SCHEDULING THE FLOW SHOP WITH BLOCKING PROBLEM WITH THE CHAOS-INDUCED DISCRETE SELF ORGANISING MIGRATING ALGORITHM

Donald Davendra
Department of Computer Science
Faculty of Electrical Engineering and Computer Science
VSB-Technical University of Ostrava
17. listopadu 15, 708 33 Ostrava-Poruba
Czech Republic.
Email: donald.davendra@vsb.cz

Magdalena Bialic-Davendra,
Tomas Bata University in Zlin,
Faculty of Management and Economics,
Nam T.G. Masaryka 5555, 760 01 Zlin,
Czech Republic.
Email: bialic@fame.utb.cz

Roman Senkerik and Michal Pluhacek,
Tomas Bata University in Zlin,
Faculty of Applied Informatics,
Nam T.G. Masaryka 5555, 760 01 Zlin,
Czech Republic.
Email: {senkerik,pluhacek}@fai.utb.cz

KEYWORDS

Discrete Self Organising Migrating algorithm, flow shop with blocking, chaos Lozi map

ABSTRACT

The dissipative Lozi chaotic map is embedded in the Discrete Self Organising Migrating Algorithm (DSOMA) algorithm, as a pseudorandom number generator (PRNG). This novel chaotic based algorithm is applied to the flow shop with blocking scheduling problem. The algorithm is tested on the Taillard problem sets and compared favourably with published heuristics.

INTRODUCTION

One of the core premises of EA's is their reliance on *stochasticity*, the ability to generate a random event, which in turn, hopefully, provides the spark of perturbation towards the desired goal. The task of generating these 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 between a specified range.

A wide variety of such PRNG's exist, however the most common in usage is the *Mersenne Twister* (Matsumoto and Nishimura, 1998). A number of its variants has been designed; for a full listing please see Matsumoto (2012). Some other common PRNG's are the *Mother Of All*, *CryptoAPI*, *Indirection, Shift, Accumulate, Add, and Count* (ISAAC), *Keep it Simple Stupid* (KISS) and *Mutiply-With-Carry* (MWC).

The first aspect of the research work presents a novel approach to generating such pseudorandom numbers, one with a lineage in chaos theory. The term *chaos* describes the complex behaviour of simple, well behaved systems. When casually observed, this behaviour can seem erratic and somewhat random, however, these systems are deterministic, whose precise knowledge of future behaviour is well known. The question is then to reconcile the notion of nonlinearity of these systems.

Sudden and dramatic changes in some nonlinear systems may give rise to a complex behaviour called chaos. The noun *chaos* and the adjective *chaotic* are used to describe the time behaviour of a system when that behaviour is aperiodic (it *never* exactly repeats) and appears apparently random or noisy (Hilborn, 2000).

This aperiodic non-repeating behaviour of chaotic systems is the core foundation of this research. The objective is then to use a valid chaotic system, and embed it in the EA's as a PRNG. Four general branches of chaotic systems exist, which are the dissipative systems, fractals, dissipative and high-dimensional systems and conservative systems. The chaos system used for this research is the *discrete* dissipative system of the Lozi map. This system is relatively simple, in terms of period density, and therefore easier to obtain data through sectional cropping.

Many chaotic maps in the literature have shown to 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 improve the performance of EA's (Alatas et al. (2009)) and Caponetto et al. (2003)). They have also been used together with some heuristic optimisation algorithms (Davendra et al. (2010) and Zuo and Fan (2006)) to express optimisation variables. The choice of chaotic sequences is justified theoretically by their unpredictability, i.e., by their spread-spectrum characteristic, non-periodic, complex temporal behaviour, and ergodic properties (Ozer, 2010).

Davendra et al. (2010) has applied the canonical Dif-

Proceedings 27th European Conference on Modelling and Simulation ©ECMS Webjørn Rekdalsbakken, Robin T. Bye, Houxiang Zhang (Editors)
ISBN: 978-0-9564944-6-7 / ISBN: 978-0-9564944-7-4 (CD)

ferential Evolution (DE) to solve the PID optimisation problem, whereas Ozer (2010) applied a sequence of chaotic maps to optimise a range of benchmark problems, with the conclusion that the Sinus map and Circle map have somewhat increased the solution quality, with the ability to escape the local optima. The economic dispatch problem was solved by Lu et al. (2011), where the *Tent Map* was utilised as a chaotic local search in order to bypass the local optima. It should also be noted that Yuan et al. (2008) has developed a chaotic hybrid DE, where the parameter selection and operation is handled by chaotic sequences to solve combinatorial optimisation problem.

This research looks at modifying the DSOMA with the Lozi map as the RPNG and using it to solve the Flow shop with blocking problem. The paper is organised as follows; The first section introduces the Lozi map followed by the flow shop with blocking problem. Thereafter, SOMA and DSOMA algorithms are introduced. Subsequently the experimentation results are presented followed by the conclusion.

