2014 IEEE Congress on Evolutionary Computation (CEC)
July 6-11, 2014, Beijing, China

Chaos-Driven Discrete Artificial Bee Colony

Magdalena Metlicka and Donald Davendra

***Abstract*— In this paper, a chaos driven Discrete Artificial Bee Algorithm is introduced. The main premise of this work is to ascertain if using chaos maps in lieu of standard pseudorandom number generators can improve the performance of the canonical algorithm. Nine unique chaos maps are embedded in the Discrete Artificial Bee Algorithm alongside the Mersenne twister and evaluated on the lot-streaming flowshop scheduling problem with setup time. Based on the obtained results, a number of chaotic maps significantly improve the performance of the algorithm. Additionally, the new algorithm is favourably compared with the chaos driven Enhanced Differential Evolution algorithm for the same problem.**

I. Introduction

ONE of the core premises of evolutionary algorithms (EA's) is their reliance on *stochasticity*, the ability to generate a random event, which in turn, provides the spark of perturbation towards the desired goal. The task of generating this stochasticity is generally in the realm of *pseudorandom number generators (PRNG)*; a structured sequence of mathematical formulation which tries to yield a generally optimal range of distributed numbers within a specified range.

A wide variety of such pseudorandom number generators exist, however, the most common in usage is the *Mersenne Twister* [1]. A number of its variants have been designed; for a full listing please see [2].

This paper explores a novel approach to generating PRNG, one with a lineage in chaos theory. The term *chaos* describes the complex behaviour of simple dynamical systems. When casually observed, this behaviour may seem erratic and somewhat random, however, these systems are deterministic, whose precise description of future behaviour is well known, given by the trajectory on the map. The proposition is then to utilise the notion of observed erratic (random) and underlying (deterministic) part of the systems.

This aperiodic non-repeating behaviour of chaotic systems is the foundation of this research. The objective is then to analyse different chaotic systems, and embed them in the EA's as Chaotic Pseudorandom Number Generators (CPRNG's). Generally, four branches of chaotic systems exist, which are the dissipative systems, fractals, dissipative and high-dimensional systems and conservative systems. The systems of interest in this line of research are the *discrete* dissipative systems. Related literature is described in section II.

Magdalena Metlicka and Donald Davendra are with the Department of Computer Science, Faculty of Electrical Engineering and Computer Science, VSB-Technical University of Ostrava, Czech Republic (email: {magdalena.metlicka.st, donald.davendra}@vsb.cz).

This work was supported by the SGS project number: SP2014/170

This paper looks to expand upon this class of research and to ascertain if chaos can improve one of the most recent and promising algorithm, the Artificial Bee Algorithm (ABC) [3], in particular its discrete version, the DABC[4]. The Mersenne twister was utilised as the canonical PRNG in ABC and compared with nine different chaotic maps.

The paper is organised as follows: section II gives a brief overview of chaos based literature, section III introduces the Discrete ABC (DABC) of [4] and presents the chaotic DABC. The different chaos maps used in this work and their mathematical descriptions are given in section IV. The lot-streaming flowshop problem with setup time, which is the benchmark problem used for validation, is described in section V. The experimental results are presented in section VI and analysis is done in section VII. Finally, the work is concluded in section VIII.

II. Related Research

Many chaotic maps in the literature possess certainty, ergodicity and the stochastic property. Recently, chaotic sequences have been adopted instead of random sequences with improved results. They have been used to enhance the performance of EA's ([5], [6]). They have also been used together with some heuristic optimisation algorithms ([7], [8]) to express optimisation variables. The choice of chaotic sequences is justified theoretically by their unpredictability, i.e. by their spread-spectrum characteristics, non-periodic, complex temporal behaviour, and ergodic properties [9].

A mathematical description of the connection between chaotic systems and random number generators has been given by [10]. In this paper, a strong linkage has been shown between the Lehmer generator [11] and the simple chaos dynamical system of Bernoulli shift [12]. The hidden periodicity of chaos system and its dependence on numerical system has been shown by [13]. A chaotic piecewise-linear one dimensional (PL1D) map has been utilised as a chaotic random number generator in [14]. The construction of the chaos random number system is based on the exploitation of the double nature of chaos, deterministic in microscopic space and by its defining equations, and random in macroscopic space. This new system is mathematically proven to overcome the major drawbacks of classical random number systems, which are its reliance on the assumed randomness of a physical process, inability to analyse and optimise the random number generator, inability to compute probabilities and entropy of the random number generator, and inconclusiveness of statistical tests.

A family of enhanced CPRNG's has been developed by [15], where the main impetus is the generation of very long series of pseudorandom number generations. This is

978-1-4799-1488-3/14/$31.00 ©2014 IEEE

accomplished through what is called the ultra weak coupling of chaotic systems, such as the Tent Map, which is enhanced in order to conceal the chaotic genuine function [16].

Recently, the very notion of using PRNG's in EA's has been explored by [17]. Additionally, extended case studies on the application of chaos on different EA's to both continuous and combinatorial optimisation problems has been done by [18], [19], [20], [21], [9], [22], [23] and [7] amongst others.

III. Discrete Artificial Bee Algorithm

The discrete variant of the ABC algorithm (DABC) has been developed by [4] to solve the flowshop lot-streaming problem. Subsequently, DABC has been applied to solve the permutative flowshop scheduling problem, with total flowtime minimisation [24], no-idle permutation flowshop scheduling problem with the total tardiness criterion [25] and multi-objective flexible job-shop scheduling problem with maintenance activities [26].

The basic outline of the DABC is now presented.

