsim u = t^{-\lambda}, \quad (1 < \lambda \leq 3), \quad (12)$$

which has an infinite variance and infinite mean.

4. Multilevel thresholding algorithm using CS

In this section, we introduce an efficient algorithm to find the optimal threshold values for multilevel thresholding. These optimal thresholds are useful for image segmentation. Here the algorithm is developed to maximize the non-extensive Tsallis entropy. The proposed method is simple and easy to implement. Different steps of our algorithm are presented below:

Step 1: Select the number of nests (different solutions). Choose an appropriate value for the mutation probability parameter.
Step 2: Select the simple bounds of the search domain i.e. lower bound and upper bound.
Step 3: Generate random initial solutions by evaluating the objective function as defined in Eq. (9) and get the current best nest subject to the constraints defined in Eq. (10). The proposed constrained optimization idea is incorporated in this algorithm to ensure stability.
Step 4: While the stopping criteria is not met, get a cuckoo randomly by Levy flights. Evaluate its fitness and keep it as the current value. Choose a nest randomly and compare its fitness with the current one. Replace the new value, if it satisfies the criteria of maximum.

Table 6
Comparison of PSNR and Standard Deviation.

Test Images	m	PSNR (dB)				Standard deviation			
		CS	BFO	PSO	GA	CS	BFO	PSO	GA
Lena	2	**18.5339**	15.2419	15.2419	15.2419	**0.0000**	0.0000	0.0000	0.0000
	3	**21.2417**	17.4715	17.1425	16.9455	**0.0000**	1.6827e−006	2.5418e−006	3.8999e−006
	4	**22.1490**	19.5070	19.4324	19.0207	**1.1102e−016**	3.4304e−006	1.3306e−005	1.9104e−005
	5	**23.2855**	20.9916	20.5637	19.8703	**3.6425e−10**	4.5355e−006	1.6797e−005	2.7208e−005
Pepper	2	**18.9283**	12.9108	12.9108	12.9108	**0.0000**	0.0000	0.0000	0.0000
	3	**19.6433**	16.6563	16.0269	15.5628	**0.0000**	2.8014e−006	7.3578e−006	2.0199e−005
	4	**21.7088**	19.2433	16.7109	16.3735	**4.4409e−016**	1.6217e−005	7.0094e−005	1.7406e−004
	5	**22.2022**	20.4910	20.2089	19.7642	**5.0064e−008**	2.0208e−004	6.3010e−004	1.1678e−003
Cameraman	2	**18.8399**	10.6258	10.6258	10.6258	**0.0000**	0.0000	0.0000	0.0000
	3	**21.2327**	15.6856	14.9951	14.5900	**1.4203e−009**	4.7916e−006	5.4543e−006	8.4892e−006
	4	**22.2643**	16.7835	15.9187	14.9756	**1.7631e−008**	3.6715e−005	7.5181e−005	1.1024e−004
	5	**24.8527**	17.8802	17.2393	16.6026	**2.8747e−007**	6.6163e−005	1.0319e−004	7.7199e−004
House	2	**18.6233**	12.9865	12.9865	12.9865	**0.0000**	0.0000	0.0000	0.0000
	3	**20.6930**	14.0213	13.8104	13.6918	**2.7660e−009**	2.5025e−006	4.3646e−005	6.9786e−005
	4	**22.4978**	16.8884	16.4428	16.1794	**1.0215e−008**	3.7689e−006	8.7702e−005	1.1385e−004
	5	**23.9747**	17.5635	16.7719	16.5772	**4.8449e−008**	7.5181e−005	9.5166e−005	1.2255e−004
Hunter	2	**16.5926**	11.3848	11.3848	11.3848	**0.0000**	0.0000	0.0000	0.0000
	3	**18.0245**	14.5772	14.5135	14.0724	**6.6715e−011**	4.6660e−007	1.8965e−006	1.0060e−005
	4	**19.2790**	16.2874	15.4496	14.1926	**3.755e−009**	1.8203e−006	4.2172e−006	1.0886e−005
	5	**19.8811**	17.3380	16.6426	15.6197	**2.7454e−008**	5.4613e−005	1.2255e−004	9.3619e−004
Map	2	**18.1283**	16.6045	16.6045	16.6045	**0.0000**	0.0000	0.0000	0.0000
	3	**19.6091**	18.4286	18.0419	16.2161	**8.4267e−011**	5.6090e−007	1.0167e−006	4.6714e−006
	4	**21.3499**	20.6499	19.7997	19.7340	**8.2589e−010**	5.0556e−004	1.1493e−003	3.9730e−003
	5	**23.6609**	22.1638	21.8968	21.5746	**3.6792e−008**	6.5988e−004	8.1623e−003	1.6169e−002
Livingroom	2	**14.6260**	13.1208	13.1208	13.1208	**0.0000**	0.0000	0.0000	0.0000
	3	**18.2750**	17.1198	16.9810	16.5873	**0.0000**	1.6980e−006	6.9103e−005	7.0160e−004
	4	**19.6732**	19.2320	18.8655	18.5189	**3.5884e−009**	4.3245e−006	8.4404e−006	2.2951e−005
	5	**21.4269**	21.3385	20.9931	20.5597	**5.8242e−008**	4.3515e−005	9.3293e−005	1.8187e−004
Butterfly	2	**16.8125**	13.0516	13.0516	13.0516	**0.0000**	0.0000	0.0000	0.0000
	3	**19.1126**	18.1337	17.8316	17.2964	**4.4408e−016**	1.4899e−006	4.8520e−005	8.5774e−004
	4	**21.4784**	20.0356	18.9792	18.8382	**4.4408e−015**	1.9529e−005	6.7992e−004	1.3908e−005
	5	**23.7384**	21.9096	21.4406	20.2055	**4.6193e−008**	6.4439e−005	9.1016e−004	5.1122e−003
Airplane	2	**16.8125**	13.7290	13.7290	13.7290	**0.0000**	0.0000	0.0000	0.0000
	3	**19.1125**	15.8742	15.5913	14.6681	**2.4517e−009**	8.3154e−007	3.1114e−006	6.9412e−006
	4	**21.4784**	16.3276	15.6294	14.9701	**7.9278e−008**	9.5166e−007	2.6305e−006	9.2004e−006
	5	**22.7385**	17.6049	17.6077	16.1579	**1.9810e−007**	5.1122e−006	3.3007e−005	6.3861e−005
Baboon	2	**18.6188**	13.1404	13.1404	13.1404	**0.0000**	0.0000	0.0000	0.0000
	3	**19.5187**	18.1076	17.0809	16.7728	**1.1102e−015**	2.9078e−006	9.3397e−006	1.2993e−005
	4	**20.4325**	17.5204	17.1462	17.1583	**2.0004e−010**	3.4997e−006	7.2225e−006	1.3714e−005
	5	**21.0668**	18.7616	18.2718	17.2903	**4.6354e−008**	9.7325e−006	1.1321e−005	1.8993e−005
46076	2	**20.6830**	19.9993	18.9983	19.9893	**0.0000**	0.0000	0.0000	0.0000
	3	**22.0758**	20.8095	20.7095	20.7005	**1.1102e−015**	2.80e−006	2.32e−006	2.15e−006
	4	**23.6611**	22.5993	21.5963	20.5983	**2.0004e−010**	3.69e−006	3.09e−006	3.00e−006
	5	**25.7109**	23.5170	22.5240	21.5150	**4.6354e−008**	9.63e−006	9.13e−006	8.63e−006