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. This system belongs to the *discrete dissipative* systems.

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 in equations (1) and (2).

$$X_{n+1} = 1 - a \cdot |X_n| + b \cdot Y_n \tag{1}$$

$$Y_{n+1} = X_n \tag{2}$$

The parameters used in this work are $a = 1.7$ and $b = 0.5$ as suggested in Caponetto et al. (2003) and the initial conditions are X_0 = -0.1 and Y_0 = 0.1. The Lozi map is given in Figure 1.

FLOW SHOP WITH BLOCKING

Consider *m* machines in series with *zero* intermediate storage between successive machines, which have to process *n* jobs. If a given machine finishes the processing of any given job, the job cannot proceed to the next machine while that machine is busy, but must remain on that machine, which therefore remains *idle*. This phenomenon is refereed to as *blocking* (Pinedo, 1995).

In this paper, only flow shops with zero intermediate storage are considered (FSSB), since any flow shop with positive (but finite) intermediate storage between machines can be modelled as a flow shop with zero intermediate storage. This is due to the fact that the storage space capable of containing one job may be regarded as

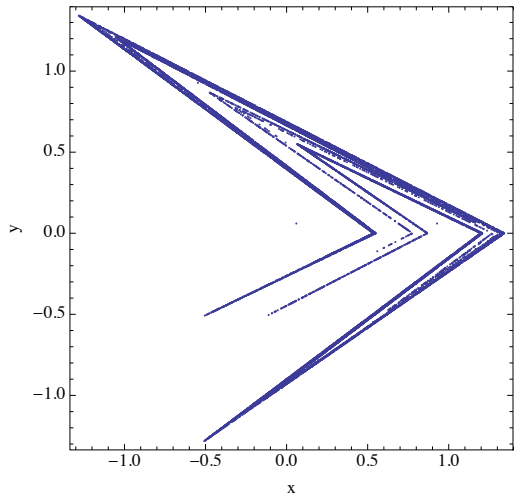

Figure 1: Lozi map

a machine on which the processing time of all machines is equal to zero.

Pinedo (1995) has defined the problem of minimising the makespan in a flow shop with zero intermediate storages is referred to in what follows as:

$$Fm\,|block\,|C_{\max}$$

Let $D_{i,j}$ denote the time that job *j* actually departs machine i. Clearly $D_{i,j} \geq C_{i,j}$. Equality holds that job *j* is not blocked. The time job *j* starts its processing at the first machine is denoted by $D_{0,j}$. The following recursive relationship hold under the job sequence $j_1, \ldots, j_n$:

$$D_{i,j_1} = \sum_{l=1}^{i} p_{l,j_1} \quad i = 1, \ldots, m \tag{3}$$

$$D_{i,j_k} = \max\left(D_{i-1,j_k} + p_{i,j_k}, D_{i+1,j_{k-1}}\right) \quad i = 2, \ldots, m \;\; k = 2, \ldots, n \tag{4}$$

$$D_{m,j_k} = D_{m-1,j_k} + p_{m,j_k} \tag{5}$$

DISCRETE SELF ORGANISING MIGRATING ALGORITHM

DSOMA (Davendra, 2009) is the discrete version of SOMA (Zelinka, 2004), developed to solve permutation based combinatorial optimisation problem. The same ideology of the sampling of the space between two individuals of SOMA is retained. Assume that there are two individuals in a search space, where the objective for DSOMA is to transverse from one individual to another, while mapping each discrete space between these two individuals.

The major input of this algorithm is the sampling of the jump sequence between the individuals in the populations, and the procedure of constructing new trial individuals from these sampled jump sequence elements. The overall outline for DSOMA can be given as:

Table 1: DSOMA parameters.

Name	Range	Type	Description
J_{min}	(1+)	Control	Min number of jumps
Population	10+	Control	Num. of individuals
Migrations	10+	Termination	Number of iterations

1. **Initial Phase**
   (a) *Population Generation*: An initial number of permutative trial individuals is generated for the initial population.
   (b) *Fitness Evaluation*: Each individual is evaluated for its fitness.
2. **DSOMA**
   (a) *Creating Jump Sequences*: Taking two individuals, a number of possible jump positions is calculated between each corresponding element.
   (b) *Constructing Trial Individuals*: Using the jump positions; a number of trial individuals is generated. Each element is selected from a jump element between the two individuals.
   (c) *Repairment*: The trial individuals are checked for feasibility and those, which contain an incomplete schedule, are repaired.
3. **Selection**
   (a) *New Individual Selection*: The new individuals are evaluated for their fitness and the best new fitness based individual replaces the old individual, if it improves upon its fitness.
4. **Generations**
   (a) *Iteration*: Iterate the population till a specified migration.

