Soft Computing (2022) 26:3293–3312
https://doi.org/10.1007/s00500-021-06665-6

OPTIMIZATION

Island-based Cuckoo Search with elite opposition-based learning and multiple mutation methods for solving optimization problems

Bilal H. Abed-alguni[1] · David Paul[2]

Accepted: 8 December 2021 / Published online: 29 January 2022

Abstract
The island Cuckoo Search (*i*CSPM) algorithm is a variation of Cuckoo Search that uses the island model and highly disruptive polynomial mutation to solve optimization problems. This article introduces an improved *i*CSPM algorithm called *i*CSPM with elite opposition-based learning and multiple mutation methods (*i*CSPM2). *i*CSPM2 has three main characteristics. Firstly, it separates candidate solutions into several islands (sub-populations) and then divides the islands among four improved Cuckoo Search algorithms: Cuckoo Search via Lévy flights, Cuckoo Search with highly disruptive polynomial mutation, Cuckoo Search with Jaya mutation and Cuckoo Search with pitch adjustment mutation. Secondly, it uses elite opposition-based learning to improve its convergence rate and exploration ability. Finally, it makes continuous candidate solutions discrete using the smallest position value method. A set of 15 popular benchmark functions indicate *i*CSPM2 performs better than *i*CSPM. However, based on sensitivity analysis of both algorithms, convergence behavior seems sensitive to island model parameters. Further, the single-objective IEEE-CEC 2014 functions were used to evaluate and compare the performance of *i*CSPM2 to four well-known swarm optimization algorithms: distributed grey wolf optimizer, distributed adaptive differential evolution with linear population size reduction evolution, memory-based hybrid dragonfly algorithm and fireworks algorithm with differential mutation. Experimental and statistical results suggest *i*CSPM2 has better performance than the four other algorithms. *i*CSPM2's performance was also shown to be favorable compared to two powerful discrete optimization algorithms (generalized accelerations for insertion-based heuristics and memetic algorithm with novel semi-constructive crossover and mutation operators) using a set of Taillard's benchmark instances for the permutation flow shop scheduling problem.

Keywords Island model · Diversity · Structured population · Cuckoo Search · Lévy flights · Highly disruptive polynomial mutation · Pitch adjustment mutation · Jaya mutation · Elite opposition-based learning

1 Introduction

Exact algorithms are usually used to find optimal solutions for simple optimization problems such as the feature selection problem (Yusta 2009; Komusiewicz and Kratsch 2020). However, NP-hard optimization problems, such as the traveling salesman problem and shortest path problem (Zhou et al. 2019; Abed-alguni and Alawad 2021), typically cannot be solved with exact algorithms and require heuristic or optimization algorithms to solve them. This is because the search area of an NP-hard optimization problem is either extremely large or infinite. An optimization algorithm can be applied iteratively to compare different candidate solutions for an optimization problem until finding a satisfactory or optimal solution. In the optimization field, the term single-solution optimization describes an optimization algorithm that iteratively optimizes a single candidate solution, while the term population-based optimization is used to describe an optimization algorithm that iteratively optimizes multiple candidate solutions at the same time. A general problem with optimization algorithms is that they may converge to unaccepted solutions (premature convergence) during an early stage of their optimization process. Besides, the performance and speed of optimization algorithms degrade with the increase of the dimensionality (i.e., number of decision

✉ Bilal H. Abed-alguni
Bilal.h@yu.edu.jo

David Paul
David.Paul@une.edu.au

[1] Department of Computer Sciences, Yarmouk University, Irbid, Jordan

[2] School of Science and Technology, The University of New England, Armidale, NSW, Australia

variables) of optimization problems (Abed-alguni and Barhoush 2018; Abed-alguni 2019; Mehta and Kaur 2019).

One frequently used option of population-based optimization algorithms is the island-based approach, which is based on the principle of divide and conquer. In island-based optimization algorithms, an island model divides the overall candidate solution population into smaller, equal-sized independent sub-populations (islands) to ensure diversity of candidate solutions through a structured, distributed approach. An optimization algorithm is then independently applied to each island, though the process of migration is modelled by the islands regularly communicating with each other (Abed-alguni 2019; Abed-Alguni et al. 2019; Al-Betar et al. 2019; Al-Betar and Awadallah 2018).

Island-based optimization algorithms [e.g., island artificial bee colony (Awadallah et al. 2020), island flower pollination algorithm (Al-Betar et al. 2019), island bat algorithm (Al-Betar and Awadallah 2018), island-based harmony search (Al-Betar et al. 2015), island-based genetic algorithm (Mohammed et al. 2016) and island-based Cuckoo Search with highly disruptive polynomial mutation (*i*CSPM) Abed-alguni 2019] have been used with a high degree of success in both continuous and discrete problems. According to Abed-alguni (2019); Alawad and Abed-alguni (2020), *i*CSPM exhibits a superior performance compared to other island-based optimization algorithms. There are three possible reasons for this. Firstly, the *i*CSPM model, especially in conjunction with its abandon method, helps increase diversity in each island. Secondly, low-fitness solutions are given the opportunity to evolve to improved solutions. Thirdly, the included highly disruptive polynomial mutation (HDP) method allows the exploration of the entire search space regardless of the decision variable's current value.

In this research study, we introduce an improved *i*CSPM algorithm, *i*CSPM2, which incorporates three modifications into *i*CSPM. Firstly, it partitions its n candidate solutions between s independent islands and then equally divides them among four Cuckoo Search (CS) algorithms: CS via Lévy flights (CS1) (Yang and Deb 2009), CS with HDP mutation (CS10) (Abed-Alguni and Paul 2018), CS with Jaya mutation (CSJ) and CS with pitch adjustment mutation (CS11) (Abed-Alguni and Paul 2018). Secondly, it uses elite opposition-based learning (EOBL) (Zhou et al. 2013) to explore solutions around the current elite solutions (best solutions) in each island's population. Finally, it employs the smallest position value (SPV) (Ali et al. 2019) with scheduling problems to convert the continuous values of any decision variables to discrete ones in a candidate solution.

There are five expected benefits for *i*CSPM2, which are mainly because of the unique advantages that each CS variation has to offer. First, CS1 uses the Lévy flight mutation, which has a better exploration capability than the random mutation method (Yang and Deb 2009). Second, CS10 uses the HDP mutation method that can sample the entire search space between the lower and upper bounds of a decision variable regardless of its value (Abed-Alguni and Paul 2018; Abed-alguni 2019; Alawad and Abed-alguni 2020). Third, CSJ uses the Jaya mutation method that mutates the candidate solutions using stochastic moves based on the best and worst candidate solutions (Rao 2016). Fourth, CS11 uses the pitch adjustment mutation, which is known for its quick convergence (Abed-Alguni and Paul 2018; Al-Betar et al. 2015). Finally, we are particularly interested to use EOBL in *i*CSPM2 because it is capable of speeding up the convergence speed of optimization algorithms by exploring the opposite solutions of the best-known candidate solutions (Paiva et al. 2017; Zhang et al. 2017). By combining these techniques, it is expected that *i*CSPM2 will be able to realize some of the advantages of each variation.