Step 5: Abandon a fraction (p_a) of worst nests and build new ones at new locations via Levy flights.
Step 6: Keep the best solutions, rank the solutions and find the current best.
Step 7: Output the optimal threshold values corresponding to the best nests.

The lower bound and the upper bound are incorporated in the algorithm according to the constraints defined above. This algorithm is validated with the help of many illustrations shown in Section 5. One of the main advantages of the proposed algorithm over other algorithms is that we can check the stability from the beginning of the procedure, since we introduced desired stability as appropriate constraints.

5. Results and discussions

In this section, experimental results are presented. The experiments are carried out on a P4 Dual-core platform with a 1.75 GHz processor and 1 GB memory, running under the Windows 7.0 operating system. The algorithms are developed using MATLAB Release 2009. Tables 1–4 represent the parameters used for GA, PSO, BFO and CS algorithms. The results obtained are then compared with BFO, PSO and GA as presented by Sathya and Kayalvizhi [13]. The parameters for GA, PSO and BFO used here are same as that of Sathya and Kayalvizhi [13]. Table 4 presents parameter values we have used for our algorithm.

It is wise to reiterate the fact that a simple CS relies on the selection of number of nests and number of maximum iterations or objective function values greater than some tolerance limit. To start the search, CS requires an initial set of points. This set of points is called population size, which is quite analogous to biological systems. A random number generator generates the initial solution. Initial points in the search space are equal to the number of nests. Here the population size is 40. Number of maximum iterations considered for this experiment is 20. Two important control parameters are mutation probability value (p_a) and the scale factor (β). In this experiment, mutation probability $p_a=0.25$ and scale factor $\beta=1.5$. The mutation probability value (p_a) decides the fraction of *worst* nests that are dropped from further calculations and build new ones at new locations via Levy flights. Mutation probability of '1' implies that a host bird recognizes the Cuckoo's egg and throws it or simply leaves its nest. Mutation probability of '0' implies that a host bird does not recognize the Cuckoo's egg and may hatch it. Therefore the range of mutation probability is chosen between 0 and 1 and a value of 0.25 gives satisfactory results for this experiment. The scale factor 'β' plays an important role in controlling the step size of Levy flights [23].

Already we have discussed about q in Section 2, which is the measure of degree of non-extensivity of the system called Tsallis parameter or entropic index. To be precise, a value of 'q' less than one leads to subextensive entropy, where $S_q(A+B)<S_q(A)+S_q(B)$. When 'q' is equal to one, the entropy is called extensive entropy and $S_q(A+B)=S_q(A)+S_q(B)$. If 'q' is greater than one, then we get superextensive entropy and $S_q(A+B)>S_q(A)+S_q(B)$. It is noteworthy to mention here that superextensive entropy, a class of entropy, is useful for multilevel thresholding for image segmentation. Here we choose $q=4$ for our calculations.

In this simulation, we tried to solve our constrained optimization problem with different population size, $n=20$, 30, 40, 50. It is observed that with increase in the population size the maximum objective function value also increases initially. But when it is increased beyond $n=40$, the increase in the objective function value was marginal, whereas the CPU run time increases considerably. Hence, in this experiment we choose the population size of 40. While doing simulation we observed that an increase in the number of iterations does not improve the objective function value further. So we choose an appropriate value of 20 for this experiment.

