Swarm and Evolutionary Computation 25 (2015) 15–28

Contents lists available at ScienceDirect

Swarm and Evolutionary Computation

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

Chaos driven discrete artificial bee algorithm for location and assignment optimisation problems

Magdalena Metlicka, Donald Davendra *

Department of Computer Science, Faculty of Electrical Engineering and Computer Science, VŠB-Technical University of Ostrava, 17. listopadu 15, 708 33 Ostrava-Poruba, Czech Republic

ARTICLE INFO

Article history:
Received 25 September 2014
Received in revised form
18 February 2015
Accepted 4 March 2015
Available online 24 March 2015

Keywords:
Artificial bee colony
Chaos maps
Quadratic assignment problem
Capacitated vehicle routing problem

ABSTRACT

The chaos driven discrete artificial bee colony (CDABC) algorithm is introduced in this paper. Four unique chaos maps of Burgers, Lozi, Delayed Logistic and Tinkerbell are embedded as chaos pseudo-random number generators and compared with the Mersenne Twister pseudo-random number generator. Two unique problems of quadratic assignment and capacitated vehicle routing problem are solved using five different variants of the algorithm and analytical comparison is conducted. Furthermore, paired *t*-test is done pairwise on all variants, and from these results it is ascertained that the Tinkerbell variant of CDABC is the best performing for both problem classes.

1. Introduction

One of the main pillars of evolutionary algorithms (EA's) is their reliance on randomness or stochasticity, which is used to spark a path towards a desired goal. The current norm is the use of pseudo-random number generators (PRNG's); a structured sequence of mathematical formulation which tries to yield a generally optimal distribution of numbers within a specified range. Of these, the Mersenne Twister is the most famous and widely used [1,2].

This research evolves around the generation of chaotic sequences, which are then used as chaos pseudo-random number generators (CPRNG's) in an EA. The chaotic behaviour when observed may seem erratic and somewhat random, however, these systems are deterministic, the precise description of their future behaviour is well known. The proposition is then to reconcile the notion of nonlinearity of these systems.

This aperiodic non-repeating behaviour of chaotic systems is the foundation of this research. The objective is then to analyse different chaotic systems, which in this case are the discrete dissipative systems, and to analyse, which of these improve the application of EA's.

A mathematical description of the connection between chaotic systems and random number generators has been given by [3]. In this paper, a strong linkage has been shown between the Lehmer generator [4] and the simple chaos dynamical system of Bernoulli shift [5]. The hidden periodicity of chaos system and its dependence on numerical system have been shown by [6]. A chaotic piecewise-linear one dimensional (PL1D) map has been utilised as a chaotic random number generator in [7]. 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 [8], where the main impetus is the generation of very long series of pseudo-random numbers. This is 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 [9]. Recently, the very notion of using CPRNG's in EA's has been explored by [10].

Current literature contains a number of research devoted to certainty, ergodicity and the stochastic property of chaotic systems. Recently, chaotic sequences have been adopted instead of random sequences with improved results. 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 [11].

A number of EA's have been improved using chaos systems as random number generators during the past few years. Genetic

* Corresponding author.
E-mail addresses: magdalena.metlicka.st@vsb.cz (M. Metlicka), donald.davendra@vsb.cz (D. Davendra).

http://dx.doi.org/10.1016/j.swevo.2015.03.002
2210-6502/

Algorithms (GA's) have been improved by chaos to solve multi-objective optimisation problems in [12,13] and tourism demand forecasting [14], whereas the Firefly algorithm has been embedded with chaos map in [15]. Differential Evolution (DE) has been improved with chaos to solve the support vector regression machine problem [16], dynamic economic dispatch for wind–thermal power systems [17], loudspeaker design problem [18], optimisation of the batch reactor [19], hydrothermal scheduling [20] and PID control problem [21].

The largest group of chaos based literature is on Particle Swarm Optimisation (PSO). Many variants of chaos embedded PSO exist, some of them are novel creations comprising different chaotic approaches in basic PSO design [22–26], accelerated chaos [27] and hybrid approaches [28] amongst others.

Applications of chaos based PSO include pattern synthesis of antenna arrays [29], image matching [30], power system stabiliser design [31], constrained predictive control [32], global numerical optimisation [28], optimisation of heat exchangers [33], reactive power optimisation [34], network intrusion detection [35], PID Controller design [36] and parameters selection [37].

Some of the other algorithm employing chaos are the Discrete Self-Organising Migrating Algorithm (DSOMA) which has been used to solve the lot-streaming problem [38] and a new artificial emotion based chaos algorithm [39].

This paper introduces the chaos driven Discrete Artificial Bee Colony (CDABC) algorithm. The Artificial Bee Colony (ABC) algorithm is one of the newest and more versatile algorithms which has proven highly effective in large scale optimisation [40]. The discrete ABC (DABC) algorithm of [41] is embedded with four unique chaos maps: Burgers, Delayed Logistic, Lozi and Tinkerbell. These new variants of CDABC are then applied to two different applications problems in location and assignment optimisation. For the location problem, the capacitated vehicle routing problem (CVRP) is selected, whereas the quadratic assignment problem

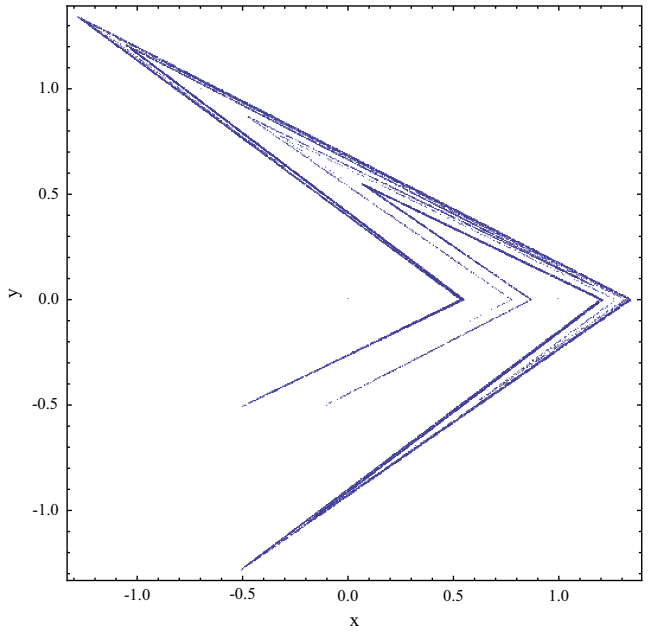

Fig. 1. Chaotic Lozi map 2D plot.

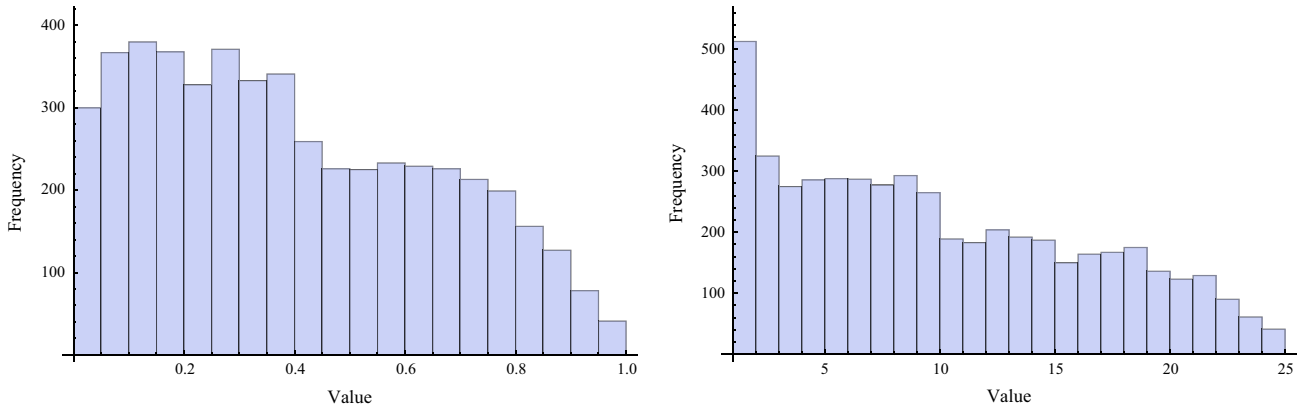

Fig. 2. Histogram of the distribution of real numbers transferred into the range $\langle 0, 1\rangle$ and integer numbers in the range $\langle 1, 25\rangle$ generated by means of the chaotic Lozi map, 5000 samples.

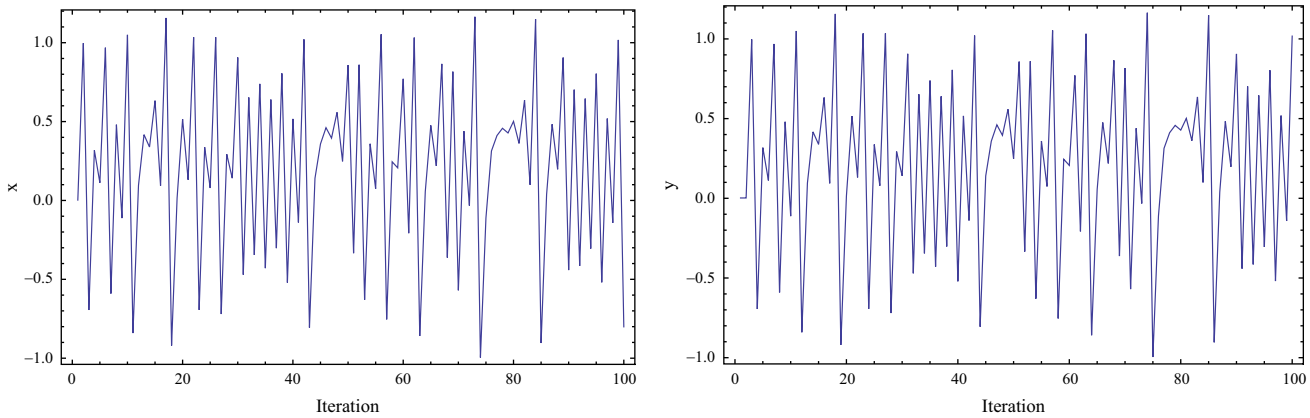

Fig. 3. Iteration of the Lozi map for the x and y values in line-plot.