The main contributions of this paper are as follows:

1. We propose *i*CSPM2, which is an improved island-based CS algorithm with multiple exploration and exploitation techniques.
2. To the best of the authors' knowledge, it is the first time that four variations of CS (CS1, CS10, CSJ and CS11) are applied synchronously to the island model. Besides, it is the first time that the EOBL method is used to improve the convergence rate and exploration ability of an island-based algorithm.
3. We use the SPV method with scheduling problems to convert continuous candidate solutions into discrete ones.
4. We present results of three sets of experiments to test and evaluate the efficiency of *i*CSPM2 compared to other popular optimization algorithms using popular benchmark suites for continuous and discrete problems.
   - Firstly, we used 15 popular benchmark functions to compare *i*CSPM2 to *i*CSPM. The experimental results indicate that *i*CSPM2 exhibits better performance than *i*CSPM.
   - Secondly, we used 30 IEEE-CEC 2014 functions to compare the performance of *i*CSPM2 against the performance of four state-of-the-art swarm optimization algorithms: distributed grey wolf optimizer (DGWO) (Abed-alguni and Barhoush 2018), distributed adaptive differential evolution with linear population size reduction evolution (L-SHADE) (Tanabe and Fukunaga 2014), memory-based hybrid dragonfly algorithm (MHDA) (Ks and Murugan 2017) and fireworks algorithm with differential mutation (FWA-DM) (Yu et al. 2014). The overall simulation and statistical results clearly indicate that *i*CSPM2 outperforms the other tested swarm optimization algorithms.
   - Finally, we used a set of Taillard's benchmark instances for the permutation flow shop schedul-

ing problem (PFSSP) (Taillard 1990) to evaluate and compare *i*CSPM2 to two powerful discrete evolutionary algorithms (generalized accelerations for insertion-based heuristics (GAIbH) (Fernandez-Viagas et al. 2020) and memetic algorithm with novel semi-constructive crossover and mutation operators (MASC) (Kurdi 2020). The simulation results indicate that *i*CSPM2 produces better schedules than GAIbH and MASC.

The remainder of the paper is organized as follows: We first review the preliminaries in Sect. 2. In Sect. 3, we provide a critical review of recently proposed island-based algorithms. Section 4 then presents our proposed variation of *i*CSPM. In Sect. 5, we demonstrate the performance of *i*CSPM2 compared to the other island-based algorithms and discuss these results in Sect. 6. Finally, Sect. 7 concludes the paper and presents future work.

2 Preliminaries

This section provides a summary of some of the underlying concepts of the Cuckoo Search algorithm (Sect. 2.1), as well as some existing improvements to Cuckoo Search, including Cuckoo Search with pitch adjustment (Sect. 2.2); Cuckoo Search with Jaya mutation (Sect. 2.3); and island-based Cuckoo Search with highly disruptive polynomial mutation (Sect. 2.4). Elite opposition-based learning, which explores opposite solutions of best-known candidates to help speed up convergence, is then discussed in Sect. 2.5. Finally, Sect. 2.6 provides a description of the permutation flow shop scheduling problem.

2.1 Cuckoo Search algorithm

The Cuckoo Search (CS) algorithm (Yang and Deb 2009; Abed-alguni et al. 2021; Alkhateeb et al. 2021) is a nature-inspired, evolutionary algorithm. It aims to optimize a population of candidate solutions for a given optimization problem using two simulation models related to the behaviors of birds in nature. The first model simulates the parasitic procreation habit of cuckoos, while the second model simulates the random actions of flying and landing of birds using Lévy flights (Yang and Deb 2009; Rakhshani and Rahati 2016).

In CS, an egg (current candidate solution) can be mutated into a cuckoo's egg. CS generates, at each iteration, a mutated solution using a function that comprises the Lévy-flight operator. It then attempts to swap the new solution with a lower quality solution in the population. Two parameters control the optimization process of CS: the population size (number of eggs) n and the fraction of the worst candidate solutions that get replaced with randomly generated solutions ($p_a \in [0, 1]$). The simulation models of CS are based on the following notations and assumptions:

- $\overrightarrow{X}^{\mathbf{i}} = \langle x_1^i, \ldots, x_m^i \rangle$ is the ith candidate solution that contains m decision variables, where x_j^i is the jth decision variable in $\overrightarrow{X}^{\mathbf{i}}$.
- The value x_j^i is a random value generated using a uniform-random, generation function $x_j^i = \mathrm{LB}_j^i + (\mathrm{UB}_j^i - \mathrm{LB}_j^i) \times U(0, 1)$, where $U(0, 1)$ represents a uniform random number in the interval $(0, 1)$, LB_j^i is the lower bound and UB_j^i is the upper bound of the search range of x_j^i.
- $f(\overrightarrow{X}^i)$ is the objective or fitness value of the candidate solution $\overrightarrow{X}^i$.
- $\overrightarrow{X}^i (i = 1, 2, \ldots, n)$ is a population of n candidate solutions.
- At the last step of each iteration, the CS algorithm replaces a fraction $p_a \in [0, 1]$ of the lowest fitted solutions with new randomly generated solutions.
- The new modified population is then moved to the next iteration of CS.

The CS algorithm initially produces a population of n candidate solutions $\overrightarrow{X}^{\mathbf{i}} (i = 1, 2, \ldots, n)$ from the range of possible solutions of the fitness function $f(\overrightarrow{X}^{\mathbf{i}})$. The maximum number of iterations ($MaxItr$) of the improvement loop of CS is specified according to of complexity of $f(\overrightarrow{X}^{\mathbf{i}})$.

CS attempts, at each iteration, to enhance a randomly selected, candidate solution ($\overrightarrow{X}^{\mathbf{i}}(t)$) using a Lévy-flight mutation function and produces a new solution ($\overrightarrow{X}^{\mathbf{i}}(t+1)$) as follows:

$$\overrightarrow{X}^{\mathbf{i}}(t+1) = \overrightarrow{X}^{\mathbf{i}}(t) + \beta \oplus \text{Lévy}(\lambda), \tag{1}$$

where $\beta > 0$ controls the distance of mutation and $\oplus$ represents entry-wise multiplication. The Lévy flight is conventionally defined as a special random walk with a dynamic step length. The step length is generated from an infinite, heavy-tailed, Lévy distribution that follows the power law:

$$\text{Lévy} \sim u = L^{-\alpha}, \tag{2}$$

where $\alpha \in (1, 3]$ is the parameter of fractal dimension and L is the step size. This distribution produces a small percentage of random solutions around the local optimal solutions, while producing the majority of random solutions away from the local optimums. Therefore, the Lévy flight-mutation method has a better exploration capability than a standard random walk, since the distance value increases gradually during

the simulation. At the end of each iteration of CS, an abandon process takes place, where a fraction (p_a) of the worst candidate solutions in the population are replaced with new randomly generated candidate solutions.

2.2 Cuckoo Search with pitch adjustment

Cuckoo Search with pitch adjustment (CS11) is an algorithm that uses the pitch adjustment (PA) method in place of Lévy flight. PA is a probabilistic method (Geem et al. 2001) that mutates $\overrightarrow{\boldsymbol{X}}^{\mathbf{j}}(t)$ by mutating each one of its decision variables with probability PAR$\in$ (0, 1):

$$x_i^j(t+1) = \begin{cases} x_i^j(t) \pm U(-1,1) \times \text{BW} & r \leq \text{PAR} \\ x_i^j(t) & r > \text{PAR} \end{cases} \quad (3)$$

where r is a random number in [0, 1], BW (bandwidth) determines the mutation distance and $U(-1, 1)$ is a random number in $(-1, 1)$.