We have also tried different values of mutation probability, for example, $p_a=0.20$, 0.25, 0.30. But it is observed that $p_a=0.25$ yields best results in this experiment. The objective function as defined in Eq. (9) subject to the constraints defined in Eq. (10) is maximized for the best nests. These nests are selected with highest values of the fitness/objective function. By repeated iterations and natural selection we get the maximum value of the objective function. A fraction (p_a) of worst nests is dropped and new ones are built at new locations via Levy flights. The termination criterion is fulfilled provided the objective function value does not improve further. The best nests correspond to the optimum threshold values for an image. It may be noted that stability and validity of the proposed algorithm is ensured through repeated checking of the constraints defined in Eq. (10). The

Table 7
Comparison of CPU time, SSIM and FSIM.

Test images	m	CPU time		SSIM		FSIM	
		CS	BFO	CS	BFO	CS	BFO
Lena	2	**8.0837**	17.0671	**0.9253**	0.8522	**0.9545**	0.9369
	3	**8.3064**	18.9397	**0.9638**	0.9318	**0.9817**	0.9694
	4	**8.7264**	19.7398	**0.9702**	0.9567	**0.9796**	0.9722
	5	**9.8660**	19.3526	**0.9737**	0.9735	**0.9825**	0.9820
Pepper	2	**7.9601**	13.6662	**0.9129**	0.9117	**0.9765**	0.9669
	3	**8.3510**	13.0076	**0.9460**	0.9205	**0.9749**	0.9702
	4	**8.4639**	14.4763	**0.9659**	0.9512	**0.9829**	0.9808
	5	**9.4155**	15.1942	**0.9773**	0.9602	**0.9869**	0.9833
Cameraman	2	**6.4983**	16.9576	**0.9438**	0.8968	**0.9831**	0.9579
	3	**6.9959**	16.5088	**0.9678**	0.9217	**0.9810**	0.9622
	4	**7.2340**	17.6882	**0.9783**	0.9534	**0.9922**	0.9858
	5	**8.1029**	17.7188	**0.9793**	0.9661	**0.9972**	0.9884
House	2	**5.6700**	17.2772	**0.9236**	0.9224	**0.9866**	0.9879
	3	**6.5587**	17.4325	**0.9547**	0.9371	**0.9944**	0.9926
	4	**8.2643**	18.0423	**0.9689**	0.9478	**0.9963**	0.9913
	5	**8.3900**	20.8635	**0.9793**	0.9698	**0.9974**	0.9970
Hunter	2	**7.2864**	16.7736	**0.9430**	0.8728	**0.9863**	0.9790
	3	**7.4230**	16.6818	**0.9634**	0.9123	**0.9887**	0.9738
	4	**7.5468**	16.8745	**0.9649**	0.9375	**0.9912**	0.9882
	5	**8.4421**	18.9907	**0.9746**	0.9467	**0.9952**	0.9904
Map	2	**7.3831**	16.3412	**0.9423**	0.8788	**0.9963**	0.9816
	3	**8.1370**	16.9876	**0.9499**	0.9383	**0.9944**	0.9901
	4	**8.2775**	17.0213	**0.9741**	0.9464	**0.9988**	0.9915
	5	**9.7967**	19.7201	**0.9767**	0.9638	**0.9975**	0.9935
Livingroom	2	**7.4939**	16.9229	**0.8446**	0.7945	**0.9561**	0.9508
	3	**8.1606**	17.7629	**0.9350**	0.9203	**0.9815**	0.9766
	4	**8.8905**	17.8943	**0.9397**	0.9346	**0.9916**	0.9750
	5	**9.5986**	19.9807	**0.9706**	0.9445	**0.9767**	0.9741
Butterfly	2	**7.0794**	16.6434	**0.9236**	0.8691	**0.9592**	0.9391
	3	**7.8238**	17.3209	**0.9547**	0.9223	**0.9768**	0.9643
	4	**8.0921**	17.9045	**0.9689**	0.9377	**0.9819**	0.9708
	5	**9.0213**	18.3502	**0.9793**	0.9408	**0.9850**	0.9699
Airplane	2	**6.1371**	15.4562	**0.8935**	0.8377	**0.9651**	0.9612
	3	**6.9829**	15.8934	**0.9499**	0.9325	**0.9765**	0.9701
	4	**9.6838**	17.3201	**0.9709**	0.9420	**0.9881**	0.9823
	5	**9.8409**	19.3254	**0.9727**	0.9491	**0.9920**	0.9885
Baboon	2	**6.8131**	15.9845	**0.9338**	0.8369	**0.9859**	0.9633
	3	**6.8874**	16.9823	**0.9467**	0.8623	**0.9823**	0.9689
	4	**8.2326**	17.0034	**0.9543**	0.9165	**0.9900**	0.9713
	5	**8.7030**	17.0231	**0.9578**	0.9287	**0.9904**	0.9865
46076	2	**8.0837**	9.96	**0.9683**	0.9669	**0.9779**	0.9704
	3	**8.3064**	10.89	**0.9771**	0.9670	**0.9765**	0.9776
	4	**8.7264**	11.03	**0.9814**	0.9812	**0.9821**	0.9792
	5	**9.8660**	11.67	**0.9912**	0.9845	**0.9907**	0.9814

algorithm is validated over a range of images. The thresholded images are shown for a comparison.