(QAP) is selected in the assignment problem class. The rationale for selecting two unique and different problem classes is that we intent to show that chaos embedded CDABC algorithm can be effectively used between different problem domains with the same level of performance, without modifications, thereby outlining its versatility and robustness.

The paper is organised as follows: Section 2 outlines the ABC algorithm with the DABC algorithm given in Section 3. The four different chaos maps of Burgers, Lozi, Delayed Logistic and Tinkerbell are described in Section 4. The chaos driven DABC (CDABC) algorithm is shown in Section 5, with the problem formulation in Section 6. Section 7 gives the experimentation results and analysis, with the work being concluded in Section 8.

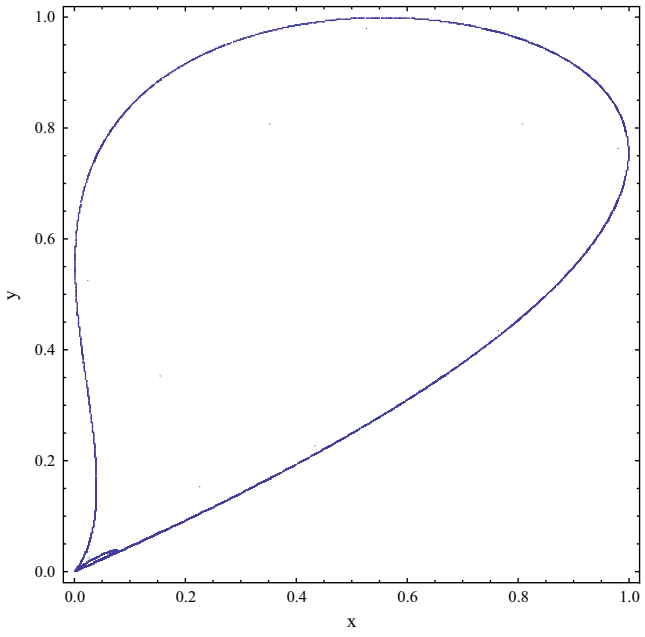

Fig. 4. Chaotic Delayed Logistic map 2D plot.

2. Artificial bee algorithm

The ABC algorithm was originally developed by Karaboga [40], for the purpose of multi-variable multi-modal continuous functions optimisation. Since then, many variations to the original ABC algorithm were created and employed in solving different types of problems (constrained optimisation [42], multi-objective optimisation [43], combinatorial optimisation [44–46,41]).

In the classification of optimisation heuristics, the Artificial Bee Colony belongs to the group of population, swarm intelligence based stochastic algorithms. Along with other algorithms of this category (such as ACO [47], PSO [48], SOMA [49,50] or Artificial Immune Algorithm (AIA) [51]), ABC searches for optimal solution employing a certain number of intelligent agents, who search independently, but also share the information on the system, in order to achieve more efficient behaviour. Amongst the advantages of such approach is inherent parallelism and scalability – it is

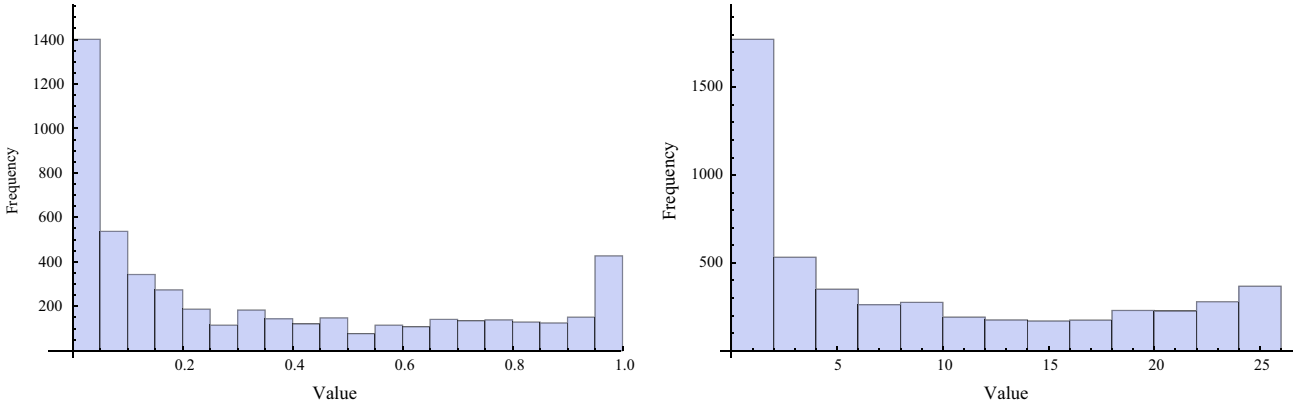

Fig. 5. Histogram of the distribution of real numbers transferred into the range $\langle 0, 1\rangle$ and integer numbers in the range $\langle 1, 25\rangle$ generated by means of the chaotic Delayed Logistic map, 5000 samples.

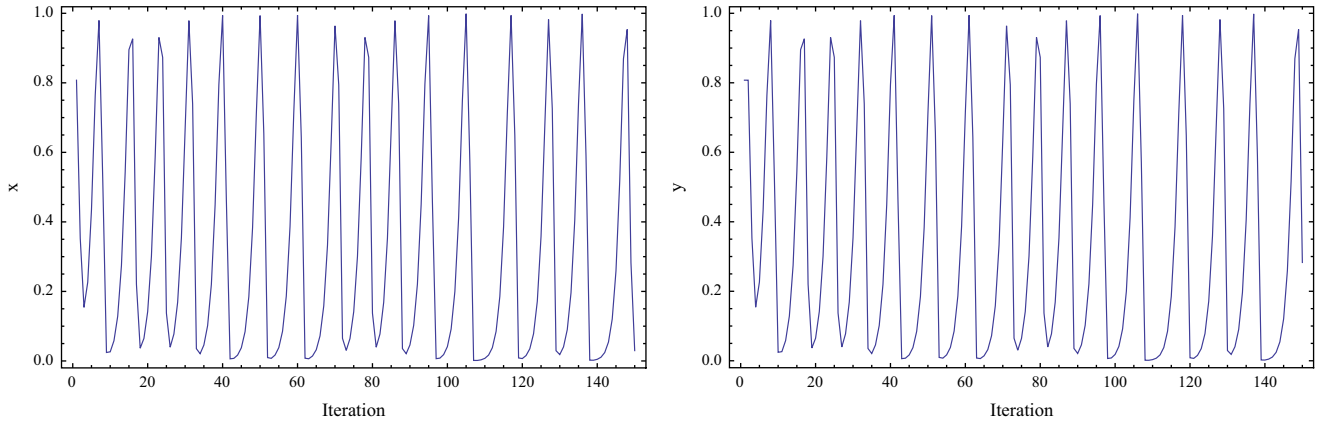

Fig. 6. Iteration of the Delayed Logistic map for the x and y values in line-plots.

possible to assign different subgroups of agents to different computational resources.

2.1. Description

As well as many other population based metaheuristics, ABC takes inspiration from nature. Namely, from the intelligent behaviour of foraging honey bee swarm. From this inspiration stems also the nomenclature. Several solutions, called food sources, form the population. These solutions are exploited by employed bees. Each employed bee tries to improve one solution at a time (which symbolises extracting a nectar from a food source). Onlooker bees are waiting in the hive to make a decision where to look for a food source. When employed bee arrives, it performs a specific dance, which points the onlooker bee to the direction of a food source. The onlooker bee then determines, based on the observation of returning employed bees, where to go. The algorithm emulates this by evaluating the cost function for each solution generated by employed bee. The onlookers choose a solution to improve, based on probability given by relative amount of nectar (cost function value relative to sum of all solutions costs). Finally, if the food source has been exhausted, a scout bee will try to find a new one, unrelated to the previous. Both employed bees and onlooker bees perform exploitation, whereas scout bees responsibility is the exploration of solutions space [52].

The overall algorithm therefore consists of initialisation of population, and three phases performed in succession iteratively, until the predefined terminating criterion is met: employed bee phase, onlooker bee phase, and eventually scout bee phase, as outlined above. The top-level pseudocode of ABC is given in Algorithm 1.

Algorithm 1. ABC top-level pseudocode.

```
initialize
repeat
    send employed bees to food sources
    send onlooker bees to selected food sources
    send scout bees to search for new food sources
    memorize best solution
until terminating criterion met
```

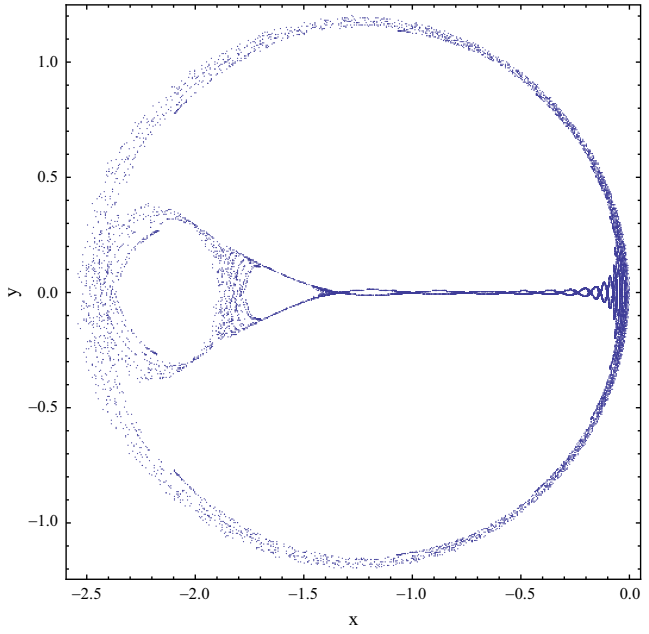

Fig. 7. Chaotic Burgers map 2D plot.

The solution in continuous space is represented as a numerical vector of dimension D, whose elements are values of optimised parameters. The population is formed by FS such solutions. The quantity associated with each solution, before it is considered exhausted (a number of trials for bees to try and improve the solution) is called limit. Limit, FS and terminating criterion are the only controlling parameters of the ABC algorithm. Dimension D is given by a definition of optimised function.