A. Algorithm Structure

The basic structure of the DABC mimics the canonical ABC. Being based on the foraging nature of honey bee swarm, the algorithm has three phases, each representing one class of bees: *employed bees*, *onlookers* and *scouts*. A bee that is currently exploiting a food source is called an *employed bee*. A bee waiting in the hive for making decision to choose a food source is named as an *onlooker*. A bee carrying out a random search for a new food source is called a *scout*. Each solution to the problem under consideration is called a food source, whereas the fitness of the solution corresponds to the *nectar* amount of the associated food resource.

B. Parameters

The initial parameters are the number of food sources (FS) which is equal to the number of the *employed bees* or *onlooker bees*, the number of trials after which a food source is assumed to be abandoned (*limit*), probability of local search (*PL*), local search iterations limit ($loop_{max}$), and a termination criterion. In the basic DABC algorithm, for every food source, there is only one employed bee.

C. Solution representation

The permutation based representation constitutes an easy procedure to decode a schedule, which has been widely used in literature for a variety of permutation flow shop scheduling problems [27]. This representation is used for DABC, where each permutation $\pi = \{x_1, x_2, .., x_D\}$, where D is the size of the permutation.

D. Employed bee phase

In the DABC, which has a permutation based neighbourhood structure, *insert* and *swap* operators are commonly used to produce neighbouring solutions in the literature [27]. The insert operator of a permutation π is defined by removing a randomly selected job from π from its original position *j* and inserting it into another position *k* such that $(k \in \{j, j-1\})$.

The *swap* operator produces a neighbour of π by interchanging two jobs from π in the different randomly determined positions. To enrich the neighbourhood structure and diversify the population, four neighbouring approaches based on the insert or swap operator are separately utilised to generate neighbouring food sources for the employed bees as follows:

- Performing one insert operation to a sequence π.
- Performing one swap operation to a sequence π.
- Performing two insert operations to a sequence π.
- Performing two swap operations to a sequence π.

The best strategy, which is generally problem dependent, is selected using an adaptive mechanism. In terms of selection, new food source is always accepted if it is better than the current food source.

E. Onlooker bee phase

A tournament selection with the size of two is used to select a new food source. In the tournament selection, an onlooker bee selects a food source x_i in such a way that two food sources are picked up randomly from the population, and compared to each other, then the better one is chosen. Onlooker bees use the same method of producing a new neighbouring solution as employed bees. Naturally, if the new food source improves upon the current food source, it replaces the latter in the population.

F. Self adaptive strategy

Both employed bees and onlookers apply a self-adaptive strategy to find neighbouring food sources. The self-adaptive strategy is presented as follows. At the beginning, an initial neighbour list (NL) with a specified length is generated by filling the list one by one randomly from four neighbouring approaches explained before. Then the DABC algorithm is started. During the evolution process, whenever the neighbouring food source is to be generated, one approach from the NL is taken and applied to the food source. If the new food source successfully replaces the current one, this approach will enter into a winning neighbouring list (WNL). Once the NL is empty, it is refilled as follows: 75% of the NL is refilled from the WNL list, and then the remaining 25% is refilled by a random selection from complete set of four possible approaches. If the WNL is empty, the latest NL is used again. The above process is repeated until a termination criterion is reached. As a result, the proper neighbouring approach can be gradually learned by the algorithm itself to suit the particular problem and the particular phase of search process [4].

G. Local search

DABC contains embedded local search. In employed bee phase, each bee may perform local search with given probability. If a random number in range $[0, 1]$ is lesser than this probability, fixed count of swap or insert operations are applied to a food source generated by a bee. Local search serves to enhance the exploitation ability of DABC algorithm.

H. Scout bee phase

Contrary to ABC, in the DABC scout bee phase, the exhausted food source is replaced by new solution generated from the best solution in the population, upon which at least three insert operations are performed. This way DABC exploits the knowledge of best food source found so far, rather than generating a new random one.

I. Chaos-Driven Discrete Artificial Bee Algorithm

In all the variants of ABC and DABC, very little attention has been paid to the stochasticity of the algorithm. The basic premise is the use of widely available PRNG's. Using the stock DABC algorithm of [4], the most popular Mersenne twister [1] has been included as the default PRNG.

Alternatively, nine unique chaotic systems have been included as CPRNG for the DABC. These new chaos embedded algorithms (hereafter referred to as variants) can be collectively labelled as **CDABC**. The basic premise of this work is to ascertain if any improvement can be achieved in DABC by using chaotic systems in place of PRNG.

The chaotic map can be utilised in two forms, the first is to generate a large chaotic sequence from *inception* of the map and use it iteratively. The second approach is to use a *random* start position of the algorithm for each experiment. The second approach has been utilised in this research, in order to have a unique sample for *each* experiment, and eliminate the need to store large values in memory. This is similar concept to having a *seed* input to a PRNG. The different chaotic systems used as CPRNG's are given in the following section IV.

IV. Chaos Systems

The most interesting chaotic systems, which can be utilised as CPRNG are discrete dissipative chaotic maps. These maps have the general description of being a linear set of equations, easily formulated, with a fine grain over the solution landscape. This last attribute allows the parsing of unique values over a period of the chaotic oscillation. In total, nine unique chaotic systems were considered for this experiment. The following sections describe the different systems. All operating parameters were obtained from [28].