In this experiment, we consider 11 different images for thresholding. An image (image 46076) Fig. 12 from Berkley University dataset for image segmentation [24] is also used for testing our method to strengthen the results. Optimal thresholds are obtained using the proposed method. Here we present 2-level, 3-level, 4-level and 5-level thresholding for visual perception. The size of all images considered for this experiment is 512×512. The image thresholding results are displayed in Figs. 2–12. The thresholding results using CS are also compared with BFO algorithm. These figures show the effectiveness of the proposed method. For instance, the output with 5-level thresholding of Lena image using CS shown in Fig. 2(j) seems qualitatively better as compared to the 5-level thresholding using BFO shown in Fig. 2(f). Similarly, all other images displayed in this paper also reveal the fact that thresholding results obtained by our method seem qualitatively better as compared to earlier methods [13]. From Figs. 2–12, it is also observed that thresholding results are better qualitatively when we increase the number of thresholds. For example, let us consider histogram image (of cameraman) shown in Fig. 4(b), which is multimodal in nature. That is why probably the thresholding result using 5-level thresholding (of cameraman) shown in Fig. 4(j) seems much better than the 2-level thresholding (of cameraman) shown in Fig. 4(g). Similar is the situation with all other images.

The effect of multilevel thresholding is visible from different images. In Fig. 4(g), the background in the cameraman image is not clearly distinct with two level thresholding. But as the number of threshold is increased to 5 (i.e. Fig. 4(j)), the background becomes identifiable. Similarly in Fig. 9(g), the butterfly image mixes up with the background objects. But as the number of threshold is increased to 5 (i.e. Fig. 9(j)), the butterfly image becomes clearly identifiable. Further, an increase in the number of thresholds from 2 to 5 in Fig. 12(g)–(j) makes the cloud object clearer. This effect of multilevel thresholding is observed in other set of images also.

Sarkar et al. [25] presented the construction equation for segmented image. However, in this paper we propose the construction logic for the thresholded image as follows:

For 2-level thresholding, let T_1 and T_2 represent the optimum threshold gray values. Let T_i is the gray value of a pixel at position i,

Table 8
Comparison of CS and ABC algorithms.

Test images	m	CPU time		PSNR		SSIM		FSIM	
		CS	ABC	CS	ABC	CS	ABC	CS	ABC
Lena	2	**8.0837**	9.96	**18.5339**	11.2131	**0.9253**	0.6820	**0.9545**	0.8999
	3	**8.3064**	10.89	**21.2417**	11.3250	**0.9638**	0.7498	**0.9817**	0.9634
	4	**8.7264**	11.03	**22.1490**	12.5290	**0.9702**	0.8033	**0.9796**	0.9670
	5	**9.8660**	11.67	**23.2855**	18.2878	**0.9737**	0.9321	**0.9825**	0.9818
Pepper	2	**7.9601**	8.57	**18.9283**	15.4509	**0.9129**	0.8570	**0.9765**	0.8790
	3	**8.3510**	9.13	**19.6433**	16.9023	**0.9460**	0.8956	**0.9749**	0.9260
	4	**8.4639**	10.45	**21.7088**	18.3478	**0.9659**	0.9101	**0.9829**	0.9398
	5	**9.4155**	11.87	**22.2022**	18.9900	**0.9773**	0.9187	**0.9869**	0.9678
Cameraman	2	**6.4983**	9.56	**18.8399**	16.9820	**0.9438**	0.6825	**0.9831**	0.8841
	3	**6.9959**	9.89	**21.2327**	17.3475	**0.9678**	0.7455	**0.9810**	0.8810
	4	**7.2340**	8.67	**22.2643**	19.4362	**0.9783**	0.8873	**0.9922**	0.9612
	5	**8.1029**	10.20	**24.8527**	19.8520	**0.9793**	0.9122	**0.9972**	0.9652
House	2	**5.6700**	7.96	**18.6233**	15.6230	**0.9236**	0.7145	**0.9866**	0.8990
	3	**6.5587**	8.45	**20.6930**	16.5363	**0.9547**	0.7596	**0.9944**	0.9601
	4	**8.2643**	10.78	**22.4978**	18.7828	**0.9689**	0.7954	**0.9963**	0.9653
	5	**8.3900**	10.98	**23.9747**	19.4747	**0.9793**	0.8045	**0.9974**	0.9710
Hunter	2	**7.2864**	9.56	**16.5926**	13.5236	**0.9430**	0.8412	**0.9863**	0.9688
	3	**7.4230**	9.78	**18.0245**	15.0265	**0.9634**	0.8614	**0.9887**	0.9698
	4	**7.5468**	9.88	**19.2790**	15.4658	**0.9649**	0.8649	**0.9912**	0.9735
	5	**8.4421**	10.45	**19.8811**	16.0023	**0.9746**	0.8926	**0.9952**	0.9785
Map	2	**7.3831**	10.53	**18.1283**	14.9454	**0.9423**	0.7789	**0.9963**	0.9823
	3	**8.1370**	11.45	**19.6091**	16.2635	**0.9499**	0.7999	**0.9944**	0.9814
	4	**8.2775**	11.86	**21.3499**	17.4635	**0.9741**	0.8512	**0.9988**	0.9878
	5	**9.7967**	12.45	**23.6609**	18.6654	**0.9767**	0.8747	**0.9975**	0.9888
Livingroom	2	**7.4939**	10.13	**14.6260**	11.2365	**0.8446**	0.6445	**0.9561**	0.8989
	3	**8.1606**	11.78	**18.2750**	15.9652	**0.9350**	0.7366	**0.9815**	0.9615
	4	**8.8905**	11.98	**19.6732**	15.2364	**0.9397**	0.7350	**0.9916**	0.9874
	5	**9.5986**	12.86	**21.4269**	18.4478	**0.9706**	0.7965	**0.9767**	0.9545
Butterfly	2	**7.0794**	10.23	**16.8125**	11.3245	**0.9236**	0.7120	**0.9592**	0.9389
	3	**7.8238**	11.45	**19.1126**	11.9876	**0.9547**	0.7234	**0.9768**	0.9421
	4	**8.0921**	12.56	**21.4784**	15.8740	**0.9689**	0.8965	**0.9819**	0.9743
	5	**9.0213**	12.87	**23.7384**	19.5436	**0.9793**	0.9688	**0.9850**	0.9849
Airplane	2	**6.1371**	9.77	**16.8125**	12.4789	**0.8935**	0.7953	**0.9651**	0.9551
	3	**6.9829**	10.45	**19.1125**	15.2322	**0.9499**	0.8945	**0.9765**	0.9665
	4	**9.6838**	12.78	**21.4784**	18.6533	**0.9709**	0.9004	**0.9881**	0.9741
	5	**9.8409**	12.98	**22.7385**	19.7458	**0.9727**	0.9044	**0.9920**	0.9821
Baboon	2	**6.8131**	8.18	**18.6188**	13.2245	**0.9338**	0.8336	**0.9859**	0.8843
	3	**6.8874**	8.78	**19.5187**	14.6533	**0.9467**	0.8396	**0.9823**	0.8823
	4	**8.2326**	10.43	**20.4325**	17.2133	**0.9543**	0.9038	**0.9900**	0.8900
	5	**8.7030**	10.73	**21.0668**	18.0546	**0.9578**	0.9078	**0.9904**	0.8954

