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

# Scatter Search Algorithm with Chaos based Stochasticity

Donald Davendra, Roman Senkerik, Ivan Zelinka and Michal Pluhacek

***Abstract*— In this paper, we introduce a Scatter Search algorithm which is driven using a set of four chaos maps. The chaos maps of Tinkerbell, Delayed Logistics, Lozi and Burgers are used as chaotic pseudorandom number generators in the Scatter Search algorithm. These variants of the algorithm are used to solve the flowshop with blocking problem. The results are compared with the Mersenne Twister version of Scatter Search. The new chaos driven Scatter Search algorithm is shown to have superior performance when compared with state of the art heuristics in literature.**

## I. Introduction

EVOLUTIONARY algorithms (EA's) have been used to solve a number of diverse and difficult engineering problems. The uniqueness of EA's arises from the methodology underpinning its premise. Broadly stated, each EA's is different from each other based on some criterion.

Genetic Algorithms (GA) was founded on the basis of evolution, and how evolutionary systems pass on knowledge and information throughout generations [1]. Ant Colony Optimisation [2] was based on the foraging behaviour of ant, and how the laying of the pheromone trail can lead to optimal paths in a graph. Differential Evolution (DE) [3] is another promising algorithm developed around 1995, which used vector differentials in a search space for exploration.

The late 1990's saw a proliferation of swarm algorithms. Swarm algorithms are those, which use the herding and flocking behaviour of various species to model EA's, which can then be applied to solve complex problems. The most famous of these is the Particle Swarm Optimisation (PSO) [4], which is modelled on the flocking and homing behaviours of migratory birds. Another quite important algorithm, which has been used quite effectively is the Tabu Search (TS) [5]. TS is based on *implicit* memory usage, using the fact that *tabus* (implying that certain things cannot be touched, they are scared) as normally conceived are transmitted by means of a *social* memory, which is subject to modification over time. This use of these memory management techniques, allows the evaluation of different search space for the algorithm. TS is heavily influenced and guided by information collected during its search.

A superset of the TS is the Scatter Search (SS) algorithm [6]. SS further improves on the memory management technique of TS, by incorporating a *reference set*, a specialised population, which contains intensified and diversified individuals. These individuals are the most extreme placed individuals in the population, therefore the search itself can now take place over a larger area. Another important aspect is that the reference set is quite small, compared to generic EA's.

This paper analyses the canonical SS and how it can be further enhanced using a new methodology of chaos.

Chaos has been shown to play a more distinct role in EA's over recent years [7]. One of these most important trends has been the use of chaotic maps as *chaos pseudorandom number generators* (CPRNG's), in place of pseudorandom number generators (PRNG's) in EA's. Generally, standard PRNG's such as the venerable Mersenne Twister [8], with a proven long period has been used as stock PRNG's in EA's. However, a number of experiments have shown that chaos maps, particularly the discrete dissipative systems can improve on the performance of EA's ([9], [10], [11]).

A number of chaotic maps have been shown to possess certainty, ergodicity and stochastic property. 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 [12].

An extended family of enhanced CPRNG's with very long series of PRNG's has been developed by [13] accomplished through the ultra weak coupling of the Tent Map, which is enhanced in order to conceal the chaotic genuine function [14].

DE has been improved using chaos maps ([15], [16] [17], [12]), whereas PSO has been successfully modified using different chaotic maps ([18], [19]). A highly improved chaos induced SOMA has been described in [20] and [21].

This research looks to expand this class of chaos driven algorithms and to validate if chaos can improve the canonical SS algorithm. For comparison, we utilise the Mersenne Twister as the canonical PRNG in SS, and compare it with four different chaotic maps for the flowshop with blocking problem.

The paper is organised as follows: section II introduces the SS algorithm and section III introduces the chaos drive SS algorithm. The four different chaos maps used in this work and its mathematical description is given in section IV.

The flowshop with blocking problem with its mathematical description is given in section V. The experimentation results

Donald Davendra and Ivan Zelinka are with the 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, ivan.zelinka}@vsb.cz). Roman Senkerik and Michal Pluhacek are with the Faculty of Applied Informatics, Tomas Bata University in Zlin, Nam. T.G. Masaryka 5555, 760 01 Zlin, Czech Republic. (email: {senkerik, pluhacek}@fai.utb.cz)

This work was fully supported by the SGS SP2014/170, IGA project No. IGA/FAI/2014/010, CEBIA-Tech No. CZ.1.05/2.1.00/03.0089, Grant Agency of the Czech Republic - GACR P103/13/08195S, and partially supported by Grant of SGS No. SP2014/42, VB - Technical University of Ostrava, Czech Republic, by the Development of human resources in research and development of latest soft computing methods and their application in practice project, reg. no. CZ.1.07/2.3.00/20.0072 funded by Operational Programme Education for Competitiveness.

TABLE I
SCATTER SEARCH NOTATION.

| Parameter | Representation |
|---|---|
| *RefSet* | Reference set of individuals |
| *b* | Size of the reference set |
| *P* | Size of the main population |
| *X* | Individuals generated by diversification generator |

are presented in section VI and the analysis with different published algorithms in given in the subsequent subsections. Finally, the work is concluded in section VII.

