Compact Bat Algorithm

Thi-Kien Dao[1], Jeng-Shyang Pan[1], Trong-The Nguyen[1],
Shu-Chuan Chu[2], and Chin-Shiuh Shieh[1]

[1] Department of Electronics Engineering,
National Kaohsiung University of Applied Sciences, Taiwan
jvnkien@gmail.com
[2] School of Computer Science, Engineering and Mathematics,
Flinders University, Australia

Abstract. Addressing to the computational requirements of the hardware devices with limited resources such as memory size or low price is critical issues. This paper, a novel algorithm, namely compact Bat Algorithm (cBA), for solving the numerical optimization problems is proposed based on the framework of the original Bat algorithm (oBA). A probabilistic representation random of the Bat's behavior is inspired to employ for this proposed algorithm, in which the replaced population with the probability vector updated based on single competition. These lead to the entire algorithm functioning applying a modest memory usage. The simulations compare both algorithms in terms of solution quality, speed and saving memory. The results show that cBA can solve the optimization despite a modest memory usage as good performance as oBA displays with its complex population-based algorithm. It is used the same as what is needed for storing space with six solutions.

Keywords: Bat algorithm, compact Bat algorithm, Optimizations, Swarm intelligence.

1 Introduction

Computational intelligence algorithms have also been successfully used to solve optimization problems in the engineering, the financial, and the management fields for recently years. For example, genetic algorithms (GA) have been successfully various applications including engineering, the financial, the security [1-3], particle swarm optimization (PSO) techniques have successfully been used to construct the portfolios of stock, human perception [3-5], ant colony optimization (ACO) techniques have successfully been used to solve the routing problem of networks, the secure watermarking [6, 7], artificial bee colony (ABC) techniques have successfully been used to solve the lot-streaming flow shop scheduling problem [8], cat swarm optimization (CSO) [9] techniques have successfully been used to discover proper positions for information hiding [10]. Some applications require the solution of a complex optimization problem event though in limited hardware conditions. These conditions are to use a computational device due to cost and space limitations. For example, wireless sensor networks (WSN) are networks of small, battery-powered, memory-constraint

J.-S. Pan et al. (eds.), *Intelligent Data Analysis and Its Applications*, *Volume 2*,
Advances in Intelligent Systems and Computing 298,
DOI: 10.1007/978-3-319-07773-4_6,

devices named sensor nodes, which have capability of wireless communication over a restricted area [11]. Due to memory and power constraints, they need to be well arranged to build a fully functional network. The other applications require a very fast solution of the optimization problem due to the communication time between a control/actuator devices and an external computer, real-time necessities within the control/actuator devices. For instance, in telecommunications[12] or in industrial plants for energy production[13]. Special applications require fault-tolerance in a high priority and/or to avoid rebooting of the device. For example, in the space shuttle control [14], or in communication underwater [15]. The mentioned problem is not enough memory of computational devices to store a population composed of numerous candidate solutions of those computational intelligence algorithms.

Compact algorithms are a promise answer for this problem. An efficient compromise is used in compact algorithms to present some advantages of population-based algorithms but the memory is not required for storing an actual population of solutions. Compact algorithms simulate the behavior of population-based algorithms by employing, instead of a population of solutions, its probabilistic representation. In this way, a much smaller number of parameters must be stored in the memory. Thus, a run of these algorithms requires much less capacious memory devices compared to their correspondent population-based structures.

The very first implementation of compact algorithms has been the compact Genetic Algorithm (cGA) [16]. The cGA simulates the behavior of a standard binary encoded Genetic Algorithm (GA). It can be seen that cGA has a performance almost as good as that of GA and that cGA requires a much less capacious memory. The compact Differential Evolution (cDE) algorithm has been introduced in [17]. The success of cDE implementation is the combination of two factors. The first is that a DE scheme seems to benefit from the introduction of a certain degree of randomization due to the probabilistic model. The second is that the one-to-one spawning survivor selection typical of DE (the offspring replaces the parent) can be naturally encoded into a compact logic. The compact Particle Swarm Optimization (cPSO) has been defined in [18]. The implementation of cPSO algorithm benefits from the same natural encoding of the selection scheme employed by DE and another ''ingredient'' of compact optimization, i.e. a special treatment for the best solution ever detected and reinterpreted as an evolutionary algorithms in order to propose a compact encoding of PSO.