2.3 Cuckoo Search with Jaya mutation

The Jaya algorithm (Rao 2016) is an evolutionary algorithm that was designed according to the concept of victory. In the improvement loop of Jaya, the candidate solutions are randomly moved between the best and worst candidate solutions. This behavior aims to explore the search area between the worst and best candidate solutions (Alawad and Abed-alguni 2021).

We propose a new variant of CS called Cuckoo Search with Jaya mutation (CSJ), which replaces the Lévy flight operator of CS with Jaya mutation. In CSJ, the decision variables of $\overrightarrow{\boldsymbol{X}}^{\mathbf{j}}(t)$ are updated one by one using the corresponding decision variables of the best candidate solution $\overrightarrow{\boldsymbol{X}}^{\mathbf{B}}(t)$ and worst candidate solution $\overrightarrow{\boldsymbol{X}}^{\mathbf{W}}(t)$ at iteration t as follows:

$$x_i^j(t+1) = x_i^j(t) + r_1\, x_i^B(t) + r_2\, x_i^W(t), \quad (4)$$

where $x_i^j(t)$, $x_i^B(t)$ and $x_i^W(t)$ are the values of ith decision variables of $\overrightarrow{\boldsymbol{X}}^{\mathbf{j}}(t)$, $\overrightarrow{\boldsymbol{X}}^{\mathbf{B}}(t)$ and $\overrightarrow{\boldsymbol{X}}^{\mathbf{W}}(t)$, respectively at iteration t. r_1 and r_2 are random parameters in [0,1]. After mutating all decision variables of $\overrightarrow{\boldsymbol{X}}^{\mathbf{j}}(t)$, if the fitness value of $\overrightarrow{\boldsymbol{X}}^{\mathbf{j}}(t+1)$ is an improvement over the fitness of $\overrightarrow{\boldsymbol{X}}^{\mathbf{j}}(t)$, the solution $\overrightarrow{\boldsymbol{X}}^{\mathbf{j}}(t)$ is replaced by $\overrightarrow{\boldsymbol{X}}^{\mathbf{j}}(t+1)$.

2.4 Island-based Cuckoo Search with highly disruptive polynomial mutation

The island Cuckoo Search (*i*CSPM) is a parallel variation that integrates two modifications to CS. Firstly, it uses a structured population model called the island model (Corcoran and Wainwright 1994) to organize candidate solutions in CS to smaller manageable sub-populations (islands). This model increases the probability that unfit candidate solutions are improved into better solutions and improves the diversity of the candidate solutions (Al-Betar et al. 2019; Al-Betar and Awadallah 2018). Secondly, it uses CS with HDP mutation (CS10) (Abed-Alguni and Paul 2018) to optimize the candidate solutions on each island. This is because, according to several experimental studies (Hasan et al. 2014; Alkhateeb and Abed-Alguni 2017; Abed-alguni and Alkhateeb 2018), CS10 is a better exploratory algorithm than the CS algorithm. Besides, the HDP mutation used in CS10 can sample the entire search space between the lower and upper bounds of a decision variable regardless of its value (Alawad and Abed-alguni 2020; Abed-alguni 2019; Abed-Alguni and Paul 2018).

CS10 uses the HDP method to alter the values of the decision variables of $\overrightarrow{\boldsymbol{X}}^{\mathbf{j}}(t)$ by mutating each of its decision variables with probability $P_m \in [0, 1]$ as follows (Doush et al. 2014):

$$x_i^j(t+1) \leftarrow x_i^j(t) + \delta_k.(\text{UB}_i - \text{LB}_i), \text{ where} \quad (5)$$

$$\delta_k = \begin{cases} (2r) + (1-2r) \times (1-\delta_1)^{\eta_m+1}]^{\frac{1}{\eta_m+1}} - 1 & r \leq P_m \\ 1 - [2(1-r) + 2(r-0.5) \times (1-\delta_2)^{\eta_m+1}]^{\frac{1}{\eta_m+1}} & r > P_m \end{cases} \quad (6)$$

$$\delta_1 \leftarrow \frac{x_i^j(t) - \text{LB}_i}{\text{UB}_i - \text{LB}_i} \quad (7)$$

$$\delta_2 \leftarrow \frac{\text{UB}_i - x_i^j(t)}{\text{UB}_i - \text{LB}_i} \quad (8)$$

The definitions of the parameters in the above equations are as follows:

- $x_i^j(t)$: the ith decision variable of the jth candidate solution $\overrightarrow{\boldsymbol{X}}^{\mathbf{j}}(t)$ at iteration t
- r: a uniform random number in [0, 1]
- P_m: the probability of mutation
- LB_i: the lower boundary of $x_i^j(t)$
- UB_i: the upper boundary of $x_i^j(t)$
- δ_1: the distance between $x_i^j(t)$ and LB_i over the closed interval $[\text{UB}_i, \text{LB}_i]$
- δ_2: the distance between UB_i and $x_i^j(t)$ over the closed interval $[\text{UB}_i, \text{LB}_i]$
- η_m: the distribution index, which is a non-negative number

A vital part of the island model is the migration procedure that periodically allows the islands to exchange candidates solutions. It enables *i*CSPM to select and transfer a subset of candidate solutions among the different islands, depending on two factors. Firstly, the number of generations that should

evolve in each island before the triggering the migration procedure (the migration frequency, M_f). Secondly, the fraction of the candidate solutions on each island that can be moved between neighboring islands (the migration rate, M_r). After each M_f iterations of *i*CSPM, the islands are organized in a random ring shape, with each island establishing one connection in from a neighboring island, and one connection out to its other neighbor.

The *i*CSPM algorithm uses a "best-worst migration policy", which determines the best candidate solutions on one island to be swapped with the worst candidate solutions on a neighboring island. The algorithmic details and the source code of *i*CSPM are available in Abed-alguni and Alkhateeb (2018).

2.5 Elite opposition-based learning

Sihwail et al. (2020) proposed a special type of opposition-based learning by the name EOBL (short for elite opposition-based learning). EOBL was inspired from the intuition that opposite solutions of elite (best) solutions are more likely to be close to the global optima than any other solutions. EOBL was used with optimization methods such as the Bat algorithm (Paiva et al. 2017) and Harris–Hawks optimization algorithm (Zhang et al. 2017) to improve their efficiency.

We can apply EOBL to decision variables of candidate solutions. Let $\overrightarrow{X} = \langle x_1, x_2 \ldots, x_m \rangle$ be an elite candidate solution with m decision variables. The elite opposite-based solution $\overrightarrow{X}^o$ can be calculated using Equation 9:

$$\overrightarrow{X}^o = \langle x_1^o, x_2^o \ldots, x_m^o \rangle, \text{ where } x_i^o = \delta(\mathrm{d}a_i + \mathrm{d}b_i) - x_i \quad (9)$$

In the above equation, $\delta \in (0, 1)$ is a parameter used to control the opposition magnitude, and $\mathrm{d}a_i$ and $\mathrm{d}b_i$ are the dynamic boundaries which are calculated as follows:

$$\mathrm{d}a_i = \min(x_i), \quad \mathrm{d}b_i = \max(x_i) \quad (10)$$

The value of an opposite decision variable x_i^o may be outside $[\mathrm{LB}_i, \mathrm{UB}_i]$. This can be solved using the following equation:

$$x_i^o = \mathrm{rand}(\mathrm{LB}_i, \mathrm{UB}_i), \text{ if } x_i^o < \mathrm{LB}_i \text{ or } x_i^o > \mathrm{UB}_i \quad (11)$$

where $\mathrm{rand}(\mathrm{LB}_i, \mathrm{UB}_i)$ is a random number between LB_i and UB_i.