## II. SCATTER SEARCH

Scatter Search (SS) and its generalised form Path Relinking (PR) are heuristics, which are build on the principles of surrogate constraint design [22]. In particular, they are designed to capture information not contained separately in the original individuals, and take advantage of auxiliary heuristic individual methods to evaluate the combinations produced and generate new individuals, based only on the original elements.

### A. Basic Principles

The SS methodology is very flexible, since each of its elements can be implemented in a variety of ways and various degrees of sophistication. The notation of SS is given in Table I.

In general, there are five main routines or methods which make up the template [23].

1) **A Diversification Generation Method** to generate a collection of diverse trial individuals, using an arbitrary trial individual (or seed individual) as an input.
2) **An Improvement Method** to transform a trial individual into one or more enhanced trial individuals. (Neither the input nor the output individuals are required to be feasible, though the output individuals will more usually be expected to be so. If there is no improvement of the input trial individual results, the "enhanced" individual is considered to be the same as the input individual.)
3) **A Reference Set Update Method** to build and maintain a reference set (*RefSet*) consisting of the b *best* individuals found (where the value of *b* is typically small, e.g., no more than 20), organised to provide efficient accessing by other parts of the method. Individuals gain membership to the reference set according to their quality or their diversity.
4) **A Subset Generation Method** to operate on the reference set, to produce a subset of its individuals as a basis for creating combined individuals.
5) **A Solution Combination Method** to transform a given subset of individuals produced by the Subset Generation Method into one or more combined individual vectors.

The basic outline is given in Figure 1.

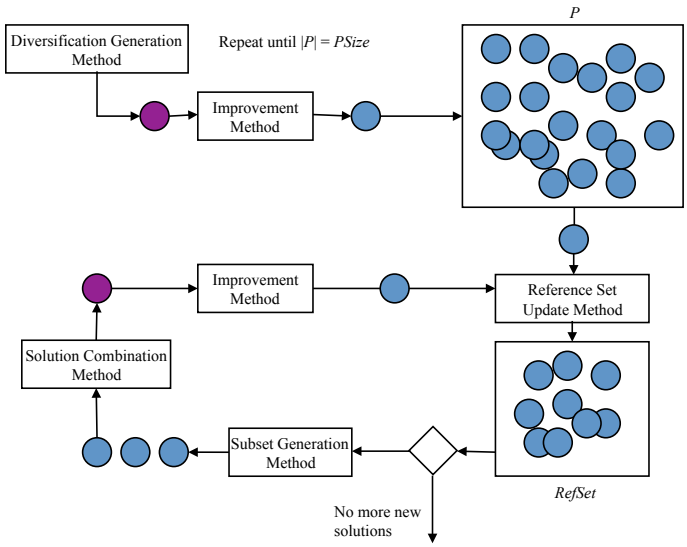


Fig. 1. Scatter Search Outline

The two principles that govern SS are:

1) Intensification,
2) Diversification.

Intensification refers to the role of isolating the best performing individuals from the populations in order to obtain a group of good individuals. Diversification, in turn, isolates the individuals which are the furthest from the best individuals and combined them with the best individuals. This new pool of individuals is the reference set, where crossover occurs in order to create individuals from new individual regions by the combination of the intensified individuals and diversified individuals. Intensification and diversification are commonly termed as adaptive memory programming.

### B. Reference Set Generation

The reference set is generated by two aspects of the population; intensified individuals and diversified individuals. The size of the reference set is defined at the beginning. It is usual to have half individuals in the refset as intensified and the rest as diversified. Intensified individuals are obtained by evaluating the population and moving the specified refset/2 best individuals from the population into the refset.

Campos *et. al.* [24] outlined how the diverse individuals are obtained from a population. The way diverse individuals are computed is through the computation of the minimum distances of each individual in the population to the individuals in refset. Then the individual with the maximum of these minimum distances is selected. Population individuals are included in refset according to the *maxmin* criterion, which maximises the minimum distance of each candidate individual to all the individuals currently in the refset. Sequentially, the refset is updated with an individual $X_i$ ($i = 1, 2, \ldots, N$, with $N$ being the size of the individual) from the population $P$. Consequentially, this individual is removed from the population, whose size is decreased by 1.

Thereafter, the distance of the newly included individual $X_i$ in the refset, to every individual currently in the population is computed to make possible the selection of a new refset individual according to the *maxmin* criterion. More formally, the selection of a population individual is given by:

$$X_i = \arg\max \min_{i=1,\ldots,|refset|} \{\varsigma_{i,j} : j = 1,\ldots,|P|\} \quad (1)$$

where $\varsigma$ is the diversity measure, which is the distance between individuals $X_i$ and $X_j$, which differ from each other by the number of edges which follows as:

$$\varsigma_{i,j} = \left|\left(X_i \cup X_j\right) \setminus \left(X_i \cap X_j\right)\right| \quad (2)$$

For extended details regrading the refset generation, the interested reader is directed to [24].