In this paper, the behavior and the characteristic of the Bat are reviewed to improve the Bat algorithms [19, 20] and to present the compact Bat Algorithm (cBA) based on the framework of the original BA (oBA). According to the experimental results, our proposed cBA presents same result in finding original Bat algorithm.

The rest of this paper is organized as follows: a briefly review of BA is given in session 2; our analysis and designs for the cBA is presented in session 3; a series of experimental results and the compare between oBA and cBA are discussed in session 4; finally, the conclusion is summarized in session 5.

2 Related Works

A random walk is a mathematical formalization of a path that consists of a succession of random steps. This work is primarily inspired by the random walk model in [21,

22]. This model focused on building block for representing individual in warms. Compact algorithms is represented the population as probability distribution based on random steps over the set of solutions. By discretizing its probability representation, the proposed algorithm reduces the oBA's memory requirements. A set of frequencies of Bats is a building block as a whole given high contribution to the fitness of an individual. There are no interactions among building blocks, so they could be solved independently. The behavior of building blocks for solving to optimality could be simulated by the dynamics of the random walk model[22].

In 2010, Xin-SheYang proposed a new optimization algorithm, namely, Bat Algorithm or original Bat Algorithm (oBA), based on swarm intelligence and the inspiration form observing the bats [19] . oBA simulates parts of the echolocation characteristics of the micro-bat in the simplicity way. Three major characteristics of the micro-bat are employed to construct the basic structure of BA. All bats utilize the echolocation to detect their prey, but not all species of the bat do the same thing. However, the micro-bat, one of species of the bat is a famous example of extensively using the echolocation. Hence, the first characteristic is the echolocation behavior. The second characteristic is the frequency that the micro-bat sends a fixed frequency f_{min} with a variable wavelength λ and the loudness A_0 to search for prey.

1. Bats fly randomly with velocity v_i at position x_i. They can adjust the wavelength (or frequency) of their emitted pulses and adjust the rate of pulse emission $r \in [0, 1]$, depending on the proximity of their target;
2. There are many ways to adjust the loudness. For simplicity, the loudness is assumed to be varied from a positive large A_0 to a minimum constant value, which is denoted by Amin.

In Yang's method, the movement of the virtual bat is simulated by equation (1) – equation (3):

$$f_i = f_{min} + (f_{max} - f_{min}) * \beta \tag{1}$$

$$v_i^t = v_i^{t-1} + (x_i^{t-1} - x_{best}) * f_i \tag{2}$$

$$x_i^t = x_i^{t-1} + \mathrm{v_i^t} \tag{3}$$

where f is the frequency used by the bat seeking for its prey, f_{min} and f_{max}, represent the minimum and maximum value, respectively. x_i denotes the location of the i^{th} bat in the solution space, v_i represents the velocity of the bat, t indicates the current iteration, β is a random vector, which is drawn from a uniform distribution, and $\beta \in [0, 1]$, and x_{best} indicates the global near best solution found so far over the whole population. In addition, the rate of the pulse emission from the bat is also taken to be one of the roles in the process. The micro-bat emits the echo and adjusts the wavelength depending on the proximity of their target. The pulse emission rate is denoted by the symbol r_i, and $r_i \in [0, 1]$, where the suffix i indicates the i^{th} bat. In every iteration, a random number is generated and is compared with r_i. If the random number is

greater than r_i, a local search strategy, namely, random walk, is detonated. A new solution for the bat is generated by equation (4):