2.6 Permutation flow shop scheduling problem

The permutation flow shop scheduling problem (PFSSP) is a popular $\mathcal{NP}$-hard combinatorial optimization problem. It is a discrete optimization problem that is commonly used as a benchmark for investigating the efficiency of discrete optimization algorithms (Alawad and Abed-alguni 2021). Several heuristic- and optimization-based scheduling algorithms have been evaluated using the PFSSP (Alawad and Abed-alguni 2020; Abed-alguni and Alawad 2021; Kurdi 2016; Wang et al. 2017). The mathematical model of the PFSSP is built around two lists. Firstly, a list of n jobs $(j_1, j_2, \ldots, j_n)$, where each job comprises m sequential operations $(O_{i,1}, O_{i,2}, \ldots, O_{i,m})$. Secondly, a list of m machines $(M_1, M_2, \ldots, M_m)$, where each machine can execute only the corresponding operation in a job. In PFSSP, $t_{i,j}$ denotes the processing time required for machine j to complete job i and $O_{i,j}$ is the processing operation of job i on machine j. There is a strict order of jobs that have to be processed by each machine, and each machine can only process one job at a time. A candidate schedule is a permutation of jobs $\pi = \{\pi_1, \pi_2, \ldots, \pi_n\}$ for which completion times of all operations can be calculated. The completion time C_j of job j is defined to be the completion of job j's last operation $O_{j,m}$ ($C_j = C_{j,m}$). The optimal solution in PFSSP is the candidate schedule that has the lowest total completion time of all jobs. The equations of the PFSSP are as follows (Wang et al. 2017; Alawad and Abed-alguni 2021):

$$C_{1,1} = t_{1,1} \quad (12)$$

$$C_{j,1} = C_{j-1,1} + t_{j,1}, \quad j = 2, \ldots, n, \quad (13)$$

$$C_{1,k} = C_{1,k-1} + t_{1,k}, \quad k = 2, \ldots, m, \quad (14)$$

$$C_{j,k} = \max\{C_{j-1,k}, C_{j,k-1}\}, \quad j = 2, \ldots, n, \ k = 2, \ldots, m, \quad (15)$$

$$C_{\max} = \max\{C_{n,m}\}, \quad (16)$$

$$f = \max\{C_{\max}\}. \quad (17)$$

We used the above equations in Sect. 5.7 to test the performance of *i*CSPM2 using 12 instances of Taillard's benchmark for the PFSSP (Taillard 1990).

3 Related Work

The island model is a structured-population strategy that has been incorporated into many evolutionary algorithms to improve their population diversity and convergence speed (Abed-alguni and Barhoush 2018; Abed-Alguni et al. 2019; Al-Betar et al. 2019; Al-Betar and Awadallah 2018; Awadallah et al. 2020; Al-Betar et al. 2015; Alawad and Abed-alguni 2020; Abed-alguni and Alawad 2021; Kurdi 2016; Lardeux and Goëffon 2010; Kushida et al. 2013; Abadlia et al. 2017; Michel and Middendorf 1998). This section provides a critical review of recently proposed island-based optimization algorithms.

The island-based differential evolution algorithm (island-based DE) is a variation of differential evolution (DE) that is based on the multi-size island model (Skakovski and Jedrzejowicz 2019). This model can be applied to the population of candidate solutions using two different approaches. In the first approach, the population of candidate solutions can be divided to islands with different sizes. In the second approach, the algorithm starts its optimization process with one population and then changes its size over the course of optimization process. In both approaches, a copy of DE is applied to each island, where communication between islands, or migration of candidate solutions, is not permitted. Island-based DE was compared in Skakovski and Jedrzejowicz (2019) to DE and IBDEAmd (island-based DE algorithm with islands of equal sizes) using complex scheduling problems. The results showed that island-based DE is the best performing algorithm. However, island-based DE requires more computational resources than the other algorithms in the beginning of its optimization process. In addition, if it is executed sequentially, it may require longer running time than the other algorithms.

The parallel multi-population chaotic bat algorithm (CBAS) (Guo et al. 2019) is an island-based algorithm that incorporates four modifications into the bat algorithm (BA): the island model (Al-Betar et al. 2019), chaotic maps (Mugemanyi et al. 2020), Lévy flight search (Liu et al. 2020) and contraction factor. The island model is used to divide the population of CBAS into islands. The chaotic maps are evolution functions with chaotic behaviors usually used to generate random numbers in BA. The contraction factor is a parameter that controls the direction of optimization in BA. The Lévy flight search is based on the Lévy flight operator, which is a special random walk with a dynamic step length. The experimental results in Guo et al. (2019), using a few general benchmark functions, suggest that CBAS outperforms some variations of BA (suggested also by the same authors) such as shrink factor BA, chaotic BA and Lévy-flight BA. However, the performance of CBAS should be compared to the performance of state-of-the-art swarm optimization algorithms such as DGWO and L-SHADE using strong benchmark suite such as the CEC 2015 test suite. This will allow us to fairly evaluate the performance of CBSA.

The HS algorithm is a widely used optimization algorithm inspired by the improvisation of music players (Geem et al. 2001). The island-based HS (*i*HS) (Al-Betar et al. 2015) is a variation of HS that integrates the concepts of island model into with HS. The optimization loop of HS is applied to each island independently. The quality of each island's solutions is periodically improved by exchanging candidate solutions among island through a migration process. The experimental study in Al-Betar et al. (2015) suggests that *i*HS provides more accurate fitness values than the original HS algorithm for a large number of continuous numerical problems. The *i*HS was utilized in Al-Betar (2021) to solve the economic load dispatch problem (i.e., the scheduling for an electrical generation system to provide the load demand in addition to the transmission loss with the objective of minimizing generation fuel cost). According to the experimental results in Al-Betar (2021), *i*HS outperformed the original HS algorithm and other algorithms (e.g., island bat algorithm Al-Betar and Awadallah 2018) by providing the best results for three out of five real-world economic load dispatch test cases.

Chen et al., proposed the chaotic multi-population hybrid DE and Harris–Hawks optimization (CMDHHO) algorithm for solving the single-objective optimization problems (Chen et al. 2020). CMDHHO integrates the chaos maps, island model and optimization operators of DE into the Harris–Hawks optimization (HHO) algorithm. The experimental and statistical results in Chen et al. (2020) using CEC 2011 and CEC 2107 suggest that CDHHO is faster and provides better performance than HHO and DE. However, CMDHHO requires more computational resources than DE and HHO in the beginning of its optimization process. Besides, the chaotic maps in CMDHHO are more suitable to continuous optimization problems than discrete problems.

Lardeux and Goëffon (2010) proposed a dynamic island model for the genetic algorithm (GA). This model uses complete graph modeling that allows the implementation of any type of migration topologies from edgeless graphs to popular island topologies such as uni-directional or bi-directional, rings and lattices. This model is called dynamic because the topology in it continues to evolve during the search process following the rewards and penalties received during previous migration waves. The experimental results in Lardeux and Goëffon (2010) using 0/1 Knapsack and MAX-SAT problems suggest that the dynamic island model helps GA to achieve good results. However, a problem with this model is that it uses a GA that is sensitive to its internal parameters and methods such as the rate of mutation, rate of crossover, population size and selection method.