then the thresholded image will have gray values,

$T_i = T_i$, if $T_i \leq T_1$
$T_i = T_1$, if $T_1 < T_i \leq T_2$ and
$T_i = T_i$, if $T_i > T_2$, for all i.

For 3-level thresholding, let T_1, T_2 and T_3 represent the optimum threshold gray values. Let T_i is the gray value of a pixel at position i, then the thresholded image will have gray values,

$T_i = T_i$, if $T_i \leq T_1$
$T_i = T_1$, if $T_1 < T_i \leq T_2$
$T_i = T_2$, if $T_2 < T_i \leq T_3$ and
$T_i = T_i$, if $T_i > T_3$, for all i.

The same logic is implemented for all other levels of thresholding.

Numerical illustrations are presented below with various parameters such as objective function values, optimal threshold values, Peak Signal to Noise ratio (PSNR in dB), CPU time (in seconds), Standard Deviation, Structure similarity (SSIM) index [26] and Feature similarity (FSIM) index [27]. Numerical findings are displayed in Tables 5–7 . A higher objective function value indicates better result. Our method has given better results as observed from Table 5, which display objective function values and their corresponding threshold values for a comparison. Our method seems to be quantitatively better as compared to BFO, PSO and GA. It is seen that the fitness/objective function values are higher for our algorithm using CS, which is desirable.

The PSNR value is calculated as

$$PSNR(\text{in dB}) = 20\log_{10}\left(\frac{255}{RMSE}\right) \tag{13}$$

where

$$RMSE = \sqrt{\frac{1}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N}[I(i,j)-\tilde{I}(i,j)]^2} \tag{14}$$

and M, N are the size of image, I is the original image and $\tilde{I}$ is the thresholded image at a particular level. A higher value of PSNR indicates a better quality of thresholding. For all the test images, the proposed method proves to be better than BFO, ABC, PSO and GA. The standard deviation

Table 9
Comparison of CS and ABC algorithms.