DSOMA requires a number of parameters as given in Table 1. The major addition is the parameter J_{min}, which gives the minimum number of jumps (sampling) between two individuals. The SOMA variables PathLength, StepSize and PRT Vector are not initialised as they are dynamically calculated by DSOMA using the adjacent elements between the individuals.

Initialisation
The population is initialised as a permutative schedule representative of the size of the problem at hand (6). As this is the initial population, the superscript of *x* its set to 0.

$$x_{i,j}^{0} = \begin{cases} 1 + INT\left(rand\left()\cdot(N-1)\right)\right. \\ \text{if } x_{i,j}^{0} \notin \left\{x_{i,1}^{0}, \ldots, x_{i,j-1}^{0}\right\} \\ i = 1, \ldots, \beta;\ j = 1, \ldots, N \end{cases} \tag{6}$$

Each individual is vetted for its fitness (7), and the best individual, whose index in the population can be assigned as *L* (leader) and it is designated the leader as X_L^0 with its best fitness given as C_L^0.

$$C_i^0 = \Im\left(X_i^0\right), \quad i = 1, \ldots, \beta \tag{7}$$

After the generation of the initial population, the migration counter *t* is set to 1 where $t = 1, \ldots, M$ and the individual index *i* is initialised to 1, where $i = 1, \ldots, \beta$. Using these values, the following sections are recursively applied.

Creating Jump Sequences
DSOMA operates by calculating the number of discrete jump steps that each individual has to circumnavigate. In DSOMA, the parameter of minimum jumps ($J_{\min}$) is used in lieu of PathLength, which states the minimum number of individuals or sampling between the two individuals.

Taking two individuals in the population, one as the incumbent (X_i^t) and the other as the leader (X_L^t), the possible number of jump individuals $J_{\max}$ is the mode of the difference between the adjacent values of the elements in the individual (8). A vector *J* of size *N* is created to store the difference between the adjacent elements in the individuals. The *mode ()* function obtains the most common number in *J* and designates it as J_{max}.

$$\begin{aligned} J_j &= \left| x_{i,j}^{t-1} - x_{L,j}^{t-1} \right|, \quad j = 1, \ldots, N \\ J_{\max} &= \begin{cases} mode\left(J\right) & if \ \ mode\left(J\right) > 0 \\ 1 & otherwise \end{cases} \end{aligned} \tag{8}$$

The step size (*s*), can now be calculated as the integer fraction between the required jumps and possible jumps (9).

$$s = \begin{cases} \left\lfloor \frac{J_{\max}}{J_{\min}} \right\rfloor & \text{if } \ J_{\max} \geq J_{\min} \\ 1 & \text{otherwise} \end{cases} \tag{9}$$

Create a jump matrix $\mathbf{G}$, which contains all the possible jump positions, that can be calculated as:

$$\mathbf{G}_{l,j} = \begin{cases} x_{i,j}^{t-1} + s \cdot l & \text{if } x_{i,j}^{t-1} + s \cdot l < x_{L,j}^{t-1} \\ & \quad \text{and } x_{i,j}^{t-1} < x_{L,j}^{t-1} \\ x_{i,j}^{t-1} - s \cdot l & \text{if } x_{i,j}^{t-1} + s \cdot l < x_{L,j}^{t-1} \\ & \quad \text{and } x_{i,j}^{t-1} > x_{L,j}^{t-1} \\ 0 & \text{otherwise} \end{cases}$$
$$j = 1, \ldots, N;\ l = 1, \ldots, J_{\min} \tag{10}$$

Constructing Trial Individuals
For each jump sequence of two individuals, a total of $J_{\min}$ new individuals can now be constructed from the jump positions. Taking a new temporary population *H* ($H = \{Y_1, \ldots, Y_{J_{\min}}\}$), in which each new individual Y_w ($w = 1, \ldots, J_{\min}$), is

constructed piecewise from $\mathbf{G}$. Each element $y_{w,j}$ ($Y_w = \{y_{w,j}, \ldots, y_{w,N}\},\ j = 1, 2, \ldots, N$) in the individual, indexes its values from the corresponding j^{th} column in $\mathbf{G}$. Each l^{th} ($l = 1, \ldots, J_{\min}$) position for a specific element is sequentially checked in $\mathbf{G}_{l,j}$ to ascertain if it already exists in the current individual Y_w. If this is a new element, it is then accepted in the individual, and the corresponding l^{th} value is set to zero as $\mathbf{G}_{l,j} = 0$. This iterative procedure can be given as in equation (11).

$$y_{w,j} = \begin{cases} \mathbf{G}_{l,j} \begin{cases} \text{if} \quad \mathbf{G}_{l,j} \notin \{y_{w,1}, \ldots, y_{w,j-1}\} \\ \qquad \text{and } \mathbf{G}_{l,j} \neq 0; \\ \text{then } \mathbf{G}_{l,j} = 0; \end{cases} \\ 0 \quad \text{otherwise} \end{cases}$$
$$l = 1, \ldots, J_{\min};\ j = 1, \ldots, N;\ w = 1, \ldots, J_{\min} \tag{11}$$

Repairing Trial Individuals

Some individuals may exist, which may not contain a permutative schedule. The jump individuals Y_w ($w = 1, 2, \ldots, J_{min}$), are constructed in such a way, that each infeasible element $y_{w,j}$ is indexed by 0.