Kurdi (2016) suggested the island model genetic algorithm (IMGA) for solving the PFSSP. In IMGA, islands use different variations of the GA (GA with insertion operator, GA with swap operator and GA with inversion operator). Results in Kurdi (2016) suggest that IMGA offers improved performance than the variations of GA that depend on a single evolutionary method.

Alawad and Abed-alguni (2020) proposed discrete *i*CSPM with opposition-based learning strategy (D*i*CSPM), a discrete variation of *i*CSPM. This algorithm provides efficient schedules for workflow applications in distributed environments. It uses random generation and opposition-based learning approaches to generate diverse initial population candidate solutions. It also optimizes decision variable values in candidate solutions using the SPV method. The simulation results in Alawad and Abed-alguni (2020) using

WorkflowSim (Casanova et al. 2014) suggest that D*i*CSPM provides efficient schedules in cloud computing environments.

Particle swarm optimization (PSO) is a well-known algorithm that was inspired from the movement behavior of birds in a group (Kennedy and Eberhart 1995). The island model was integrated into PSO in a new algorithm called the dynamic island model PSO (DIMPSO) algorithm (Abadlia et al. 2017). The goal of DIMPSO is to increase the diversity of the population and the convergence speed of PSO. DIMPSO was evaluated using the CEC 2005 test suite, and the results suggest it is more efficient and accurate than PSO.

The island bat algorithm (*i*BA) (Al-Betar and Awadallah 2018) incorporates the island model into BA. Similar to *i*HS, the population in *i*BA is divided into smaller sub-populations, where the improvement loop of BA is applied independently to each sub-population. A migration process and the ring topology are used in *i*BA to improve the diversity of the candidate solutions in each sub-population and improve the convergence speed. *i*BA was evaluated in Al-Betar and Awadallah (2018) using 25 IEEE-CEC2005 benchmark functions. Results indicate that *i*BA improves the performance of BA.

The ant colony algorithm has also been modified to an island model in the ant colony optimization with lookahead (ACO) algorithm (Michel and Middendorf 1998). It was proposed to help solve the problem of finding the shortest supersequence *str* of two sequences *str1* and *str2* such that both sequences are part of *str*. AOC incorporates the same principles of the island model as *i*HS and *i*BA, but it additionally uses a lookahead function to mutate the candidate solutions taking into account the influence of the current mutation on the mutation in the next iteration.

The distributed grey wolf optimizer (DGWO) algorithm (Abed-alguni and Barhoush 2018) is a distributed variation of the grey wolf optimizer (GWO) algorithm (Mirjalili et al. 2014). DGWO attempts to increase the convergence speed of GWO by improving diversity through the incorporation of the principles of the island model into GWO. DGWO was compared to four optimization algorithms (GWO, distributed adaptive differential evolution with linear population size reduction evolution (L-SHADE) (Tanabe and Fukunaga 2014), memory-based hybrid dragonfly algorithm (MHDA) (Ks and Murugan 2017) and fireworks algorithm with differential mutation (FWA-DM) Yu et al. 2014) using 30 CEC 2014 functions. DGWO performs significantly better than the other tested algorithms.

Abed-alguni and Alawad (2021) proposed a discrete variation of DGWO for scheduling of dependent tasks to virtual machines in distributed environments. It uses the largest order value (LOV) method to produce discrete job permutations from continuous candidate solutions. The simulation results using WorkflowSim Casanova et al. (2014) indicate that the discrete variation of DGWO is efficient for solving scheduling problems.

The flower pollination algorithm (FPA) is an evolutionary algorithm based on the biological evolution of pollination of flowers (Yang 2012). In Al-Betar et al. (2019), the island model was applied to FPA to control its diversity and solve its premature convergence problem. The proposed algorithm was called IsFPA. 23 test functions were used to evaluate the performance of IsFPA compared to GA, PSO, gravitational search algorithm (GSA), multi-verse optimizer (MVO), *i*BA and *i*HS. IsFPA was found to perform better than each of the other algorithms.

The artificial bee colony (ABC) algorithm (Karaboga and Basturk 2007) simulates honey bee foraging behavior to solve various optimization problems. It is an efficient algorithm, but may very quickly converge to sub-optimal solutions at the beginning of its optimization process. The island artificial bee colony (*i*ABC) algorithm (Awadallah et al. 2020) helps overcome this problem by distributing ABC's optimization process over multiple islands. Evaluation of the performance of *i*ABC using the IEEE-CEC 2015 indicated that *i*ABC indeed improved diversity and performance of ABC and also provided interesting performance compared to 18 other tested algorithms.

The island-based differential evolution (*i*DE) algorithm (Kushida et al. 2013) modifies the DE algorithm to use an island model. *i*DE divides the population of candidate solutions to islands with varying population size and parameters. Therefore, each island has different convergence behavior compared to the other islands. *i*DE was evaluated in Kushida et al. (2013) using a set of basic test functions and found to be more efficient than DE.

The whale optimization algorithm (WOA) simulates humpback whale bubble-net hunting mechanisms to solve optimization problems (Mirjalili and Lewis 2016). However, WOA may suffer from the premature convergence, degrading the performance of the evolutionary algorithm (Abed-alguni and Klaib 2018). To mitigate this issue, Abed-Alguni et al. (2019) modified WOA to incorporate the island model. Results show the *i*WOA has improved accuracy over WOA for 18 test functions.

In conclusion, island-based optimization algorithms have been used extensively in the literature (Abed-alguni and Barhoush 2018; Abed-Alguni et al. 2019; Al-Betar et al. 2019; Al-Betar and Awadallah 2018; Awadallah et al. 2020; Al-Betar et al. 2015; Alawad and Abed-alguni 2020; Abed-alguni and Alawad 2021; Kurdi 2016; Lardeux and Goëffon 2010; Kushida et al. 2013; Abadlia et al. 2017; Michel and Middendorf 1998) to improve the diversity of the population in population-based optimization algorithms and improve their convergence rate to optimality. However, typical island-based optimization algorithms apply only a single optimization algorithm to all islands. Therefore, in this paper,

we investigate the simultaneous use of several variations of CS on different islands. Such an algorithm should benefit from some of the unique advantages each CS variation has to offer.

4 Proposed algorithm: *i*CSPM with elite opposition-based learning and multiple mutation methods

In this section, we present *i*CSPM2 which is a variation of *i*CSPM with EOBL and four mutation methods. Fig. 1 illustrates the flowchart of *i*CSPM2. The main steps of the flowchart are as follows:

1. The first step is to randomly generate a population of n candidate solutions using a generating function. The n candidate solutions are then randomly divided among s islands.
2. The s islands are then divided equally among four variations of CS: CS1, CS10, CSJ and CS11.
3. Each variation is then independently executed for M_f iterations on the islands assigned to it, with EBOL executed at the end of each generation to replace any elite candidate solution with its opposite if the opposite is determined to be a better candidate (as described in Sect. 2.5). This step can be performed in parallel because each variation of CS is applied to an independent set of islands. Even the optimization loop in each variation of CS can be executed in parallel. This issue is explained in Algorithm 1, which shows the algorithmic details of *i*CSPM2.
4. The n islands are arranged based on a random ring topology. Some example random ring topologies are presented in Fig 2. The direction of migration between an island and its neighboring island is represented in the figure as a unidirectional edge between the two islands. An important constraint in the topology is that each island has exactly one incoming and one outgoing edge.
5. The migration process takes place among the islands following the chosen ring topology.
6. If the maximum number of iterations has not been reached the algorithm repeats from step 3. Otherwise, the algorithm ends.