Test images	m	Best objective function value		Optimum threshold values		SD	
		CS	ABC	CS	ABC	CS	ABC
Lena	2	**0.888976**	0.888888	98,170	58,196	**0.0000**	0.0000
	3	**1.296296**	1.296294	79,116,169	23,132,199	**0.0000**	1.68e−006
	4	**1.654320**	1.654319	73,101,129,185	23,124,132,175	**1.1102e−016**	2.13e−006
	5	**1.995884**	1.995881	56,106,153,175,213	17,103,122,136,212	**3.6425e−10**	3.43e−006
Pepper	2	**0.888983**	0.888888	51,161	46,198	**0.0000**	0.0000
	3	**1.296296**	1.296295	109,157,218	99,161,183	**0.0000**	1.70e−006
	4	**1.654319**	1.654308	47,61,81,147	37,89,138,246	**4.4409e−016**	2.65e−005
	5	**1.995874**	1.995864	51,90,134,171,203	28,97,107,193,206	**5.0064e−008**	4.12e−004
Cameraman	2	**0.888960**	0.888950	72,155	62,185	**0.0000**	0.0000
	3	**1.296294**	1.296289	36,66,145	26,86,175	**1.4203e−009**	3.69e−006
	4	**1.654319**	1.654290	45,86,104,160	35,106,124,195	**1.7631e−008**	5.66e−005
	5	**1.995882**	1.995774	51,94,133,171,202	31,107,184,198,220	**2.8747e−007**	7.68e−005
House	2	**0.888886**	0.888771	79,173	59,195	**0.0000**	0.0000
	3	**1.296292**	1.296192	56,127,182	38,133,199	**2.7660e−009**	2.57e−006
	4	**1.654316**	1.653630	61,93,152,212	41,105,176,236	**1.0215e−008**	3.79e−006
	5	**1.995879**	1.994227	49,92,131,170,203	26,95,141,185,220	**4.8449e−008**	7.61e−005
Hunter	2	**0.888930**	0.888920	98,170	84,187	**0.0000**	0.0000
	3	**1.296295**	1.296275	70,117,170	52,128,191	**6.6715e−011**	4.96e−007
	4	**1.654320**	1.654228	61,91,150,193	41,110,172,212	**3.755e−009**	1.22e−006
	5	**1.995883**	1.995760	51,91,133,172,203	45,93,153,180,242	**2.7454e−008**	5.16e−005
Map	2	**0.888885**	0.881216	98,172	84,176	**0.0000**	0.0000
	3	**1.296192**	1.283982	88,147,194	74,149,199	**8.4267e−011**	5.60e−007
	4	**1.654020**	1.589902	56,109,152,206	33,113,159,213	**8.2589e−010**	5.35e−004
	5	**1.993884**	1.928422	49,92,131,170,202	25,112,147,184,216	**3.6792e−008**	6.79e−004
Livingroom	2	**0.888888**	0.888881	75,168	61,184	**0.0000**	0.0000
	3	**1.296292**	1.296280	50,115,169	39,143,197	**0.0000**	1.49e−006
	4	**1.654320**	1.654260	61,91,150,213	47,107,165,226	**3.5884e−009**	4.62e−006
	5	**1.995864**	1.995763	51,92,134,171,201	32,111,139,184,219	**5.8242e−008**	4.55e−005
Butterfly	2	**0.888889**	0.888888	78,150	45,178	**0.0000**	0.0000
	3	**1.296290**	1.296286	84,113,167	24,145,222	**4.4408e−016**	1.68e−006
	4	**1.654320**	1.654319	63,92,150,210	16,78,222,255	**4.4408e−015**	1.85e−005
	5	**1.995884**	1.995883	50,90,132,165,198	39,48,93,131,150	**4.6193e−008**	6.74e−005
Airplane	2	**0.888982**	0.888920	78,171	62,153	**0.0000**	0.0000
	3	**1.296293**	1.296253	71,125,182	59,143,193	**2.4517e−009**	8.61e−007
	4	**1.654318**	1.654275	68,107,135,190	38,103,139,182	**7.9278e−008**	9.21e−007
	5	**1.995878**	1.995790	50,89,125,156,182	31,94,121,150,195	**1.9810e−007**	5.81e−006
Baboon	2	**0.888976**	0.888906	62,126	51,147	**0.0000**	0.0000
	3	**1.296296**	1.296286	60,131,147	35,148,188	**1.1102e−015**	2.80e−006
	4	**1.654320**	1.654260	59,99,126,186	45,114,146,175	**2.0004e−010**	3.69e−006
	5	**1.995884**	1.995784	57,69,123,156,198	38,96,136,167,215	**4.6354e−008**	9.63e−006

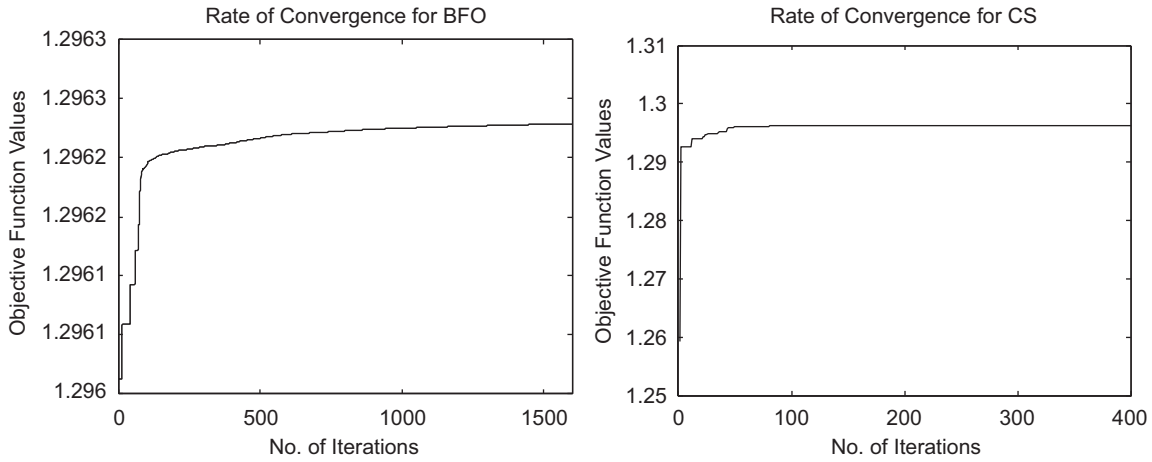

Fig. 13. Rate of convergence of objective function values with BFO and CS.

σ is defined as:

$$\sigma = \sqrt{\frac{1}{k}\sum_{i=1}^{k}(s_i-\mu)^2}. \tag{15}$$

where k is the number of runs for each algorithm. Here we use a value of $k=50$. Note that s_i is the best objective value obtained by the i-th run of the algorithm. Both PSNR and standard deviation σ are displayed in Table 6. A lower value of sigma 'σ' indicates a better quality of thresholding. From sigma values listed in Table 6, it is clear that our method outperform other methods. Note that the standard deviation is smaller in our case, which shows a better stability.