A. Arnold's Cat Map

The Arnold's cat map is a two dimensional discrete chaotic map, which is a torus into itself. The equations are given in (1). The parameter of k = 2.0.

$$\begin{aligned} X_{n+1} &= X_n + Y_n \cdot (\mathrm{mod} 1) \\ Y_{n+1} &= X_n + k \cdot Y_n \cdot (\mathrm{mod} 1) \end{aligned} \tag{1}$$

B. Burgers Map

The Burgers map arose from the study of hydrodynamics, where the discretization of coupled differential equations led to a bifurcation effect of the system. The equation is given in (2) and the control parameters are α = 0.75 and β = 1.75.

$$\begin{aligned} X_{n+1} &= (\alpha \cdot X_n) - Y_n^2 \\ Y_{n+1} &= (\beta \cdot Y_n) + (X_n \cdot Y_n) \end{aligned} \tag{2}$$

C. Delayed Logistic

The Delayed Logistic is a two-dimensional map which is a phase shifted one-dimensional logistic equation. The equation is given in (2) and the parameter α = 2.27.

$$\begin{aligned} X_{n+1} &= \alpha \cdot X_n \cdot (1 - Y_n) \\ Y_{n+1} &= X_n \end{aligned} \tag{3}$$

D. Dissipative Standard Map

The Dissipative Standard Map is a two-dimensional chaotic system. The equation is given in (4) and the operating parameters are β = 0.1 and k = 8.8.

$$\begin{aligned} X_{n+1} &= X_n + Y_{n-1} \cdot (\mathrm{mod} 2\pi) \\ Y_{n+1} &= (\beta \cdot Y_n) + (k \cdot \sin X_n \, (\mathrm{mod} 2\pi)) \end{aligned} \tag{4}$$

E. Henon Map

The Henon map is a discrete-time dynamical system, which was introduced as a simplified model of the Poincare map for the Lorenz system. The equation is given in (5) and the control parameters are α = 1.4 and β = 0.3.

$$\begin{aligned} X_{n+1} &= \alpha - X_n^2 + (\beta \cdot Y_n) \\ Y_{n+1} &= X_n \end{aligned} \tag{5}$$

F. Ikeda Map

The Ikeda map is a discrete-time dynamical system derived as a model of light going around across a nonlinear optical resonator. A 2D real example of the Ikeda map is given in equation (6). The operating parameters are α = 0.75, β = 1.75, γ = 1 and μ = 0.9.

$$\begin{aligned} X_{n+1} &= \gamma + \mu \cdot ((X_n \cdot \cos\phi) - (Y_n \cdot \sin\phi)) \\ Y_{n+1} &= \mu \cdot ((X_n \cdot \sin\phi) + (Y_n \cdot \cos\phi)) \\ \phi &= \beta - \tfrac{\alpha}{(1+X_n^2+Y_n^2)} \end{aligned} \tag{6}$$

G. Lozi Map

The Lozi map is a simple discrete two-dimensional chaotic map. The equation is given in (7) and the control parameters are α = 1.7 and β = 0.5.

$$\begin{aligned} X_{n+1} &= 1 - (\alpha \cdot |X_n|) + (\beta \cdot Y_n) \\ Y_{n+1} &= X_n \end{aligned} \tag{7}$$

H. Sinai Map

The Sinai map is a simple two-dimensional discrete system similar to the Arnolds Cat map. The equation is given in (8) and the control parameter is δ = 0.1.

$$\begin{aligned} X_{n+1} &= X_n + Y_n + (\delta \cdot \cos 2\pi \cdot Y_n \cdot (\mathrm{mod} 1)) \\ Y_{n+1} &= X_n + 2 \cdot Y_n \cdot (\mathrm{mod} 1) \end{aligned} \tag{8}$$

I. Tinkerbell Map

The Tinkerbell map is a two-dimensional complex discrete-time dynamical system. The equation is given in (9) and the operating parameters are α = 0.9, β = -0.6, ρ = 2 and υ = 0.5.

$$\begin{aligned} X_{n+1} &= X_n^2 - Y_n^2 + (\alpha \cdot X_n) + (\beta \cdot Y_n) \\ Y_{n+1} &= (2 \cdot X_n \cdot Y_n) + (\rho \cdot X_n) + (\upsilon \cdot Y_n) \end{aligned} \tag{9}$$

V. Lot Streaming Problem

The lot-steaming problem with setup time considered in this paper is a subset of the generic flowshop scheduling problem. Whereas, in the permutative flowshop problem, each job *n* is processed by a single machine *m*, in a lot-streaming variant, each job is divided into smaller tasks called *lots* (*l*) [29]. Once the processing of a sub-lot on its preceding machine is completed, it can be transferred to the downstream machine immediately. However, all $l(j)$ sub-lots of job *j* should be processed continuously as no intermingling or exchanging is allowed. A separable sequence-dependent setup time is necessary for the first sub-lot of each job *j* before it can be processed on any machine *k* [30].

Two different cases of the problem are available; the idling and the non-idling case. The idling case is the simpler variant of the problem, where only the schedule of the lots is taken into consideration. A non-idling case on the other hand is more practical. A non-idle case arises when the machine is not allowed to be idle. This is beneficial, especially in the case when a number of machines are in operation, and resources, such as electricity, are wasted. Another practical example is when expensive machinery is employed. Idling of such expensive equipment is often not desired. In this paper, only the non-idling case is considered.

For a detailed description of the lot-streaming problem please refer to [31].

A. Non-Idling Case

The constraint in this case is that at any given time a machine can process only one sub-lot, and each sub-lot can only be assessed individually. Let the processing time of each sub-lot of job j on machine m be $P(m,j)$, and the setup time of job j on machine m, after having processed job j is $s(m,j,j)$, which can also represent the setup time of job j if it is the first job to be proceeded on the machine. The objective is to find a sequence with the optimal sub-lot starting and completion times to minimise the makespan.