Algorithm 1 has two differences compared to *i*CSPM. Firstly, it divides the islands among four variations of CS: CS1, CS10, CSJ and CS11 (Line 7). The optimization loop (Lines 12–25) is applied to each island according to the CS variation assigned to it (Line 13). This allows the optimization loops to be performed in parallel because the islands only communicate with each other during the migration process. Secondly, it applies EOBL to a fraction (p_b) of the elite solutions in each island (Line 20). The elite-opposite solutions are generated using the procedure and equations discussed in Sect. 2.5. An opposite-elite solution replaces its corresponding elite solution if it has a better fitness than the elite solution.

Algorithm 1 *i*CSPM with elite opposition-based learning and multiple mutation methods (*i*CSPM2).

1: Determine n, the total population size of each island and $MaxItr$, the maximum number of iterations.
2: Initialize s (the number of islands), M_f (migration frequency) and M_r (migration rate).
3: Determine $k = n/s$, each island's population size
4: Determine $M_w = MaxItr/M_f$, the number of migration waves
5: Determine $n_r = \frac{n}{s} \times M_r$, the number of migrant solutions
6: For each island (j=1,2,…,s), initialize the k candidate solutions $\vec{X}_i^j (i = 1, 2, \ldots, k)$
7: Divide the s islands among four variations of CS: CS1, CS10, CSJ and CS11
8: **for** $c = 1$ to M_w **do**
9: **for** $j = 1$ to s **do**
10: $t = 0$
11: Determine $f(\vec{X}_i^j)$, the fitness value of each candidate solution
12: **while** $t < M_f$ **do**
13: Select a random solution (say, $\vec{X}_a^j$) from island j's current population and replace it by applying the CS variation assigned to island j
14: Calculate $f(\vec{X}_a^j)$, the quality/fitness value of $\vec{X}_a^j$
15: Select a random solution (say, $\vec{X}_b^j$) from island j's current population
16: **if** $f(\vec{X}_a^j)$ is better than $f(\vec{X}_b^j)$ **then**
17: Replace $\vec{X}_b^j$ by $\vec{X}_a^j$
18: **end if**
19: Replace a fraction p_a of island j's worst solutions with new ones
20: Apply EOBL:
- Select a fraction p_b of island j's elite candidate solutions
- Generate each elite solution's opposite solution using EOBL (Section 2.5)
- If any elite solution has a lower fitness than its opposite-elite solution, replace the elite solution with its corresponding opposite-elite solution

21: Keep the best quality solutions
22: Rank the solutions and find the current best ($\vec{X}_{Best}^j$)
23: Replace island j's n_r best candidate solutions with island $((j + 1) \bmod s)$'s n_r worst candidate solutions
24: $t = t + 1$
25: **end while**
26: **end for**
27: **end for**
28: Determine $\vec{X}_{Best}$ ($\vec{X}_{Best}^j$ with the best fitness)
29: Return $\vec{X}_{Best}$

The generated candidate solutions at lines 6, 13 and 20 in Algorithm 1 are composed of continuous values that can be used directly with continuous optimization problems. However, in order to solve discrete optimization problems such as