$$x_{new} = x_{old} + \varepsilon A^t \tag{4}$$

where ε is a random number and $\varepsilon \in [-1, 1]$, and at represents the average loudness of all bats at the current time step. After updating the positions of the bats, the loudness A_i and the pulse emission rate r_i are also updated only when the global near best solution is updated and the random generated number is smaller than A_i. The update of A_i and r_i are operated by equation (5) and equation (6):

$$A_i^{t+1} = \alpha A_i^t \tag{5}$$

$$r_i^{t+1} = r_i^0[1 - e^{-\gamma t}] \tag{6}$$

where α and γ are constants. In Yang's experiments, $\alpha = \gamma = 0.9$ is used for the simplicity. The process of oBA is depicted as follows:

Step 1. Initialize the bat population, the pulse rates, the loudness, and define the pulse frequency
Step 2. Update the velocities to update the location of the bats, and decide whether detonate the random walk process.
Step 3. Rank the bats according to their fitness value, find the current near best solution found so far, and then update the loudness and the emission rate.
Step 4. Check the termination condition to decide whether go back to step 2 or end the process and output the result.

3 Compact Bat Algorithm

As mentioned above that compact algorithms process an actual population of solution as a virtual population. This virtual population is encoded within a data structure, namely Perturbation Vector *(PV)* as probabilistic model of a population of solutions. The distribution of the individual in the hypothetical swarms must be described by a probability density function (PDF) [23] defined on the normalized interval is from -1 to +1. The distribution of the each Bat of swarms could be assumed as Gaussian PDF with mean μ and standard deviation σ [16]. A minimization problem is considered in an *m*-dimensional hyper-rectangle in Normalization of two truncated Gaussian curves (*m* is the number of parameters). Without loss of generality, the parameters assume to be normalized so that each search interval is [-1,+1] . Therefore PV is a vector of *m*×2 matrix specifying the two parameters of the PDF of each design variable being defined as:

$$PV^t = [\mu^t, \sigma^t] \tag{7}$$

where μ and σ are mean and standard deviation values a Gaussian (PDF) truncated within the interval [-1, +1], respectively. The amplitude of the PDF is normalized in order to keep its area equal to 1. The apex *t* is time steps. The initialization of the virtual population is generated for each design variable *i*, $\mu_i^1 = 0$ and $\delta_i^1 = k$ where *k* is set as a large positive constant (e.g. $k = 10$). The PDF height normalization is obtained approximately sufficient in well the uniform distribution with a wide shape. The generating for a candidate solution x_i is produced from $PV(\mu_i, \delta_i)$. The value of

mean μ and standard deviation δ in *PV* are associated equation of a truncated Gaussian PDF is described as following:

$$PDF\big(trucNormal(x)\big) = \frac{e^{\frac{(x-\mu_i)^2}{2\delta_i^2}}}{\delta_i(\mathrm{erf}\left(\frac{\mu_i+1}{\sqrt{2}\delta_i}\right)-\mathrm{erf}\left(\frac{\mu_i+1}{\sqrt{2}\delta_i}\right))} \tag{8}$$

The PDF in formula (8) is then used to compute the corresponding Cumulative Distribution Function (CDF). The CDF is constructed by means of Chebyshev polynomials by following the procedure described in [24], the codomain of CDF is arrange from 0 to 1. The distribution function or cumulative distribution function (*CDF*) describes the probability that a real-valued random variable X with a given probability distribution will be found at a value less than or equal to x_i CDFs are also used to specify the distribution of multivariate random variables.

$$CDF = \int_0^1 PDF * dx \tag{9}$$

The sampling of the design variable x_i from *PV* is performed by generating a random number rand (0, 1) from a uniform distribution and then computing the inverse function of CDF in rand (0, 1). The newly calculated value is x_i.

$$\mathrm{x_i}=\mathrm{inverse}(CDF) \tag{10}$$

When the comparison between two design variables for individuals of the swarm (or better two individuals sampled from PV) is performed the winner solution biases the PV. Let us indicate with winner the vector that scores a better fitness value and with loser the individual losing the (fitness based) comparison. Regarding the mean values l, the update rule for each of its elements is $\mu_i^t, \delta_i^t => \mu_i^{t+1}, \delta_i^{t+1}$.