The permissible job permutation can be presented as $\pi = \{\pi_1, \pi_2, \ldots, \pi_n\}$, and the earliest start and completion time as $S(m,j,r)$ and $C(m,j,r)$, where r represents the specific sub-lot on job j being processed on machine m.

For the non-idling case, the earliest start time for the first sub-lot is given in equations (10) and (11), where the start time is the maximum of the setup time of the job in the current machine, the completion time of the first sub-lot on the previous machine, and the difference between the completion time of the whole job on the previous machine and the total processing time of the whole job on the preceding machine except the last sub-lot. This ensures that there is no idling time between two adjacent sub-lots. The last two directives of these equations calculate the completion time for the first job.

The subsequent processing times of the following job sequence are given in equations (12) and (13).

$$\begin{aligned} &S(1,\pi_1,1) = s(1,\pi_i,\pi_i) \\ &C(1,\pi_1,l(\pi_1)) = S(1,\pi_1,1) + l(\pi_1) \times P(1,\pi_1) \end{aligned} \tag{10}$$

$$\begin{aligned} S(w,\pi_1,1) = \max \left\{ \begin{array}{l} s(w,\pi_1,\pi_1), S(w-1,\pi_1,1) + \\ \quad p(w-1,\pi_1), \\ C(k-1,\pi_1,l(\pi_1)) - \\ \quad (l(\pi_1)-1) \times P(1,\pi_1) \end{array} \right\}, \\ C(w,\pi_1,l(\pi_1)) = S(w,\pi_1,1) + l(\pi_1) \times P(w,\pi_i), \\ w = 2,3,\ldots,m \end{aligned} \tag{11}$$

$$\begin{aligned} &S(1,\pi_1,1) = C(1,\pi_{i-1},l(\pi_{i-1})) + s(1,\pi_{i-1},\pi_i), \\ &C(1,\pi_1,l(\pi_1)) = S(1,\pi_1,1) + l(\pi_1) \times P(1,\pi_1), \\ &\quad i = 2,3,\ldots,n \end{aligned} \tag{12}$$

$$\begin{aligned} S(w,\pi_1,1) = \max \left\{ \begin{array}{l} S(w-1,\pi_i,1) + P(w-1,\pi_1), \\ C(w-1,\pi_1,l(\pi_1)) - \\ \quad (l(\pi_1)-1) \times P(1,\pi_1), \\ C(w-1,\pi_{i-1},l(\pi_{i-1})) + \\ \quad s(1,\pi_{i-1},\pi_i) \end{array} \right\}, \\ i = 2,3,\ldots,n, \quad w = 2,3,\ldots,m \\ C(w,\pi_1,l(\pi_1)) = S(w,\pi_1,1) + l(\pi_1) \times P(w,\pi_i), \\ i = 2,3,\ldots,n, \quad w = 2,3,\ldots,m \end{aligned} \tag{13}$$

The makespan for the non-idling case can be then calculated as equation (14).

$$C_{\max}(\pi) = C_T(m,\pi_n,l(\pi_n)) \tag{14}$$

The objective of the lot-streaming flow shop scheduling problem with makespan criterion is to find a permutation π^* in the set of all permutations $\prod$. It can be given as in the equation (15) [30].

$$C_{\max}(\pi^*) \leq C_{\max}(\pi), \forall \pi \in \Pi \tag{15}$$

VI. Experimentation

In keeping with the theme of utilising the chaotic maps in lieu of PRNG, the data sets have been generated using two unique chaotic maps; the Lozi and the Dissipative map. Five unique sizes of data sets have been generated. They are from 10 jobs x 5 machines, 20 jobs x 10 machines, 50 jobs x 25 machines, 75 jobs x 30 machines and 100 jobs x 50 machines. There are 5 instances for each data set size, therefore, in total 25 data set instances for each of the Lozi and Dissipative data sets.

In order to have *unique* data sets, each *instance* was initialised from a unique *start* position of the respective

TABLE I

DABC OPERATING PARAMETERS

Parameter	Value
Food Source (FS)	30
Limit (food source)	50
$Loop_{max}$ (Local Search)	200
Local search probability (P_L)	0.2
Neighbourhood List (NL)	20
Winning Neighbourhood List (WNL)	0.75 x NL
Iterations	100

chaotic system. Additionally, the map was not allowed to be reinitialised. Two different maps were used in order to gain more diversity, due to their different maps in the data sets, and to remove any particular bias when using any one system.

The datasets are available at [32] for download.

The operating parameters of CDABC are given in Table I. All parameters were kept constant for all the experimentation, in order not to introduce a bias. All experiments were conducted on the machine having Intel i7-3610QM CPU processor running at 2.3GHz with 8GB of RAM. All codes were written in the C programming language, compiled with the gcc 4.6.2.

For each instance, fifteen (15) repeated experimentations were conducted in order to obtain statistical variance. Therefore, 375 individual experiments were conducted on the Lozi and Dissipative data sets, a total of 750 experimentations.

The *average* results obtained by the fifteen experiments are given in Table II for the Lozi data sets and Table III for the Dissipative data sets.

From the average results for the Lozi data sets, Tinkerbell has the lowest average values for 18 data instances. It also has the lowest collective average value of 10241.89. The second best performing variant is the Delayed Logistic with 10252.36 for the collective average value. Mersenne twister is the fifth best performing variant with 10359.71.

As in the Lozi case, Tinkerbell and Delayed Logistic are the two best performing variants in the Dissipative data sets. Tinkerbell obtains 12 best results, whereas Delayed Logistic obtains seven. For the collective average results, Tinkerbell has 14058.66 compared to 14067.54 for the Delayed Logistic. Once again Mersenne Twister is the fifth best performing variant with the average of 14184.32.

