Island-based Cuckoo Search with Highly Disruptive Polynomial Mutation

Article *in* International Journal of Artificial Intelligence · March 2019

CITATIONS
2

READS
74

1 author:

Bilal Abed-alguni
Yarmouk University
19 PUBLICATIONS **136** CITATIONS

SEE PROFILE

Island-based Cuckoo Search with Highly Disruptive Polynomial Mutation

Bilal H. Abed-alguni

Department of Computer Sciences
Yarmouk University
21163, University Street, Irbid, Jordan
Bilal.h@yu.edu.jo

ABSTRACT

The island model is one of the most well-known structured population strategies used to control the diversity in evolutionary algorithms. The population of an island-based evolutionary algorithm is normally partitioned into several sub-populations (islands). An evolutionary algorithm is then applied to each island independently. After a number of predefined generations, a migration process takes place to exchange specific candidate solutions between the islands. Recently, the Cuckoo search (CS) algorithm has been proposed as a population-based algorithm that mimics the nesting and parasitic reproduction behaviors of some cuckoo species. The main drawback of the CS algorithm is that its evolutionary operators may not adequately preserve the diversity of its population during the evolution process which may cause it to converge earlier than expected to suboptimal solutions. This paper introduces an improved variation of CS called island-based CS with polynomial mutation (iCSPM) that adapts two improvements to CS. First, the strategy of island model is incorporated into the CS algorithm to empower its capability in controlling the diversity of its population. Second, the Lévy flight method in CS is replaced with the highly disruptive polynomial mutation method in an attempt to enhance the exploration of CS. The iCSPM algorithm was evaluated using 15 standard benchmark functions in terms of the accuracy and reliability of the obtained results over multiple simulations. The sensitivity analysis of the main parameters of iCSPM was carried out to show their effect on the convergence behavior of iCSPM. The experimental results suggest that iCSPM provides more accurate and reliable results than 3 competitive methods.

Keywords: Cuckoo search, Evolutionary algorithm, Island model, Structured population, Population diversity

Computing Classification System (CCS): k.3.2
Mathematics Subject Classification (MSC): 80M50

1 Introduction

Cuckoo search (CS) is a mathematical optimization algorithm inspired by the obligate brood parasitic behavior of some cuckoo species and the Lévy flight behavior of some fruit flies and birds (Yang and Deb, 2009). CS has recently attracted the attention of the evolutionary research community due to its efficiency and usability in solving NP-hard problems and problems with high complexity (Abed-alguni, 2017a; Abed-alguni and Alkhateeb, 2018; Abed-alguni and Alkhateeb, 2017; Alkhateeb and Abed-alguni, 2017; Marichelvam and Geetha, 2016; Srivastav and Agrawal, 2018). However, the evolutionary operators of CS (Section 3) may not adequately preserve the diversity of its population during the evolution process which may cause it to converge earlier than expected to suboptimal solutions (Alkhateeb and Abed-alguni, 2017; Abed-alguni and Alkhateeb, 2018).

Several research studies have attempted recently to improve CS for both continuous and discrete optimization problems (Abed-alguni and Alkhateeb, 2017; Marichelvam, 2012; Marichelvam, Prabaharan and Yang, 2014; Marichelvam and Tosun, 2016; Abed-alguni and Alkhateeb, 2018). In (Abed-alguni and Alkhateeb, 2017), several variations of the CS algorithm were proposed by replacing the uniformly random-based selection method used in CS with six randomized selection schemes: greedy, proportional, exponential, ε-greedy, softmax and reinforcement learning selection schemes. The experimental results showed that the proposed variations provide minor enhancement to the original cuckoo search algorithm for twenty well-known benchmark functions. Four variations of a hybrid CS and Simulated annealing (SA) algorithm (CSA1, CSA2, CSA3, CSA4) were proposed in (Alkhateeb and Abed-alguni, 2017). These variations aim to improve the performance CS by incorporating the exploration strategy of SA into CS to minimize the likelihood of getting stuck in local optima. As suggested in the experiments, the four variations of CSA provide more accurate results compared to the basic CS algorithm, but they require more computations compared to the basic CS algorithm. A hybrid CS algorithm (CSBHC) that incorporates the β-hill climbing into CS was proposed in (Abed-alguni and Alkhateeb, 2018). The β-hill climbing algorithm (i.e., an exploratory version of the Hill climbing algorithm (Al-Betar, 2016)) is used in CSBHC to balance between the exploration and exploitation of candidate solutions. Although it seems that CSBH requires more computational time than standard CS, CSBHC is ca-

pable of solving standard test functions and reaching semi-optimal solutions in shorter time than CS. A hybrid optimization algorithm that combines the exploitation strategy of the CS algorithm and the exploration strategy of harmony search was suggested in (Feng, Wang and Gao, 2016) for solving 0-1 knapsack problems. The simulation results indicated that the proposed algorithm performs better compared to the basic CS algorithm. A dynamic adaptive CS algorithm with a crossover operator was proposed in (Liao, Zhou, Shi and Shi, 2017) to finely tune the parameters of CS to maximize its effectiveness. However, dynamic adaptive CS showed limited improvement in the experiments compared to CS. An improved cuckoo search algorithm with two modifications was proposed in (Walton, Hassan, Morgan and Brown, 2011). The first modification involves the addition of information exchange between the best solutions, while the second modification deals with modifying the step size of the Lévy flight. The experimental results using standard test functions showed that the modified cuckoo search outperforms the original CS. In summary, the hybrid CS algorithms that incorporate other search algorithms or exploratory methods into CS (e.g., CSA, CSBHC) require more computational time than standard optimization algorithms, particularly when the hybrid CS algorithm uses a search method at each iteration of CS. Other versions of CS (e.g., improved CS) provide insignificant enhancement in the performance of CS and can be applied only to particular applications.

The island model is one of the most well-known structured population strategies used to control the diversity in evolutionary algorithms (e.g., island-based Genetic algorithm (Lardeux and Goëffon, 2010), island-based Harmony search (Al-Betar, Awadallah, Khader and Abdalkareem, 2015; Saadat, Moallem and Koofigar, 2017), island-based Bat algorithm (Al-Betar and Awadallah, 2018), island-based Ant colony optimization (Michel and Middendorf, 1998), island-based Particle swarm optimization (Romero and Cotta, 2005; Zhou and Tan, 2011), distributed Grey wolf optimizer (Abed-alguni and Barhoush, 2018; Precup, David and Petriu, 2017) and island-based Differential evolution (Kushida, Hara, Takahama and Kido, 2013)). The population of an island-based evolutionary algorithm is normally partitioned into smaller populations known as islands. An evolutionary algorithm is then applied to each island independently. After a number of predefined generations, a migration process takes place to exchange specific candidate solutions between the islands.

Mutation operators are basically exploratory methods used to generate new candidate solutions and to minimize the likelihood of getting trapped in local optima when solving a given optimization problem (Hasan, Doush, Al Maghayreh, Alkhateeb and Hamdan, 2014; Abed-alguni, Klaib and Nahar, 2019). Mutation methods such as random

(Michalewicz, 1994), boundary (Hasan et al., 2014), non-uniform (Michalewicz, Logan and Swaminathan, 1994), MPT (Toivanen, Makinen, Périaux and Cloud Cedex, 1999), power (Deep and Thakur, 2007), highly disruptive polynomial (HDP) (Deb and Tiwari, 2008) and pitch adjustment (Geem, Kim and Loganathan, 2001) methods have been used with evolutionary algorithms such as artificial bee colony (Doush, Hasan, Al-Betar, Al Maghayreh, Alkhateeb and Hamdan, 2014) and harmony search (Hasan et al., 2014) to solve continuous optimization problems. The HDP mutation has been found to be more accurate and stable than the other mutation methods for large number of the benchmark functions. This may be because the HDP method can explore the entire search space of a decision variable even if the variable's value is close to one of its boundaries.

The current research study introduces an improved variation of CS called island-based CS with polynomial mutation (*i*CSPM) that adapts two improvements to CS. First, the strategy of island model is incorporated into the CS algorithm to empower its capability in controlling the diversity of its population. Second, the Lévy flight method in CS is replaced with the HDP mutation method in an attempt to enhance the exploration of CS. In *i*CSPM, the population is divided into several islands. The candidate solutions on each island are independently evolved using CS with HDP mutation. The islands periodically interact using a migration process based on a random ring topology. The proposed algorithm was evaluated using a set of 15 popular benchmark functions over multiple simulations. The obtained results suggest that *i*CSPM provides more accurate results than well-known evolutionary algorithms. Interestingly, *i*CSPM can be applied to parallel machines, which means that its computational time can be significantly reduced compared to existing variations of CS (e.g., CSA, CSBHC, dynamic adaptive CS).

The rest of the paper is organized as follows: Section 2 presents the basic concepts of the island model, Section 3 introduces the Cuckoo search algorithm, Section 4 discusses the highly disruptive polynomial mutation, Section 5 throughly discusses the proposed algorithm, Section 6 presents the experimental results and finally Section 7 presents the conclusion of this paper.