### C. Solution Combination Method

A number of individual combination methods exist, however through experimentation, we have narrowed down the choice of the method to the generic two point crossover as implemented in Genetic Algorithms (GA).

The advantages of the two point crossover operator is that there is no requirement for domain conversion, as all values are inherently discrete. Therefore, the only application is to check for in-feasibility and correct the individuals. The individuals are corrected using the repairment schema given in [25].

### D. Improvement method

SS employs the 2 Opt local search as its improvement strategy. The 2 Opt local search is a very simple routine. It utilises two iterators; one for the outer loop $k = 1, 2, \ldots, N-1$ and one for the inner loop $l = k+1, \ldots, N$. Using the iterators as indexes to the individuals, the indexed variables in the individuals are exchanged $X_i = \{\ldots, x_{i,l}, \ldots, x_{i,k}, \ldots\}$ and the new trial individual evaluated. If any improvement is seen, the individual is adapted in the population. The complexity of this local search is $O\left(n^2\right)$, where *n* is the input schedule.

## III. Chaos Induced Scatter Search

The chaos induced SS algorithm ($SS_c$) is one where the generic PRNG is replaced by a chaos map. Essentially, two aspects are required pertaining to random number; real and integer random numbers. A chaos sequence is one where a two dimensional coordinate is generated for each point, the *x* or *y* coordinate. For our needs, we can utilise either of these points, as both are aperiodic.

For a given chaos sequence *R*, a specific *x* coordinate at index *j* where $j \in R$ can be changed to a *real* number $r_{\text{real}}$ as given in equation (3).

$$r_{\text{real}} = \text{mod}\left(abs\left|x_j\right|, 1.0\right) \quad (3)$$

The corresponding integer value $r_{\text{int}}$ can be calculated as in equation (4).

$$r_{\text{int}} = \text{mod}\left(abs\left|x_j\right|, 1.0\right) \cdot UB + 1 \quad (4)$$

The $SS_c$ algorithm is described in Algorithm 1. Each occasion where the chaos used variable is used is marked with $\leftarrow C_r$.

**input** : Population ($P$), Generations ($G$), RefSet ($RS$), ChaosMap ($C$)
**output**: Best individual ($X_{best}$)

*Start Chaotic Sequence* $C_r \leftarrow (r_{real}, r_{int})$;
*Generate Population (P)* $\leftarrow C_r$;
*Calculate Fitness of P*;
  $fitness_P \leftarrow$ evaluate $f(P)$;
*Reference Set Generation*;
  $maxmin_P \leftarrow$ evaluate $f_{maxmin}(P)$;
*Create Refset (RS)*;
  $RS \leftarrow maxmin_P \cup fitness_P$;
**for** $i \leftarrow 1$ **to** $G$ **do**
  **for** $j \leftarrow 1$ **to** $RS$ **do**
    *Solution Combination Method*;
      $X_t \leftarrow \{RS_j \Leftrightarrow RS_{j+n}\} \leftarrow C_r$;
      $fitness_{X_t} \leftarrow$ evaluate $f(X_t)$;
    *Employ 2 OPT local search*;
      $fitness_{X_t} \leftarrow$ evaluate $f(LS(X_t))$;
    **if** $fitness_{X_t} < fitness_{RS_{worst}}$ **then**
      $RS_{worst} \leftarrow X_t$;
    **end**
  **end**
  **if** *No refset update in previous iteration* **then**
    *Obtain best solution from RefSet* ($X_{best}$);
    *Regenerate Population (P)* $\leftarrow C_r$;
    *Insert* $X_{best}$ *into P*;
      $P \leftarrow X_{best}$;
      $fitness_P \leftarrow$ evaluate $f(P)$;
      $maxmin_P \leftarrow$ evaluate $f_{maxmin}(P)$;
    *Regenerate RefSet*;
      $RS \leftarrow maxmin_P \cup fitness_P$;
  **end**
**end**
*Output best solution* ($X_{best}$);

**Algorithm 1:** $SS_c$ algorithm

## IV. Chaos Maps

Discrete dissipative systems, which are based on linear set of equations, and which can be easily formulated, is of the most interest as CPRNG's. For this research, four promising chaotic map of Burgers, Delayed Logistic, Lozi and Tinkerbell have been selected. The four chaotic maps are described in the following sections.

### A. Burgers Map

The Burgers map arose from the study of hydrodynamics, where the discretization of coupled differential equations led to a bifurcation effect of the system. The control parameters are $\alpha$ = 0.75 and $\beta$ = 1.75 [26].

$$X_{n+1} = (\alpha X_n) - Y_n^2$$
$$Y_{n+1} = (\beta Y_n) + (X_n Y_n) \quad (5)$$

### B. Delayed Logistic

The Delayed Logistic is a two-dimensional map, which is a phase shifted one-dimensional logistic equation. The control parameter $\alpha$ = 2.27 [26].

$$X_{n+1} = \alpha X_n (1 - Y_n)$$
$$Y_{n+1} = X_n \quad (6)$$

### C. Lozi Map

The Lozi map is a simple discrete two-dimensional chaotic map. The control parameters are $\alpha$ = 1.7 and $\beta$ = 0.5 [26].