$$\mu_i^{t+1} = \mu_i^t + \frac{1}{N_p}(winner_i - loser_i) \tag{11}$$

where Np is virtual population size. Regarding δ values, the update rule of each element is given by:

$$\delta_i^{t+1} = \sqrt{(\delta_i^t)^2 + (\mu_i^t)^2 - (\mu_i^{t+1})^2 + \frac{1}{N_p}(winner_i^2 - loser_i^2)} \tag{12}$$

$$[winner, loser] = complete(x_{best}, x^{t+1}) \tag{13}$$

The construction of formulas (11) and (12) are persistent and non-persistent structures with tested results given in [25]. Similar to the binary cGA case, it was impossible to assess whether one or another elitist strategy was preferable. As reported in [17], it is fundamental to remember that the virtual population size N_p is parameter typical of compact algorithms and does not strictly correspond to the population size in a population-based algorithm. The virtual population size, in real-valued compact optimization, is a parameter which biases the convergence speed of

the algorithm. In [25] has been mentioned that, for a given problem (with its dimensionality), this parameter should be set several times bigger than the population size of a corresponding population-based algorithm. In elitist compact schemes, at each moment of the optimization process, the solution displaying the best performance is retained in a separate memory slot. If a new candidate solution is computed, the fitness based comparison between it and the elite is carried out. If elite is a winner solution, it biases the *PV* as shown in formulas (11) and (12).

1) *Initialization probability vector* (*PV*(μ, δ))
   for *i=1:n* **do** μ_i^t *=0;* δ_i^t*=k= 10;*
2) *Initialization parameters: pulse rate* r_i , the loudness A_i , β *= random, and* v_i^1 *= 0*;
   Generate global best solution x_{best} from *PV; Define pulse frequency* f_{min}, f_{max} *as search range;*
3) Evaluate new solutions
   while *termination is not satisfied* **do**
   x^t = generateFrom(*PV*)
   Update velocities and locations
   Use equations (1),(2), and (3)
   if (β > ri)
   Select a solution among the best solution,
   Generate a local solution around selected best solution
   endif
   [*winner, loser*]=**compete**(x^{t+1}, x_{best})
4) *Update PV*
   Use equations (11), and (12)
5) *Global best update*
   x_{best} =*winner*; $\mu_i^{t+1} = \mu_i^t$
6) *Accept new solutions*,
   Fnew =fitness(x);
   Rank the bats and find the current best
   Update if the solution improves, or not too loud
   if (Fnew<=fmin) & (β <A)
   best=xt; fmin=Fnew;
   endif
   endwhile

Fig. 1. The pseudo code of compact Bat algorithm

The fitness value of the position x^{t+1} is calculated and compared with x_{best} to determine a winner and a loser. Equation (11) and equation (12) are then applied to update the probability vector *PV*. If f(x^{t+1}) <= *f*(*min*) and β <*A*, the value of the global best is then updated: $f_{min}=F_{new}$ and the process is repeated over for the subsequent steps. Current values for x^{t+1} and v^{t+1} are retained for subsequent algorithm steps. Figure 1 shows the pseudo code of algorithm working principles of cBA.

4 Experimental Results

This section presents simulation results and compares the cBA with the oBA, both in terms of solution quality and in the number of function evaluations taken. To evaluate the accuracy and the computational speed of the proposed cBA, four benchmark functions are chosen to use in the experiments. All experiments are averaged over different random seeds with 25 runs. All benchmark functions are listed in equation (14) - equation (19).

$$f_1(x) = \sum_{i=1}^{n-1}(100(x_{i-1} - x_i^2)^2 + (1 - x_i)^2 \quad (14)$$

$$f_2(x) = \sum_{i=1}^{n}(\sum_{k=1}^{i} x_i) \quad (15)$$

$$f_3(x) = 1 + \sum_{i=1}^{N} \frac{x_i^2}{4000} + \prod_{i=1}^{N} cos\frac{x_i}{\sqrt{i}} \quad (16)$$

$$f_4(x) = \sum_{i=1}^{N}[10 + x_i^2 - 10cos2\pi x_i \ ] \quad (17)$$

The initial range and the total iteration number for all test functions are listed in Table 1.