2 Concepts of Island model

The island model is a popular structured evolutionary model in which the population of a given evolutionary problem is partitioned into smaller populations called islands (Corcoran and Wainwright, 1994). In this model, a sequential evolutionary algorithm can be applied separately to each island (Lardeux and Goëffon, 2010; Vaščák, 2012).

The islands interact periodically through a migration process that specifies how candidate solutions can be exchanged between islands based on a migration rate and migration frequency. There are two main advantages for the island model (Skolicki, 2005). First, the existence of independent islands in the island model provide a chance for unfit solutions to evolve which may increase the probability of finding global optimal solutions. Second, the island model can be executed on several processing devices, which can significantly reduce the search time for optimal solutions.

Several parameters and issues have to be considered to incorporate the island model into the framework of any evolutionary algorithm. First, the number of islands and the size of each island where the islands may have the same size or variable sizes. Second, the migration rate which determines the number of candidate solutions to be sent and received among islands after each predefined number of iterations. Third, the migration frequency which determines the number of iterations between migrations. Finally, the decision whether the exchange of candidate solutions among islands should occur at the same time (synchronous migration) or not at the same time (asynchronous migration).

Various migration topologies have been described in (Corcoran and Wainwright, 1994) that can be used to determine the feasible paths for exchanging candidate solutions across islands based on a migration policy (i.e., a selection process that is used to select that candidate solutions to be sent and received between islands). Migration topologies can be categorized into static topologies (e.g., ring, mesh and star) or dynamic topologies (e.g., random ring, random mesh and random star) (Corcoran and Wainwright, 1994).The possible paths between the neighboring islands in static topologies have to be specified before the breeding step and remain static during the evolution, while the possible paths between the neighboring islands in dynamic topologies can be dynamically modified before every migration process. For instance, Figure 1 displays a random ring topology that is represented as a unidirectional-connected graph. The nodes in the graph represent the islands and each unidirectional edge represents the direction of migration between an island and its neighboring island. It is worth noting that each island has one neighboring island in the ring or random ring topologies.

The migration process is triggered when a predefined number of iterations is reached (migration frequency). The migration rate is used to determine the number of migrant candidate solutions to be exchanged between islands based on a migration topology. The migrant solutions are normally selected using a migration policy (Kushida et al., 2013). Migration policies are broadly categorized into either random-based migration policies (e.g., random-random policy) or greedy-based migration policies (e.g., best-

worst policy) (Skolicki, 2005). The random-random policy selects random candidate solutions in an island and then send them to replace random solutions in a neighboring island. On the other hand, the best-worst migration policy selects the best candidate solutions in an island and then send them to replace the worst solutions in a neighboring island.

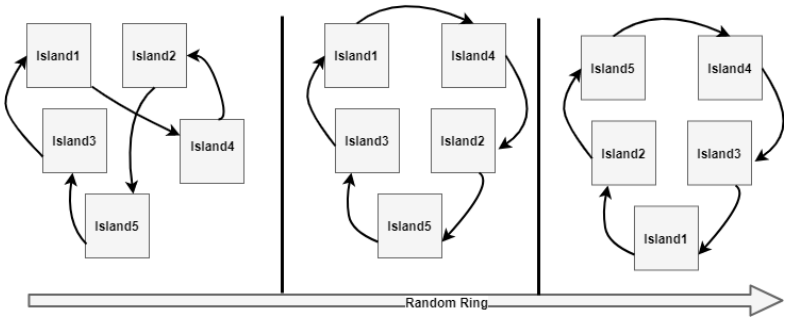

Figure 1: Random ring topology.

3 Cuckoo Search Algorithm

Cuckoo search (CS) is a mathematical optimization algorithm inspired by the obligate brood parasitic behavior of some cuckoo species and the Lévy flight behavior of some fruit flies and birds (Yang and Deb, 2009). The common cuckoo in the wild is an obligate brood parasite because it depends on other birds (host birds) to raise its chicks. At the appropriate moment, the female cuckoo flies down to the nest of a host bird, throws an egg out of the nest, lays an egg and flies away. However, there is a risk that the cuckoo's egg may be discovered and discarded by the host bird. The CS algorithm attempts to solve optimization algorithms based on two simulation models: a simulation model that represents the obligate brood parasitic behavior of the cuckoo birds and another simulation model that represents the Lévy flight behavior of some fruit flies and bird species (Rakhshani and Rahati, 2016; Yang and Deb, 2009).

An existing candidate solution in the population of CS is called an egg and a new generated solution is called a cuckoo's egg. The CS algorithm iteratively attempts to generate new solutions (cuckoos' eggs) using a Lévy fligh function (i.e., a mutation like function based on the Lévy flight operator) for potential replacement with lower quality solutions in the population (eggs in the nests). The original CS algorithm (Figure 2) is controlled by two parameters: the population size n and the fragment of the population to be dis-

carded at each iteration of CS ($p_a \in [0,1]$). The following notations and assumptions formulate the simulation models of CS for continuous optimization problems:

- The ith candidate solution $\boldsymbol{x}^i$ is a vector of m decision variables $\boldsymbol{x}^i = \langle x_1^i, x_2^i ..., x_m^i \rangle$, where each variable x_j is between $[LB_j, UB_j]$. LB_j is the lower bound and UB_j is the upper bound of the j^{th} decision variable.
- The value of a decision variable x_j can be generated using the random-based function $x_j = LB_j + (UB_j - LB_j) \times U(0,1)$ where $U(0,1)$ is a uniform random number between $[0,1]$.
- $f(\boldsymbol{x}^i)$ is the fitness value of the ith candidate solution $\boldsymbol{x}^i$.
- n is a fixed number that denotes the number of candidate solutions (population size). The population of n candidate solutions is conventionally represented as an augmented matrix of size $n \times m$:

$$\textbf{\textit{population}} = \begin{bmatrix} x_1^1 & x_2^1 & \dots & x_m^1 \\ x_1^2 & x_2^2 & \dots & x_m^2 \\ \vdots & \vdots & \dots & \vdots \\ x_1^n & x_2^n & \dots & x_m^n \end{bmatrix} \tag{3.1}$$

- p_a is a floating point number in the range $[0,1]$ that represents a fraction of the population (i.e., solutions with the lowest fitness values) to be replaced with new randomly generated solutions at the last step of each generation of CS.
- The best candidate solutions move to the next generation of CS at the end of each generation.

Figure 2 shows that the evolution process of CS starts from a population of randomly generated population of n nominated solutions $\boldsymbol{x}^i (i = 1, 2, ..., n)$. These solutions are generated from the range of potential solutions of the fitness function $f(\boldsymbol{x}^i)$. Furthermore, the stopping condition or the maximum number of iterations (*MaxItr*) of CS is determined depending on the optimal value/s of $f(\boldsymbol{x}^i)$.
The population in each iteration of CS is called a generation. The CS algorithm iteratively tries to enhance a randomly selected candidate solution ($\boldsymbol{x}_t^i$) at generation $t+1$ using a Lévy function to produce a new candidate solution ($\boldsymbol{x}_{t+1}^i$) as follows:

$$\boldsymbol{x}_{t+1}^i = \boldsymbol{x}_t^i + \beta \oplus \text{Lévy}(\lambda), \tag{3.2}$$

1: **Begin**
2: Objective function $f(\boldsymbol{x}_i)$, where $\boldsymbol{x}_i = \langle x_1, ..., x_m \rangle$ is a nominated solution
3: Generate initial population of n solutions $\boldsymbol{x}_i (i = 1, 2, ..., n)$
4: **while** ($t <$*MaxItr) or (stop criterion)* **do**
5: Select a solution i randomly from the current population and replace its solution $\boldsymbol{x}_i$ by Lévy flights
6: Calculate the quality/fitness value $f(\boldsymbol{x}_i)$ of $\boldsymbol{x}_i$
7: Select a solution randomly from the current population (say, j)
8: **if** $f(\boldsymbol{x}_i)$ is better than $f(\boldsymbol{x}_j)$ **then**
9: Replace $\boldsymbol{x}_j$ by $\boldsymbol{x}_i$
10: **end if**
11: A fraction (p_a) of worst solutions are replaced with new ones;
12: Keep the best solutions (i.e., solutions with quality solutions)
13: Rank the solutions and find the current best;
14: **end while**
15: Post-process results and visualization
16: **End**

Figure 2: The Cuckoo Search Algorithm (CS).

where $\beta > 0$ represents a parameter that specifies the distance of mutation and the symbol $\oplus$ denotes the entry-wise product operator. The Lévy flight operator (Lévy(λ)) is basically a random walk with a step length that is randomly generated from a heavy tailed Lévy distribution with a power law:

$$\text{Lévy} \sim u = D^{-\alpha}, \tag{3.3}$$

where D represents the step size and $\alpha \in (1, 3]$ represents a parameter related to fractal dimension (i.e., a ratio that gives a statistical index of complexity and that compares how detail in a fractal pattern). It is interesting to note that the Lévy distribution function has an infinite variance (Yang and Deb, 2009) which means that some of the generated random solutions using the Lévy function will be near to the local optima, while a large percentage of the generated solutions will be random solutions far from the local optima. This indicates that the Lévy flight function is a better exploratory method than the random walk method because the distance value in the Lévy flight operator gets longer over the course of iterations.