$$X_{n+1} = 1 - (\alpha |X_n|) + (\beta Y_n)$$
$$Y_{n+1} = X_n \quad (7)$$

### D. Tinkerbell Map

The Tinkerbell map is a two-dimensional complex discrete-time dynamical system. The operating parameters as given by [26] are $\alpha$ = 0.9, $\beta$ = -0.6, $\rho$ = 2 and $\upsilon$ = 0.5.

$$X_{n+1} = X_n^2 - Y_n^2 + (\alpha X_n) + (\beta Y_n)$$
$$Y_{n+1} = (2 \cdot X_n Y_n) + (\rho X_n) + (\upsilon Y_n) \quad (8)$$

## V. 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** [27].

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 [27] 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 $j_1, \ldots, j_n$:

$$D_{i,j_1} = \sum_{l=1}^{i} p_{l,j_1} \quad i = 1, \ldots, m \quad (9)$$

$$D_{i,j_k} = \max\left(D_{i-1,j_k} + p_{i,j_k}, D_{i+1,j_{k-1}}\right) \quad i = 2, \ldots, m \;\; k = 2, \ldots, n \quad (10)$$

$$D_{m,j_k} = D_{m-1,j_k} + p_{m,j_k} \quad (11)$$

TABLE II
SS Operating parameters

| Parameter | Value |
|---|---|
| Population size | 60 |
| RefSet size | 20 |
| Local Search | 2 Opt |
| Crossover | 2 Point |
| Generations | 100 |

## VI. Experimentation

The Taillard benchmark problems used for the experiments is referenced from [28]. 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 [29].

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

$$PRD = \frac{100 \times \left(C^{Ron} - C^{SS_C}\right)}{C^{Ron}} \quad (12)$$

where $C^{Ron}$ is the referenced makespan provided by [30], and $C^{SS_C}$ is the makespan found by the $SS_c$ 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.

The operating parameters of SS is given in II. The population size is kept at 60, which is basically three times the size of the reference set. The number of generations is kept at 100. These parameters are kept stagnant for all experimentations.

### A. Comparison between different Chaos maps and Mersenne Twister

The initial experiment was conducted on the SS algorithm with the Mersenne Twister and with the four variants of the chaos map. A total of ten (10) experiments was conducted on *each* instance, resulting in a total of 1200 experiments *per* SS variant. Therefore, a cumulative total of 6000 experiments were conducted on the data sets to validate the findings of this research.

The average results for each problem size is given in Table III. From the obtained results, Tinkerbell is the best performing variant, with eight better average values out of the twelve instance sizes. Most importantly, it performs better for the lager instances of 200 x 10, 200 x 20 and 500 x 20. The second best performing variant is the Delayed Logistic with three better averages. A total cumulative average is also computed for all the data sets and Tinkerbell obtains the best value of 3.56.

When comparing with Mersenne Twister, the stock PRNG, it is obvious that the chaos variant are better performing for all data sets. In total average comparisons, Mersenne Twister is the worst performing for all the problem instances.

TABLE III

COMPUTATION COMPARISON OF DIFFERENT VARIANTS OF SS

| J x M | Mersenne Twister | Tinkerbell | Burgers | Lozi | Delayed Logistic |
|---|---|---|---|---|---|
| 20 x 5 | 0.32 | **0.44** | 0.36 | 0.42 | 0.39 |
| 20 x 10 | 2.16 | **2.39** | 2.14 | 2.21 | 2.28 |
| 20 x 20 | 2.94 | **3.14** | 2.87 | 3.02 | 3.12 |
| 50 x 5 | 4.24 | 4.55 | 4.42 | 4.54 | **4.65** |
| 50 x 10 | 5.65 | 6.04 | 5.96 | 5.85 | **6.14** |
| 50 x 20 | 6.12 | **6.21** | 6.04 | 5.96 | 6.13 |
| 100 x 5 | 1.43 | 1.53 | 1.61 | 1.32 | **1.68** |
| 100 x 10 | 4.53 | **4.92** | 4.42 | 4.76 | 4.72 |
| 100 x 20 | 4.65 | 4.72 | **4.79** | 4.67 | 4.71 |
| 200 x 10 | 2.87 | **3.11** | 2.88 | 2.97 | 3.02 |
| 200 x 20 | 3.01 | **3.32** | 3.21 | 3.05 | 3.21 |
| 500 x 20 | 2.33 | **2.45** | 2.32 | 2.29 | 2.36 |
| Mean | 3.35 | **3.56** | 3.41 | 3.41 | 3.53 |

[1] MacBook Pro, 2.3GHz Intel Core i7 (2nd gen), 8 GB RAM

It obtains 3.35 compared to 3.56 for Tinkerbell, 3.53 for Delayed Logistic and 3.41 for both the Burgers and Lozi maps.

Therefore, based on the obtained results, we can infer that the chaos maps, when used as CPRNG's, improve the basic performance of SS algorithm for the FSS with blocking problem. The graphical plot for all the variants is given in Figure 2.