Table 1. The initial range and the total iteration of test standard functions

Function	Initial range $[x_{min}, x_{max}]$	Total iteration
$f_1(x)$	[-100,100]	5000
$f_2(x)$	[-100,100]	5000
$f_3(x)$	[-100,100]	5000
$f_4(x)$	[-5.12,5.12]	5000

The optimization goal for all of these test functions is to minimize the outcome. The parameters setting for both cBA and oBA: are the initial loudness $A_i^0 = 0.25$ (range 0.1 to 0.9), pulse rate $r_i^0 = 0.5$ (range 0.5 to 0.9) the total population size n = 20 and the dimension of the solution space M = 30, frequency minimum f_{min} = the lowest of initial range function and frequency minimum f_{max} = the highest of initial range function. Each function contains the full iterations of 5000 is repeated by different random seeds with 25 runs. The final results are obtained by taking the average of the outcomes from all runs. The results are compared with the original BA (oBC).

4.1 Comparison Optimizing Performance Algorithms

Table 2 compares the quality of performance and time running for numerical problem optimization between cBA and oBA. It is clearly seen that, almost cases of benchmark functions for optimizing in compact Bat algorithm are faster convergence. It is special case with test function $f_2(x)$ quadric has the mean of value function

minimum of total 25 runs is 44865 with average time running equal 3.1764 seconds for oBA evaluation. However, for cBA this value of function minimum of total 25 runs is 36321 with time running equal 0.9936 seconds in same executing computer. The mean of four test functions evaluation of minimum function 25 runs is 9.40E+08 with average time consuming 7.0093 for oBA but, 8.87E+08 with average time consuming 3.5843 for cBA respectively. It improved the accuracy and the convergence is up to 4%.

Table 2. The comparison between oBA and cBA in terms of quality performance evaluation and speed

Function	Performance evaluation		Time running evaluation	
	oBA	***cBA***	***oBA***	***cBA***
$f_1(x)$	9.4E+08	8.87E+08	1.2659	1.0141
$f_2(x)$	44865	36321	3.1764	0.9936
$f_3(x)$	6.5331	6.1153	1.3823	0.8845
$f_4(x)$	209.356	256.2685	1.1847	0.6921
Average value	**9.40E+08**	**8.87E+08**	**7.0093**	**3.5843**

Figure 2 shows the average of function minimum of four test functions in 25 runs output in the same iteration of 5000. It can be clearly seen that the curves of compact BA (red and star lines) are faster in convergence.

