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 Dis-
crete Self Organising Migrating Algorithm (DSOMA) al-
gorithm, as a pseudorandom number generator (PRNG).
This novel chaotic based algorithm is applied to the
flow shop with blocking scheduling problem. The algo-
rithm 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 pertur-
bation towards the desired goal. The task of generating
these stochasticity is generally in the realm of pseudoran-
dom number generators (PRNG); a structured sequence
of mathematical formulation which tries to yield a gen-
erally 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 (Mat-
sumoto 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 de-
scribes the complex behaviour of simple, well behaved
systems. When casually observed, this behaviour can
seem erratic and somewhat random, however, these sys-
tems are deterministic, whose precise knowledge of fu-
ture behaviour is well known. The question is then to
reconcile the notion of nonlinearity of these systems.

Sudden and dramatic changes in some nonlinear sys-
tems 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 be-
haviour is aperiodic (it never exactly repeats) and appears
apparently random or noisy (Hilborn, 2000).

This aperiodic non-repeating behaviour of chaotic sys-
tems is the core foundation of this research. The objec-
tive 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 conserva-
tive 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 crop-
ning.

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 unpre-
dictability, 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 Diff-
ferential Evolution (DE) to solve the PID optimisation problem, whereas Otter (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).

\[
X_{n+1} = 1 - a \cdot |X_n| + b \cdot Y_n
\]

\[
Y_{n+1} = X_n
\]

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 referred 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 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:

\[
Fm | block | C_{\text{max}}
\]

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 \(j_1, \ldots, j_n\):

\[
D_{i,j} = \sum_{l=1}^{i} p_{i,j_l} \quad i = 1, \ldots, m
\]

\[
D_{i,j} = \max \left( D_{i-1,j_k} + p_{i,j_k}, D_{i+1,j_{k-1}} \right) \quad i = 2, \ldots, m \quad k = 2, \ldots, n
\]

\[
D_{m,j} = D_{m-1,j} + p_{m,j_k}
\]

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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$J_{\text{min}}$</td>
<td>$(1^+)$</td>
<td>Control</td>
<td>Min number of jumps</td>
</tr>
<tr>
<td>Population</td>
<td>$10^+$</td>
<td>Control</td>
<td>Num. of individuals</td>
</tr>
<tr>
<td>Migrations</td>
<td>$10^+$</td>
<td>Termination</td>
<td>Number of iterations</td>
</tr>
</tbody>
</table>

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 $J_{\text{min}}$, which gives the minimum number of jumps (sampling) between two individuals. The SOMA variables PathLength, Step-Size 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 $x$ is set to 0.

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

$$ C_i^0 = \max (X_i^0), \quad i = 1, \ldots, \beta $$ (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 ($J_{\text{min}}$) 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 $J_{\text{max}}$ 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 $\text{mode}(J)$ function obtains the most common number in $J$ and designates it as $J_{\text{max}}$.

$$ J_j = [x_{i,j}^{t-1} - x_{L,j}^{t-1}], \quad j = 1, \ldots, N $$

$$ J_{\text{max}} = \begin{cases} \text{mode}(J) & \text{if } \text{mode}(J) > 0 \\ 1 & \text{otherwise} \end{cases} $$ (8)

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

$$ s = \begin{cases} \frac{J_{\text{max}}}{J_{\text{min}}} & \text{if } J_{\text{max}} \geq J_{\text{min}} \\ 1 & \text{otherwise} \end{cases} $$ (9)

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

$$ G_{i,j} = \begin{cases} x_{i,j}^{t-1} + s \cdot l & \text{if } x_{i,j}^{t-1} + s \cdot l < x_{L,j}^{t-1} \\ x_{i,j}^{t-1} - s \cdot l & \text{if } x_{i,j}^{t-1} + s \cdot l > x_{L,j}^{t-1} & \text{and } x_{i,j}^{t-1} > x_{L,j}^{t-1} \\ 0 & \text{otherwise} \end{cases} $$

$$ j = 1, \ldots, N; \quad l = 1, \ldots, J_{\text{min}} $$ (10)

**Constructing Trial Individuals**

For each jump sequence of two individuals, a total of $J_{\text{min}}$ new individuals can now be constructed from the jump positions. Taking a new temporary population $H$ ($H = \{Y_1, \ldots, Y_{J_{\text{min}}}\}$), in which each new individual $Y_w$ ($w = 1, \ldots, J_{\text{min}}$), is
constructed piecewise from \( G \). 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 \( G \). Each \( l^{th} (l = 1, \ldots, J_{\text{min}}) \) position for a specific element is sequentially checked in \( G_{i,j} \) 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 \( j^{th} \) value is set to zero as \( G_{i,j} = 0 \). This iterative procedure can be given as in equation (11).

\[
y_{w,j} = \begin{cases} 
G_{i,j} & \text{if } G_{i,j} \notin \{y_{w,1}, \ldots, y_{w,j-1}\} \\
0 & \text{and } G_{i,j} \neq 0; \\
G_{i,j} = 0; & \text{otherwise}
\end{cases} \quad l = 1, \ldots, J_{\text{min}}; \ j = 1, \ldots, N; \ w = 1, \ldots, J_{\text{min}} 
\tag{11}
\]

### Repairing Trial Individuals

Some individuals may exist, which may not contain a permutative schedule. The jump individuals \( Y_w \) \((w = 1, 2, \ldots, J_{\text{min}})\), 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).

\[
B = A \setminus Y_w 
\tag{12}
\]

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 \( B_{\text{size}} \) 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).