Taking each jump individual Y_w iteratively from *H*, the following set of procedures can be applied recursively.

Take *A* and *B*, where *A* is initialised to the permutative schedule $A = \{1, 2, \ldots, N\}$ and *B* is the complement of individual Y_w relative to *A* as given in equation (12).

$$B = A \backslash Y_w \tag{12}$$

If after the complement operation, *B* is an empty set without any elements; $B = \{\}$, then the individual is correct with a proper permutative schedule and does not require any repairment.

However, if *B* contains values, then these values are the missing elements in individual Y_w. The repairment procedure is now outlined. The first process is to randomise the positions of the elements in set *B*. Then, iterating through the elements $y_{w,j}$ ($j = 1, \ldots, N$) in the individual Y_w, each position, where the element $y_{w,j} = 0$ is replaced by the value in *B*. Assigning B_{size} as the total number of elements present in *B* (and hence missing from the individual Y_w), the repairment procedure can be given as in equation (13).

$$y_{w,j} = \begin{cases} B_h & \text{if } \ y_{w,j} = 0 \\ y_{w,j} & \text{otherwise} \end{cases} \qquad h = 1, \ldots, B_{size};\ j = 1, \ldots, N \tag{13}$$

After each individual is repaired in *H*, it is then evaluated for its fitness value as in equation (14) and stored in γ, the fitness array of size J_{min}.

$$\gamma_w = \Im\left(Y_w\right), \quad w = 1, \ldots, J_{\min} \tag{14}$$

Population update

2 Opt local search is applied to the best individual Y_{best} obtained with the minimum fitness value ($\min(\gamma_w)$). After the local search routine, the new individual is compared with the fitness of the incumbent individual X_i^{t-1}, and if it improves on the fitness, then the new individual is accepted in the population (15).

$$X_i^t = \begin{cases} Y_{best} & \text{if } \Im\left(Y_{best}\right) < C_i^{t-1} \\ X_i^{t-1} & \text{otherwise} \end{cases} \tag{15}$$

If this individual improves on the overall best individual in the population, it then replaces the best individual in the population (16).

$$X_{best}^t = \begin{cases} Y_{best} & \text{if } \Im\left(Y_{best}\right) < C_{best}^t \\ X_{best}^{t-1} & \text{otherwise} \end{cases} \tag{16}$$

Iteration

Sequentially, incrementing *i*, the population counter by 1, another individual X_{i+1}^{t-1} is selected from the population, and it begins its own sampling towards the designated leader X_L^{t-1}. It should be noted that the leader does not change during the evaluation of one migration.

Migrations

Once all the individuals have executed their sampling towards the designated leader, the migration counter *t* is incremented by 1. The individual iterator *i* is reset to 1 (the beginning of the population) and the migration loop is re-initiated.

2 Opt local search

The local search utilised in DSOMA is the 2 Opt local search algorithm (Lin and Kernighan, 1973). The reason as to why the 2 Opt local search was chosen, is that it is the simplest in the *k*-opt class of routines. As the DSOMA sampling occurs between two individuals in *k*-dimension, the local search refines the individual. This in turn provides a better probability to find a new leader after each jump sequence. The placement of the local search was refined heuristically during experimentation.

The complexity of this local search is $O\left(n^2\right)$. As local search makes up the majority of the complexity time of DSOMA, the overall computational complexity of DSOMA for a single migration is $O\left(n^3\right)$.

CHAOS DRIVEN DISCRETE SELF ORGANISING MIGRATING ALGORITHM

The Lozi map is embedded in the DSOMA algorithm in place of the PRNG, and the new algorithm (DSOMA_C) is used to solve the FSSB problem. The operational parameters of DSOMA_C are given in Table 2.

Table 2: Operating parameters for DSOMA_C for FSSB.

Parameter	Value
Individuals	100
Migrations	20
Sampling (J_{min})	20
Local Search	2 Opt

DSOMA$_C$ experiment on FSSB problem

The Taillard benchmark problems used for the experiments is referenced from Taillard (1993). These benchmarks comprise of 12 different sets of problems ranging from 20 jobs and 5 machines to 500 jobs and 20 machines. Each set contains 10 unique instances, hence a total of 120 instances Beasley (2009).