VII. ANALYSIS

A. T-test analysis

From the experimentations, the top four performing chaotic systems Tinkerbell, Delayed Logistics, Burgers and Lozi together with the Mersenne Twister are compared pairwise for their performance. From the results it is obvious that the significant divergence of the results occurs from the medium to large data sets, therefore a comprehensive *t-test* analysis is conducted from the results of data set of instance 11 to instance 25. As mentioned, 15 experiments have been conducted for each instance by each variant of the algorithm.

TABLE IV

LOZI T-TEST RESULTS

	Tinkerbell		DL		Burgers		Lozi	
	t	*p*	*t*	*p*	*t*	*p*	*t*	*p*
DL	2.15	**0.032**	-	-	-	-	-	-
Burgers	7.21	**0.00**	5.07	**0.00**	-	-	-	-
Lozi	16.85	**0.00**	15.42	**0.00**	12.9	**0.00**	-	-
MT	14.38	**0.00**	13.98	**0.00**	11.87	**0.00**	1.47	0.141

TABLE V

DISSIPATIVE T-TEST RESULTS

	Tinkerbell		DL		Burgers		Lozi	
	t	*p*	*t*	*p*	*t*	*p*	*t*	*p*
DL	2.02	**0.044**	-	-	-	-	-	-
Burgers	21.76	**0.00**	6.64	**0.00**	-	-	-	-
Lozi	8.71	**0.00**	20.67	**0.00**	15.14	**0.00**	-	-
MT	22.77	**0.00**	21.69	**0.00**	17.14	**0.00**	1.41	0.157

The *t-test* experiments takes all the 15 results for each problem instance by the selected variant and conducts a pairwise comparison. The *t* and *p* values for the *paired t-test* are given in Table IV for the Lozi test instances and Table V for the Dissipative test instances.

The *t-tests* were conducted at a 95% confidence level, so all pairwise compared variants, which have a value of *p* of less than 0.05 can be interpreted as being significantly different from each other. From the obtained *t-test* results (Table VI) all the variants are significantly different from each other apart from Mersenne Twister and Lozi Map. Based on these results, it can be inferred that the hierarchy of the five best performing variants based on average performance are Tinkerbell, Delayed Logistic, Burgers, Lozi and Mersenne Twister for the Lozi data sets. For the Dissipative data sets the best five variants are Tinkerbell, Delayed Logistic, Lozi, Burgers and Mersenne Twister.

The basic premise of this research is therefore achieved as it has been shown that a number of different chaotic systems improves DABC, under the same operating parameters.

B. Comparison with Enhanced Differential Evolution

An algorithm comparison is done with the chaos driven Enhanced Differential Evolution (EDE_C) algorithm of [33]. EDE algorithm is an extension of the canonical DE algorithm, with *backward/forward* transformation structure and embedded local search. EDE_C has been shown to significantly improve upon EDE. The comparison between EDE_C and CDABC for the Lozi data sets is given in Table VII. In this case, the Tinkerbell variant of CDABC ($CDABC_T$) is chosen as it is the best performing.

TABLE VI

COMBINED T-TEST RESULTS

	Tinkerbell		DL		Burgers		Lozi	
	D	*L*	*D*	*L*	*D*	*L*	*D*	*L*
DL	$\neq$	$\neq$	-	-	-	-	-	-
Burgers	$\neq$	$\neq$	$\neq$	$\neq$	-	-	-	-
Lozi	$\neq$	$\neq$	$\neq$	$\neq$	$\neq$	$\neq$	-	-
MT	$\neq$	$\neq$	$\neq$	$\neq$	$\neq$	$\neq$	$=$	$=$

D = Dissipative data sets
L = Lozi data sets

TABLE II

LOZI DATA SETS

	MT Δavg	Arnold Cat Δavg	Burgers Δavg	Delayed Logistic Δavg	Dissipative Δavg	Henon Δavg	Ikeda Δavg	Lozi Δavg	Sinai Δavg	Tinkerbell Δavg
1	541	541	541	541	541	541	541	541	541	541
2	430	430	430	430	430	430	430	430	430	430
3	502	502	502	502	502	502	502	502	502	502
4	551	551	551	551	551	551	551	551	551	551
5	531	531	531	531	531	531	531	531	531	531
6	1985.8	1997.067	1983.733	**1983.2**	1994.667	2000.6	1986.933	1986.333	1999.333	1983.333
7	2210	2210	2210	2210	2210	2210	2210	2210	2210	2210
8	2105.2	2110.133	2104.533	2103.6	2108.467	2113.667	2105.667	2105.067	2111.133	**2103.333**
9	2174.267	2184.133	2168.2	2165.867	2179.2	2184.067	2172.067	2168.867	2185	**2164.067**
10	2043.667	2048.4	2042.6	2042.467	2047.8	2053.667	2044.6	2043.8	2049.667	**2042.4**
11	28976.867	29239.801	28701.334	28579.133	29114	29248	28977	28954.934	29273	**28611.467**
12	27658.133	27986.533	27440.467	27411.199	27951.199	28036.801	27639.801	27649.199	28109.867	**27333.334**
13	27985	28217.4	27846.268	27755.801	28144.867	28298.666	27999.801	27967.133	28302.268	**27722.4**
14	28839.801	29036.732	28418.732	28443.801	28997	29241.934	28751.801	28735.867	29135.732	**28380.732**
15	28803.801	29060.801	28564.867	28469.4	29039.467	29175.6	28799.732	28696.066	29198.934	**28453.467**
16	8409.134	8556.4	8372.533	8341.667	8541.333	8560.333	8462.066	8430.4	8567.134	**8329.4**
17	8301.733	8393.934	8226.667	8198.6	8362.267	8435.6	8292.866	8295.6	8416.6	**8175.933**
18	8446.733	8522.533	8391.533	8363.934	8521.134	8551.2	8471.467	8435.667	8568.2	**8378.333**
19	8176.933	8287.866	8118.933	8097.333	8261.533	8291	8212.2	8212.6	8286.467	**8093.733**
20	8045	8102.2	7954.267	7933.533	8085	8138.733	8035.4	8012.4	8116.8	**7938.067**
21	12225.733	12370.066	12115.934	12107.333	12307.4	12391.8	12218.066	12198.667	12408.267	**12092.733**
22	12306.533	12466	12234.934	12194.934	12415	12428.333	12335.134	12300.733	12486.6	**12194.4**
23	12364.4	12493.2	12244	12201.333	12470.733	12552.333	12356.467	12361.134	12520.066	**12192.533**
24	12319.6	12500.8	12240	12206.6	12436.6	12537.733	12383.934	12337	12518.733	**12186.2**
25	13059.533	13186.533	12971.066	12944.267	13154.467	13204.8	13105.866	13077.066	13239.934	**12906.467**
Average	10359.71	10461.02	10276.22	10252.36	10435.89	10488.39	10364.63	10349.34	10490.35	**10241.89**