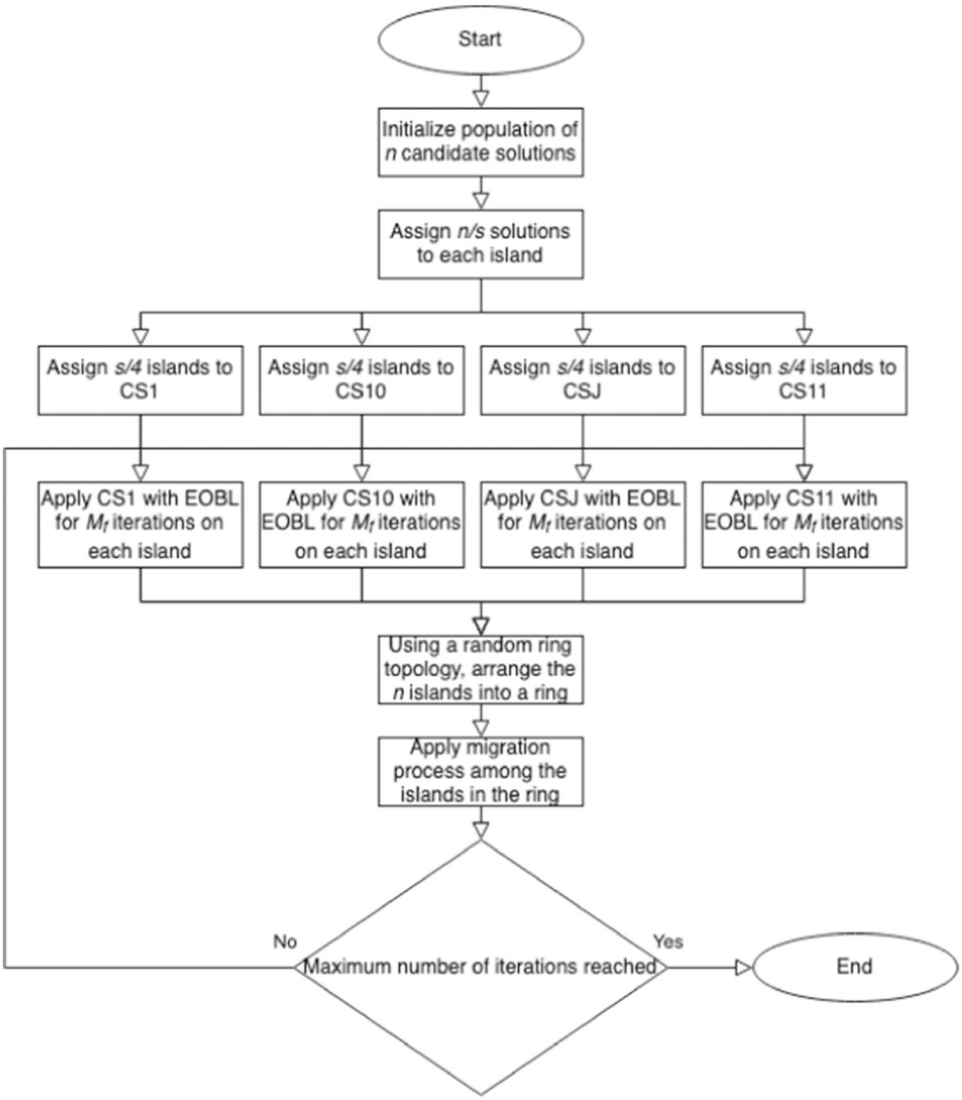

Fig. 1 General framework of *i*CSPM2

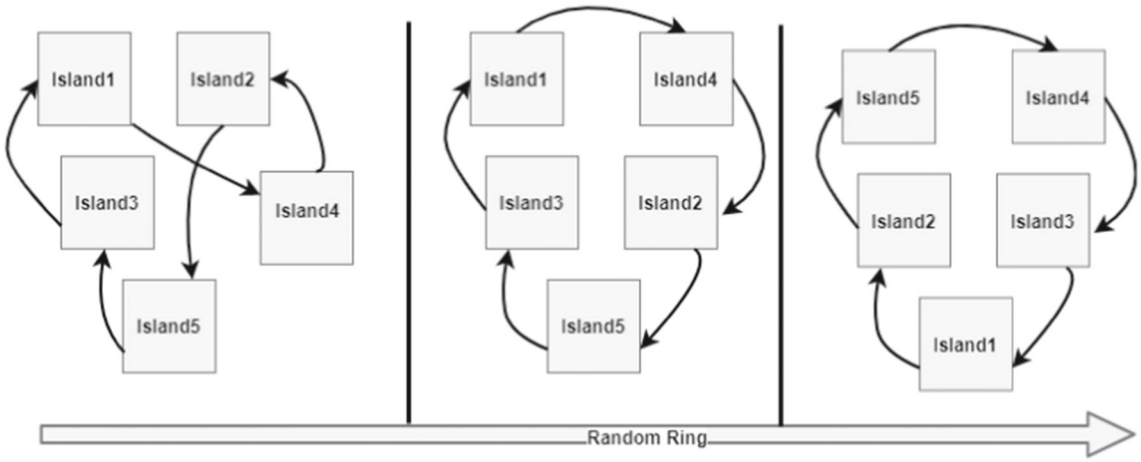

Fig. 2 Random ring topologies (Abed-alguni 2019)

Table 1 A candidate solution using the SPV method

Dimension (j)	1	2	3	4
x_j^i	7.86	12.11	10.46	5.66
Dimension (j)	4	1	3	2
x_j^i in ascending order	5.66	7.86	10.46	12.11
π_j^i	4	1	3	2

the PFSSP, these values should be converted to meaningful discrete values. In this paper, we also show how *i*CSPM2 can be used with the PFSSP. To this end, the SPV method (Ali et al. 2019) can be used with *i*CSPM2 to convert candidate solutions' continuous decision variables into discrete ones (job numbers).

SPV is applied as follows. Let $\vec{X}^{\mathbf{i}} =< x_1^i, x_2^i, \ldots, x_m^i >$ be a candidate solution that contains m real values. The position values in $\vec{X}^{\mathbf{i}}$ are sorted in ascending order and the ordered positions are used to generate a job permutation π. An example of the SPV method is shown in Table 1, where $\vec{X}^{\mathbf{i}} =< x_1, x_2, x_3, x_4 >=< 7.86, 12.11, 10.46, 5.66 >$ and the calculated job permutation is $\mathbf{ß^i} =< 4, 1, 3, 2 >$, since $x_4 > x_1 > x_3 > x_2$.

5 Experiments

This section describes the experiments conducted to compare *i*CSPM2 to existing algorithms. Section 5.1 summarizes the experimental setup for all the algorithms and Sect. 5.2 describes the benchmark functions used in the experiments. The experimental design of the paper is described in Sect. 5.3. Section 5.4 then compares the sensitivity of *i*CSPM2 to its island model parameters to that of *i*CSPM. After this, Sect. 5.5 provides a comparison of the simulation results of *i*CSPM2 to the reported simulation results in Abed-alguni and Barhoush (2018) of four popular optimization algorithms (DGWO, L-SHADE, FWA-DM and MHDA), with statistical analysis provided in Sect. 5.6. Finally, Sect. 5.7 provides a comparison of the performance of *i*CSPM2 to the performance of GAIbH and MASC using 12 instances of Taillard's benchmark.

5.1 Setup

The experiments were executed using a 3.5GHz 8-core Intel Xeon W processor, Turbo Boost up to 4.0GHz with 32GB of DDR4 ECC memory running macOS 11.0.1, Big Sur. The algorithms were all programmed in Java.

5.2 Benchmarking

The 15 benchmark functions described in Table 2 are well-recognized functions that have frequently been used to evaluate the efficiency of optimization algorithms (Hasan et al. 2014; Doush et al. 2014). These functions were used in Sect. 5.4 to test the sensitivity of *i*CSPM and *i*CSPM2 to the island model parameters.

The 30 single-objective real-parameter optimization functions of CEC2014 are various complex functions: F_1 - F_3 are unimodal functions, F_4 - F_{16} are multimodal functions, F_{17} - F_{22} are hybrid functions, and F_{23} - F_{30} are composite functions (Liang et al. 2013). The search range of each function is $[-100, 100]^D$. We have previously used the single-objective functions of CEC2014 to compare the convergence speeds and the reliability of six popular optimization algorithms: CS, GWO, DGWO (Abed-alguni and Barhoush 2018), L-SHADE (Tanabe and Fukunaga 2014), MHDA (Ks and Murugan 2017) and FWA-DM (Yu et al. 2014). In Sect. 5.5, we compared the simulation results of *i*CSPM2 using the single-objective functions of CEC2014 to the four best performing algorithms in Abed-alguni and Barhoush (2018) (DGWO, L-SHADE, MHDA and FWA-DM).

Taillard's benchmark (Taillard 1990) is a benchmark for evaluating discrete optimization problems. In Sect. 5.7, we compared *i*CSPM2 to two powerful optimization-based scheduling algorithms (generalized accelerations for insertion-based heuristics (GAIbH) (Fernandez-Viagas et al. 2020) and memetic algorithm with novel semi-constructive crossover and mutation operators (MASC) Kurdi 2020) using 12 instances of Taillard's benchmark for the PFSSP.

5.3 Experimental design

In order to determine how sensitive the performance of *i*CSPM and *i*CSPM2 are to the island model parameters (s, M_f, M_r), the experiments in Sect. 5.4 were run over nine scenarios (presented in Table 3). The main goal of Scenarios 1–3 is to understand the relationships between the number of islands s and the performance of *i*CSPM and *i*CSPM2. The purpose of Scenarios 4-6 is to measure the effect of migration frequency M_f on the performance of *i*CSPM and *i*CSPM2. The goal of Scenarios 7–9 is to investigate the migration rate M_r's influence on the performance of *i*CSPM and *i*CSPM2. Fig. 3 shows the graphical representation of the values in Table 3.