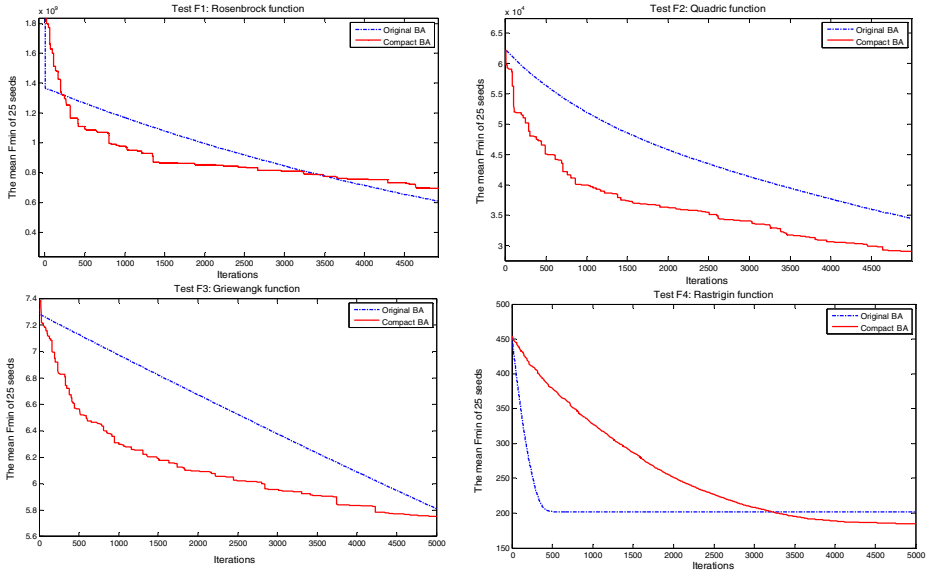

Fig. 2. The mean of function minimum curves in comparing cBA and oBA algorithms for four benchmark functions

4.2 Comparison Saving-Memory and Time-Complexity Algorithms

Table 3 compares the saving-memory computations of two algorithms cBA and oBA. It can be clearly seen that the number memory variables of cBA is smaller than that of oBA in the same condition of computation such as iterations. The real number of population or population size of oBA is *N*, but that size for cBA is only one. Even though, the number equations used for optimizing computation in cBA is six such as equations (1), (2), (3), (12), (13) and (14), and the number equations used for optimizing computation in oBA is only three of them such as equations (1), (2), and (3), the computing complexity of cBA is $6\times T\times iteration$ and it for oBA *is* $3\times T\times N\times iteration$. Thus, the rate of saving-memory equals the computing complexity of cBA per the computing complexity of oBA as given: rate = 2/N.

Table 3. The saving-memory comparison between cBA and oBA

Algo-rithms	Population size	#Memory variable	# Equ-ations	Computing complexity
oBA	*N*	$3\times N$	*(1),(2),(3)*	$3\times T\times N \times iteration$
cBA	*1*	*6*	*(1),(2),(3),(12), (13),(14)*	$6\times T \times iteration$

The considered computational times, for both the algorithms cBA and oBA, have been calculated by means of a PC Intel Core 2 Duo 2.4 GHz with 4 GB RAM employing in Windows7-OS, with Matlab (R2011b), version 7.13.0.564 32bits. Figure 4 illustrates the comparison of executing time between cBA and oBA in 25 seed runs with iteration 5000 for four benchmark functions. It is clearly seen that the most cases of test functions time executing in the proposed cBA (red colored bar) are smaller than that executing in oBA (blue colored bar). The bar of test function number 2 is longest distance different oBA and cBA more than double time executed.

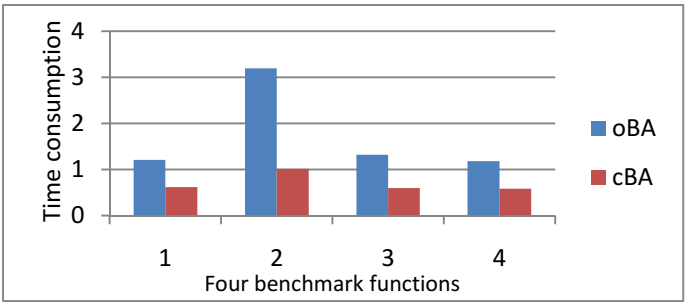

Fig. 3. Comparison two algorithms in term of time running for 6 chosen benchmark functions

For the computational times, several dimensionality levels such as 5, 10, 20, 30, and 40 dimensions should be tested in standard test functions. Figure 4 compares two algorithms in term of different dimensions for test functions. The most of cases test functions for convergence of cBA is smaller than that for oBA.