TABLE III

DISSIPATIVE DATA SETS

	MT Δavg	Arnold Cat Δavg	Burgers Δavg	Delayed Logistic Δavg	Dissipative Δavg	Henon Δavg	Ikeda Δavg	Lozi Δavg	Sinai Δavg	Tinkerbell Δavg
1	701	701	701	701	701	701	701	701	701	701
2	621	621	621	621	621	621	621	621	621	621
3	769	769	769	769	769	769	769	769	769	769
4	743	743	743	743	743	743	743	743	743	743
5	691	691	691	691	691	691	691	691	691	691
6	2230	2230	2230	2230	2230	2230	2230	2230	2230	2230
7	2189.66	2212.93	2188.87	2189.73	2209.67	2217.07	2201.13	2199.93	2216.07	**2188.73**
8	2130.733	2133.467	2129.533	**2128.667**	2132.667	2134.467	2130.867	2130	2130	2128.867
9	2204.2	2215.533	2202.067	**2200.067**	2210.6	2224.667	2205	2205.6	2220.133	2200.2
10	2426.6	2439.73	2418.33	2416	2441.533	2447.333	2426.8	242.467	2441.6	**2412.8**
11	14677.333	14820.934	14601.4	14574.4	14799.467	14856.467	14703.934	14700.333	14839.8	**14547.4**
12	15626.2	15810.267	15569.134	**15491.467**	15762.733	15807.4	15679	15668.134	15806.667	15509.066
13	15115.533	15249.134	15055.134	15057.6	15209.866	15284.8	15158.134	15139.667	15268.733	**15033.267**
14	13565.934	13696.934	13472.533	13473.733	13656.134	13736.066	13578.467	13572.934	13710.2	**13441.134**
15	12959.6	13096.6	12914.866	12863.934	13068.866	13137.733	12998.467	13004.8	13123.467	**12879**
16	21409.934	21560.533	21255.268	21219.666	21512.334	21628.133	21420.666	21375.934	21612.334	**21174.867**
17	20869.666	21028.467	20723.934	**20668.199**	21009.467	21042	20895.6	20840.666	21091.867	20678.801
18	20698.801	20892.801	20542.934	20507.334	20775	20933.268	20696.4	20663.801	20886	**20505.133**
19	21009	21256.334	20875.934	20870.666	21195.867	21298.666	21056.934	21034.666	21335.066	**20832.4**
20	21074.467	21210.533	20917.334	**20863.801**	21198.666	21286.934	21062.732	21032.801	21276	20864
21	31887.133	32214.066	31735.533	**31610.133**	32144.066	32263	31937.934	31915.867	32257.268	31697.533
22	32409	32647.467	32193.467	32130.6	32610.268	32717.533	32418.6	32414.066	32743.334	**32079.801**
23	33096	33305.535	32833.867	32793.133	33255.066	33376.801	33100.266	33049.332	33446.934	**32735.467**
24	33999.734	34190.934	33798.602	**33637.668**	34253.066	34314.332	33996.801	33951.266	34318.734	33657.535
25	31503.334	31811	31282	31236.732	31719.334	31894	31550.334	31509.4	31842.4	**31145.666**
Average	14184.315	14301.928	14098.629	14067.542	14276.787	14334.227	14198.923	14096.266	14332.864	**14058.666**

Four different parameters are presented; minimum, maximum, average and execution time for each instance *class*. The instance *class* here refers to the grouping of the problems according to size; 10 x 5, 20 x 10, 50 x 25, 75 x 30 and 100 x 50.

The parameter of most interest is the *average*, as it presents the overall performance of the algorithm. $CDABC_T$ has three better *class* averages of size 20 x 10, 50 x 25 and 100 x 50, whereas EDE_C performs better for the 10 x 5 and 75 x 30. Also, EDE_C has the better cumulative average of 10435.73 compared to 10917.93. However, it is quite obvious that the bias of the 75 x 30 (8297.76 against 11561.49) data class greatly influences the cumulative average in EDE_C favour.