4 Highly Disruptive Polynomial Mutation

The CS algorithm attempts to improve a selected candidate solution using a mutation like function based on the Lévy flight operator. The Lévy flight mutation method uses equation 3.2 to alter the values of decision variables of a candidate solution.
Two factors determine the efficiency of mutation operators: the population part affected by the mutation and the mutation strength (i.e., how far is the generated solution from the initial population in the search space) (Hasan et al., 2014). Mutation methods such as random (Michalewicz, 1994), boundary (Hasan et al., 2014), non-uniform (Michalewicz et al., 1994), MPT (Makinen, Periaux and Toivanen) (Toivanen et al., 1999), power (Deep and Thakur, 2007), highly disruptive polynomial (HDP) (Deb and Tiwari, 2008) and pitch adjustment (Geem et al., 2001) have been used with evolutionary algorithms such as artificial bee colony (Doush et al., 2014) and harmony search (Hasan et al., 2014) to solve continuous optimization problems. The polynomial mutation has been found to be more accurate and stable than the other algorithms for large number of the benchmark functions.
The basic polynomial mutation method (Deb and Agrawal, 1994) tends to get stuck in local optima when the value of the target decision variable for mutation is close to one of its boundaries. The HDP mutation (Deb and Tiwari, 2008) is an improved version of the polynomial mutation method. The algorithm in Figure 3 shows how the HDP method can mutate the value of decision variable x_i. In the algorithm, r is a random number ($0 \leq r \leq 1$), P_m represents the probability of mutation, LB_i denotes the lower boundary of x_i, UB_i denotes the upper boundary of x_i, δ_1 represents the difference between x_i and LB_i divided by $UB_i - LB_i$, δ_2 represents the difference between UB_i and x_i divided by $UB_i - LB_i$, and η_m denotes a non-negative number that represents the distribution index. The main advantage of the HDP method over the polynomial method is that it can explore the entire search space of a decision variable even if the variable's value is near to one of its boundaries.

5 Island-based Cuckoo Search Algorithm

The Cuckoo search (CS) algorithm proved its efficiency in solving complex optimization problems based on the obligate brood parasitic behavior of some cuckoo species and the Lévy flight behavior of some fruit flies and birds (Yang and Deb, 2009). As any other evolutionary algorithm, CS may converge earlier than expected to suboptimal solutions (premature convergence) because of the loss of diversity in its population of candidate

1: $i \leftarrow 0$
2: **repeat**
3: $r \leftarrow U[0,1]$
4: **if** $(r \leq P_m)$ **then**
5: $\delta_1 \leftarrow \frac{x_i - LB_i}{UB_i - LB_i}$
6: $\delta_2 \leftarrow \frac{UB_i - x_i}{UB_i - LB_i}$
7: $r \leftarrow U[0,1]$

$$\delta_k \leftarrow \begin{cases} [(2r) + (1-2r) \times (1-\delta_1)^{\eta_m+1}]^{\frac{1}{\eta_m+1}-1} & \text{If } r \leq 0.5 \\ 1 - [2(1-r) + 2(r-0.5) \times (1-\delta_2)^{\eta_m+1}]^{\frac{1}{\eta_m+1}} & \text{otherwise} \end{cases}$$

8: $x_i \leftarrow x_i + \delta_k.(UB_i - LB_i)$
9: **end if**
10: **until** i++==n

Figure 3: Highly Disruptive Polynomial Mutation.

solutions (Alkhateeb and Abed-alguni, 2017; Abed-alguni and Alkhateeb, 2018).

The premature convergence problem of CS is caused by the optimization operators of CS (i.e., the selection and Lévy flight operators (Section 3)). These operators are not perfect and may not be able to amend and maintain the diversity of the solutions over multiple generations. The hybrid CS algorithms that were discussed in Section 1 (e.g., CSA, CSBHC), attempt to decrease the likelihood of premature convergence of CS, but they require more computational time than the standard CS algorithm. This is because a hybrid CS algorithm normally uses a search method at each iteration of CS which increases its complexity. Other variations of CS (e.g., improved CS) provide insignificant enhancement in the performance of CS and can be applied only to particular applications.

The island model (Section 2) is one of the most popular structured population strategies used to control the diversity problem in optimization algorithms. In addition, an island-based optimization algorithm can be applied to parallel machines. The HDP mutation method (Section 4) has been found to be more accurate and stable than several mutation methods (Doush et al., 2014; Hasan et al., 2014; Abed-alguni and Paul, 2018). This section introduces a distributed variation of CS called island-based CS with polynomial mutation (*i*CSPM). The *i*CSPM algorithm is based on the island model and uses an exploration function based on the HDP mutation method instead of the exploration function that is based the Lévy flight operator. The *i*CSPM algorithm attempts to increase the

probability of finding global optimal solutions by giving a chance to unfit candidate solutions to evolve in isolated islands. In addition, *i*CSPM can be implemented to several processing devices and hence its computation time can be reduced.

In *i*CSPM, the population of candidate solutions is divided equally into k populations (islands) where each island is of size s. The total population of the islands can be represented as $\sum_{i=1}^{k} s$. The evolutionary loop of CS is applied independently to each island at the same time (asynchronous execution). The migration process in *i*CSPM takes place whenever a predefined number of generations (migration frequency (F_m)) is met by exchanging a number of candidate solutions specified by a migration rate (R_m) between the islands. The *i*CSPM algorithm implements the best-worst migration policy which selects the best candidate solutions and send them from one island to replace the worst candidate solutions in a neighboring island. The islands in *i*CSPM are arranged based on a random ring migration topology to determine possible migration paths. In detail, the *i*CSPM algorithm comprises the following steps:

Step 1 **Initialize the problem and *i*CSPM parameters**. In this step, a given optimization problem is modeled as a minimization function $min\{f(\boldsymbol{x}^i)|\boldsymbol{x}^i \in \boldsymbol{X}\}$, where $f(\boldsymbol{x}^i)$ is the objective function of the ith nominated solution that is composed of m decision variables $\boldsymbol{x}^i = \langle x_1^i, x_2^i ..., x_m^i \rangle$. The following parameters are initialized at this step:

(a) The main parameters of CS (the size of population n, the maximum number of iterations *MaxItr* and the fraction of solutions to be discarded p_a).

(b) The parameters of the island model including k which represents the number of islands, s which represents the island size where $s = population/k$, F_m which denotes the migration frequency, and R_m which is the migration rate.

(c) The probability of mutation P_m in the HDP mutation method.

Step 2 **Initialize the population of n nests (candidate solutions)**. The population of n candidate solutions $\boldsymbol{X} = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_n\}$ is represented as an augmented matrix of size $n \times m$.

$$\boldsymbol{X} = \begin{bmatrix} x_1^1 & x_2^1 & \dots & x_m^1 \\ x_1^2 & x_2^2 & \dots & x_m^2 \\ \vdots & \vdots & \dots & \vdots \\ x_1^n & x_2^n & \dots & x_m^n \end{bmatrix} \tag{5.1}$$

The values of the decision variables of each solution $\boldsymbol{x}^i = \{x_1^i, ..., x_N^i\}$ are randomly generated using a random function $x_j^i = LB_j^i + (UB_j^i - LB_j^i) \times U(0,1), \forall j = 1, 2, \ldots, m$, where $U(0,1)$ is a uniform random-based function that generates random numbers in the range 0 and 1.