Each instance has 10 independent replications and in each replication, the percentage relative difference *(PRD)* is computed as follows:

$$PRD = \frac{100 \times \left(C^{Ron} - C^{DSOMA_C}\right)}{C^{Ron}} \quad (17)$$

where C^{Ron} is the referenced makespan provided by Ronconi (2005), and C^{DSOMA_C} is the makespan found by the DSOMA$_C$ algorithm. Furthermore, average percentage relative difference *(APRD)*, maximum percentage relative difference *(MaxPRD*), minimum percentage relative difference *(MinPRD)* and the standard deviation (SD) of *PRD* are calculated. The average execution time for each set (T(s)) is also displayed.

Termination and Starting Criteria

Comparison of DSOMA$_C$ is done with a number of published algorithms. The first comparison is with the DE variant and its hybrid equivalent (HDE) of Qian et al. (2009). The HDE algorithm was modified by Ling et al. (2010) for the blocking flow shop scheduling problem. The second variant of DE is the discrete DE (DDE) and the hybrid discrete DE (HDDE) of Ling et al. (2010). The third algorithm is the GA of Caraffa et al. (2001), and finally the TS and its multimoves (TS+M) variant of Grabowski and Pempera (2007).

The principle termination criteria for DSOMA$_C$ is set at 20 migrations (see Table 2). This is against the termination criteria of other recent algorithms of HDDE, DDE, HDE and DE, which have a termination criteria based on computational time set as $5\,(m \cdot n)\; ms$. This however is in line with TS, TS+M and GA who have the termination set for the number of generation of which the minimum is 100.

Another important aspect is the starting population of the different algorithms. DSOMA$_C$ has a random population, whereas the DDE, HDDE, DE, HDE, TS and TS+M have a population based on a constructed seed individual utilising the NEH (Nawaz et al., 1983) algorithm, which is one of the well known and better performing heuristics that guarantees an initial solution with a certain quality and diversity. In order to bring parity to the algorithms, the experiment for the Taillard sets is redesigned with a seed individual created using the NEH (Nawaz et al., 1983) algorithm included in the initial population.

Comparison of DSOMA$_C$ with DE/HDE algorithms

A very good analytical presentation is given in Ling et al. (2010), where the authors compared their own discrete DDE and hybrid HDDE algorithms with the HDE algorithm of Qian et al. (2009).

The comparison of DSOMA$_C$ is done with the HDE algorithm of Qian et al. (2009) and the DE (without local search) variant of HDE given in Table 3.

DSOMA$_C$ has better performance indices for all the specifications compared to HDE and DE. It has better ARPD values for all the 12 different sets and on average has a better APRD, MinPRD and MaxPRD.

Comparison of DSOMA$_C$ with DDE/HDDE algorithms

Table 4 compares the results for DSOMA$_C$ with the DDE and its hybrid variant HDDE of Ling et al. (2010). The hybrid component of HDDE is a novel insert-neighbourhood-based speed-up method.

From the results obtained, DSOMA$_C$ performs better than both DDE and HDDE in two of the five performance specifications. In ten out of the 12 sets, it obtains better APRD values as well as higher MaxPRD values. Overall DSOMA$_C$ has on average a superior APRD, which is the key measure of performance, and MaxPRD.

Comparison of DSOMA$_C$ with GA algorithm

This comparison of the result of DSOMA$_C$ with that of GA algorithm of Caraffa et al. (2001) is given in Table 5. For all the problem instances, DSOMA$_C$ produces better results for the PRD.

Comparison of DSOMA$_C$ with TS and TS+M

Table 6 compares the results of DSOMA$_C$ with the TS and its hybrid variant TS+M algorithms of Grabowski and Pempera (2007). TS is considered one of the best constructive heuristics, and it has successfully been applied to a number of complex optimisation tasks (Grabowski and Pempera, 2007).

The comparison of results are given are Table 6. Based on the results DSOMA$_C$ is the better performing heuristic in all 12 problem sets. Overall, it obtains four times better results in the APRD than the TS+M algorithm (4.03 - 0.81). As the termination criteria of TS and TS+M are set to 100 generations, the average execution time of DSOMA$_C$ is significantly lower.

T-test of DSOMA$_C$ with HDDE, TS+M and RON

To test the performances of DSOMA$_C$ with that of HDDE, TS+M and RON, a series of paired *t*-test at the 95% significance level was carried out. The point of interest in the paired *t*-test is in the difference in two observations within the pairs. The paired results of DSOMA$_C$ with HDDE, TS+M and RON are given in Table 7.

In all three two paired *t*-tests, the null hypothesis is rejected and the alternate hypothesis is accepted, stating that there is significant differences between the two compared heuristics in terms of the average relative percent deviations generated. What this implies is that DSOMA$_C$ is significantly better performing than HDDE, TS+M and RON algorithms, given the better results obtained by DSOMA$_C$.