### *B. Comparison of SS$_c$ with DE/HDE algorithms*

The chaos based SS algorithm (SS$_c$) is compared with some state-of-the art heuristics from literature. The first comparison is done with the Differential Evolution (DE) and Hybrid DE algorithm of [31].

The best results obtained from *all* the chaos variants of SS is compared with DE and HDE and presented in Table IV.

From the results, SS$_c$ 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 obtains a better APRD, MinPRD and MaxPRD.

### *C. Comparison of SS$_c$ with DDE/HDDE algorithms*

The second comparison is done with the Discrete Differential Evolution (DDE) and Hybrid DDE (HDDE) of [32]. The hybrid component of HDDE is a novel insert-neighbourhood-based speed-up method. The comparison results is given in Table V.

From the results obtained, SS$_c$ performs better than both DDE and HDDE in the APRD, where it obtains 3.58 compared to 2.79 for DDE and 3.54 for HDDE. In eight out of twelve data classes, SS$_c$ has a superior APRD. HDDE, on the other hand has better ARPD on the largest data sets, and it generally obtains higher Maximum PRD values compared to SS$_c$. The total execution time is generally 3 sec more for the SS$_c$ algorithm.

TABLE VI

COMPUTATION COMPARISON OF SS$_C$ WITH GA OF [33].

| J x M | SS$_c$ [1] | | | | | GA [2] | |
|---|---|---|---|---|---|---|---|
| | APRD | MinPRD | MaxPRD | SD | T(s) | PRD | T(s) |
| 20 x 5 | **0.44** | 0.00 | 0.93 | 0.30 | 0.09 | -6.36 | 0.1 |
| 20 x 10 | **2.39** | 1.27 | 4.32 | 0.53 | 0.6 | -4.35 | 0.2 |
| 20 x 20 | **3.14** | 1.76 | 4.77 | 0.54 | 0.67 | -1.26 | 0.4 |
| 50 x 5 | **4.65** | 1.03 | 6.75 | 1.54 | 0.78 | -8.53 | 0.3 |
| 50 x 10 | **6.14** | 4.32 | 6.82 | 0.33 | 1.32 | -5.97 | 0.5 |
| 50 x 20 | **6.21** | 4.43 | 7.53 | 0.43 | 3.43 | -4.33 | 1.1 |
| 100 x 5 | **1.68** | 0.76 | 3.21 | 0.56 | 4.22 | -14.4 | 0.5 |
| 100 x 10 | **4.92** | 3.43 | 6.32 | 0.43 | 6.34 | -7.89 | 1.1 |
| 100 x 20 | **4.79** | 3.65 | 5.64 | 0.32 | 13.44 | -5.64 | 2.1 |
| 200 x 10 | **3.11** | 2.21 | 3.86 | 0.43 | 24.33 | -11.04 | 2.2 |
| 200 x 20 | **3.32** | 2.16 | 3.98 | 0.34 | 30.45 | -7 | 4.3 |
| 500 x 20 | **2.45** | 1.33 | 3.63 | 0.32 | 50.24 | -8.08 | 10.8 |
| Mean | **3.58** | 2.19 | 4.81 | 0.50 | 11.32 | -7.07 | 1.97 |

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

TABLE VII

COMPUTATION COMPARISON OF SS$_C$ WITH TS AND TS+M OF [34].

| J x M | SS$_c$ [1] | | | | | TS [2] | | TS+M [2] | |
|---|---|---|---|---|---|---|---|---|---|
| | APRD | MinPRD | MaxPRD | SD | T(s) | PRD | T(s) | PRD | T(s) |
| 20 x 5 | **0.44** | 0.00 | 0.93 | 0.30 | 0.09 | -1.64 | 2.4 | -0.34 | 2.7 |
| 20 x 10 | **2.39** | 1.27 | 4.32 | 0.53 | 0.6 | 1.45 | 4.1 | 1.76 | 4.6 |
| 20 x 20 | **3.14** | 1.76 | 4.77 | 0.54 | 0.67 | 2.88 | 7.1 | 2.94 | 7.6 |
| 50 x 5 | **4.65** | 1.03 | 6.75 | 1.54 | 0.78 | -0.55 | 6 | 0.55 | 6.2 |
| 50 x 10 | **6.14** | 4.32 | 6.82 | 0.33 | 1.32 | 1.98 | 10.6 | 3.52 | 10.8 |
| 50 x 20 | **6.21** | 4.43 | 7.53 | 0.43 | 3.43 | 3.68 | 19 | 4.26 | 19.3 |
| 100 x 5 | **1.68** | 0.76 | 3.21 | 0.56 | 4.22 | -3.03 | 12.2 | -2.62 | 12.4 |
| 100 x 10 | **4.92** | 3.43 | 6.32 | 0.43 | 6.34 | 1.71 | 21.9 | 2.66 | 22.1 |
| 100 x 20 | **4.79** | 3.65 | 5.64 | 0.32 | 13.44 | 2.01 | 39.2 | 3.03 | 39.4 |
| 200 x 10 | **3.11** | 2.21 | 3.86 | 0.43 | 24.33 | -0.6 | 44.1 | 0.58 | 44.3 |
| 200 x 20 | **3.32** | 2.16 | 3.98 | 0.34 | 30.45 | 1.24 | 79.2 | 2.31 | 79.4 |
| 500 x 20 | **2.45** | 1.33 | 3.63 | 0.32 | 50.24 | 0.63 | 207 | 1.47 | 209 |
| Mean | **3.58** | 2.19 | 4.81 | 0.50 | 11.32 | 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