The comparison results between EDE_C and $CDABC_T$ for the Dissipative data sets are given in Table VIII. Apart from the 10 x 5 data class, $CDABC_T$ obtains better results for all the remaining data classes, in addition to the cumulative average value of 14058.74 against 14152.97.

Therefore, it can be stated that $CDABC_T$ is a better performing algorithm compared to EDE_C for the non-idling problem.

VIII. Conclusion

The main premise of this research is the application and validation of chaos induced Discrete Artificial Bee Colony algorithm. To this effect, nine unique chaotic systems have been embedded in DABC and compared with the Mersenne twister. The lot-streaming flowshop scheduling problem with setup time has been used as the stock problem for this validation process.

From the obtained results, the Tinkerbell variant has been shown to be the best performing for both the Lozi and Dissipative data sets. For the individual data instances, the Tinkerbell variant obtains 18 better average results for the Lozi data set and 12 for the Dissipative data set.

In terms of comparison between Tinkerbell and Mersenne Twister, the twosided *t-test* pairwise comparison shows that the two variants are significantly different with a 95% confidence level. In fact, the top three performing algorithms are significantly different from each other and Mersenne twister.

These obtained results lend weight to the argument that using chaos maps as CPRNG's in DABC significantly improves its performance. This is in line with what has been reported for other algorithms in recent literature.

Moreover, $CDABC_T$ has been compared with EDE_C for the non-idling lot streaming problem, and from the obtained results, has shown to be better performing on the majority of the data classes.

Therefore, it can be concluded that chaos variant of DABC is a significant improvement of the canonical DABC of [4], and has comparable performance with other algorithms in the lot-streaming flowshop scheduling problem with setup time.

A future direction could be the application of parallel processing of the bee hive, either using multi-core or GPU based approach. The application of different chaos map in different computing cores operating in parallel could provide a better application for the $CDABC_T$ for different problem classes.

References

[1] M. Matsumoto and T. Nishimura, "Mersenne twister: A 623-dimensionally equidistributed uniform pseudorandom number generator," *ACM Transaction on Modeling and Computer Simulation*, vol. 8, no. 1, pp. 3–30, 1998.

[2] M. Matsumoto, "Mersenne twister webpage," 2012, http://www.math.sci.hiroshima-u.ac.jp/ m-mat/MT/ARTICLES/earticles.html.

[3] D. Karaboga, "An idea based on honey bee swarm for numerical optimization," *Technical Report TR06, Computer Engineering Department, Erciyes University, Turkey*, 2005.

[4] Q.-K. Pan, M. F. Tasgetiren, P. Suganthan, and T. Chua, "A discrete artificial bee colony algorithm for the lot-streaming flow shop scheduling problem," *Information Sciences*, vol. 181, pp. 2455 – 2468, 2011.

[5] B. Alatas, E. Akin, and A. Ozer, "Chaos embedded particle swarm optimization algorithms," *Chaos, Solitons and Fractals*, vol. 40, no. 4, pp. 1715–1734, 2009.

[6] R. Caponetto, L. Fortuna, S. Fazzino, and M. Xibilia, "Chaotic sequences to improve the performance of evolutionary algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 3, pp. 289–304, 2003.

[7] D. Davendra, R. I. Zelinka, Senkerik, and M. Bialic-Davendra, "Chaos driven evolutionary algorithm for the traveling salesman problem," in *Traveling Salesman Problem, Theory and Applications,*, D. Davendra, Ed. Croatia: InTech Publishing, 2010, pp. 55–70.

[8] X. Zuo and Y. Fan, "A chaos search immune algorithm with its application to neuro-fuzzy controller design," *Chaos, Solitons and Fractals*, vol. 30, no. 1, pp. 94–109, 2006.

[9] A. B. Ozer, "Cide: Chaotically initialized differential evolution," *Expert Systems with Applications*, vol. 37, no. 6, pp. 4632 – 4641, 2010.

[10] C. Herring and P. Julian, "Random number generators are chaotic," *ACM SIGPLAN*, vol. 11, pp. 1–4, 1989.

[11] D. Lehmer, "Mathematical methods in large-scale computing units," *Ann. Computing Lab, Harvard University*, vol. 26, pp. 141–146, 1951.

[12] J. Palmore and J. McCauley, "Shadowing by computable chaotic orbits," *Physics Letters A*, vol. 121, p. 399, 1987.

[13] I. Zelinka, M. Chadli, D. Davendra, R. Senkerik, M. Pluhacek, and J. Lampinen, "Hidden periodicity - chaos dependance on numerical precision," *Advances in Intelligent Systems and Computing*, vol. 210, pp. 47–59, 2013.

[14] T. Stojanovski and L. Kocarev, "Chaos-based random number generatorspart i: Analysis," *IEEE Transactions on Circuits and Systems - I: Fundamental Theory and Applications*, vol. 48, no. 3, pp. 281–288, 2001.

[15] R. Lozi, "New enhanced chaotic number generators," *Indian Journal of Industrial and Applied Mathematics*, vol. 1, no. 1, pp. 1–23, 2008.

[16] ——, "Chaotic pseudo random number generators via ultra weak coupling of chaotic maps and double threshold sampling sequences," in *ICCSA 2009 The 3rd International Conference on Complex Systems and Applications*, University of Le Havre, France, Jun. 2009, pp. 1–5.

[17] I. Zelinka, M. Chadli, D. Davendra, R. Senkerik, M. Pluhacek, and J. Lampinen, "Do evolutionary algorithms indeed require random numbers? extended study," *Advances in Intelligent Systems and Computing*, vol. 210, pp. 61–75, 2013.