Further, Table 7 displays CPU time (seconds), SSIM and FSIM index values for both CS and BFO for a comparison. CS and ABC algorithms are compared in terms of CPU time (seconds), PSNR, SSIM, FSIM and results are displayed in Table 8. CS and ABC algorithms are also compared in terms of best objective function values, optimal threshold values, standard deviation and results are shown in Table 9. We choose two more measures SSIM and FSIM index to support our claim. The CPU time is less in our proposed method as compared to BFO and ABC. The SSIM is used to compare the structures of the original and thresholded image [26]. The SSIM index is calculated as:

$$SSIM(I,\tilde{I}) = \frac{(2\mu_I\mu_{\tilde{I}}+C1)(2\sigma_{I\tilde{I}}+C2)}{(\mu^2{}_I+\mu^2\tilde{I}+C1)(\sigma^2{}_I+\sigma^2\tilde{I}+C2)} \tag{16}$$

where $\mu_I = 1/N\sum_{i=1}^{N} I_i$ is the mean intensity of image I and $\sigma_{I\ \tilde{I}} = 1/(N-1)\sum_{i=1}^{N}(I_i-\mu_I)\ (\tilde{I}_i-\mu_{\tilde{I}})$ is the standard deviation. $C1$ and $C2$ are constants and are included to avoid instability when $\mu^2{}_I+\mu^2\tilde{I}$ are very close to zero. Here $C1=C2=0.065$. A higher value of SSIM shows better performance and our method gives higher values as compared to BFO and ABC.

The FSIM is used to calculate the similarity between two images [27]. It is calculated between two images $I,\tilde{I}$ as

$$FSIM = \frac{\sum_{X\in\Omega} S_L(X)PC_m(X)}{\sum_{X\in\Omega} PC_m(X)} \tag{17}$$

where

$$S_L(X) = S_{PC}(X)S_G(X);$$

$$S_{PC}(X) = \frac{2PC_1(X)PC_2(X)+T_1}{PC_1^2(X)+PC_2^2(X)+T_1};$$

$$S_G(X) = \frac{2G_1(X)G_2(X)+T_2}{G_1^2(X)+G_2^2(X)+T_2}$$

and T_1 and T_2 are constants. Here we choose $T_1=0.85$ and $T_2=160$.

G represents the gradient magnitude (GM) of an image and is defined as:

$$G = \sqrt{G_x^2+G_y^2}$$

PC is the phase congruence and is defined as:

$$PC(X) = \frac{E(X)}{(\varepsilon+\sum_n A_n(X))}$$

$A_n(X)$ is the local amplitude on scale n and $E(X)$ is the magnitude of the response vector at position X on scale n. Note that ε is a small positive constant. A higher value of FSIM implies better performance. Interestingly, in our case, we get higher values of FSIM as compared to BFO and ABC.

Rate of convergence of objective function values with BFO and CS are shown in Fig. 13 for a comparison. These convergence graphs are obtained for the Lena image with $m=3$. Recently, we find many applications of BFO in the literature [28–32]. But the convergence rate still seems to be slow. It may be reiterated the fact that number of bacteria chosen here is equal to 20, number of chemotaxis steps is equal to 10, number of reproduction steps is 4 and number of elimination and dispersal steps is 2 while implementing BFO. Therefore total number of iterations is equal to $20\times10\times4\times2=1600$. In case of BFO, the maximum objective function value is obtained after about 1500 iterations. Whereas, the maximum objective function value is obtained after about 100 iterations only in the case of CS algorithm. It is observed from Fig. 13 that the rate of convergence is much faster in our case. The parameters for ABC algorithm are as follows: Number of colony size is 40, Number of food sources is 20, and Number of iterations is 40. This gives us 1600 evaluations of the objective function.

6. Conclusions

An extensive study on the application of Cuckoo search algorithm for multilevel thresholding for image segmentation is made. As seen from the experimental results, non-extensive entropy based image thresholding using Cuckoo search algorithm is useful for image segmentation. An interesting feature of the proposed method is that Tsallis entropy uses global and objective property of the image histogram and is easily implemented. The Tsallis parameter 'q' can be used as a tuning parameter for improvising image thresholding results. It is observed that the results obtained are superior to that of BFO, ABC, PSO and GA. The proposed method is faster (CPU time is less) than other techniques. The numerical illustrations demonstrate that the proposed algorithm

outperforms other methods. The validity and stability of the method is justified both qualitatively and quantitatively. The future work will include introduction of multi-objective criteria for multilevel thresholding, optimization of Tsallis parameter 'q', use of penalty function in the optimization problem and using other methods of measuring non-extensive entropy. Further, the idea can be easily extended to other applications of Image Processing like image enhancement and image fusion.