2.2. Initialisation

Unless some preliminary information on the system is available, the population of solutions is initialised randomly as follows:

$$\mathbf{x}_{i,j} = L_j + (U_j - L_j) \cdot r \tag{1}$$

where $\mathbf{x}_{i,j}$ denotes jth element of ith vector, $j \in \{1, 2, \ldots, D\}$, $i \in \{1, 2, \ldots, FS\}$. L_j and U_j are, respectively, lower and upper bounds of jth dimension of the search space (minimal and maximal values, respectively, allowed for the jth element of a vector), and r represents a random number in range $[0, 1]$.

2.3. Employed bee phase

In an employed bee phase, as mentioned earlier, each employed bee i tries to improve a single solution $\mathbf{x}_i$, by performing

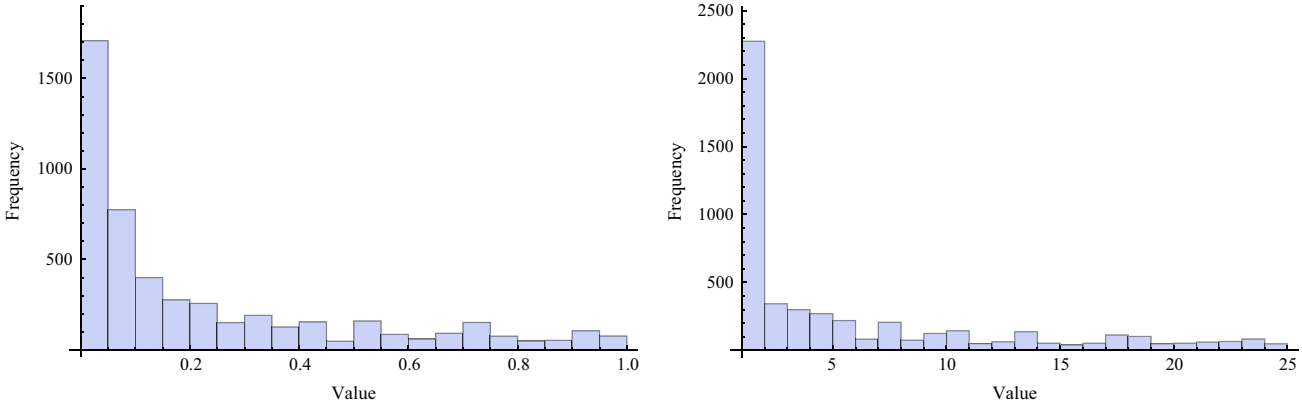

Fig. 8. Histogram of the distribution of real numbers transferred into the range $\langle 0, 1\rangle$ and integer numbers in the range $\langle 1, 25\rangle$ generated by means of the chaotic Burgers map, 5000 samples.

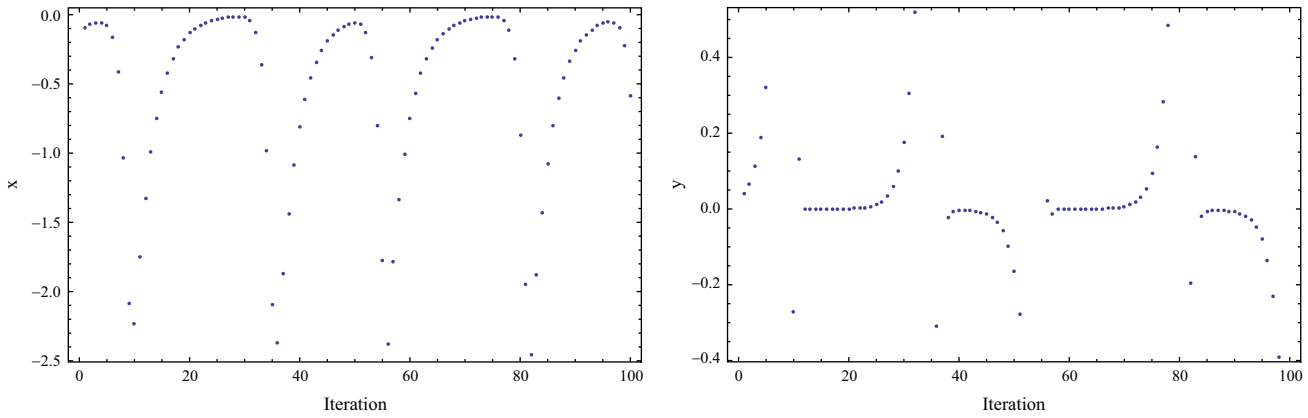

Fig. 9. Iteration of the Burgers map for the x and y values in point-plot.

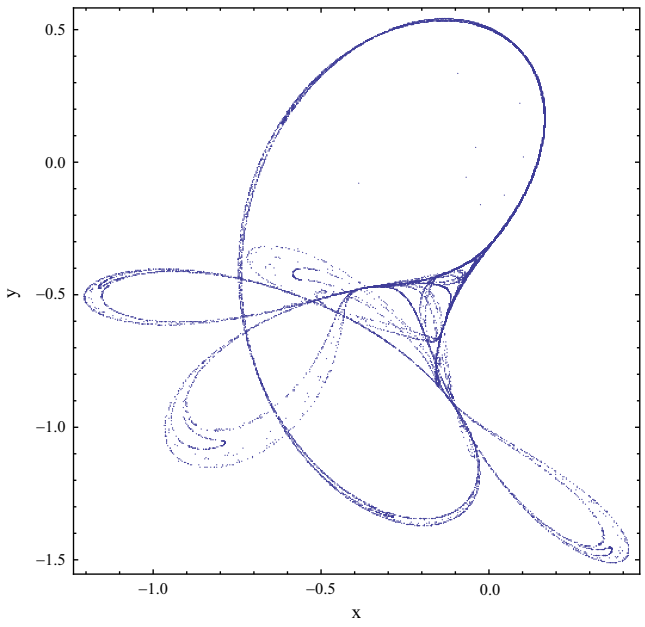

Fig. 10. Chaotic Tinkerbell map 2D plot.

an operation on it, according to Eq. (2). The newly created candidate solution $\mathbf{n}_i$ is evaluated, and greedy selection is applied – if a solution $\mathbf{n}_i$ is better than or equal to previous $\mathbf{x}_i$, the previous one is replaced in the population:

$$\mathbf{n}_{i,j} = \mathbf{x}_{i,j} + (\mathbf{x}_{i,j} - \mathbf{x}_{k,j}) \cdot \mathbf{r}_{i,j} \tag{2}$$

where $\mathbf{x}_i$ is the previous solution in the population, $\mathbf{n}_i$ is the candidate solution, $i \in \{1, 2, \ldots, FS\}$, index j is randomly chosen in range $[1, D]$, index k is randomly selected from range $[1, FS]$ so that $k \neq i$ and $\mathbf{r}_{i,j}$ is a random number in range $[-1, 1]$.

2.4. Onlooker bee phase

In an onlooker bee phase, a selection from all the food sources (solutions) $\{\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_{FS}\}$ is performed, based on the probability of each solution, $p_i, i \in \{1, 2, \ldots, FS\}$, defined in Eq. (3). The selected solution is modified using Eq. (2) and the greedy selection is again performed between the original and the modified solution, to determine the winning solution to be left in the population, in the same way as described in Section 2.3:

$$p_i = \frac{f_i}{\sum_{k=1}^{FS} f_k} \tag{3}$$

In Eq. (3), p_i represents the probability, f_i is the fitness of ith solution, $i \in \{1, 2, \ldots, FS\}$. FS is the total number of solutions in the population.

2.5. Scout bee phase

As mentioned earlier, each solution has a limit of attempts to improve, associated with it. It is being incremented in both employed bee and onlooker bee phase, in case the solution fails to improve (the newly created solution is worse and therefore not selected). If this limit is exceeded, a scout bee generates new, random solution, according to Eq. (1), which replaces the exhausted one. In each iteration of the original ABC algorithm, at most one scout bee is released. The increase in the number of scout bees encourages exploration (escaping from local optima and searching in the global space), but also reduces the possibility of exploitation of good solutions already found, since these are removed from the population by this process [40].

3. Discrete artificial bee algorithm

The DABC algorithm by [41] is a modification to the ABC algorithm for solving combinatorial optimisation problems. As opposed to the continuous optimisation, a solution in combinatorial optimisation is naturally encoded as a permutation of elements. Several approaches are possible for transforming continuous optimisation algorithm for solving combinatorial problems, as described for example in [53].

DABC replaces the neighbourhood generation operation of original ABC by a set of operations which transform one permutation to another, thus completely avoiding generation of infeasible solutions. These operations (4 operations altogether, described in Section 3.2) are based on swapping two random elements, or inserting an element into permutation.

Each of the operations explores a different type of neighbourhood of a solution, furthermore, each of them is suitable for exploring solution space of different problems. In order to maximally adapt to given problem and explore the neighbourhood in an efficient way, all 4 defined operations are used in an adaptive mechanism. This adaptive strategy, described in Section 3.3, decides which operation to use, partly depending on the list of previous successful operations (those which produced improved solution), partly randomly.

To enhance the exploitation of solution space, DABC contains embedded local search, described in Section 3.4.

3.1. Solution as permutation

As mentioned earlier, a solution is represented as a D-dimensional permutation of elements, as shown in Eq. (4). The objective of optimisation is to find the least cost permutation. π_i represents the ith solution, $\pi_{i,j}$ the element at jth position in the permutation:

$$\pi_i = \{\pi_{i,1}, \pi_{i,2}, \ldots, \pi_{i,D}\} \tag{4}$$

3.2. Operations

As mentioned earlier, DABC makes use of 4 operations: Insert, Swap, 2 × Insert, 2 × Swap, defined as follows:

Insert removes a randomly selected permutation element from its position j and reinserts at different randomly selected position k:

$$\begin{aligned}\pi^{(i)} &= (\pi_1, \ldots, \pi_{j-1}, \pi_j, \pi_{j+1}, \ldots, \pi_{k-1}, \pi_k, \pi_{k+1}, \ldots, \pi_d)\\ \pi^{(i+1)} &= (\pi_1, \ldots, \pi_{j-1}, \pi_{j+1}, \ldots, \pi_{k-1}, \pi_k, \pi_j, \pi_{k+1}, \ldots, \pi_d)\end{aligned} \tag{5}$$

Swap exchanges 2 different randomly selected elements j and k:

$$\begin{aligned}\pi^{(i)} &= (\pi_1, \ldots, \pi_{j-1}, \pi_j, \pi_{j+1}, \ldots, \pi_{k-1}, \pi_k, \pi_{k+1}, \ldots, \pi_d)\\ \pi^{(i+1)} &= (\pi_1, \ldots, \pi_{j-1}, \pi_k, \pi_{j+1}, \ldots, \pi_{k-1}, \pi_j, \pi_{k+1}, \ldots, \pi_d)\end{aligned} \tag{6}$$

2 × Insert performs 2 successive Insert operations, as defined above. 2 × Swap performs 2 successive Swap operations. Which operation is applied to modify given solution depends on adaptive strategy list.