Step 3 **Divide the population into k islands of size s.** An individual vector is used in the partitioning process $I =< I_1, I_2, ..., I_k >$, where the value range of $I_j \in (1, 2, ..., k)$ and $j \in (1, 2, ..., n)$. The following algorithm adopted from (Al-Betar et al., 2015) shows how the population can be divided into k islands. The output of the algorithm is a set of islands where $\boldsymbol{X} = (SubPop_1, SubPop_2, ...SubPop_k)$, where $SubPop_i$ is the sub-population of island I_i.

$$
\begin{aligned}
\textit{SubPop}_1 &= \begin{bmatrix} x_1^1 & x_2^1 & \ldots & x_m^1 \\ x_1^2 & x_2^2 & \ldots & x_m^2 \\ \vdots & \vdots & \ldots & \vdots \\ x_1^s & x_2^s & \ldots & x_m^s \end{bmatrix} \\
\textit{SubPop}_2 &= \begin{bmatrix} x_1^1 & x_2^1 & \ldots & x_m^1 \\ x_1^2 & x_2^2 & \ldots & x_m^2 \\ \vdots & \vdots & \ldots & \vdots \\ x_1^s & x_2^s & \ldots & x_m^s \end{bmatrix} \\
&\ddots \\
\textit{SubPop}_k &= \begin{bmatrix} x_1^1 & x_2^1 & \ldots & x_m^1 \\ x_1^2 & x_2^2 & \ldots & x_m^2 \\ \vdots & \vdots & \ldots & \vdots \\ x_1^s & x_2^s & \ldots & x_m^s \end{bmatrix}
\end{aligned} \tag{5.2}
$$

1: **Divide-POP()**
2: **Begin**
3: **for** $j = 1...N$ **do**
4: **repeat**
5: $h = U(1...k)$
6: **if** $SubPop_h \leq s$ **then**
7: $I_j = h$
8: **end if**
9: **until** I_j is assigned to isand h
10: **end for**

11: **End**

Step 3 **Generate a cuckoo using the HDP method in each island.** This step of *i*CSPM is applied independently to each island. In this step, a candidate solution (cuckoo, say $\boldsymbol{x}^i$) is selected randomly from the population of an island and then updated using the HDP mutation method (Figure 3). The solution is also evaluated using the objective function $f(\boldsymbol{x}^i)$.

Step 4 **Randomly select a nest from the population in each island.** In each island, a candidate solution (nest, say $\boldsymbol{x}^j$) is randomly selected from the s candidate solutions and then evaluated using the objective function $f(\boldsymbol{x}^j)$.

Step 5 **Compare the objective values of $\boldsymbol{x}^i$ and $\boldsymbol{x}^j$ in each island.** In each island, if $f(\boldsymbol{x}^i) \leq f(\boldsymbol{x}^j)$ (for minimization problems), replace $\boldsymbol{x}^j$ with $\boldsymbol{x}^i$.

Step 6 **Update the population of each island**. In each island, determine the p_a worst solutions and then generate new solutions to replace them.

Step 7 **Rank the solutions in each island.** In each island, ascendingly order the solutions based on their fitness values.

Step 8 **Trigger the migration process between islands.** The migration process is activated each time a predefined number of generations is reached as specified by F_m to send and receive solutions between islands. The isolated islands are randomly arranged to form a unidirectional ring based on the random ring topology. The migrant solutions are transferred between islands based on the best-worst migration policy. Let $SubPop_i = \{\boldsymbol{x}_1^i, \boldsymbol{x}_2^i ..., \boldsymbol{x}_s^i\}$ and $SubPop_j = \{\boldsymbol{x}_1^j, \boldsymbol{x}_2^j ..., \boldsymbol{x}_s^j\}$ be two islands I_i and I_j. Let $R_m = 20\%$, $n = 150$ and $s = 10$, thus the number of migrant solutions is $R_m \times (n/s) = 20\% \times (150/10) = 3$. Let $SubPop_i$ and $SubPop_j$ be two ascendingly ordered lists based on their fitness values where $f(\boldsymbol{x}_1^i) \leq f(\boldsymbol{x}_2^i) \leq ... \leq f(\boldsymbol{x}_s^i)$ and $f(\boldsymbol{x}_1^j) \leq f(\boldsymbol{x}_2^j) \leq ... \leq f(\boldsymbol{x}_s^j)$. Assuming that there is a unidirectional edge from I_i to I_j. Hence, the three best candidate solutions in I_i ($\boldsymbol{x}_1^i, \boldsymbol{x}_2^i, \boldsymbol{x}_3^i$) replace the three worst solutions in I_j ($\boldsymbol{x}_{s-2}^j, \boldsymbol{x}_{s-1}^j, \boldsymbol{x}_s^j$). The same process applies to all of the islands in the ring.

Step 9 **Return the best solution in all islands.** The solutions in each island are ranked based on their fitness values and then the best solution in all islands is calculated.

Step 10 **Check the stopping condition**. **Steps 3-9** are repeated as long as the stopping condition is not satisfied.

6 Experiments

This section is divided into five subsections as follows: Section 6.1 provides a summary of the test functions used in the experiments, Section 6.2 describes the experimental setup for the algorithms, Section 6.3 provides an analysis of the sensitivity of *i*CSPM to its parameters, Section 6.4 provides a comparison between *i*CSPM, CS and two well-known island-based algorithms (island-based Harmony search (*i*HS) (Al-Betar et al., 2015) and island-based genetic algorithm (*i*GA) (Lardeux and Goëffon, 2010)), and finally Section 6.5 provides a discussion about the convergence behavior of *i*CSPM compared to *i*HS and *i*GA.

6.1 Benchmark Functions

The algorithms in the experiments were compared and evaluated using 15 benchmark functions. Tables 1-2 show a summary of these test functions reporting the function name, function definition, search range, optimum value $f(\overrightarrow{X^*})$ and function mode of each benchmark function. These benchmark functions have been utilized in the literature to evaluate the performance of various evolutionary algorithms (Abed-alguni and Klaib, 2018; Abed-alguni and Alkhateeb, 2018; Abed-alguni and Alkhateeb, 2017; Alkhateeb and Abed-alguni, 2017; Omran and Mahdavi, 2008). Note that these test functions provide trade-off between unimodal functions (f_1-f_3, f_5, f_{11}, f_{12}) and multimodal functions (f_4, f_6-f_{10}, f_{13}-f_{15}). In addition, the selected test functions are continuous (i.e., the decision variables in the functions are continuous, meaning that a variable is allowed to take out any value within a range of real values), which means that they are suitable to test and evaluate continuous optimization algorithms (i.e., optimization algorithms designed to solve continuous optimization problems) such as *i*CSPM.

Table 1: Benchmark functions used to evaluate CS variations.

Function Name	Function Definition
f_1 :Sphere Function	$f_1(\boldsymbol{X}) = \sum_{i=1}^{D} x_i^2$

Continued on next page

Table 1 – continued from previous page

Function Name	Function Definition
f_2 :Schwefel's Problem 2.22	$f_2(\boldsymbol{X}) = -\sum_{i=1}^{D}(x_i) + \prod_{i=1}^{D}(x_i)$
f_3 :Step Function	$f_3(\boldsymbol{X}) = \sum_{i=1}^{D}(\lfloor x_i + 0.5 \rfloor)^2$
f_4 :Rosenbrock's Function	$f_4(\mathbf{X}) = \sum_{i=1}^{n-1}\left[100\left(x_{i+1} - x_i^2\right)^2 + (x_i - 1)^2\right]$
f_5 :Rotated Hyper-ellipsoid Function	$f_5(\boldsymbol{X}) = \sum_{i=1}^{D}\left(\sum_{j=1}^{i} x_j\right)^2$
f_6 :Schwefel's problem 2.26	$f_6(X) = -\sum_{i=1}^{n}\left[x_i \sin\left(\sqrt{\lvert x_i \rvert}\right)\right]$
f_7 :Rastrigin's Function	$f_7(\boldsymbol{X}) = \sum_{i=1}^{D}(x_i^2 - 10\cos(2\pi x_i) + 10)$
f_8 :Ackley's Function	$f_8(\mathbf{X}) = -20\, e^{\left(-0.2 \times \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right)} - e^{\left[\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right]} + 20 + e^{(1)}$
f_9 :Griewank's Function	$f_9(\mathbf{X}) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$
f_{10} :Six-Hump Camel-Back Function	$f_{10}(x, y) = 4x^2 - 2.1x^4 + \frac{1}{3}x^6 + xy + 4y^2 + 4y^4$
f_{11} :Shifted Sphere function	$f_{11}(\boldsymbol{X}) = \sum_{i=1}^{D} z_i^2 + f\_bias_1$, where $\boldsymbol{z} = \boldsymbol{X} - \boldsymbol{o}$
f_{12} :Shifted Schwefel's problem 1.2	$f_{12}(\boldsymbol{X}) = \sum_{i=1}^{D}(\sum_{j=1}^{i} z_j)^2 + f\_bias_2$, where $\boldsymbol{z} = \boldsymbol{X} - \boldsymbol{o}$
f_{13} :Shifted Rosenbrock's Function	$f_{13}(\boldsymbol{X}) = \sum_{i=1}^{D}(100(z_i^2 - z_{i+1})^2 + (z_i - 1)^2) + f\_bias_{13}$, where $\boldsymbol{z} = \boldsymbol{X} - \boldsymbol{o} + 1$
f_{14} :Shifted Rastrigin's Function	$f_{14}(\boldsymbol{X}) = \sum_{i=1}^{D}(z_i^2 - 10\cos(2\pi z_i) + 10) + f\_bias_{14}$, where $\boldsymbol{z} = \boldsymbol{X} - \boldsymbol{o}$

Continued on next page

Table 1 – continued from previous page

Function Name	Function Definition
f_{15} :Shifted Expanded Griewank's plus Rosenbrock's function	$f_{15}(\boldsymbol{X})$ (for the definition see CEC 2005)

Table 2: Details of Benchmark functions.