[18] M. Pluhacek, R. Senkerik, D. Davendra, Z. Kominkova Oplatkova, and I. Zelinka, "On the behavior and performance of chaos driven pso algorithm with inertia weight," *Computers and Mathematics with Applications*, vol. 66, no. 2, pp. 122–134, 2013.

[19] M. Pluhacek, R. Senkerik, I. Zelinka, and D. Davendra, "Chaos pso algorithm driven alternately by two different chaotic maps-an initial study," *2013 IEEE Congress on Evolutionary Computation, CEC 2013*, pp. 2444–2449, 2013.

[20] R. Senkerik, M. Pluhacek, D. Davendra, I. Zelinka, and Z. Kominkova Oplatkova, "Chaos driven evolutionary algorithm: A new approach for evolutionary optimization," *International Journal of Mathematics and Computers in Simulation*, vol. 7, no. 4, pp. 363–368, 2013.

[21] D. Davendra, I. Zelinka, and R. Senkerik, "Chaos driven evolutionary algorithms for the task of pid control," *Computers & Mathematics with Applications*, vol. 60, no. 4, pp. 1088 – 1104, 2010.

TABLE VII

COMPARISON WITH EDE_C FOR LOZI NON-IDLING RESULTS

	EDE_C				**$CDABC_T$**			
Instance	Min	Max	Average	Time (sec)	Min	Max	Average	Time (sec)
10 x 5	425	554	**510.92**	0.89	511	511	511	0.88
20 x 10	2017	2230	2139.68	38.78	2099	2102.2	**2100.62**	3.12
50 x 25	27603	30066	28837.28	494.40	27847.4	28274	**28102.08**	22.11
75 x 30	7912	8584	**8297.76**	773.70	11484.5	11631.83	11561.49	32.2
100 x 50	12039	13191	12393.0	3309.70	12226.6	12387.40	**12314.46**	67.81
Average	9999.20	10925.00	**10435.73**	923.49	10833.7	10981.29	10917.93	25.23

TABLE VIII

COMPARISON WITH EDE_C FOR DISSIPATIVE NON-IDLING RESULTS

	EDE_C				**$CDABC_T$**			
Instance	Min	Max	Average	Time (sec)	Min	Max	Average	Time (sec)
10 x 5	613	762	**700.60**	0.20	705	705	705	0.72
20 x 10	2145	2494	2276.27	40.32	2228.8	2237.2	**2232.49**	2.62
50 x 25	12897	15936	14517.47	514.40	14181.6	14355.2	**14281.97**	17.36
75 x 30	20437	21958	20931.2	772.3	20686.4	20938	**20811.04**	32.87
100 x 50	30904	33984	32339.33	3401.40	32092	32411	**32263.2**	80
Average	13399.20	15026.80	14152.97	945.62	13978.76	14129.28	**14058.74**	26.71

[22] Y. Lu, J. Zhou, H. Qin, Y. Wang, and Y. Zhang, "Chaotic differential evolution methods for dynamic economic dispatch with valve-point effects," *Engineering Applications of Artificial Intelligence*, vol. 24, no. 2, pp. 378 – 387, 2011.

[23] X. Yuan, B. Cao, B. Yang, and Y. Yuan, "Hydrothermal scheduling using chaotic hybrid differential evolution," *Energy Conversion and Management*, vol. 49, no. 12, pp. 3627 – 3633, 2008.

[24] M. F. Tasgetiren, Q.-K. Pan, P. Suganthan, and A. Chen, "A discrete artificial bee colony algorithm for the total flowtime minimization in permutation flow shops," *Information Sciences*, vol. 181, pp. 3459 – 3475, 2011.

[25] M. F. Tasgetiren, Q.-K. Pan, P. Suganthan, and A. Oner, "A discrete artificial bee colony algorithm for the no-idle permutation flowshop scheduling problem with the total tardiness criterion," *Applied Mathematical Modelling*, vol. 37, pp. 6758 – 6799, 2013.

[26] J.-Q. Li, Q.-K. Pan, and M. F. Tasgetiren, "A discrete artificial bee colony algorithm for the multi-objective flexible job-shop scheduling problem with maintenance activities," *Applied Mathematical Modelling*, 2013.

[27] L. Wang, *Shop Scheduling with Genetic Algorithms*. Beijing, China: Tsinghua Univ Press, 2003.

[28] J. Sprott, *Chaos and Time-Series Analysis*. UK: Oxford University Press, 2003.

[29] Q.-K. Pan, M. Fatih Tasgetiren, P. N. Suganthan, and T. J. Chua, "A discrete artificial bee colony algorithm for the lot-streaming flow shop scheduling problem," *Inf. Sci.*, vol. 181, no. 12, pp. 2455–2468, Jun. 2011.

[30] Q.-K. Pan and R. Ruiz, "An estimation of distribution algorithm for lot-streaming flow shop problems with setup times," *Omega*, vol. 40, no. 2, pp. 166–180, April 2012.

[31] S. C. Sarin and P. Jaiprakash, *Flow Shop Lot Streaming*. Berlin: Springer, 2007.

[32] D. Davendra. (2012) Flowshop lot-streaming problem data sets. [Online]. Available: http://mrl.cs.vsb.cz/people/davendra/research.html

[33] D. Davendra, M. Bialic-Davendra, and R. Senkerik, "Scheduling the lot-streaming flowshop scheduling problem with setup time with the chaos-induced enhanced differential evolution," *Proceedings of the 2013 IEEE Symposium on Differential Evolution, SDE 2013 - 2013 IEEE Symposium Series on Computational Intelligence, SSCI 2013*, pp. 119–126, 2013.