3.3. Adaptive strategy

Adaptive strategy, as described in [41], maintains two lists of operations, behaving like stacks – list of available operations *NL*, and list of previously successful operations *WNL*. On the beginning of the optimisation, the *NL* is filled with randomly selected operations (4 available operators). Before a solution is modified, operation is taken from the top of *NL*. If the solution improves upon the previous one, the operation is inserted into *WNL*. If a *NL* is empty, part of it is refilled from operations stored in *WNL*, the rest is refilled again with randomly selected operations. If *WNL* is empty, the last *NL* is used again.

3.4. Local search

Local search is embedded within DABC. In the employed bee phase, after a new successful solution is generated, the local search is performed with probability *PL*, in order to further enhance it. The pseudocode is given in Algorithm 2.

Algorithm 2. Local search.

Input: $\pi, fitness, last\_operation$
begin select operation:
  if $last\_operation \in \{insert, 2 \times insert\}$ **then**
    $operation \leftarrow swap$
  else
    $operation \leftarrow insert$
  end
end
begin
  $\pi^{(1)} \leftarrow \pi$
  $fitness^{(1)} \leftarrow fitness$
  for $i = 1$ **to** $loop_{max}$ **do**
    $\pi_c \leftarrow operation(\pi^{(i)})$
    $fitness_c \leftarrow$ evaluate π_c
    if $fitness_c$ *better than or equal to* $fitness^{(i)}$ **then**
      $\pi^{(i+1)} \leftarrow \pi_c$
      $fitness^{(i+1)} \leftarrow fitness_c$
    else
      $\pi^{(i+1)} \leftarrow \pi^{(i)}$
      $fitness^{(i+1)} \leftarrow fitness^{(i)}$
    end
  end
  $\pi \leftarrow \pi^{(i)}$
  $fitness \leftarrow fitness^{(i)}$
end

3.5. Algorithm structure

DABC has similar structure to original ABC algorithm (pseudocode is shown in Algorithm 3). It consists of initialisation, and several iterations of bee phases performed sequentially, until stopping criterion is met. Unless some preliminary information on the problem solution space is known, the population is initialised as a set of random permutations.

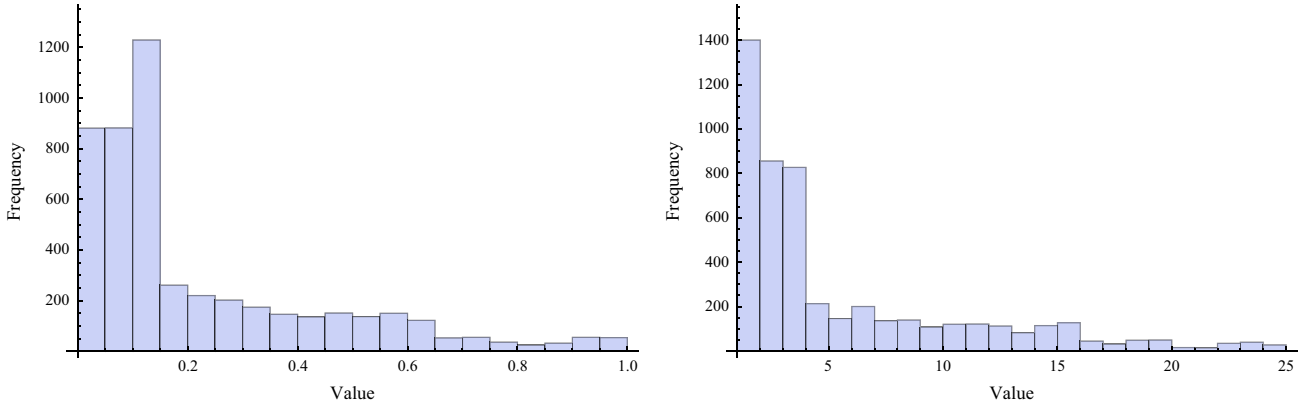

Fig. 11. Histogram of the distribution of real numbers transferred into the range $\langle 0, 1\rangle$ and integer numbers in the range $\langle 1, 25\rangle$ generated by means of the chaotic Tinkerbell map, 5000 samples.

Algorithm 3. DABC pseudocode.

begin initialize:
  generate food sources as random permutations
  evaluate food sources
  randomly fill NL
end
repeat
  begin 1.send employed bees to food sources :
    foreach *food source* π_i **do**
      /* `get operation from Adaptive strategy list :` */
      $operation \leftarrow get\_operation(NL)$
      $\pi_i^{(c)} \leftarrow operation(\pi_i)$
      $fitness_i^{(c)} \leftarrow evaluate\ \pi_i^{(c)}$
      if $fitness_i^{(c)}$ *better than or equal to* $fitness_i$ **then**
        local search on $\pi_i^{(c)}$ with probability *PL*
        $\pi_i \leftarrow \pi_i^{(c)}$
        $fitness_i \leftarrow fitness_i^{(c)}$
        update adaptive strategy WNL
      end
      update $limit_i$
    end
  end
  begin 2.let onlooker bees select food sources :
    foreach *onlooker bee* **do**
      randomly pick two food sources π_{r_1}, π_{r_2}
      $\pi_s \leftarrow$ better of π_{r_1}, π_{r_2}
      /* `get operation from Adaptive strategy list` */
      $operation \leftarrow get\_operation(NL)$
      $\pi_s^{(c)} \leftarrow operation(\pi_s)$
      $fitness_s^{(c)} \leftarrow$ evaluate $\pi_s^{(c)}$
      if $fitness_s^{(c)}$ *better than or equal to* $fitness_s$ **then**
        $\pi_s \leftarrow \pi_s^{(c)}$
        $fitness_s \leftarrow fitness_s^{(c)}$
        update adaptive strategy WNL
      end
      update $limit_s$
    end
  end
  begin 3.send the scout to search new food source :
    /* `select a solution which was not successfully improved given number of trials :` */
    $\pi_s \leftarrow \pi_i$, where $limit_i \geq Limit$
    $\pi_s \leftarrow$ perform 3 insert operations on π_{best}
    $fitness_s \leftarrow$ evaluate π_s
    $limit_s \leftarrow 0$
  end
  begin 4.memorize best food source :
    $\pi_{best} \leftarrow \pi_i$, where : $\forall k, k \neq i$: $fitness_i$ better than or equal to $fitness_k$
    $fitness_{best} \leftarrow fitness_i$
  end
until stopping criterion met

3.6. Parameters

There are 7 control parameters of DABC, 2 of which are only for usage in adaptive strategy. *FS* is a number of solutions or food sources in the population, and also the number of employed bees and onlooker bees. *Limit* is a maximum number of unsuccessful trials to improve the solution, before it is abandoned. $Loop_{max}$ defines a number of iterations in local search, *PL* is a probability of local search to happen. Stopping criterion in this variant is specified as a number of iterations of DABC algorithm to perform, *T*. The adaptive strategy requires two parameters, NL_L, WNL_L, defining length of *NL* list and *WNL* list, respectively [41]. The range and recommended values of parameters are described in Table 1.

4. Chaos maps

4.1. Lozi map

The Lozi map is a two-dimensional piecewise linear map whose dynamics are similar to those of the better known Henon map and it admits strange attractors (Fig. 1).

The advantage of the Lozi map is that one can compute every relevant parameter exactly, due to the linearity of the map, and the successful control can be demonstrated rigorously.

The Lozi map equations are given as

$$X_{n+1} = 1 - a|X_n| + bY_n \tag{7}$$

$$Y_{n+1} = X_n \tag{8}$$

The parameters used are $a=1.7$ and $b=0.5$ as suggested in [54] and the initial conditions are $X_0=-0.1$ and $Y_0=0.1$. The real number and integer number plots for a sample iteration are given in Fig. 2 and the *x* and *y* value 2D plots are given in Fig. 3. The figures presented of the chaotic maps are referenced from [55].

4.2. Delayed logistics map

The Delayed Logistic map is a dissipative map with a smooth invariant circle interspersed among parameter intervals for which the attractor appears to be strange [56]. This phenomena has given rise to its application in population growth models (Fig. 4). The equations of the Delayed Logistic are given as

$$X_{n+1} = AX_n(1 - Y_n) \tag{9}$$

$$Y_{n+1} = X_n \tag{10}$$

The operating parameters are $A=2.27$ and the initial conditions are $X_0=0.001$ and $Y_0=0.001$. The real and integer value histogram is given in Fig. 5, and the *x* and *y* value plots in Fig. 6.

4.3. Burgers map

The Burgers mapping is a discretisation of a pair of coupled differential equations which were used by Burgers [57] to illustrate the relevance of the concept of bifurcation to the study of hydrodynamic flows (Fig. 7). It has been numerically shown to produce a much richer set of dynamic patterns than those observed in continuous case [58]. The equations are given as

$$X_{n+1} = aX_n - Y_n^2 \tag{11}$$

$$Y_{n+1} = bY_n + X_nY_n \tag{12}$$

The operating parameters are $a=0.75$ and $b=1.75$ with the initial conditions being $X_0=-0.1$ and $Y_0=0.1$. As in the previous

Table 1
Parameter values.

Parameter	Recommended value	Range
FS	30	[5, ∞]
Limit	50	[1, ∞]
$Loop_{max}$	200	[1, *n*]
PL	0.2	[0, 1]
T	100	[1, *n*]
NL_L	20	[1, *n*]
WNL_L	0.75 NL_L	[1, *n*]

Table 2
Operating parameters of CDABC algorithm.