Function Name	Search Range	Global optimum	Function Mode
f_1	$x_i \in [-100, 100]$	$\min(f_1) = f(0, \ldots, 0) = 0$	unimodal
f_2	$x_i \in [-10, 10]$	$\min(f_2) = f(0, \ldots, 0) = 0$	unimodal
f_3	$x_i \in [-100, 100]$	$\min(f_3) = f(0, \ldots, 0) = 0$	unimodal
f_4	$x_i \in [-2.048, 2.048]$	$\min(f_4) = f4(1, \ldots, 1) = 0$	multimodal
f_5	$x_i \in [-100, 100]$	$\min(f_5) = f(0, \ldots, 0) = 0$	unimodal
f_6	$x_i \in [-500, 500]$	$\min(f_6) = f(420.9687, \ldots, 420.9687) = -12569.5$	multimodal
f_7	$x_i \in [-5.12, 5.12]$	$\min(f_7) = f(0, \ldots, 0) = 0$	multimodal
f_8	$x_i \in [-32, 32]$	$\min(f_8) = f(0, \ldots, 0) = 0$	multimodal
f_9	$x_i \in [-600, 600]$	$\min(f_9) = f(0, \ldots, 0) = 0$	multimodal
f_{10}	$x_i \in [-5, 5]$	$\min(f_{10}) = f_{10}(0.08983, 0.712) = -1.031628$	multimodal
f_{11}	$x_i \in [-100, 100]$	$\min(f_{11}) = f(o_1, \ldots, o_D) = f\_bias_{11} = -450$	unimodal
f_{12}	$x_i \in [-100, 100]$	$\min(f_{12}) = f(o_1, \ldots, o_D) = f\_bias_{12} = -450$	unimodal
f_{13}	$x_i \in [-100, 100]$	$\min(f_{13}) = f(o_1, \ldots, o_D) = f\_bias_{13} = 390$	multimodal
f_{14}	$x_i \in [-5, 5]$	$\min(f_{14}) = f(o_1, \ldots, o_D) = f\_bias_{14} = -330$	multimodal
f_{15}	$x_i \in [-5, 5]$	$\min(f_{15}) = f(o_1, \ldots, o_D) = f\_bias_{15} = -130$	multimodal

6.2 Setup

The experiments were conducted using an Intel Core i5 6th Gen CPU (3.4 GHz) with 8 GB RAM running macOS 10.13, High Sierra. All of the algorithms were implemented

using Java programming language.
All the variations of CS used the same parameter setting as in (Abed-alguni and Alkhateeb, 2017; Alkhateeb and Abed-alguni, 2017; Abed-alguni, 2017a): population size $n = 1000$ and percentage of abandon $p_a = 0.25$. The values of n and p_a were dynamically tuned over multiple simulations in (Abed-alguni and Alkhateeb, 2017). The test functions were implemented with the dimensions (i.e., number of decision variables) 10 and 100 except for the six-hump camel-back function which is a two dimensional function. The maximum number of iterations of each algorithm is 10,000.

6.3 Sensitivity analysis of *i*CSPM to its parameters

This section investigates different settings of the island model in order to discover which of these settings increase the convergence speed of *i*CSPM towards good solutions. The parameters of the island model (number of islands k, migration frequency F_m, and migration rate R_m) were investigated based on different experimental cases (Table 3) to measure their effect on the convergence speed of *i*CSPM. The purpose of the first three cases in the table is to measure the effect of s on the convergence behavior of *i*CSPM. The purpose of the second three experimental cases is to measure the effect of F_m on the convergence behavior of *i*CSPM. Finally, the purpose of the last three experimental cases is to measure the impact of R_m on the convergence behavior of *i*CSPM.

Table 3: Nine experimental cases used to evaluate the sensitivity of *i*CSPM to its parameters.

Experimental Case	s	F_m	R_m (%)
Case1	2	100	20
Case2	5	100	20
Case3	10	100	20
Case4	10	50	20
Case5	10	100	20
Case6	10	500	20
Case7	10	50	10
Case8	10	50	20
Case9	10	50	30

Tables 4-6 report the results of the nine experimental cases in Table 3 for the 15 test

functions in Table 1. The dimension of each test function was 10 except for the six-hump camel-back function which is a two dimensional function. The values in the tables are the average of 50 independent runs for the best objective values. The best solutions are highlighted in **bold**.

Table 4 summarizes the results produced for the first three cases (Case 1, Case 2, and Case 3). Obviously, the results in Table 4 indicate that the performance of *i*CSPM gets better with every increase in the value of s. This is because 10 search regions will be explored simultaneous when s is 10 compared to 2 regions when s is 2 and 5 regions when s is 5. In addition, higher values of s lead to better diffusion on the population. However, any increase in the value of s increases the computational complexity of *i*CSPM. Hence, it is recommended to run the *i*CSPM algorithm in parallel devices to control its computational complexity. Based on the results in Table 4, $s = 10$ was selected to perform the other consecutive six cases (Cases 4 to 9).

Table 4: The effect of various values of s on the convergence behavior of *i*CSPM. The dimension is 10, number of iterations is 10,000 and number of runs is 50.

Problem	Case1 $s = 2$	Case2 $s = 5$	Case3 $s = 10$
f_1	0.00E+00	0.00E+00	0.00E+00
f_2	0.00E+00	0.00E+00	0.00E+00
f_3	0.00E+00	0.00E+00	0.00E+00
f_4	8.29E+02	8.29E+02	8.29E+02
f_5	0.00E+00	0.00E+00	0.00E+00
f_6	2.09E+05	8.38E+04	**4.19E+04**
f_7	0.00E+00	0.00E+00	0.00E+00
f_8	4.44E-16	4.44E-16	4.44E-16
f_9	0.00E+00	0.00E+00	0.00E+00
f_{10}	-1.0307	**-1.0310**	-1.0301
f_{11}	-4.41E+02	-4.42E+02	**-4.46E+02**
f_{12}	-3.45E+02	-3.32E+02	**-3.46E+02**
f_{13}	5.25E+02	**3.92E+02**	**3.92E+02**
f_{14}	-3.274E+02	-3.289E+02	**-3.298E+02**
f_{15}	-7.92E+01	-8.9142E+01	**-1.03E+02**

Table 5 summarizes the results calculated for the second three cases (Case 4, Case 5,

and Case 4). Note that the results in Case 3 were repeated in Case 5. The purpose of Table 5 is to study the effect of different values of F_m on the convergence behavior of *i*CSPM. The best results produced in Table 5 were mostly obtained by Case 4 ($F_m = 50$) followed by Case 5 ($F_m = 100$) and finally Case 6 ($F_m = 500$). These results indicate that low-migration frequencies lead to better results than large-migration frequencies. This may because large-scale migrations negatively affect the diversity of populations in the islands while low-scale migrations provide better chances for the migrant solutions to move without affecting the diversity of any island and so convergence will take longer to occur.

Table 5: The effect of various values of F_m on the convergence behavior of *i*CSPM. The dimension is 10, number of iterations is 10,000 and number of runs is 50.

Problem	Case4 $F_m = 50$	Case5 $F_m = 100$	Case6 $F_m = 500$
f_1	0.00E+00	0.00E+00	0.00E+00
f_2	0.00E+00	0.00E+00	0.00E+00
f_3	0.00E+00	0.00E+00	0.00E+00
f_4	8.29E+02	8.29E+02	8.29E+02
f_5	0.00E+00	0.00E+00	0.00E+00
f_6	**4.1864E+04**	4.1866E+04	4.1867E+04
f_7	0.00E+00	0.00E+00	0.00E+00
f_8	4.44E-16	4.44E-16	4.44E-16
f_9	0.00E+00	0.00E+00	0.00E+00
f_{10}	-1.0300	**-1.0301**	-1.0265
f_{11}	-4.43E+02	**-4.46E+02**	-4.37E+02
f_{12}	**-3.68E+02**	-3.46E+02	-2.73E+02
f_{13}	3.94E+02	**3.92E+02**	8.95E+02
f_{14}	**-3.300E+02**	-3.298E+02	-3.274E+02
f_{15}	-9.36E+01	-1.03E+02	**-1.10E+02**

Table 6 shows the produced results for Case 7, Case 8 and Case 9. Note that the value of $F_m = 50$ which produced the best results in Table 5 was selected to be used in the last three convergence cases. There is no indication in Table 6 of the effects of low and high values of R_m on the performance of *i*CSPM which may be because of the small dimension of the test problems.

Table 6: The effect of various values of R_m on the convergence behavior of *i*CSPM. The dimension is 10, number of iterations is 10,000 and number of runs is 50.