### *D. Comparison of SS$_c$ with GA algorithm*

The third comparison of SS$_c$ is done with the GA algorithm of [33] and is given in Table VI. For all the problem instances, SS$_c$ produces better results for the PRD.

### *E. Comparison of SS$_c$ with TS and TS+M*

The final comparison is done with the Tabu Search (TS) and its hybrid variant TS+M algorithms of [34]. TS is considered one of the best constructive heuristics, and it has successfully been applied to a number of complex optimisation tasks [34].

The comparison of results are given are Table VII. Based on the results, SS$_c$ is the better performing heuristic in all 12 problem sets. Overall, it obtains significantly better results in the APRD than the TS+M algorithm (3.58 - 0.81). As the termination criteria of TS and TS+M are set to 100 generations, the average execution time of SS$_c$ is significantly lower (11.32 sec compared to 38.15 sec).

## VII. CONCLUSION

A recent trend has emerged where the contribution of pseudorandom number generators are being analysed in evolutionary algorithms. To this effect, different stochasticity generators are being analysed in terms of *propagators* of evolutionary algorithms.

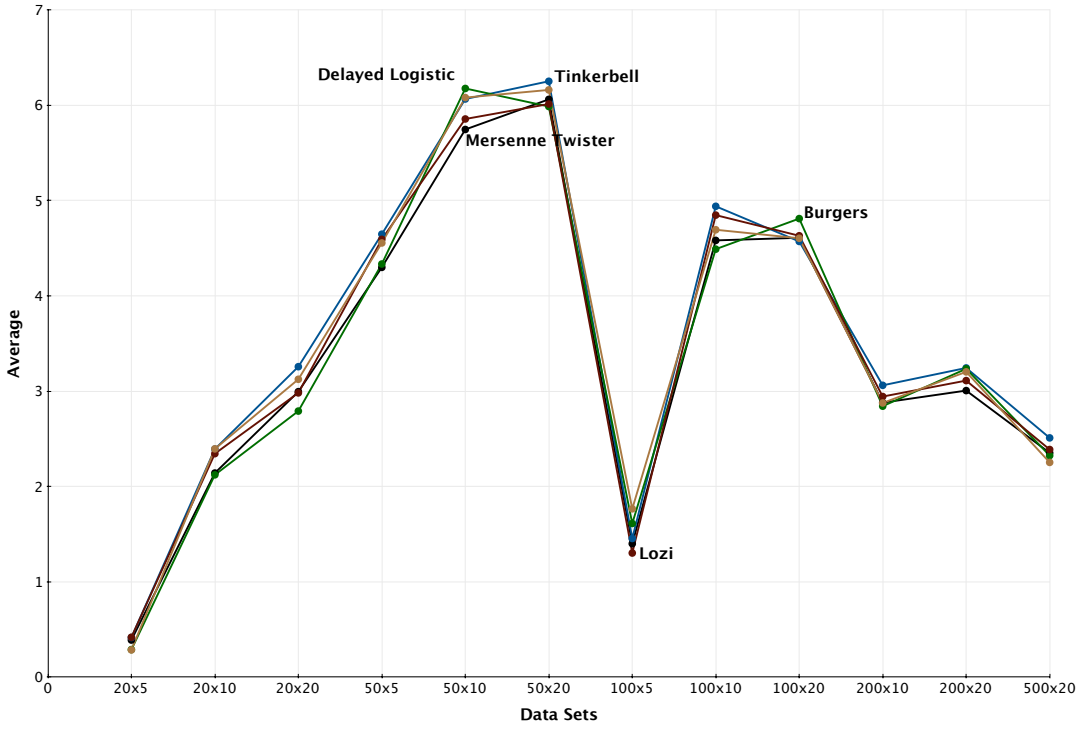


Fig. 2. Mean plot of different variants

TABLE IV
COMPUTATION COMPARISON OF $SS_C$ WITH DE AND HDE OF [31]