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

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

\[
\gamma_w = \gamma(Y_w), \quad w = 1, \ldots, J_{\text{min}} 
\tag{14}
\]

### Population update

2 Opt local search is applied to the best individual \( \gamma_{\text{best}} \) 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 \( X^t_{\text{best}} \), and if it improves on the fitness, then the new individual is accepted in the population (15).

\[
X^t = \begin{cases} 
\gamma_{\text{best}} & \text{if } \exists \gamma_{\text{best}} < C^t_{\text{best}} \\
X^t_{\text{best}} & \text{otherwise}
\end{cases} 
\tag{15}
\]

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

\[
X^t_{\text{best}} = \begin{cases} 
\gamma_{\text{best}} & \text{if } \exists \gamma_{\text{best}} < C^t_{\text{best}} \\
X^t_{\text{best}} & \text{otherwise}
\end{cases} 
\tag{16}
\]

### Iteration

Sequentially, incrementing \( i \), the population counter by 1, another individual \( X^t_{\text{best}} \) is selected from the population, and it begins its own sampling towards the designated leader \( X^t_{L-1} \). 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 \( O(n^2) \). As local search makes up the majority of the complexity time of DSOMA, the overall computational complexity of DSOMA for a single migration is \( O(n^3) \).

### 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>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Individuals</td>
<td>100</td>
</tr>
<tr>
<td>Migrations</td>
<td>2</td>
</tr>
<tr>
<td>Sampling (J_{\text{min}})</td>
<td>20</td>
</tr>
<tr>
<td>Local Search</td>
<td>2 Opt</td>
</tr>
</tbody>
</table>

Table 2: Operating parameters for DSOMA_C for FSSB.
DSOMAC 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:

$$\text{PRD} = \frac{100 \times (C_{\text{Ron}} - C_{\text{DSOMAC}})}{C_{\text{Ron}}}$$  (17)

where $C_{\text{Ron}}$ is the referenced makespan provided by Ronconi (2005), and $C_{\text{DSOMAC}}$ is the makespan found by the DSOMAC algorithm. Furthermore, the 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 DSOMAC 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 DSOMAC 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. DSOMAC 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 DSOMAC 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 DSOMAC is done with the HDE algorithm of Qian et al. (2009) and the DE (without local search) variant of HDE given in Table 3. DSOMAC has better performance indices for all the specifications compared to HDE and DE. It has better APRD values for all the 12 different sets and on average has a better APRD, MinPRD and MaxPRD.

Comparison of DSOMAC with DDE/HDDE algorithms

Table 4 compares the results for DSOMAC 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, DSOMAC 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 DSOMAC has on average a superior APRD, which is the key measure of performance, and MaxPRD.

Comparison of DSOMAC with GA algorithm

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

Comparison of DSOMAC with TS and TS+M

Table 6 compares the results of DSOMAC 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 DSOMAC 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 DSOMAC is significantly lower.

T-test of DSOMAC with HDDE, TS+M and RON

To test the performances of DSOMAC 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 DSOMAC 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 DSOMAC is significantly better performing than HDDE, TS+M and RON algorithms, given the better results obtained by DSOMAC.
<table>
<thead>
<tr>
<th>J x M</th>
<th>APRD</th>
<th>MinAPRD</th>
<th>MaxAPRD</th>
<th>SD</th>
<th>T(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>20 x 2</td>
<td>0.4</td>
<td>0.04</td>
<td>0.83</td>
<td>0.63</td>
<td>2.21</td>
</tr>
<tr>
<td>50 x 4</td>
<td>0.4</td>
<td>0.04</td>
<td>0.83</td>
<td>0.63</td>
<td>2.21</td>
</tr>
</tbody>
</table>

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

**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, DSOMAC 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 not 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**


---

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

<table>
<thead>
<tr>
<th>J x M</th>
<th>APRD</th>
<th>MinAPRD</th>
<th>MaxAPRD</th>
<th>SD</th>
<th>T(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>20 x 10</td>
<td>0.4</td>
<td>0.04</td>
<td>0.83</td>
<td>0.63</td>
<td>2.21</td>
</tr>
<tr>
<td>50 x 4</td>
<td>0.4</td>
<td>0.04</td>
<td>0.83</td>
<td>0.63</td>
<td>2.21</td>
</tr>
</tbody>
</table>

---

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

<table>
<thead>
<tr>
<th>J x M</th>
<th>APRD</th>
<th>MinAPRD</th>
<th>MaxAPRD</th>
<th>SD</th>
<th>T(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>20 x 10</td>
<td>0.4</td>
<td>0.04</td>
<td>0.83</td>
<td>0.63</td>
<td>2.21</td>
</tr>
<tr>
<td>50 x 4</td>
<td>0.4</td>
<td>0.04</td>
<td>0.83</td>
<td>0.63</td>
<td>2.21</td>
</tr>
</tbody>
</table>

---

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

<table>
<thead>
<tr>
<th>J x M</th>
<th>APRD</th>
<th>MinAPRD</th>
<th>MaxAPRD</th>
<th>SD</th>
<th>T(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>20 x 2</td>
<td>0.4</td>
<td>0.04</td>
<td>0.83</td>
<td>0.63</td>
<td>2.21</td>
</tr>
</tbody>
</table>

---

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

<table>
<thead>
<tr>
<th>DSOMAC vs</th>
<th>t-value</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDDE</td>
<td>3.8813</td>
<td>0.0001</td>
</tr>
<tr>
<td>TS + M</td>
<td>11.4269</td>
<td>0.0001</td>
</tr>
<tr>
<td>RON</td>
<td>11.4052</td>
<td>0.0001</td>
</tr>
</tbody>
</table>


**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