Problem	Case7 $R_m = 10\%$	Case8 $R_m = 20\%$	Case9 $R_m = 30\%$
f_1	0.00E+00	0.00E+00	0.00E+00
f_2	0.00E+00	0.00E+00	0.00E+00
f_3	0.00E+00	0.00E+00	0.00E+00
f_4	8.29E+02	8.29E+02	8.29E+02
f_5	0.00E+00	0.00E+00	0.00E+00
f_6	4.1865E+04	**4.1864E+04**	4.1865E+04
f_7	0.00E+00	0.00E+00	0.00E+00
f_8	4.44E-16	4.44E-16	4.44E-16
f_9	0.00E+00	0.00E+00	0.00E+00
f_{10}	**-1.0311**	-1.0300	**-1.0311**
f_{11}	**-4.43E+02**	**-4.43E+02**	-4.42E+02
f_{12}	-3.42E+02	**-3.68E+02**	-3.53E+02
f_{13}	**3.93E+02**	3.94E+02	**3.93E+02**
f_{14}	-3.300E+02	-3.300E+02	-3.300E+02
f_{15}	-9.50E+01	-9.36E+01	**-9.89E+01**

Tables 7-9 report the results of the nine experimental cases in Table 3 for the 15 test functions in Table 1. The dimension of each test function was 100 except for the six-hump camel-back function which is a two dimensional function. The values in the tables are the average of 50 independent runs for the best objective values. The best solutions are highlighted in **bold**.

The obtained results in Table 7 and Table 8 for the problems with dimension 100 confirm the observations that were previously discussed about Table 4 and Table 5 for the problems with dimension 10. The results in Table 7 suggest that the average objective values get better with every increase in the value of s. The results in Table 8 indicate that low-scale migrations provide better results than low-scale migrations.

Interestingly, there was no indication of the impact of low and high migration rates on the convergence behavior of *i*CSPM for the problems of dimension 10 (Table 6). But, when the problem dimensionality was increased to 100 (Table 9), low migration rates enhanced the performance of *i*CSPM compared to high migration rates.

Table 10 shows a summary of Tables 4 to 9. The overall results suggest that large values of s and low values of F_m and R_m enhance the performance of *i*CSPM.

Table 7: The effect of various values of s on the convergence behavior of *i*CSPM. The dimension is 100, number of iterations is 10,000 and number of runs is 50.

Problem	Case1 $s = 2$	Case2 $s = 5$	Case3 $s = 10$
f_1	0.00E+00	0.00E+00	0.00E+00
f_2	0.00E+00	0.00E+00	0.00E+00
f_3	0.00E+00	0.00E+00	0.00E+00
f_4	9.83E+03	9.83E+03	9.83E+03
f_5	0.00E+00	0.00E+00	0.00E+00
f_6	2.0937E+05	8.3696E+04	**4.1781E+04**
f_7	0.00E+00	0.00E+00	0.00E+00
f_8	4.44E-16	4.44E-16	4.44E-16
f_9	0.00E+00	0.00E+00	0.00E+00
f_{10}	-1.0307	**-1.0310**	-1.0301
f_{11}	2.12E+02	2.26E+02	**1.67E+02**
f_{12}	1.98E+06	1.91E+06	**1.81E+06**
f_{13}	6.45E+03	5.57E+02	**4.50E+02**
f_{14}	**-3.2974E+02**	-3.2882E+02	-3.2920E+02
f_{15}	4.18E+06	3.4586E+06	**3.0247E+06**

6.4 Comparison between *i*CSPM and other algorithms

The performance of *i*CSPM (Case8) was compared against the performance of Cuckoo search (CS) (Yang and Deb, 2009), island-based Harmony search (*i*HS) (Al-Betar et al., 2015) and island-based genetic algorithm (*i*GA) (Lardeux and Goëffon, 2010) as shown in Table 11. The parameters of the algorithms in Table 11 were set as follows:

1. For the *i*HS algorithm, the harmony memory size = 100, harmony memory consideration rate= 0.98, pitch adjustment rate= 0.30 and fret width= 0.01 as recommended in (Al-Betar et al., 2015; Al-Betar and Khader, 2012).
2. For the *i*GA algorithm, the mutation rate= 0.05 and crossover rate= 0.7 as in (Lardeux and Goëffon, 2010).

Table 8: The effect of various values of F_m on the convergence behavior of *i*CSPM. The dimension is 100, number of iterations is 10,000 and number of runs is 50.

Problem	Case4 $F_m = 50$	Case5 $F_m = 100$	Case6 $F_m = 500$
f_1	0.00E+00	0.00E+00	0.00E+00
f_2	0.00E+00	0.00E+00	0.00E+00
f_3	0.00E+00	0.00E+00	0.00E+00
f_4	9.83E+03	9.83E+03	9.83E+03
f_5	0.00E+00	0.00E+00	0.00E+00
f_6	**4.1776E+04**	4.1781E+04	4.1792E+04
f_7	0.00E+00	0.00E+00	0.00E+00
f_8	4.44E-16	4.44E-16	4.44E-16
f_9	0.00E+00	0.00E+00	0.00E+00
f_{10}	-1.0300E	**-1.0301**	-1.0265E
f_{11}	**1.57E+02**	1.67E+02	1.76E+02
f_{12}	**1.69E+06**	1.81E+06	1.95E+06
f_{13}	**4.01E+02**	4.50E+02	5.53E+02
f_{14}	-3.2929E+02	-3.2920E+02	**-3.293E+02**
f_{15}	3.85E+06	**3.0247E+06**	3.13E+06

3. For all the island-based algorithms, $s = 10$, $F_m = 50$ and $R_m = 10\%$.

4. The dimension of each benchmark function is 10 except for the six-hump camel-back function (f_{10}) which is a two dimensional function.

Tables 11 shows the experimental results with respect to each of the 15 benchmark functions described in Table 1. The average of 50 independent runs for the best solutions are shown in the table. The results in the Table show that *i*CSPM outperformed the other algorithms for 11 out of 15 functions. The results also show that *i*HS is the second best performing algorithm (best results for 3 out of 15 functions).

6.5 Convergence Behavior of *i*CSPM compared to *i*HS and *i*GA

Three difficult benchmark functions (f_4 (Figure 4), f_{13} (Figure 5), f_{14} (Figure 6)) have been selected to show the convergence behavior of *i*CSPM compared to *i*HS and *i*GA

Table 9: The effect of various values of R_m on the convergence behavior of *i*CSPM. The dimension is 100, number of iterations is 10,000 and number of runs is 50.

Problem	Case7 $R_m = 10\%$	Case8 $R_m = 20\%$	Case9 $R_m = 30\%$
f_1	0.00E+00	0.00E+00	0.00E+00
f_2	0.00E+00	0.00E+00	0.00E+00
f_3	0.00E+00	0.00E+00	0.00E+00
f_4	9.83E+03	9.83E+03	9.83E+03
f_5	0.00E+00	0.00E+00	0.00E+00
f_6	**4.1776E+04**	**4.1776E+04**	4.1783E+04
f_7	0.00E+00	0.00E+00	0.00E+00
f_8	4.44E-16	4.44E-16	4.44E-16
f_9	0.00E+00	0.00E+00	0.00E+00
f_{10}	**-1.0311**	-1.0300	**-1.0311**
f_{11}	**1.54E+02**	1.57E+02	1.61E+02
f_{12}	1.80E+06	**1.69E+06**	1.78E+06
f_{13}	4.47E+02	**4.01E+02**	4.51E+02
f_{14}	**-3.299E+02**	-3.2929E+02	-3.298E+02
f_{15}	3.54E+06	3.85E+06	**3.46E+06**

during the first 1,000 iterations. Figure 5 and Figure 6 show that *i*CSPM converges to a solution faster than *i*HS and *i*GA. This is expected because *i*CSPM is not only based on the island model but it also uses an exploration function based on the HDP mutation method. The HDP mutation has been found in (Doush et al., 2014; Hasan et al., 2014) to be more accurate and stable than many mutation methods. This may be because the HDP method can explore the entire search space of a decision variable even if the variable's value is close to one of its boundaries. In Figure 4, *i*HS is the fastest algorithm followed by *i*CSPM in approaching a solution for f_4.This is possibly because the pitch adjustment mutation method in *i*HS is an efficient method for generating diverse population as suggested in (Hasan et al., 2014; Doush et al., 2014).

Table 10: Summary of sensitivity analysis of *i*CSPM to its parameters (Tables 4 to 9).

Problem Size	Case 1	Case 2	Case 3	Case 4	Case 5	Case 6	Case 7	Case 8	Case 9
D=10	0	2	6	3	3	1	3	3	3
D=100	1	1	5	4	2	1	4	3	1
Total	1	3	11	7	5	2	7	6	4

Table 11: Simulation results of *i*CSPM compared to other algorithms. The dimension is 10, number of iterations is 10,000 and number of runs is 50.