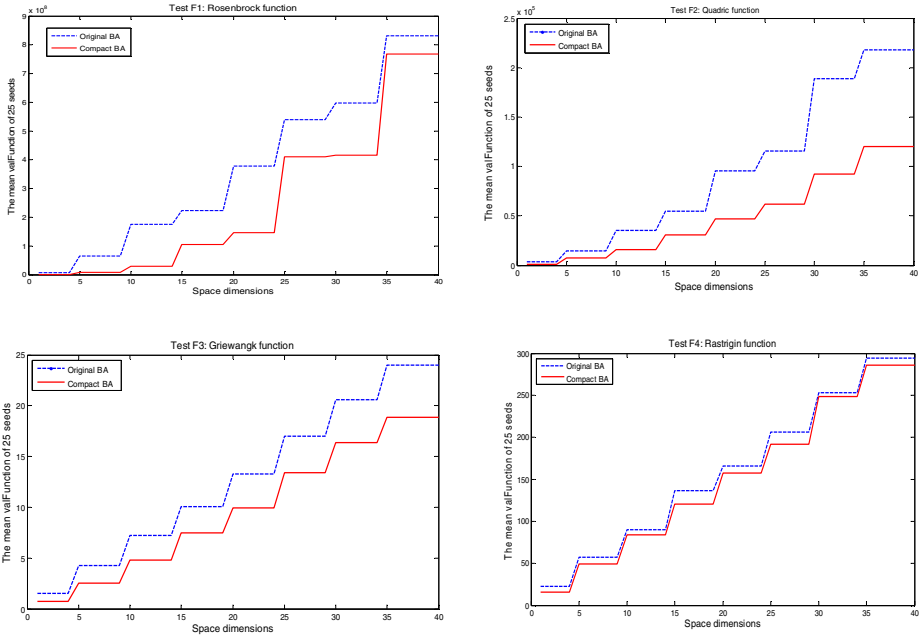

Fig. 4. Comparison two algorithms in term of different dimension for test functions

For variety population size of swarms *N*, the most cases of test functions employing in cBA is not effected much in comparison with oBA because of population size in cBA is virtual population, so the mean of value functions test for cBA are more stable.

5 Conclusion

This paper, a novel proposed optimization algorithm is presented, namely compact Bat algorithm (cBA). The implementation of compact for optimization algorithms could have important significance for the development of embedded devices with small size, low price and being suitable for trend of ubiquitous computing today. In new proposed algorithm, the actual design variable of solutions search space of Bat algorithm is replaced with a probabilistic representation of the population. This feature is important for application problems characterized by a limited memory since it allows the embedded implementation in small and cheap devices. The performance of cBA algorithm is as good as the other previous works in literature with respect to compact algorithms. The results of proposed algorithm on a set of various test

problems show that cBA seems to be a valid alternative for optimization problems plagued by a limited memory. Experimental results on this real-world application also show the applicability of the proposed approach and highlight the good cBA performance within the category of memory-saving algorithms.

Acknowledgement. The authors would like to express their sincere thanks to the National Science Council, Taiwan (ROC), for financial support under the grants NSC 102-2221-E-151-037-.