Parameter	Value
Food source (FS)	30
Limit (food source)	50
Loop (Local Search)	200
Local search probability (P_L)	0.2
Neighbourhood List (NL_L)	20
Winning neighbourhood list (WNL_L)	$0.75 \times NL_L$
Iterations	100

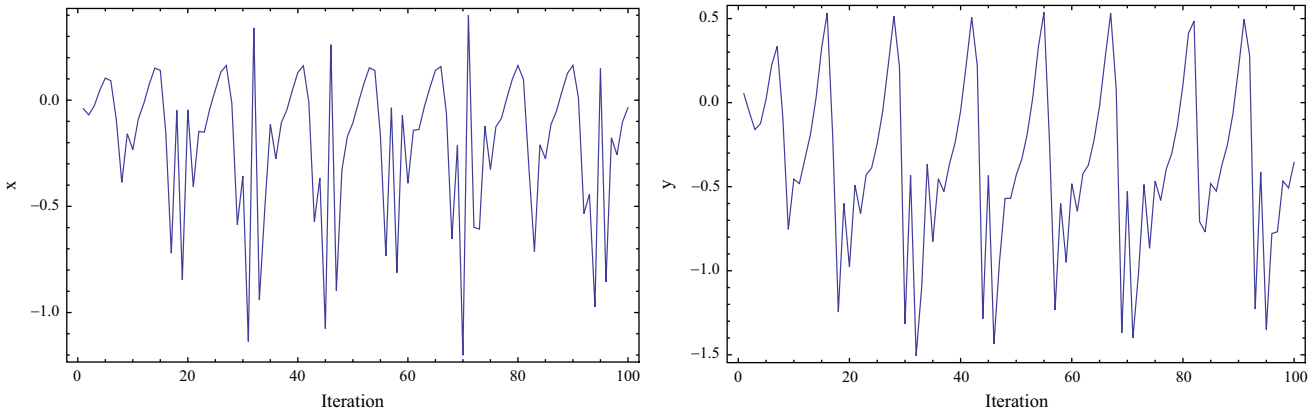

Fig. 12. Iteration of the Tinkerbell map for the *x* and *y* values in line-plot.

cases the frequency plot of the real and integer values is given in Fig. 8, whereas the x and y value plots are given in Fig. 9.

4.4. Tinkerbell map

The Tinkerbell is yet another Dissipative map, which has been proven to be chaotic [59] and studied extensively for its unique chaotic attractor (Fig. 10). The equations of the Tinkerbell is given as

$$X_{n+1} = X_n^2 - Y_n^2 + aX_n + bY_n \tag{13}$$

$$Y_{n+1} = 2X_nY_n + cX_n + dY_n \tag{14}$$

The usual operating parameters for Tinkerbell are $a=0.9$, $b=-0.6$, $c=2$ and $d=0.5$. The initial conditions are $X_0=0$ and $Y_0=0.5$. The real and integer frequency plots are given in Fig. 11, whereas the x and y values are given in Fig. 12.

Algorithm 4. CPRNG pseudocode.

begin CPRNG_init:
  Input : *seed*
  /* set initial point on map : */
  $\leftarrow seed \pmod{0.1}$
  $Y \leftarrow X$
end
begin CPRNG_get_next:
  /* compute next X and Y according to given chaotic map equation : */
  $X^{(n+1)} \leftarrow f_x(X^{(n)}, Y^{(n)}, Y^{(n+1)})$
  $Y^{(n+1)} \leftarrow f_y(X^{(n)}, Y^{(n)}, X^{(n+1)})$
  return $X^{(n+1)} (mod\ 1)$
end

5. Chaos driven discrete artificial bee algorithm

The DABC algorithm, as described in Section 3, makes heavy use of randomness in its flow. The stock DABC does not pay much attention to the applied pseudo-random generator, and presumes the usage of one of the standard PRNG's. The most commonly used PRNG is Mersenne Twister, owing to its good stochastic properties and speed [1]. This article explores the possibility to improve the standard DABC algorithm by exchanging the common PRNG for a CPRNG, for the purpose of solving QAP and CVRP problems. Expanding upon the previous work solving permutative flow shop with no wait constraint [60], four of the previously best performing CPRNGs were utilised to modify DABC algorithm, creating four new variants of DABC: $CDABC_L$ using Lozi map, $CDABC_{DL}$ using Delayed Logistic map, $CDABC_B$ with Burgers map and $CDABC_T$ with Tinkerbell map.

5.1. Chaotic pseudo-random number generator

There are two options how to implement the CPRNG. Either the large sequence of numbers – the inception of chaotic map – is stored in a file, and used iteratively. This approach, however, consumes large amount of memory. The second option is to use random starting point on the map, and a mathematical equation to get the next points. This concept is similar to PRNG seed. In this way, it is no longer necessary to store and read large data. For CDABC variants, the latter approach was used. The pseudocode describing this concept is given in Algorithm 4. Values on the X-axis are used as the output sequence, whereas values on the Y-axis are merely internal memory of the map.

The DABC algorithm structure then remains almost the same, the only alteration lies in replacement of calls to get next pseudo-random number from PRNG for calls to CPRNG.

6. Problem formulation

6.1. Quadratic assignment problem

The quadratic assignment problem (QAP) is a combinatorial optimisation problem stated for the first time by [61] and is widely regarded as one of the most difficult problems in this class. The objective is to assign $\boldsymbol{n}$ facilities to $\boldsymbol{n}$ locations in such a way as to minimise the assignment cost.

The assignment cost is the sum, over all pairs, of the flow between a pair of facilities multiplied by the distance between their assigned locations.

Let C and D be two $n \times n$ matrices such that $C=[c_{i,j}]$ and $D=[d_{i,j}]$. Consider the set of positive integers $\{1,2,\ldots,n\}$, and let S_n be the set of permutations of $\{1,2,\ldots,n\}$. Then the quadratic assignment problem can be defined as follows:

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{i,j} d_{\pi(i)\pi(j)} \tag{15}$$

over all permutations $\pi \in S_n$. The above formulation is known as the Koopmans–Beckman QAP [61].

Stated in other words, the objective of the quadratic assignment problem with cost matrix C and distance matrix D is to find the permutation $\pi_0 \in S_n$ that minimises the double summation over all i, j.

It should be understood that the notation $d_{\pi(i)\pi(j)}$ as used above refers to permuting the rows and columns of the matrix D by some permutation π.

Table 3
Average results for the QAP problem instances.

Instance	$DABC_{MT}$			$CDABC_{B}$			$CDABC_{DL}$			$CDABC_{L}$			$CDABC_{T}$		
	Average	Std	Time	Average	Std	Time	Average	Std	Time	Average	Std	Time	Average	Std	Time
bur26a	5 435 351.47	3.01E+03	0.518	5 434 405.067	1.27E+03	1.242	5 433 440.13	1.92E+03	1.447	5 434 966.73	2.02E+03	0.775	**5 433 251.93**	2.79E+03	1.696
bur26b	3 825 372.67	2.88E+03	0.503	3 821 792.4	2.69E+03	1.247	**3 819 926.40**	1.94E+03	1.438	3 822 431.60	3.10E+03	0.779	3 820 955.67	2.75E+03	1.695
bur26c	5 432 327.67	2.42E+03	0.509	5 429 092.933	1.12E+03	1.247	5 428 807.27	7.51E+02	1.461	5 430 760.73	1.35E+03	0.786	**5 428 716.93**	1.18E+03	1.729
bur26d	3 824 011.60	1.44E+03	0.537	3 822 009.867	3.68E+02	1.242	3 821 930.33	3.38E+02	1.459	3 822 180.13	5.58E+02	0.768	**3 821 924.87**	2.20E+02	1.724
bur26e	5 390 063.80	1.79E+03	0.497	**5 388 476.267**	6.26E+02	1.263	5 388 615.20	3.38E+02	1.439	5 389 240.40	1.05E+03	0.786	5 388 511.40	4.41E+02	1.723
bur26f	3 783 475.40	4.29E+02	0.511	3 782 924.733	4.14E+02	1.264	3 782 829.73	2.10E+02	1.437	3 783 184.20	3.93E+02	0.773	**3 782 653.47**	3.19E+02	1.702
bur26g	10 125 473.87	4.05E+03	0.505	10 121 200	2.12E+03	1.238	10 120 160.53	1.22E+03	1.457	10 122 399.40	1.80E+03	0.775	**10 120 082.33**	1.30E+03	1.716
bur26h	7 101 955.53	3.07E+03	0.506	7 100 231.4	8.52E+02	1.247	7 100 119.73	5.85E+02	1.504	7 100 971.07	1.01E+03	0.771	**7 099 876.00**	5.42E+02	1.698
kra30a	94 978.00	5.93E+02	0.669	93 702.66667	5.77E+02	1.628	**93 458.67**	6.59E+02	1.902	94 444.67	6.42E+02	1.021	93 534.67	5.03E+02	2.260
kra30b	95 906.67	7.82E+02	0.679	94 616	6.05E+02	1.611	94 158.67	6.26E+02	1.906	94 699.33	8.57E+02	1.029	**94 076.67**	7.62E+02	2.262
chr25a	5576.80	3.76E+02	0.471	5388	3.31E+02	1.160	5314.93	2.01E+02	1.332	5453.60	2.54E+02	0.718	**5341.20**	1.72E+02	1.572
tai20b	123 609 163.20	4.63E+05	0.313	123 131 974.4	2.93E+05	0.769	123 150 328.00	3.84E+05	0.888	123 425 646.40	4.74E+05	0.470	**122 822 634.67**	2.96E+05	1.044
tai25b	351 931 195.73	2.73E+06	0.474	348 934 201.6	1.60E+06	1.145	347 994 845.87	8.41E+05	1.338	349 421 038.93	2.08E+06	0.729	**347 097 013.33**	1.49E+06	1.569
tai30b	651 098 227.20	6.43E+06	0.677	644 277 290.7	3.74E+06	1.653	643 316 868.27	3.07E+06	1.914	646 544 123.73	3.29E+06	1.040	**642 278 387.20**	1.38E+06	2.221
tai35b	292 725 510.40	1.61E+06	0.974	288 727 129.6	1.30E+06	2.338	288 440 763.73	9.26E+05	2.743	290 941 043.20	2.00E+06	1.481	**287 770 026.67**	1.00E+06	3.155
tai40b	667 226 658.13	1.04E+07	1.241	660 323 575.5	1.32E+07	2.945	**655 818 901.33**	1.10E+07	3.551	664 875 080.53	1.14E+07	1.875	656 662 830.93	1.01E+07	4.124
tai50b	482 051 266.13	3.00E+06	1.877	474 744 234.7	3.95E+06	4.462	473 119 225.60	4.03E+06	5.393	479 978 762.67	3.92E+06	2.917	**471 260 149.33**	4.04E+06	6.209
Average	153 750 383.19	1.44E+06	**0.67**	152 072 485	1.42E+06	1.629	151 584 099.67	1.19E+06	1.918	152 958 025.14	1.36E+06	1.028	**151 351 762.78**	**1.08E+06**	2.241

That is, $D^{\pi} = [d^{\pi}_{i,j}] = d_{\pi(i)\pi(j)}$, for $1 \le i, j \le n$. In the same manner, given an n-dimensional vector $V = [v_i]$, a permutation of the elements of V by a permutation π will be denoted as $V^{\pi} = [v^{\pi}_i] = v_{\pi(i)}$.