Problem	CS	*i*CSPM	*i*HS	*i*GA
f_1	1.03E-05	**0.00E+00**	5.44E-11	2.22E-07
f_2	3.60E-03	**0.00E+00**	1.41E-09	6.48E-05
f_3	0.00E+00	0.00E+00	0.00E+00	0.00E+00
f_4	**8.29E+02**	**8.29E+02**	4.84E+03	7.65E+04
f_5	1.13E-02	**0.00E+00**	**0.00E+00**	2.35E-04
f_6	4.1896E+05	4.1864E+04	**5.2300E+02**	1.2594E+04
f_7	1.11E-01	**0.00E+00**	6.20E-04	6.24E-04
f_8	1.32E-03	**4.44E-16**	9.26E-05	8.25E-04
f_9	7.56E-06	**0.00E+00**	9.11E-03	8.48E-04
f_{10}	-9.3353E-01	**-1.0300E+00**	**-1.0300E+00**	**-1.0300E+00**
f_{11}	-4.16E+02	**-4.43E+02**	5.22E+02	1.23E+03
f_{12}	1.01E+02	-3.68E+02	**-4.01E+02**	-3.58E+02
f_{13}	4.14E+09	**3.94E+02**	3.98E+02	4.03E+02
f_{14}	-2.5273E+02	**-3.300E+02**	-3.286E+02	-2.994E+02
f_{15}	-1.05E+02	-9.36E+01	**-1.26E+02**	-5.54E+01

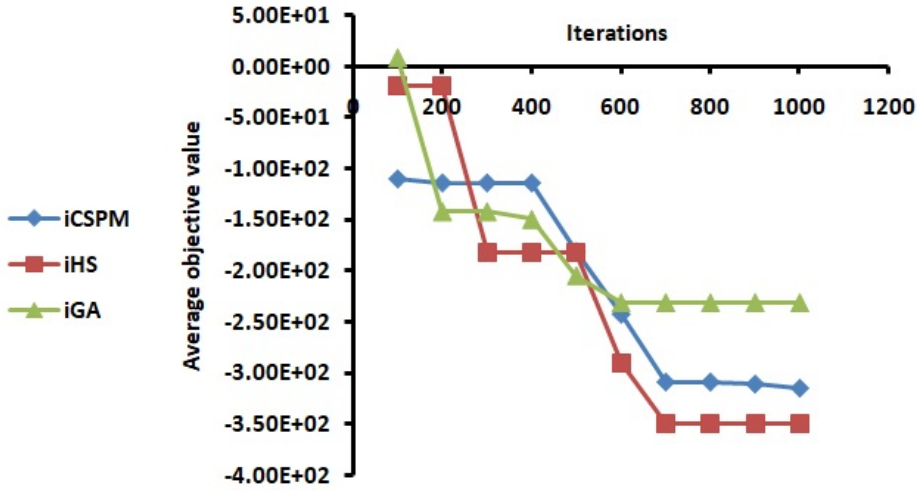

Figure 4: Convergence plots of three island-based algorithms for f_4.

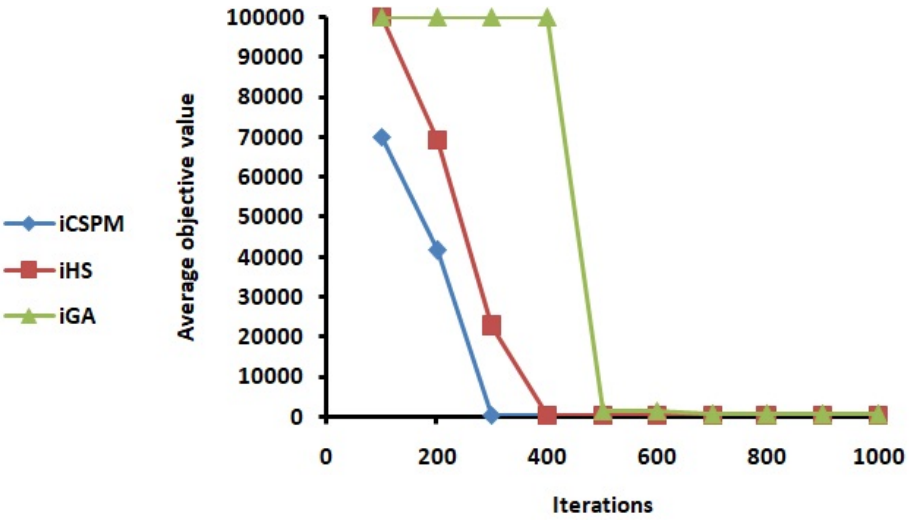

Figure 5: Convergence plots of three island-based algorithms for f_{13}.

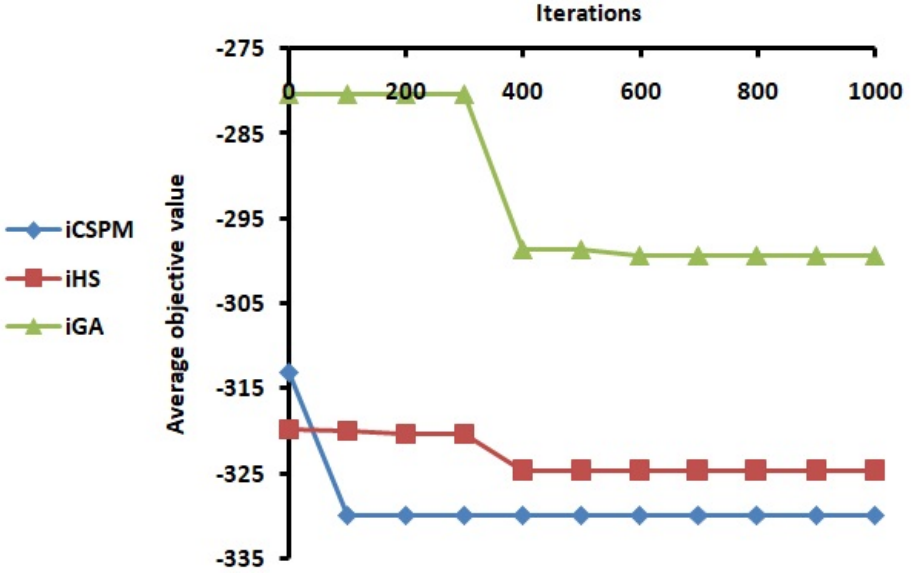

Figure 6: Convergence plots of three island-based algorithms for f_{14}.

7 Conclusion

The current paper presented a new variation of CS called island-based CS with polynomial mutation (*i*CSPM). The *i*CSPM algorithm is based on the island model and uses an exploration function based on the HDP mutation method instead of the exploration function that is based the Lévy flight operator. The *i*CSPM algorithm attempts to increase the probability of finding global optimal solutions by giving a chance to unfit candidate solutions to evolve in isolated islands.

Extensive simulations were conducted using a set of 15 well-known benchmark functions to evaluate the performance of *i*CSPM compared to the performance of Cuckoo search (CS) (Yang and Deb, 2009), island-based Harmony search (*i*HS) (Al-Betar et al., 2015) and island-based Genetic algorithm (*i*GA) (Lardeux and Goëffon, 2010). *i*CSPM was found to be more accurate than the other algorithms for a large number of the benchmark functions. These results indicate that the island model enhances the diversity of *i*CSPM and the HDP mutation method enhances the exploration of CS.

Furthermore, the parameters of *i*CSPM that are related to the island model (number of islands s, migration frequency F_m, and migration rate R_m) were investigated based on different experimental cases. The overall experimental results indicate that large values of s and low values of F_m and R_m enhance the performance of *i*CSPM.

Future work will be directed to implementing *i*CSPM to cooperative Q-learning (Abed-alguni, Chalup, Henskens and Paul, 2015a; Abed-alguni, Chalup, Henskens and Paul, 2015b; Abed-alguni and Ottom, 2018; Abed-alguni, Paul, Chalup and Henskens, 2016; Abed-alguni, 2014) as described in (Abed-alguni, 2017a; Abed-alguni, 2017b).

References

Abed-alguni, B. H. (2017a). Action-selection method for reinforcement learning based on cuckoo search algorithm, *Arabian Journal for Science and Engineering* **0**(0): to appear.

Abed-alguni, B. H. (2017b). Bat Q-learning algorithm, *Jordanian Journal of Computers and Information Technology (JJCIT)* **3**(1): 56–77.

Abed-alguni, B. H. and Alkhateeb, F. (2017). Novel selection schemes for cuckoo search, *Arabian Journal for Science and Engineering* **42**(8): 3635–3654.

Abed-alguni, B. H. and Alkhateeb, F. (2018). Intelligent hybrid cuckoo search and β-hill climbing algorithm, *Journal of King Saud University - Computer and Information Sciences* **0**(0): 1–43.

Abed-alguni, B. H. and Barhoush, M. (2018). Distributed grey wolf optimizer for numerical optimization problems, *Jordanian Journal of Computers and Information Technology* **4**(3): 130–149.

Abed-alguni, B. H., Chalup, S. K., Henskens, F. A. and Paul, D. J. (2015a). Erratum to: A multi-agent cooperative reinforcement learning model using a hierarchy of consultants, tutors and workers, *Vietnam Journal of Computer Science* **2**(4): 227–227.