References

1. Srinivas, M., Patnaik, L.M.: Genetic algorithms: a survey. Computer 27(6), 17–26 (1994)
2. Wang, S., Yang, B., Niu, X.: A Secure Steganography Method based on Genetic Algorithm. Journal of Information Hiding and Multimedia Signal Processing 1(1), 8 (2010)
3. Ruiz-Torrubiano, R., Suarez, A.: Hybrid Approaches and Dimensionality Reduction for Portfolio Selection with Cardinality Constraints. IEEE Computational Intelligence Magazine 5(2), 92–107 (2010)
4. Jui-Fang, C., Shu-Wei, H.: The Construction of Stock's Portfolios by Using Particle Swarm Optimization, pp. 390–390
5. Bajaj, P., Puranik, P., Abraham, A., Palsodkar, P., Deshmukh, A.: Human Perception-based Color Image Segmentation Using Comprehensive Learning Particle Swarm Optimization. Journal of Information Hiding and Multimedia Signal Processing 2(3), 227–235 (2011)
6. Pinto, P.C., Nagele, A., Dejori, M., Runkler, T.A., Sousa, J.M.C.: Using a Local Discovery Ant Algorithm for Bayesian Network Structure Learning. IEEE Transactions on Evolutionary Computation 13(4), 767–779 (2009)
7. Chouinard, J.-Y., Loukhaoukha, K., Taieb, M.H.: Optimal Image Watermarking Algorithm Based on LWT-SVD via Multi-objective Ant Colony Optimization. Journal of Information Hiding and Multimedia Signal Processing 2(4), 303–319 (2011)
8. Pan, Q.-K., Tasgetiren, M.F., Suganthan, P.N., Chua, T.J.: A discrete artificial bee colony algorithm for the lot-streaming flow shop scheduling problem. Inf. Sci. 181(12), 2455–2468 (2011)
9. Chu, S.-C., Tsai, P.W.: Computational Intelligence Based on the Behavior of Cats. International Journal of Innovative Computing, Information and Control 3(1), 8 (2006)
10. Wang, Z.-H., Chang, C.-C., Li, M.-C.: Optimizing least-significant-bit substitution using cat swarm optimization strategy. Inf. Sci. 192, 98–108 (2012)
11. Akyildiz, I.F., Weilian, S., Sankarasubramaniam, Y., Cayirci, E.: A survey on sensor networks. IEEE Communications Magazine 40(8), 102–114 (2002)
12. Gene, C., Wai-tian, T., Yoshimura, T.: Real-time video transport optimization using streaming agent over 3G wireless networks. IEEE Transactions on Multimedia 7(4), 777–785 (2005)
13. Pourmousavi, S.A., Nehrir, M.H., Colson, C.M., Caisheng, W.: Real-Time Energy Management of a Stand-Alone Hybrid Wind-Microturbine Energy System Using Particle Swarm Optimization. IEEE Transactions on Sustainable Energy 1(3), 193–201 (2010)
14. Norman, P.G.: The new AP101S general-purpose computer (GPC) for the space shuttle. Proceedings of the IEEE 75(3), 308–319 (1987)

15. Simpson, J.A., Hughes, B.L., Muth, J.F.: Smart Transmitters and Receivers for Underwater Free-Space Optical Communication. IEEE Journal on Selected Areas in Communications 30(5), 964–974 (2012)
16. Harik, G.R., Lobo, F.G., Goldberg, D.E.: The compact genetic algorithm. IEEE Transactions on Evolutionary Computation 3(4), 287–297 (1999)
17. Mininno, E., Neri, F., Cupertino, F., Naso, D.: Compact Differential Evolution. IEEE Transactions on Evolutionary Computation 15(1), 32–54 (2011)
18. Neri, F., Mininno, E., Iacca, G.: Compact Particle Swarm Optimization. Information Sciences 239, 96–121 (2013)
19. Yang, X.-S.: A New Metaheuristic Bat-Inspired Algorithm. In: González, J.R., Pelta, D.A., Cruz, C., Terrazas, G., Krasnogor, N. (eds.) NICSO 2010. SCI, vol. 284, pp. 65–74. Springer, Heidelberg (2010)
20. Tsai, P.W., Pan, J.S., Liao, B.Y., Tsai, M.J., Istanda, V.: Bat Algorithm Inspired Algorithm for Solving Numerical Optimization Problems. Applied Mechanics and Materials 148-149, 134–137 (2012)
21. Pearson, K.: The Problem of the Random Walk. Nature, 72 (1905)
22. Pemantle, R.: A survey of random processes with reinforcement. Probability Surveys 4(2007), 9 (2007)
23. Billingsley, P.: Probability and Measure. John Wiley and Sons (1979)
24. Cody, W.J.: Rational Chebyshev approximations for the error function. Mathematics of Computation 23(107), 631–637 (1969)
25. Mininno, E., Cupertino, F., Naso, D.: Real-Valued Compact Genetic Algorithms for Embedded Microcontroller Optimization. IEEE Transactions on Evolutionary Computation 12(2), 203–219 (2008)