A number of heuristics have been developed to handle large scale QAP problems; some notable ones being simulated annealing [62], tabu search [63] and the hybrid genetic-tabu search [64].

6.2. Capacitated vehicle routing problem

The vehicle routing problem is a well known problem in the field of transportation [65–70]. The basics of capacitated vehicle routing problem can be stated as follows [71]. Each vehicle has the same loading capacity, and starts off from only one delivery depot and then routes through customers. All customers have known demands and required service time. Each customer can only be visited by one vehicle, and each vehicle has to return to the depot. The service time unit can be transformed into the distance unit. The loading and the traveling distance of each vehicle cannot exceed the loading capacity and the maximum traveling distance of vehicle, respectively. The objective of CVRP is to minimize the traveling cost. The capacitated vehicle routing problem can be modeled as a mixed integer programming as follows:

$$\min \sum_{i=0}^{N} \sum_{j=0}^{N} \sum_{K=1}^{K} C_{ij} X_{ij}^{k} \tag{16}$$

Subject to

$$\sum_{i=0}^{N} \sum_{j=0}^{N} X_{ij}^{k} d_i \le Q^{k}, \quad 1 \le k \le K, \tag{17}$$

$$\sum_{i=0}^{N} \sum_{j=0}^{N} X_{ij}^{k} (C_{ij} + S_i) \le T^{k}, \quad 1 \le k \le K, \tag{18}$$

$$\sum_{j=1}^{N} X_{ijk} = \sum_{j=1}^{N} X_{jik} \le 1 \quad \text{for } i = 0 \text{ and } k \in \{1, \ldots, k\}, \tag{19}$$

$$\sum_{k=1}^{K} \sum_{j=1}^{N} X_{ijk} \le K \quad \text{for } i = 0, \tag{20}$$

where C_{ij} is the cost incurred on customer i to customer j, K the number of vehicles, N the number of customers, S_i the service time at customer i, Q^k the loading capacity of vehicle k, T^k the maximal traveling (route) distance of vehicle k, d_i the demand at customer i, $X_{ij}^k \in 0$ and 1 $(i \ne j; i, j \in 0, 1, \ldots, N)$.

Eq. (16) is the objective function of the problem. Eq. (17) is the constraint of loading capacity, where $X_{ij}^K = 1$ if vehicle k travels from customer i to customer j directly, and 0 otherwise. Eq. (18) is the constraint of maximum traveling distance. Eq. (19) makes sure that every route starts and ends at the delivery depot. Eq. (20) specifies that there are maximum K routes going out of the delivery depot.

7. Experimentation and analysis

Two separate and unique experimentations were conducted to validate the CDABC approach. The first experimentation was performed on the QAP problem and the second experiment on the CVRP problem. The data set was obtained from the OR Library [72].

The experimentation were conducted on the Tesla server with the following specifications: Intel Xeon, CPU E5-2640 v2 running at 2 GHz with 8 cores and 2 GB of cache, hosted at the Media Research Lab [73].

The operating CDABC parameters are given in Table 2. All the parameters were kept fixed for all the different simulations in order to not introduce any bias into the experimentations.

7.1. Quadratic assignment problem

The first experiment was on the QAP problem set. Seventeen different problem instances were selected, with different flow dominance. For each problem instance, 15 experiments were conducted. Therefore, 255 experiments were conducted for each CDABC variant, leading to a total of 1275 experiments. Three different statistical measures of importance the cumulative average, the standard deviation of the results and the average time were calculated for each problem instance. These measures for the DABC with Mersenne Twister ($DABC_{MT}$), $CDABC_B$, $CDABC_{DL}$, $CDABC_L$ and $CDABC_T$ are given in Table 3.

According to the results, the $CDABC_T$ is the best performing variant of the CDABC algorithm. Of the 17 instances, it has the better average for 13 instances. These are spread over the different problem instances of `bur`, `kra`, `tai` and `chr` instance types in the dataset. $CDABC_B$ map has one better average for the `bur26e` instance, while $CDABC_{DL}$ has three better average instances of `kra30a`, `bur26b` and `tai40b`.

In terms of cumulative average values of all the 17 instances, $CDABC_T$ has better cumulative average of 151 351 762.78 and the smallest cumulative standard deviation of 1.08E+06. In terms of execution time, $DABC_{MT}$ has the lowest execution time of 0.67 s compared to 2.241 s for the $CDABC_T$.

Paired *t*-test analysis was conducted on the raw data for the different variants in order to analyse if there is any significant difference between them. All 255 instances for the different variants were pairwise compared and the respective *t*-value and *p*-value was calculated. The *t*-test results are given in Table 4.

The variants were tested at 95% confidence level, and from the results only the $CDABC_B$, $CDABC_{DL}$ and $CDABC_{DL}$, $CDABC_T$ variants are not significantly different. All the other variants are significantly different pairwise.

Based on these results, we can confidently state that chaos maps significantly improve on the Mersenne Twister PRNG in DABC algorithm. In terms of hierarchy obtained from the cumulative average values, the order can be given as $CDABC_T$, $CDABC_{DL}$, $CDABC_B$, $CDABC_L$ and $DABC_{MT}$.

7.2. Capacitated vehicle routing problem

The second experiment was conducted on the CVRP problem. Twelve different instances were obtained from [74], with the range from 75 to 385 customers. Fifteen experiments were conducted on each instance, each variant having 180 simulations, leading to a total of 900 experiments.

The same three attributes average, standard deviation and execution time were obtained for each problem instance. The tabulated results are given in Table 5. For the CVPR problem, $CDABC_T$ is the best performing variant for all the problem instances, $CDABC_B$ the second best performing.

In terms of cumulative average values, $CDABC_T$ has the best cumulative average value of 6826.437 and standard deviation of 112.973. $DABC_{MT}$ once again has the lowest execution time of 1.1195 s.

Paired *t*-test was again applied to the different algorithms in order to ascertain if there was significant difference between each two of the variants. The *t*-test results are given in Table 6. At 95% confidence level, only the $CDABC_B$, $CDABC_{DL}$ maps are not significantly different, whereas all the other variants are significantly different pairwise. Through the analysis of the cumulative values in Table 5, we can state that the order of best performing variants is $CDABC_T$, $CDABC_B$, $CDABC_{DL}$, $CDABC_L$ and $DABC_{MT}$.

8. Conclusion

This research introduces a chaos driven discrete artificial bee colony algorithm. The main premise is to gauge the effectiveness of using different chaos maps instead of Mersenne Twister as the PRNG in the DABC algorithm. Two different and difficult optimisation problems the quadratic assignment problem and the capacitated vehicle routing problem are solved using four different chaos maps: Burgers ($CDABC_B$), Delayed Logistic $CDABC_{DL}$, Lozi $CDABC_L$ and Tinkerbell $CDABC_T$, in addition to the Mersenne Twister $DABC_{MT}$.

For the QAP problem, 17 different instances were solved with 15 experiments for each instance. This leads to 225 experimentations for each variant and a total of 1275 experimentations. Three different attributes average, standard deviation and execution time are measured for each experiment. From the results, the Tinkerbell map is the best performing algorithm for the QAP problem. Paired *t*-test at 95% confidence level was conducted pairwise on the different variants, and apart from $CDABC_B$, $CDABC_{DL}$ and $CDABC_{DL}$, $CDABC_T$, all the variants are significantly different from each other. $CDABC_T$ variant of DABC is the best performing, followed by $CDABC_DL$, $CDABC_B$, $CDABC_L$ and $DABC_{MT}$.

The CVPR problem was likewise solved with the five different variants. In this case, 12 instances were evaluated, with 15 experiments for each instance. A total of 180 simulations was done for each variant, with the cumulative total of 900 experiments. From the obtained results, the $CDABC_T$ variant was the best performing for all the different instances in terms of the average and standard deviation values. The *t*-test results showed that only the $CDABC_B$ and $CDABC_{DL}$ variants were not significantly different. From the cumulative average values, the $CDABC_T$ is the best performing followed by the $CDABC_B$, $CDABC_{DL}$, $CDABC_L$ and $DABC_{MT}$.

In both problem classes, the $CDABC_T$ variant is far superior to the $DABC_{MT}$ variant which is the worst performing of the five tested variants. This research lends weight to the argument that chaos driven algorithms are significantly better performing compared to the canonical variants. Also, the concepts of PRNG's, random seeds and data periods are called into question, especially for EA's. Additionally, the concept of population propagation,

Table 4
T-test results for the QAP problem instances.

	$DABC_{MT}$		$CDABC_B$		$CDABC_{DL}$		$CDABC_L$	
	t-value	*p*-value	*t*-value	*p*-value	*t*-value	*p*-value	*t*-value	*p*-value
$CDABC_B$	5.432	**1.298**e^{-7}						
$CDABC_{DL}$	5.6864	**3.555**e^{-8}	1.667	0.0967				
$CDABC_L$	2.711	**0.0072**	2.903	**0.0040**	4.4266	**1.0**e^{-5}		
$CDABC_T$	7.298	**3.727**e^{-12}	2.4407	**0.0153**	0.994	0.3207	4.6278	**5.8973**e^{-6}

Table 5
Average results for the CVRP problem instances.