Abed-alguni, B. H., Chalup, S. K., Henskens, F. A. and Paul, D. J. (2015b). A multi-agent cooperative reinforcement learning model using a hierarchy of consultants, tutors and workers, *Vietnam Journal of Computer Science* **2**(4): 213–226.

Abed-alguni, B. H. K. (2014). *Cooperative reinforcement learning for independent learners*, PhD thesis, Faculty of Engineering and Built Environment, School of Electrical Engineering and Computer Science, The University of Newcastle, Australia.

Abed-alguni, B. H. and Klaib, A. F. (2018). Hybrid whale optimization and β-hill climbing algorithm, *International Journal of Computing Science and Mathematics* **0**(0): 1–13.

Abed-alguni, B. H., Klaib, A. F. and Nahar, K. M. (2019). Island-based whale optimization algorithm for continuous optimization problems, *International Journal of Reasoning-based Intelligent Systems* **0**(0): 1–11.

Abed-alguni, B. H. and Ottom, M. A. (2018). Double delayed Q-learning, *International Journal of Artificial Intelligence™* **16**(2): 41–59.

Abed-alguni, B. H. and Paul, D. J. (2018). Hybridizing the cuckoo search algorithm with different mutation operators for numerical optimization problems, *Journal of Intelligent Systems* pp. 1–32.

Abed-alguni, B. H., Paul, D. J., Chalup, S. K. and Henskens, F. A. (2016). A comparison study of cooperative Q-learning algorithms for independent learners, *International Journal of Artificial Intelligence™* **14**(1): 71–93.

Al-Betar, M. A. (2016). β-hill climbing: an exploratory local search, *Neural Computing and Applications* pp. 1–16.

Al-Betar, M. A. and Awadallah, M. A. (2018). Island bat algorithm for optimization, *Expert Systems with Applications* **107**: 126–145.

Al-Betar, M. A., Awadallah, M. A., Khader, A. T. and Abdalkareem, Z. A. (2015). Island-based harmony search for optimization problems, *Expert Systems with Applications* **42**(4): 2026–2035.

Al-Betar, M. A. and Khader, A. T. (2012). A harmony search algorithm for university course timetabling, *Annals of Operations Research* **194**(1): 3–31.

Alkhateeb, F. and Abed-alguni, B. H. (2017). A hybrid cuckoo search and simulated annealing algorithm, *Journal of Intelligent Systems* p. to appear.

Corcoran, A. L. and Wainwright, R. L. (1994). A parallel island model genetic algorithm for the multiprocessor scheduling problem, *Proceedings of the 1994 ACM symposium on Applied computing, Phoenix, Arizona, USA*, ACM, New York, NY, USA, pp. 483–487.

Deb, K. and Agrawal, R. B. (1994). Simulated binary crossover for continuous search space, *Complex Systems* **9**(3): 1–15.

Deb, K. and Tiwari, S. (2008). Omni-optimizer: A generic evolutionary algorithm for single and multi-objective optimization, *European Journal of Operational Research* **185**(3): 1062–1087.

Deep, K. and Thakur, M. (2007). A new mutation operator for real coded genetic algorithms, *Applied mathematics and Computation* **193**(1): 211–230.

Doush, I. A., Hasan, B. H. F., Al-Betar, M. A., Al Maghayreh, E., Alkhateeb, F. and Hamdan, M. (2014). Artificial bee colony with different mutation schemes: A comparative study., *Computer Science Journal of Moldova* **22**(1).

Feng, Y., Wang, G.-G. and Gao, X.-Z. (2016). A novel hybrid cuckoo search algorithm with global harmony search for 0-1 knapsack problems, *International Journal of Computational Intelligence Systems* **9**(6): 1174–1190.

Geem, Z. W., Kim, J. H. and Loganathan, G. (2001). A new heuristic optimization algorithm: harmony search, *Simulation* **76**(2): 60–68.

Hasan, B. H. F., Doush, I. A., Al Maghayreh, E., Alkhateeb, F. and Hamdan, M. (2014). Hybridizing harmony search algorithm with different mutation operators for continuous problems, *Applied Mathematics and Computation* **232**: 1166–1182.

Kushida, J.-i., Hara, A., Takahama, T. and Kido, A. (2013). Island-based differential evolution with varying subpopulation size, *2013 IEEE Sixth International Workshop on Computational Intelligence & Applications (IWCIA)*, IEEE, Hiroshima, Japan, pp. 119–124.

Lardeux, F. and Goëffon, A. (2010). A dynamic island-based genetic algorithms framework, *Asia-Pacific Conference on Simulated Evolution and Learning, Kanpur, India*, SEAL'10, Springer, Berlin, Heidelberg, pp. 156–165.

Liao, Q., Zhou, S., Shi, H. and Shi, W. (2017). Parameter estimation of nonlinear systems by dynamic cuckoo search, *Neural Computation* **29**(4): 1103–1123.

Marichelvam, M. (2012). An improved hybrid cuckoo search (ihcs) metaheuristics algorithm for permutation flow shop scheduling problems, *International Journal of Bio-Inspired Computation* **4**(4): 200–205.

Marichelvam, M. and Geetha, M. (2016). A hybrid cuckoo search metaheuristic algorithm for solving single machine total weighted tardiness scheduling problems with sequence dependent setup times, *International Journal of Computational Complexity and Intelligent Algorithms* **1**(1): 23–34.

Marichelvam, M., Prabaharan, T. and Yang, X.-S. (2014). Improved cuckoo search algorithm for hybrid flow shop scheduling problems to minimize makespan, *Applied Soft Computing* **19**: 93–101.

Marichelvam, M. and Tosun, Ö. (2016). Performance comparison of cuckoo search algorithm to solve the hybrid flow shop scheduling benchmark problems with makespan criterion, *International Journal of Swarm Intelligence Research (IJSIR)* **7**(2): 1–14.

Michalewicz, Z. (1994). *Genetic Algorithms + Data Structures = Evolution Programs (2Nd, Extended Ed.)*, Springer-Verlag New York, Inc., New York, NY, USA.

Michalewicz, Z., Logan, T. and Swaminathan, S. (1994). Evolutionary operators for continuous convex parameter spaces, *Proceedings of the 3rd Annual conference on Evolutionary Programming, San Diego, California, USA*, World Scientific, River Edge, NJ, pp. 84–97.

Michel, R. and Middendorf, M. (1998). An island model based ant system with lookahead for the shortest supersequence problem, *International Conference on Parallel Problem Solving from Nature, Amsterdam, The Netherlands*, Springer, pp. 692–701.

Omran, M. G. H. and Mahdavi, M. (2008). Global-best harmony search, *Applied Mathematics and Computation* **198**(2): 643–656.

Precup, R.-E., David, R.-C. and Petriu, E. M. (2017). Grey wolf optimizer algorithm-based tuning of fuzzy control systems with reduced parametric sensitivity, *IEEE Transactions on Industrial Electronics* **64**(1): 527–534.

Rakhshani, H. and Rahati, A. (2016). Intelligent multiple search strategy cuckoo algorithm for numerical and engineering optimization problems, *Arabian Journal for Science and Engineering* pp. 1–27.

Romero, J. F. and Cotta, C. (2005). Optimization by island-structured decentralized particle swarms, *Computational Intelligence, Theory and Applications*, Springer, pp. 25–33.

Saadat, J., Moallem, P. and Koofigar, H. (2017). Training echo state neural network using harmony search algorithm, *International Journal of Artificial Intelligence* **15**(1): 163–179.

Skolicki, Z. (2005). An analysis of island models in evolutionary computation, *Proceedings of the 7th annual workshop on Genetic and evolutionary computation, Washington DC, USA*, ACM, New York, NY, USA, pp. 386–389.

Srivastav, A. and Agrawal, S. (2018). Multi-objective optimization of mixture inventory system experiencing order crossover, *Annals of Operations Research* pp. 1–18.

Toivanen, J., Makinen, R., Périaux, J. and Cloud Cedex, F. (1999). Multidisciplinary shape optimization in aerodynamics and electromagnetics using genetic algorithms, *Intl J. Numer. Meth. Fluids* **30**: 149–159.

Vaščák, J. (2012). Adaptation of fuzzy cognitive maps by migration algorithms, *Kybernetes* **41**(3/4): 429–443.

Walton, S., Hassan, O., Morgan, K. and Brown, M. (2011). Modified cuckoo search: a new gradient free optimisation algorithm, *Chaos, Solitons & Fractals* **44**(9): 710–718.

Yang, X.-S. and Deb, S. (2009). Cuckoo search via lévy flights, *World Congress on Nature & Biologically Inspired Computing, 2009. NaBIC 2009, Coimbatore, India*, IEEE, pp. 210–214.

Zhou, Y. and Tan, Y. (2011). Gpu-based parallel multi-objective particle swarm optimization, *International Journal of Artificial Intelligence* **4**(3): 130–149.