Table 3: Computation comparison of $DSOMA_C$ with DE and HDE of Qian et al. (2009)

J x M	$DSOMA_C$[1]					DE[2]					HDE[2]				
	APRD	MinPRD	MaxPRD	SD	T(s)	APRD	MinPRD	MaxPRD	SD	T(s)	APRD	MinPRD	MaxPRD	SD	T(s)
20 x 5	0.46	0.00	1.83	0.60	0.25	-0.78	-1.32	-0.05	0.4	0.5	0.26	0	0.44	0.16	0.5
20 x 10	2.40	0.57	5.36	1.48	0.6	1.73	1.29	2.14	0.3	1	2.3	2.13	2.39	0.1	1
20 x 20	3.30	2.13	4.48	0.72	1.08	2.86	2.49	3.18	0.22	2	3.25	3.14	3.3	0.06	2
50 x 5	4.81	0.97	7.45	1.77	2.31	-4.32	-5.2	-3.33	0.61	1.25	3.09	2.59	3.62	0.36	1.25
50 x 10	6.23	4.86	7.36	0.89	3.54	-0.3	-0.89	0.43	0.44	2.5	4.57	4.1	5.06	0.31	2.5
50 x 20	6.62	4.48	8.40	1.42	4.63	1.99	1.47	2.51	0.33	5	5	4.58	5.51	0.3	5
100 x 5	2.24	1.00	3.74	0.95	8.31	-13.99	-14.62	-13.06	0.48	2.5	-0.32	-0.84	0.31	0.36	2.5
100 x 10	5.90	4.67	7.94	0.93	17.08	-5.7	-6.25	-5.11	0.35	5	3.4	2.88	3.99	0.35	5
100 x 20	5.14	4.00	6.01	0.62	28.42	-2.42	-2.8	-1.93	0.29	10	3.05	2.69	3.47	0.26	10
200 x 10	4.03	2.56	5.49	1.09	195.33	-11.12	-11.46	-10.63	0.26	10	0.33	-0.14	0.88	0.34	10
200 x 20	3.99	2.82	4.81	0.66	243.33	-5.82	-6.1	-5.46	0.2	20	1.36	1	1.82	0.26	20
500 x 20	3.23	2.53	4.03	0.46	435.34	-8.2	-8.33	-8.02	0.1	50	0.25	-0.06	0.59	0.2	50
Mean	4.03	2.55	5.57	0.97	78.35	-3.84	-4.31	-3.28	0.33	9.15	2.21	1.84	2.62	0.26	9.15

[1] MacBook Pro, 2.3GHz Intel Core i7 (2nd gen), 8 GB RAM
[2] Pentium P-IV, 3.0 GHz, 512 MB

Table 4: Computation comparison of $DSOMA_C$ with DDE and HDDE of Ling et al. (2010)

J x M	$DSOMA_C$[1]					DDE[2]					HDDE[2]				
	APRD	MinPRD	MaxPRD	SD	T(s)	APRD	MinPRD	MaxPRD	SD	T(s)	APRD	MinPRD	MaxPRD	SD	T(s)
20 x 5	0.46	0.00	1.83	0.60	0.25	0.34	0.18	0.45	0.1	0.5	0.43	0.33	0.46	0.05	0.5
20 x 10	2.40	0.57	5.36	1.48	0.6	2.34	2.16	2.39	0.08	1	2.38	2.36	2.4	0.02	1
20 x 20	3.30	2.13	4.48	0.72	1.08	3.25	3.15	3.3	0.06	2	3.29	3.24	3.3	0.02	2
50 x 5	4.81	0.97	7.45	1.77	2.31	4.15	3.73	4.61	0.28	1.25	4.24	3.88	4.67	0.25	1.25
50 x 10	6.23	4.86	7.36	0.89	3.54	5.36	4.94	5.83	0.28	2.5	5.75	5.43	6.12	0.23	2.5
50 x 20	6.62	4.48	8.40	1.42	4.63	5.55	5.25	5.87	0.2	5	6.03	5.74	6.34	0.2	5
100 x 5	2.24	1.00	3.74	0.95	8.31	0.37	0.03	0.79	0.24	2.5	1.42	1.04	1.86	0.26	2.5
100 x 10	5.90	4.67	7.94	0.93	17.08	3.9	3.59	4.25	0.21	5	5.17	4.92	5.56	0.21	5
100 x 20	5.14	4.00	6.01	0.62	28.42	3.62	3.36	3.88	0.16	10	4.68	4.39	5.01	0.19	10
200 x 10	4.03	2.56	5.49	1.09	195.33	1.29	1.04	1.58	0.17	10	3.09	2.8	3.47	0.2	10
200 x 20	3.99	2.82	4.81	0.66	243.33	2.17	1.99	2.35	0.11	20	3.57	3.31	3.86	0.17	20
500 x 20	3.23	2.53	4.03	0.46	435.34	1.19	1.08	1.34	0.08	50	2.47	2.16	2.78	0.2	50
Mean	4.03	2.55	5.57	0.97	78.35	2.79	2.54	3.05	0.16	9.15	3.54	3.3	3.82	0.17	9.15

[1] MacBook Pro, 2.3GHz Intel Core i7 (2nd gen), 8 GB RAM
[2] Pentium P-IV, 3.0 GHz, 512 MB

Table 5: Computation comparison of $DSOMA_C$ with GA of Caraffa et al. (2001).