Instance	$DABC_{MT}$			$CDABC_{B}$			$CDABC_{DL}$			$CDABC_{L}$			$CDABC_{T}$		
	Average	Std	Time	Average	Std	Time	Average	Std	Time	Average	Std	Time	Average	Std	Time
tai75a	2342.845	49.762	0.704	2141.405	41.472	1.680	2114.515	53.104	2.037	2222.681	51.401	1.092	**2096.185**	50.166	2.401
tai75b	1869.640	62.235	0.715	1745.197	46.052	1.685	1714.855	39.542	2.037	1816.956	33.931	1.089	**1678.957**	28.920	2.385
tai75c	1938.842	44.641	0.711	1761.108	37.366	1.652	1748.206	41.322	2.049	1843.879	51.041	1.063	**1716.844**	31.109	2.417
tai75d	2109.960	48.108	0.700	1841.167	46.703	1.683	1826.861	49.557	1.995	1983.603	69.140	1.063	**1784.023**	32.802	2.411
tai100a	3516.282	105.034	0.916	3162.238	86.045	2.154	3165.510	80.824	2.607	3349.617	92.724	1.451	**3060.056**	61.203	3.057
tai100b	3408.972	98.248	0.917	3033.001	74.036	2.156	3019.551	86.953	2.623	3250.870	60.472	1.391	**2935.121**	73.899	3.063
tai100c	2498.816	70.574	0.908	2186.654	55.190	2.159	2150.959	47.176	2.619	2380.911	68.574	1.393	**2055.812**	76.071	3.081
tai100d	2729.089	87.405	0.939	2498.172	39.744	2.159	2440.982	69.503	2.679	2616.968	63.197	1.415	**2402.249**	34.891	3.139
tai150a	6450.524	148.854	1.373	5496.065	197.539	3.126	5536.911	160.523	3.835	6097.841	157.913	2.042	**5346.892**	139.102	4.488
tai150b	6125.898	174.801	1.316	5150.960	148.461	3.129	5082.495	187.743	3.809	5756.215	135.314	2.051	**4974.448**	100.401	4.450
tai150c	5302.500	213.994	1.343	4487.373	126.304	3.147	4340.375	139.333	3.832	5012.526	124.796	2.055	**4093.157**	131.397	4.491
tai150d	5769.406	142.592	1.330	4970.953	75.503	3.143	4957.906	131.110	3.817	5405.630	125.025	2.089	**4780.173**	114.187	4.455
tai385	64,679.206	1026.345	3.664	52,628.353	1314.226	8.732	53,691.375	1003.324	10.487	60089.740	1243.935	5.660	**51819.760**	594.504	12.181
Average	8364.768	174.815	**1.195**	7007.896	176.049	2.816	7060.808	160.770	3.417	7832.880	175.189	1.835	**6826.437**	**112.973**	4.001

Table 6
T-test results for the CVRP problem instances.

	$DABC_{MT}$		$CDABC_{B}$		$CDABC_{DL}$		$CDABC_{L}$	
	t-value	*p*-value	*t*-value	*p*-value	*t*-value	*p*-value	*t*-value	*p*-value
$CDABC_{B}$	6.023	$\mathbf{8.403}e^{-9}$						
CDABCDL	6.399	$\mathbf{1.145}e^{-9}$	1.4114	0.159				
CDABCL	6.043	$\mathbf{7.603}e^{-9}$	5.759	$\mathbf{3.268}e^{-8}$	6.408	$\mathbf{1.09}e^{-9}$		
CDABCT	6.476	$\mathbf{7.515}e^{-10}$	5.63614	$\mathbf{6.061}e^{-8}$	5.604	$\mathbf{7.112}e^{-8}$	6.523	$\mathbf{5.858}e^{-10}$

relying on random mutation, can also be improved by chaotic mutation.

In summation, we can confidently state that chaos based evolutionary algorithms, in this case the discrete artificial bee algorithm, are significantly better than those using stock pseudo-random number generators. The one fallibility is the execution time of the chaos systems. This is due to the high level of optimisation of the Mersenne Twister. It can be resolved in the future by using GPU based implementation of these algorithms.

Acknowledgements

This work was principally supported by the Grant of SGS SP2015/141 and Grant Agency of the Czech Republic – GAČR P103/15/06700S.

References

[1] M. Matsumoto, T. Nishimura, Mersenne twister: a 623-dimensionally equidistributed uniform pseudorandom number generator, ACM Trans. Model. Comput. Simul. 8 (1) (1998) 3–30.
[2] M. Matsumoto, Mersenne Twister Webpage, ⟨http://www.math.sci.hiroshima-u.ac.jp⟩, 2012.
[3] C. Herring, P. Julian, Random number generators are chaotic, ACM SIGPLAN 11 (1989) 1–4.
[4] D. Lehmer, Mathematical methods in large-scale computing units, Ann. Comput. Lab Harv. Univ. 26 (1951) 141–146.
[5] J. Palmore, J. McCauley, Shadowing by computable chaotic orbits, Phys. Lett. A 121 (1987) 399.
[6] I. Zelinka, M. Chadli, D. Davendra, R. Senkerik, M. Pluhacek, J. Lampinen, Hidden periodicity—chaos dependance on numerical precision, in: Advances in Intelligent Systems and Computing, vol. 210, 2013, pp. 47–59.
[7] T. Stojanovski, L. Kocarev, Chaos-based random number generators Part I: analysis, IEEE Trans. Circuits Syst. I: Fund. Theory Appl. 48 (3) (2001) 281–288.
[8] R. Lozi, New enhanced chaotic number generators, Indian J. Ind. Appl. Math. 1 (1) (2008) 1–23.
[9] R. Lozi, 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, 2009, pp. 1–5.
[10] I. Zelinka, M. Chadli, D. Davendra, R. Senkerik, M. Pluhacek, J. Lampinen, Do evolutionary algorithms indeed require random numbers? Extended study, in: Advances in Intelligent Systems and Computing, vol. 210, 2013, pp. 61–75.
[11] A.B. Ozer, Cide: chaotically initialized differential evolution, Expert Syst. Appl. 37 (6) (2010) 4632–4641.
[12] R.-Q. Wang, C.-H. Zhang, K. Li, Multi-objective genetic algorithm based on improved chaotic optimization, Kongzhi yu Juece/Control Decis. 26 (9) (2011) 1391–1397.
[13] H. Lu, R. Niu, J. Liu, Z. Zhu, A chaotic non-dominated sorting genetic algorithm for the multi-objective automatic test task scheduling problem, Appl. Soft Comput. J. 13 (5) (2013) 2790–2802.
[14] W.-C. Hong, Y. Dong, L.-Y. Chen, S.-Y. Wei, Svr with hybrid chaotic genetic algorithms for tourism demand forecasting, Appl. Soft Comput. J. 11 (2) (2011) 1881–1890.
[15] A. Kazem, E. Sharifi, F. Hussain, M. Saberi, O. Hussain, Support vector regression with chaos-based firefly algorithm for stock market price forecasting, Appl. Soft Comput. J. 13 (2) (2013) 947–958.
[16] W. Liang, L. Zhang, M. Wang, The chaos differential evolution optimization algorithm and its application to support vector regression machine, J. Softw. 6 (7) (2011) 1297–1304.
[17] C. Peng, H. Sun, J. Guo, G. Liu, Dynamic economic dispatch for wind-thermal power system using a novel bi-population chaotic differential evolution algorithm, Int. J. Electr. Power Energy Syst. 42 (1) (2012) 119–126.
[18] L. Coelho, T. Bora, L. Lebensztajn, A chaotic approach of differential evolution optimization applied to loudspeaker design problem, IEEE Trans. Magn. 48 (2) (2012) 751–754.
[19] R. Senkerik, D. Davendra, I. Zelinka, Z. Oplatkova, M. Pluhacek, Optimization of the batch reactor by means of chaos driven differential evolution, in: Advances in Intelligent Systems and Computing (AISC), vol. 188, 2013, pp. 93–102.
[20] X. Yuan, B. Cao, B. Yang, Y. Yuan, Hydrothermal scheduling using chaotic hybrid differential evolution, Energy Convers. Manag. 49 (12) (2008) 3627–3633.
[21] D. Davendra, I. Zelinka, R. Senkerik, Chaos driven evolutionary algorithms for the task of pid control, Comput. Math. Appl. 60 (4) (2010) 1088–1104.
[22] X.-B. Xu, K.-F. Zheng, D. Li, B. Wu, Y.-X. Yang, New chaos-particle swarm optimization algorithm, Tongxin Xuebao/J. Commun. 33 (1) (2012) 24–30+37.
[23] M. Pluhacek, R. Senkerik, I. Zelinka, D. Davendra, Chaos pso algorithm driven alternately by two different chaotic maps-an initial study, in: IEEE Congress on Evolutionary Computation, CEC 2013, 2013, pp. 2444–2449.
[24] R. Senkerik, M. Pluhacek, D. Davendra, I. Zelinka, Z. Kominkova Oplatkova, Chaos driven evolutionary algorithm: a new approach for evolutionary optimization, Int. J. Math. Comput. Simul. 7 (4) (2013) 363–368.
[25] M. Pluhacek, R. Senkerik, D. Davendra, Z. Kominkova Oplatkova, I. Zelinka, On the behavior and performance of chaos driven pso algorithm with inertia weight, Comput. Math. Appl. 66 (2) (2013) 122–134.
[26] B. Alatas, E. Akin, A. Ozer, Chaos embedded particle swarm optimization algorithms, Chaos Solitons Fractals 40 (4) (2009) 1715–1734.
[27] A. Gandomi, G. Yun, X.-S. Yang, S. Talatahari, Chaos-enhanced accelerated particle swarm optimization, Commun. Nonlinear Sci. Numer. Simul. 18 (2) (2013) 327–340.
[28] L.-Y. Chuang, S.-W. Tsai, C.-H. Yang, Chaotic catfish particle swarm optimization for solving global numerical optimization problems, Appl. Math. Comput. 217 (16) (2011) 6900–6916.
[29] W.-B. Wang, Q.-Y. Feng, D. Liu, Application of chaotic particle swarm optimization algorithm to pattern synthesis of antenna arrays, Progr. Electromagn. Res. 115 (2011) 173–189.
[30] F. Liu, H. Duan, Y. Deng, A chaotic quantum-behaved particle swarm optimization based on lateral inhibition for image matching, Optik 123 (21) (2012) 1955–1960.
[31] M. Eslami, H. Shareef, A. Mohamed, Power system stabilizer design using hybrid multi-objective particle swarm optimization with chaos, J. Cent. South Univ. Technol. (English Edition) 18 (5) (2011) 1579–1588.
[32] H. Jiang, C. Kwong, Z. Chen, Y. Ysim, Chaos particle swarm optimization and t-s fuzzy modeling approaches to constrained predictive control, Expert Syst. Appl. 39 (1) (2012) 194–201.
[33] V. Mariani, A. Duck, F. Guerra, L. Coelho, R. Rao, A chaotic quantum-behaved particle swarm approach applied to optimization of heat exchangers, Appl. Therm. Eng. 42 (2012) 119–128.
[34] J. Li, L. Yang, J.-L. Liu, D.-L. Yang, C. Zhang, Multi-objective reactive power optimization based on adaptive chaos particle swarm optimization algorithm, Dianli Xitong Baohu yu Kongzhi/Power Syst. Prot. Control 39 (9) (2011) 26–31.
[35] M. Zhang, G. Li, Network intrusion detection based on least squares support vector machine and chaos particle swarm optimization algorithm, J. Converg. Inf. Technol. 7 (4) (2012) 169–174.
[36] M. Pluhacek, R. Senkerik, D. Davendra, I. Zelinka, Pid controller design for 4th order system by means of enhanced pso algorithm with Lozi chaotic map, in: Proceedings of the 18th International Conference on Soft Computing, MENDEL, 2012, pp. 35–39.
[37] Q. Wu, A self-adaptive embedded chaotic particle swarm optimization for parameters selection of wv-svm, Expert Syst. Appl. 38 (1) (2011) 184–192.
[38] D. Davendra, R. Senkerik, I. Zelinka, M. Pluhacek, M. Bialic-Davendra, Utilising the chaos-induced discrete self organising migrating algorithm to solve the lot-streaming flowshop scheduling problem with setup time, Soft Comput. 18 (4) (2014) 669–681. http://dx.doi.org/10.1007/s00500-014-1219-7.
[39] Y. Yang, Y. Wang, X. Yuan, F. Yin, Hybrid chaos optimization algorithm with artificial emotion, Appl. Math. Comput. 218 (11) (2012) 6585–6611.
[40] D. Karaboga, B. Basturk, A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm, J. Glob. Optim. 39 (3) (2007) 459–471.
[41] M.F. Tasgetiren, Q.-K. Pan, P.N. Suganthan, A.H.-L. Chen, A discrete artificial bee colony algorithm for the total flowtime minimization in permutation flow shops, Inf. Sci. 181 (16) (2011) 3459–3475.
[42] D. Karaboga, B. Basturk, Artificial bee colony (abc) optimization algorithm for solving constrained optimization problems, in: P. Melin, O. Castillo, L. Aguilar, J. Kacprzyk, W. Pedrycz (Eds.), Foundations of Fuzzy Logic and Soft Computing, Lecture Notes in Computer Science, vol. 4529, Springer, Berlin, Heidelberg, 2007, pp. 789–798.
[43] J.-Q. Li, Q.-K. Pan, K.-Z. Gao, Pareto-based discrete artificial bee colony algorithm for multi-objective flexible job shop scheduling problems, Int. J. Adv. Manuf. Technol. 55 (9–12) (2011) 1159–1169.
[44] Y.-Y. Han, J.J. Liang, Q.-K. Pan, J.-Q. Li, H.-Y. Sang, N.N. Cao, Effective hybrid discrete artificial bee colony algorithms for the total flowtime minimization in the blocking flowshop problem, Int. J. Adv. Manuf. Technol. 67 (1–4, SI) (2013) 397–414.
[45] S. Hongyan, G. Liang, P. Quanke, Discrete artificial bee colony algorithm for lot-streaming flowshop with total flowtime minimization, Chin. J. Mech. Eng. 25 (5) (2012) 990–1000.
[46] Q.-K. Pan, M.F. Tasgetiren, P.N. Suganthan, T.J. Chua, A discrete artificial bee colony algorithm for the lot-streaming flow shop scheduling problem, Inf. Sci. 181 (12) (2011) 2455–2468.
[47] M. Dorigo, M. Birattari, T. Stutzle, Ant colony optimization, Comput. Intell. Mag. IEEE 1 (4) (2006) 28–39.
[48] J. Kennedy, R. Eberhart, Particle swarm optimization, in: 1995 IEEE International Conference on Neural Networks Proceedings, vols. 1–6, 1995, pp. 1942–1948.
[49] I. Zelinka, Soma – self-organizing migrating algorithm, in: New Optimization Techniques in Engineering, Studies in Fuzziness and Soft Computing, vol. 141, Springer, Berlin, Heidelberg, 2004, pp. 167–217.
[50] D. Davendra, I. Zelinka, M. Bialic-Davendra, R. Senkerik, R. Jasek, Discrete self-organising migrating algorithm for flow-shop scheduling with no-wait makespan, Math. Comput. Model. 57 (1–2) (2013) 100–110.
[51] A. Bagheri, M. Zandieh, I. Mahdavi, M. Yazdani, An artificial immune algorithm for the flexible job-shop scheduling problem, Future Gener. Comput. Syst.—The International Journal of Grid Computing and eScience 26 (4) (2010) 533–541.