| J x M | $SS_c$[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 | **0.44** | 0.00 | 0.93 | 0.30 | 0.09 | -0.78 | -1.32 | -0.05 | 0.4 | 0.5 | 0.26 | 0 | 0.44 | 0.16 | 0.5 |
| 20 x 10 | **2.39** | 1.27 | 4.32 | 0.53 | 0.6 | 1.73 | 1.29 | 2.14 | 0.3 | 1 | 2.3 | 2.13 | 2.39 | 0.1 | 1 |
| 20 x 20 | **3.14** | 1.76 | 4.77 | 0.54 | 0.67 | 2.86 | 2.49 | 3.18 | 0.22 | 2 | 3.25 | 3.14 | 3.3 | 0.06 | 2 |
| 50 x 5 | **4.65** | 1.03 | 6.75 | 1.54 | 0.78 | -4.32 | -5.2 | -3.33 | 0.61 | 1.25 | 3.09 | 2.59 | 3.62 | 0.36 | 1.25 |
| 50 x 10 | **6.14** | 4.32 | 6.82 | 0.33 | 1.32 | -0.3 | -0.89 | 0.43 | 0.44 | 2.5 | 4.57 | 4.1 | 5.06 | 0.31 | 2.5 |
| 50 x 20 | **6.21** | 4.43 | 7.53 | 0.43 | 3.43 | 1.99 | 1.47 | 2.51 | 0.33 | 5 | 5 | 4.58 | 5.51 | 0.3 | 5 |
| 100 x 5 | **1.68** | 0.76 | 3.21 | 0.56 | 4.22 | -13.99 | -14.62 | -13.06 | 0.48 | 2.5 | -0.32 | -0.84 | 0.31 | 0.36 | 2.5 |
| 100 x 10 | **4.92** | 3.43 | 6.32 | 0.43 | 6.34 | -5.7 | -6.25 | -5.11 | 0.35 | 5 | 3.4 | 2.88 | 3.99 | 0.35 | 5 |
| 100 x 20 | **4.79** | 3.65 | 5.64 | 0.32 | 13.44 | -2.42 | -2.8 | -1.93 | 0.29 | 10 | 3.05 | 2.69 | 3.47 | 0.26 | 10 |
| 200 x 10 | **3.11** | 2.21 | 3.86 | 0.43 | 24.33 | -11.12 | -11.46 | -10.63 | 0.26 | 10 | 0.33 | -0.14 | 0.88 | 0.34 | 10 |
| 200 x 20 | **3.32** | 2.16 | 3.98 | 0.34 | 30.45 | -5.82 | -6.1 | -5.46 | 0.2 | 20 | 1.36 | 1 | 1.82 | 0.26 | 20 |
| 500 x 20 | **2.45** | 1.33 | 3.63 | 0.32 | 50.24 | -8.2 | -8.33 | -8.02 | 0.1 | 50 | 0.25 | -0.06 | 0.59 | 0.2 | 50 |
| Mean | **3.58** | 2.19 | 4.81 | 0.50 | 11.32 | -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 V
COMPUTATION COMPARISON OF $SS_C$ WITH DDE AND HDDE OF [32]

| J x M | $SS_c$[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 | **0.44** | 0.00 | 0.93 | 0.30 | 0.09 | 0.34 | 0.18 | 0.45 | 0.1 | 0.5 | 0.43 | 0.33 | 0.46 | 0.05 | 0.5 |
| 20 x 10 | **2.39** | 1.27 | 4.32 | 0.53 | 0.6 | 2.34 | 2.16 | 2.39 | 0.08 | 1 | 2.38 | 2.36 | 2.4 | 0.02 | 1 |
| 20 x 20 | 3.14 | 1.76 | 4.77 | 0.54 | 0.67 | 3.25 | 3.15 | 3.3 | 0.06 | 2 | **3.29** | 3.24 | 3.3 | 0.02 | 2 |
| 50 x 5 | **4.65** | 1.03 | 6.75 | 1.54 | 0.78 | 4.15 | 3.73 | 4.61 | 0.28 | 1.25 | 4.24 | 3.88 | 4.67 | 0.25 | 1.25 |
| 50 x 10 | **6.14** | 4.32 | 6.82 | 0.33 | 1.32 | 5.36 | 4.94 | 5.83 | 0.28 | 2.5 | 5.75 | 5.43 | 6.12 | 0.23 | 2.5 |
| 50 x 20 | **6.21** | 4.43 | 7.53 | 0.43 | 3.43 | 5.55 | 5.25 | 5.87 | 0.2 | 5 | 6.03 | 5.74 | 6.34 | 0.2 | 5 |
| 100 x 5 | **1.68** | 0.76 | 3.21 | 0.56 | 4.22 | 0.37 | 0.03 | 0.79 | 0.24 | 2.5 | 1.42 | 1.04 | 1.86 | 0.26 | 2.5 |
| 100 x 10 | 4.92 | 3.43 | 6.32 | 0.43 | 6.34 | 3.9 | 3.59 | 4.25 | 0.21 | 5 | **5.17** | 4.92 | 5.56 | 0.21 | 5 |
| 100 x 20 | **4.79** | 3.65 | 5.64 | 0.32 | 13.44 | 3.62 | 3.36 | 3.88 | 0.16 | 10 | 4.68 | 4.39 | 5.01 | 0.19 | 10 |
| 200 x 10 | **3.11** | 2.21 | 3.86 | 0.43 | 24.33 | 1.29 | 1.04 | 1.58 | 0.17 | 10 | 3.09 | 2.8 | 3.47 | 0.2 | 10 |
| 200 x 20 | 3.32 | 2.16 | 3.98 | 0.34 | 30.45 | 2.17 | 1.99 | 2.35 | 0.11 | 20 | **3.57** | 3.31 | 3.86 | 0.17 | 20 |
| 500 x 20 | 2.45 | 1.33 | 3.63 | 0.32 | 50.24 | 1.19 | 1.08 | 1.34 | 0.08 | 50 | **2.47** | 2.16 | 2.78 | 0.2 | 50 |
| Mean | **3.58** | 2.19 | 4.81 | 0.50 | 11.32 | 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