J x M	$DSOMA_C$[1]					GA[2]	
	APRD	MinPRD	MaxPRD	SD	T(s)	PRD	T(s)
20 x 5	0.46	0	1.83	0.6	0.25	-6.36	0.1
20 x 10	2.4	0.57	5.36	1.48	0.6	-4.35	0.2
20 x 20	3.3	2.13	4.48	0.72	1.08	-1.26	0.4
50 x 5	4.81	0.97	7.45	1.77	2.31	-8.53	0.3
50 x 10	6.23	4.86	7.36	0.89	3.54	-5.97	0.5
50 x 20	6.62	4.48	8.4	1.42	4.63	-4.33	1.1
100 x 5	2.24	1	3.74	0.95	8.31	-14.4	0.5
100 x 10	5.9	4.67	7.94	0.93	17.08	-7.89	1.1
100 x 20	5.14	4	6.01	0.62	28.42	-5.64	2.1
200 x 10	4.03	2.56	5.49	1.09	195.33	-11.04	2.2
200 x 20	3.99	2.82	4.81	0.66	243.33	-7	4.3
500 x 20	3.23	2.53	4.03	0.46	435.34	-8.08	10.8
Mean	4.03	2.55	5.57	0.97	78.35	-7.07	1.97

[1] MacBook Pro, 2.3GHz Intel Core i7 (2nd gen), 8 GB RAM
[2] Pentium P-IV 1000MHz

Table 6: Computation comparison of $DSOMA_C$ with TS and TS+M of Grabowski and Pempera (2007).

J x M	$DSOMA_C$[1]					TS[2]		TS+M[2]	
	APRD	MinPRD	MaxPRD	SD	T(s)	PRD	T(s)	PRD	T(s)
20 x 5	0.46	0	1.83	0.6	0.25	-1.64	2.4	-0.34	2.7
20 x 10	2.4	0.57	5.36	1.48	0.6	1.45	4.1	1.76	4.6
20 x 20	3.3	2.13	4.48	0.72	1.08	2.88	7.1	2.94	7.6
50 x 5	4.81	0.97	7.45	1.77	2.31	-0.55	6	0.55	6.2
50 x 10	6.23	4.86	7.36	0.89	3.54	1.98	10.6	3.52	10.8
50 x 20	6.62	4.48	8.4	1.42	4.63	3.68	19	4.26	19.3
100 x 5	2.24	1	3.74	0.95	8.31	-3.03	12.2	-2.62	12.4
100 x 10	5.9	4.67	7.94	0.93	17.08	1.71	21.9	2.66	22.1
100 x 20	5.14	4	6.01	0.62	28.42	2.01	39.2	3.03	39.4
200 x 10	4.03	2.56	5.49	1.09	195.33	-0.6	44.1	0.58	44.3
200 x 20	3.99	2.82	4.81	0.66	243.33	1.24	79.2	2.31	79.4
500 x 20	3.23	2.53	4.03	0.46	435.34	0.63	207	1.47	209
Mean	4.03	2.55	5.57	0.97	78.35	0.81	37.73	1.68	38.15

[1] MacBook Pro, 2.3GHz Intel Core i7 (2nd gen), 8 GB RAM
[2] Pentium P-IV, 1000 MHz

Table 7: Paired t-test for $DSOMA_C$ against HDDE, TS+M and RON.

$DSOMA_C$ vs	t-value	p-value	$p < 0.05$	Better
$HDDE$	3.8813	0.0001	Yes	$DSOMA_C$
$TS + M$	11.4269	0.0001	Yes	$DSOMA_C$
RON	11.4052	0.0001	Yes	$DSOMA_C$

CONCLUSION

In this research, the DSOMA algorithm is embedded with the Lozi chaotic map and applied to the FSSB problem and compared with a number of current best performing algorithms for this problem. The initial population was modified with a NEH heuristic, in order to provide a decent seed individual. The termination criteria were modified from the problem sized based to one of migration based. The reasoning is that with evolving hardware specifications, and the extensive usage of greedy search based techniques, using the problem size as the benchmark is illogical. While this topic remains open to debate, based on the experimentation results, $DSOMA_C$ was better performing than all other compared heuristics of DE, HDDE, TS, TS+M and GA, based on the APRD. In terms of operational time, however, it was more expensive especially for the large sized problems. Based on the t-test comparison $DSOMA_C$ was significantly better than the compared heuristics.