[52] D. Karaboga, B. Basturk, On the performance of artificial bee colony ABC algorithm, Appl. Soft Comput. 8 (1) (2008) 687–697. http://dx.doi.org/10.1016/j.asoc.2007.05.007.
[53] G. Onwubolu, D. Davendra, Differential evolution for permutation—based combinatorial problems, in: G. Onwubolu, D. Davendra (Eds.), Differential Evolution: A Handbook for Global Permutation-Based Combinatorial Optimization, Studies in Computational Intelligence, vol. 175, Springer, Berlin, Heidelberg, 2009, pp. 13–34.
[54] J. Sprott, Chaos and Time-Series Analysis, Oxford University Press, UK, 2001.
[55] R. Senkerik, I. Zelinka, M. Pluhacek, D. Davendra, Z.O. Kominkova, Chaos enhanced differential evolution in the task of evolutionary control of selected set of discrete chaotic systems, Sci. World J., (2014). http://dx.doi.org/10.1155/2014/836484.
[56] D. Aronson, M. Chory, G. Hall, R. McGehee, A discrete dynamical system with subtly wild behavior, in: D. Davendra (Ed.), New Approaches to Nonlinear Problems in Dynamics, SIAM Publications, Philadelphia, Pennsylvania, 1980, pp. 339–359.
[57] J. Burgers, Mathematical examples illustrating relations occurring in the theory of turbulent fluid motion, in: F. Nieuwstadt, J. Steketee (Eds.), Selected Papers of J. M. Burgers, Springer, Netherlands, 1995, pp. 281–334.
[58] R. Whitehead, N. MacDonald, A chaotic mapping that displays its own homoclinic structure, Physica D: Nonlinear Phenom. 13 (3) (1984) 401–407.
[59] K. Alligood, T. Sauer, J. Yorke, Chaos, Springer, Germany, 1997.
[60] M. Metlick, D. Davendra, Scheduling the flowshop with zero intermediate storage using chaotic discrete artificial bee algorithm, in: I. Zelinka, P.N. Suganthan, G. Chen, V. Snasel, A. Abraham, O. Rssler (Eds.), Nostradamus 2014: Prediction, Modeling and Analysis of Complex Systems, Advances in Intelligent Systems and Computing, vol. 289, Springer International Publishing, 2014, pp. 141–152. http://dx.doi.org/10.1007/978-3-319-07401-6_14.
[61] T. Koopans, M. Beckman, Assignment problems and the location of economic activities, Econometrica 25 (1957) 53–76.
[62] D. Connolly, An improved annealing scheme for the qap, Eur. J. Oper. Res. 46 (1990) 93–100.
[63] E. Taillard, Robust taboo search for the quadratic assignment problem, Parallel Comput. 17 (1991) 443–455.
[64] C. Fleurent, J. Ferland, Genetic hybrids for the quadratic assignment problem, Oper. Res. Q. 28 (1994) 167–179.
[65] G.B. Dantzig, J.H. Ramser, The truck dispatching problem, Manag. Sci. 6 (1) (1959) 80–91.
[66] G. Laporte, The vehicle routing problem: an overview of exact and approximate algorithms, Eur. J. Oper. Res. 59 (3) (1992) 345–358.
[67] P. Toth, D. Vigo, The vehicle routing problem, Society for Industrial and Applied Mathematics, Philadelphia, USA, 2001.
[68] R. Baldacci, N. Christofides, A. Mingozzi, An exact algorithm for the vehicle routing problem based on the set partitioning formulation with additional cuts, Math. Program. 115 (2) (2008) 351–385.
[69] B.L. Golden, S. Raghavan, E.A. Wasil, The Vehicle Routing Problem: Latest Advances and New Challenges, Society for Industrial and Applied Mathematics, Springer, Germany, 2008.
[70] R. Baldacci, P. Toth, D. Vigo, Exact algorithms for routing problems under vehicle capacity constraints, Ann. Oper. Res. 175 (1) (2010) 213–245.
[71] S.-W. Lin, Z.-J. Lee, K.-C. Ying, C.-Y. Lee, Applying hybrid meta-heuristics for capacitated vehicle routing problem, Expert Syst. Appl. 36 (2, Part1) (2009) 1505–1512 http://dx.doi.org/10.1016/j.eswa.2007.11.060, URL ⟨http://www.sciencedirect.com/science/article/pii/S095741740700588X⟩.
[72] J.E. Beasley, Operations research library, ⟨http://www.brunel.ac.uk/~mastjjb/jeb/info.html⟩, 2014.
[73] M.R. Lab, Media Research Lab, ⟨http://mrl.cs.vsb.cz⟩, 2014.
[74] E. Taillard, Vehicle Routing Problem Instances, ⟨http://mistic.heig-vd.ch/taillard/⟩, 2012.