References

[1] M. Portes de Albuquerque, I.A. Esquef, A.R. Gesualdi Mello, Image thresholding using Tsallis entropy, Pattern Recognition Letters 25 (9) (2004) 1059–1065.
[2] P.L. Rosin, Unimodal thresholding, Pattern Recognition Letters 34 (2001) 2083–2096.
[3] C.H. Li, C.K. Lee, Minimum cross entropy thresholding, Pattern Recognition Letters 26 (1993) 617–625.
[4] P.K. Sahoo, S. Soltani, A.K.C. Wong, A survey of thresholding techniques, Computer Vision Graphics Image Processing 41 (1988) 233–260.
[5] N.R. Pal, On minimum cross entropy thresholding, Pattern Recognition Letters 29 (1996) 575–580.
[6] E. Zahara, S.K.S. Fan, M.D. Tsai, Optimal multi-thresholding using a hybrid optimization approach, Pattern Recognition Letters 26 (8) (2005) 1082–1095.
[7] N. Otsu, A threshold selection method from gray-level histograms, IEEE Transactions on Systems Man and Cybernetics SMC-9 (1) (1979) 62–66.
[8] J. Kittler, J. Illingworth, Minimum error thresholding, Pattern Recognition Letters 19 (1) (1986) 41–47.
[9] J.N. Kapur, P.K. Sahoo, A.K.C. Wong, A new method for gray-level picture thresholding using the entropy of the histogram, Computer Vision Graphics Image Processing 29 (1985) 273–285.
[10] Peng-Yeng Yin, A fast scheme for multilevel thresholding using genetic algorithms, Signal Processing 72 (1999) 85–95.
[11] P.Y. Yin, Multilevel minimum cross entropy threshold selection based on particle swarm optimization algorithm, Applied Mathematics and Computation 184 (2) (2007) 503–513.
[12] Yudong Zhang, Lenan Wu, Optimal multi-level thresholding based on maximum Tsallis entropy via an artificial bee colony approach, Entropy 13 (4) (2011) 841–859.
[13] P.D. Sathya, R. Kayalvizhi, Optimum multilevel image thresholding based on Tsallis entropy method with bacterial foraging algorithm, International Journal of Computer Science 7 (5) (2010) 336–343.
[14] Kamal Hammouche, Moussa Diaf, Patrick Siarry, A comparative study of various meta-heuristic techniques applied to the multilevel thresholding problem, Engineering Applications of Artificial Intelligence 23 (5) (2010) 676–688.
[15] S. Sarkar, G.R. Patra, S Das, A differential evolution based approach for multilevel image segmentation using minimum cross entropy thresholding, SEMCCO LNCS 7076 (2011) 51–58.
[16] C. Tsallis, In Abe, Y.S. Okamoto, Non-extensive statistical mechanics and its applications, Series Lecture Notes in Physics, Springer, Berlin, 2001.
[17] C. Tsallis, Entropic nonextensivity: a possible measure of complexity, Chaos, Solitons, & Fractals 13 (2002) 371–391.
[18] A.R ´enyi, On Measures of Entropy and Information, University California Press, Berkeley, California, 1988 547–561.
[19] Luis Rueda, A polynomial-time algorithm for optimal multilevel thresholding, Structural, syntactic, and statistical pattern recognition, Joint IAPR International Workshop, SSPR & SPR 2008, Orlando, USA, LNCS 5342, Springer, 2008 pp. 1–28.
[20] X. Yang, S. Deb, Cuckoo search via levey flights, in: Proceedings of the World congress on nature and biologically inspired computing, NABIC, Coimbatore, vol. 4, 2009 pp. 210–214.
[21] X. Yang, S. Deb, Engineering optimization by cuckoo search, International Journal of Mathmatical Modelling & Numerical Optimization 1 (4) (2010) 330–343.
[22] Ramin Rajabioun, Cuckoo Optimization Algorithm, Applied Soft Computing, vol. 11, 2011, pp. 5508–5518.
[23] Pinar Civicioglu, Erkan Besdok, A conceptual comparison of the cuckoo-search, particle swarm optimization, differential evolution and artificial bee colony algorithms (2011) Artificial Intelligence Review, 39 (4) (2013) 315–346, 10.1007/s10462-011-9276-0.
[24] Berkeley University dataset for image segmentation, ⟨http://www.eecs.berkeley.edu/Research/Projects/CS/vision/bsds⟩.
[25] S. Sarkar, N. Sen, A. Kundu, S. Das, S.S. Chaudhuri, A differential evolutionary multilevel segmentation of near infra-red images using Renyis entropy, in: Proceedings of the International Conference on Frontiers of Intelligent Computing: Theory and Applications, Advances in Intelligent Systems and Computing, vol. 199, 2013 pp. 699–706.
[26] Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, Eero P. Simoncelli, Image quality assessment: from error visibility to structural similarity, IEEE Transactions on Image Processing 13 (4) (2004) 600–612.
[27] Lin Zhang, Lei Zhang, Xuanqin Mou, David Zhang, FSIM: A feature similarity index for image quality assessment, IEEE Transactions on Image Processing 20 (8) (2011) 2378–2386.
[28] S. Dasgupta, S. Das, A. Abraham, A. Biswas, Adaptive computational chemotaxis in bacterial foraging optimization: an analysis, IEEE Transaction on Evolutionary Computation 13 (4) (2009) 919–941.
[29] S. Dasgupta, S. Das, A. Abraham, A. Biswas, Automatic circle detection on digital images using an adaptive bacterial foraging algorithm, Soft Computing: A Fusion of Foundations Methodologies and Applications, 14, Springer-Verlag, Germany. 1151–1164.
[30] S. Das, A. Biswas, S. Dasgupta, A. Abraham, The bacterial foraging optimization-algorithm, analysis and applications, in: Aboul-Ella Hassanien, Ajith Abraham (Eds.), Foundations on Computational Intelligence, Studies in Computational Intelligence, Springer-Verlag, Germany, 2008.
[31] R. Panda, S. Agrawal, EBFS-ICA—An efficient algorithm for CT-MRI image fusion, SEMCCO 2010, LNCS 6466, Springer Verlag, Berlin, Heidelberg356–361.
[32] R. Panda, M.K. Naik, B.K. Panigrahi, Face recognition using bacterial foraging strategy, Swarm and Evolutionary Computation 1 (2011) 138–146.