ACKNOWLEDGEMENT

This work was supported by the Technology Agency of the Czech Republic under the Project TE01020197 and the Internal Grant Agency of Tomas Bata University under the project No. IGA/FAI/2013/012.

REFERENCES

Alatas, B., Akin, E., and Ozer, A. (2009). Chaos embedded particle swarm optimization algorithms. *Chaos, Solitons and Fractals*, 40(4):1715–1734.

Beasley, J. (2009). Operations research library. http://people.brunel.ac.uk/ mastjjb/jeb/info.htm.

Caponetto, R., Fortuna, L., Fazzino, S., and Xibilia, M. (2003). Chaotic sequences to improve the performance of evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 7(3):289–304.

Caraffa, V., Ianes, S., Tapan, P., and Sriskandarajah, C. (2001). Minimizing makespan in a blocking flowshop using genetic algorithms. *International Journal of Production Economics*, 70(2):101 – 115.

Davendra, D. (2009). *Chaotic Attributes and Permutative Optimization*. PhD thesis, Tomas Bata University in Zlin.

Davendra, D., Zelinka, I., and Senkerik, R. (2010). Chaos driven evolutionary algorithms for the task of pid control. *Computers amp; Mathematics with Applications*, 60(4):1088 – 1104.

Grabowski, J. and Pempera, J. (2007). The permutation flow shop problem with blocking. a tabu search approach. *Omega*, 35(3):302 – 311.

Hilborn, R. (2000). *Chaos and Nonlinear Dynamics: An Introduction for Scientists and Engineers*. OUP Oxford, UK.

Lin, S. and Kernighan, B. (1973). An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 21(2):498–516.

Ling, W., Quan-Ke, P., Suganthan, P., Wen-Hong, W., and Ya-Min, W. (2010). A novel hybrid discrete differential evolution algorithm for blocking flow shop scheduling problems. *Computers & Operations Research*, 37(3):509 – 520.

Lu, Y., Zhou, J., Qin, H., Wang, Y., and Zhang, Y. (2011). Chaotic differential evolution methods for dynamic economic dispatch with valve-point effects. *Engineering Applications of Artificial Intelligence*, 24(2):378 – 387.

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

Matsumoto, M. and Nishimura, T. (1998). Mersenne twister: A 623-dimensionally equidistributed uniform pseudorandom number generator. *ACM Transaction on Modeling and Computer Simulation*, 8(1):3–30.

Nawaz, M., Enscore, E., and Ham, I. (1983). A heuristic algorithm for the m- machine, n-job flowshop sequencing problem. *OMEGA The International Journal of Management Science*, 11:91–95.

Ozer, A. (2010). Cide: Chaotically initialized differential evolution. *Expert Systems with Applications*, 37(6):4632 – 4641.

Pinedo, M. (1995). *Scheduling: theory, algorithms and systems*. Prentice Hall, Inc., New Jersey.

Qian, B., Wang, L., Huang, D., and Wang, X. (2009). An effective hybrid de-based algorithm for flow shop scheduling with limited buffers. *International Journal of Production Research*, 47.

Ronconi, D. (2005). A branch-and-bound algorithm to minimize the makespan in a flowshop with blocking. *Annals OR*, 138(1):53–65.

Taillard, E. (1993). Benchmarks for basic scheduling problems. *European Journal of Operations Research*, 64:278–285.

Yuan, X., Cao, B., Yang, B., and Yuan, Y. (2008). Hydrothermal scheduling using chaotic hybrid differential evolution. *Energy Conversion and Management*, 49(12):3627 – 3633.

Zelinka, I. (2004). Soma - self organizing migrating algorithm. In G., O. and B., B., editors, *New Optimization Techniques in Engineering*. Springer-Verlag, Germany.

Zuo, X. and Fan, Y. (2006). A chaos search immune algorithm with its application to neuro-fuzzy controller design. *Chaos, Solitons and Fractals*, 30(1):94–109.

AUTHOR BIOGRAPHIES

DONALD DAVENDRA is an Assistant Professor of Computer Science at VSB - Technical University of Ostrava, Czech Republic. His email is donald.davendra@vsb.cz

MAGDALENA BIALIC-DAVENDRA is a Post-Doctoral researcher in the Centre of Applied Economic Research, at Tomas Bata University in Zlin. Her email is bialic@fame.utb.cz

ROMAN SENKERIK is an Assistant Professor of Informatics at Tomas Bata University in Zlin, Czech Republic. His email is senkerik@fai.utb.cz

Michal Pluhacek is a doctoral student at Tomas Bata University in Zlin, Czech Republic. His email is pluhacek@fai.utb.cz