This research aims to extend this field of knowledge by analysing the SS algorithm in terms of its performance under different stochasticity generators. To this effect, the Mersenne Twister was selected as the best PRNG in literature. Additionally, four unique chaos maps of Tinkerbell, Burgers, Delayed logistic and Lozi were selected as CPRNG's. Using the same operating parameters, the five variants were used to solve the flowshop with blocking problem.

From the obtained results, it can be concluded that all the chaos variants performed better than the Mersenne Twister for the entire problem data sets. Tinkerbell was the best performing variant, followed by Delayed logistic, Burgers and Lozi map.

The chaos variant of SS was then compared with the current best algorithms in literature. $SS_c$ was shown to be better performing than DE, HDE of [31] and DDE, HDDE of [32]. Additionally, $SS_c$ was shown to have superior performance to GA of [33] and TS and TS+M of [34] for all the data sets and on all performance indices.

Therefore, it can be summarised that the performance of SS is improved with the usage of different chaos maps and $SS_c$ is a competitive algorithm for the flowshop with blocking problem.

## REFERENCES


[1] Z. Michalewicz, *Genetic Algorithm + Data Structures = Evolution*. Springer, Germany, 1994.

[2] D. M. and G. L., "Ant colony system: A co-operative learning approach to the traveling salesman problem," *IEEE Transactions on Evolutionary Computation*, vol. Vol. 1, pp. 53–65, 1997.

[3] K. Price, "An introduction to differential evolution," in *New ideas in Optimisation*, US, 1999.

[4] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of the 1995 IEEE International Conference on Neural Networks*, 1995, pp. 1942 – 1948.

[5] F. Glover and M. Laguna, *Tabu Search*. Kluwer, Norwell, MA, 1997.

[6] G. F., L. M., and R. Martí, "Fundamentals of scatter search and path relinking," *Control and Cybernetics*, vol. 39, pp. 653–684, 2000.

[7] I. Zelinka, S. Celikovsky, H. Richter, and G. Chen, *Evolutionary Algorithms and Chaotic Systems*. Springer, Germany, 2010.

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

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

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

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

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

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

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

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

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

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

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

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

[20] D. Davendra, R. Senkerik, I. Zelinka, M. Pluhacek, and M. Bialic-Davendra, "Utilising the chaos-induced discrete self organising migrating algorithm to solve the lot-streaming flowshop scheduling problem with setup time," *Soft Computing*, vol. 18, pp. 669 – 681, 2014.

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

[22] F. Glover, "Heuristics for integer programming using surrogate constraints," *Decision Sciences*, vol. 8, no. 1, pp. 156–166, 1977.

[23] R. Marti, M. Laguna, and F. Glover, "Principles of scatter search," *European Journal of Operations Research*, vol. 169, no. 2, pp. 359–372, 2008.

[24] V. Campos, F. Glover, M. Laguna, and R. Marti, "An experimental evaluation of a scatter search for the linear ordering problem," *Journal of Global Optimization*, vol. 21, pp. 397–414, 2001.

[25] D. Davendra and G. Onwubolu, "Forward backward transformation," in *Differential Evolution: A Handbook for Global Permutation-Based Combinatorial Optimization*, O. G. and D. D., Eds. Germany: Springer, 2009.

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

[27] M. Pinedo, *Scheduling: theory, algorithms and systems*. Prentice Hall, Inc., New Jersey, 1995.

[28] E. Taillard, "Benchmarks for basic scheduling problems," *European Journal of Operations Research*, vol. 64, pp. 278–285, 1993.

[29] J. Beasley, "Operations research library," 2009, http://people.brunel.ac.uk/ mastjjb/jeb/info.htm.

[30] D. Ronconi, "A branch-and-bound algorithm to minimize the makespan in a flowshop with blocking," *Annals OR*, vol. 138, no. 1, pp. 53–65, 2005.

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

[32] W. Ling, P. Quan-Ke, P. Suganthan, W. Wen-Hong, and W. Ya-Min, "A novel hybrid discrete differential evolution algorithm for blocking flow shop scheduling problems," *Computers & Operations Research*, vol. 37, no. 3, pp. 509 – 520, 2010.

[33] V. Caraffa, S. Ianes, P. Tapan, and C. Sriskandarajah, "Minimizing makespan in a blocking flowshop using genetic algorithms," *International Journal of Production Economics*, vol. 70, no. 2, pp. 101 – 115, 2001.

[34] J. Grabowski and J. Pempera, "The permutation flow shop problem with blocking. a tabu search approach," *Omega*, vol. 35, no. 3, pp. 302 – 311, 2007.