In all the experiments, the size of the population, n, and the fraction of abandon solutions, p_a, in *i*CSPM and *i*CSPM2 were dynamically tuned over multiple simulations as recommended in Abed-alguni and Alkhateeb (2017). The maximum number of iterations of each algorithm was limited to 10,000. The control parameters of *i*CSPM2 were as follows: $L = 1$ in the Lévy flight method, PAR=0.3 in the

Table 2 Details of popular 15 benchmark functions

Function name	Search boundaries	Global optimum	Function type
f_1: Sphere	$[-100, 100]$	0	Unimodal
f_2: Schwefel's problem 2.22	$[-10, 10]$	0	Unimodal
f_3: Step	$[-100, 100]$	0	Unimodal
f_4: Rosenbrock	$[-2.048, 2.048]$	0	Multimodal
f_5: Rotated hyper-ellipsoid	$[-100, 100]$	0	Unimodal
f_6: Schwefel's problem 2.26	$[-500, 500]$	$-418.98 * d$	Multimodal
f_7: Rastrigin	$[-5.12, 5.12]$	0	Multimodal
f_8: Ackley	$[-32, 32]$	0	Multimodal
f_9: Griewank	$[-600, 600]$	0	Multimodal
f_{10}: Six-Hump Camel-Back	$[-5, 5]$	-1.031628	Multimodal
f_{11}: Shifted sphere	$[-100, 100]$	-450	Unimodal
f_{12}: Shifted Schwefel's problem 1.2	$[-100, 100]$	-450	Unimodal
f_{13}: Shifted Rosenbrock	$[-100, 100]$	390	Multimodal
f_{14}: Shifted Rastrigin	$[-5, 5]$	-330	Multimodal
f_{15}: Shifted expanded Griewanks plus Rosenbrock's function	$[-5, 5]$	-130	multimodal

Table 3 Nine experimental scenarios to evaluate the sensitivity of *i*CSPM and *i*CSPM2 to the parameters of island model

Experimental Scenario	s	M_f	M_r (%)
Scenario1	4	100	20
Scenario2	8	100	20
Scenario3	12	100	20
Scenario4	12	50	20
Scenario5	12	100	20
Scenario6	12	500	20
Scenario7	12	100	10
Scenario8	12	100	20
Scenario9	12	100	30

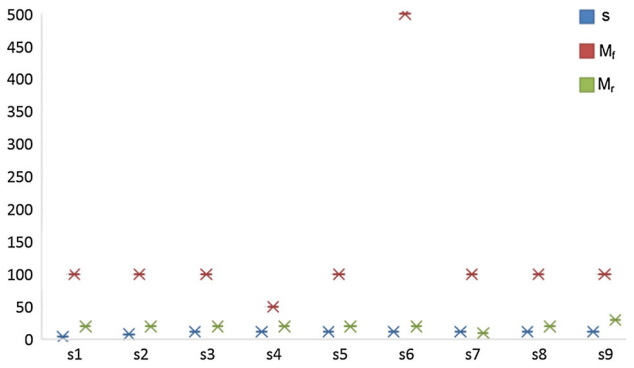

Fig. 3 Boxplot of the experimental scenarios

PA method and the fraction of elite solutions p_b was set to a value equal to the value of p_a.

Section 5.5 presents a comparison of *i*CSPM2 to the reported simulation results in Abed-alguni and Barhoush (2018) for four popular optimization algorithms (DGWO Abed-alguni and Barhoush 2018, L-SHADE Tanabe and Fukunaga 2014, FWA-DM Yu et al. 2014 and MHDA Ks and Murugan 2017). In Sect. 5.6, a nonparametric statistical test (Friedman's test Derrac et al. 2011; Friedman 1940) is applied to the FEV for each of the 30 CEC 2014 functions in Table 7.

5.4 Sensitivity analysis of *i*CSPM2 to the parameters of Island model

Tables 4, 5 and 6 show the average over 50 runs of the best objective values of the experimental scenarios in Table 3 for the 15 benchmark functions in Table 2. The best objective value for a benchmark function is its lowest value that is achieved after applying the tested optimization algorithms to it for a specified number of iterations (MaxItr). The number of decision variables (dimension) of each benchmark function was 10 except for the two-dimensional six-hump camel-back function (f_10). In each table, two rows correspond to each function. The first row contains the simulation results of *i*CSPM, while the second row contains the results of *i*CSPM2. Each row's best results are marked with bold font.

Table 4 shows the experimental results of the first three scenarios in Table 3. The results clearly indicate that both *i*CSPM and *i*CSPM2 performance are sensitive to the value of s. Their performance becomes better with every increase in the value of s. This is expected because any increase in the value of s means that more search regions with smaller sizes are explored simultaneously. Besides, larger islands have better diffusion than smaller islands (Abed-alguni 2019;

Table 4 The effect of various values of s on the performance of *i*CSPM and *i*CSPM2. D = 10, runs = 50 and number of iterations = 10,000

Function	Scenario1 $s = 4$	Scenario2 $s = 8$	Scenario3 $s = 12$
f_1	**0.00E+00**	**0.00E+00**	**0.00E+00**
	0.00E+00	**0.00E+00**	**0.00E+00**
f_2	**0.00E+00**	**0.00E+00**	**0.00E+00**
	0.00E+00	**0.00E+00**	**0.00E+00**
f_3	**0.00E+00**	**0.00E+00**	**0.00E+00**
	0.00E+00	**0.00E+00**	**0.00E+00**
f_4	**8.29E+02**	**8.29E+02**	**8.29E+02**
	2.14E–03	2.55E–03	**7.10E–04**
f_5	**0.00E+00**	**0.00E+00**	**0.00E+00**
	0.00E+00	**0.00E+00**	**0.00E+00**
f_6	2.09E+05	8.38E+04	**4.15E+04**
	– 2.453E+03	– 2.614E+03	**– 4.001+03**
f_7	**0.00E+00**	**0.00E+00**	**0.00E+00**
	0.00E+00	**0.00E+00**	**0.00E+00**
f_8	**4.44E–16**	**4.44E–16**	**4.44E–16**
	4**.44E–16**	**4.44E–16**	**4.44E–16**
f_9	**0.00E+00**	**0.00E+00**	**0.00E+00**
	0.00E+00	**0.00E+00**	**0.00E+00**
f_{10}	– 1.0307	**– 1.0310**	– 1.0301
	– 1.0310	**– 1.0315**	– 1.0312
f_{11}	– 4.42E+02	– 4.43E+02	**– 4.47E+02**
	– 4.50E+02	**– 4.50E+02**	**– 4.50E+02**
f_{12}	– 3.42E+02	– 3.38E+02	**– 3.47E+02**
	– 4.50E+02	**– 4.50E+02**	**– 4.50E+02**
f_{13}	4.85E+02	**3.92E+02**	**3.92E+02**
	3.90E+02	**3.90E+02**	**3.90E+02**
f_{14}	– 3.283E+02	– 3.295E+02	**– 3.298E+02**
	– 3.30E+02	**– 3.30E+02**	**– 3.30E+02**
f_{15}	– 9.69E+02	– 1.04E+02	**– 1.12E+02**
	– 1.10E+01	– 1.13E+02	**– 1.15E+02**

Table 5 The effect of various values of M_f on the performance of *i*CSPM and *i*CSPM2. D = 10, runs = 50 and number of iterations = 10,000

Function	Scenario4 $M_f = 50$	Scenario5 $M_f = 100$	Scenario6 $M_f = 500$
f_1	**0.00E+00**	**0.00E+00**	**0.00E+00**
	0.00E+00	**0.00E+00**	**0.00E+00**
f_2	**0.00E+00**	**0.00E+00**	**0.00E+00**
	0.00E+00	**0.00E+00**	**0.00E+00**
f_3	**0.00E+00**	**0.00E+00**	**0.00E+00**
	0.00E+00	**0.00E+00**	**0.00E+00**
f_4	**8.29E+02**	**8.29E+02**	**8.29E+02**
	1.02E–03	**7.10E–04**	1.23E–03
f_5	**0.00E+00**	**0.00E+00**	**0.00E+00**
	0.00E+00	**0.00E+00**	**0.00E+00**
f_6	**4.124E+04**	4.15E+04	4.1867E+04
	– 2.636E+03	**– 4.001+03**	– 2.863E+03
f_7	**0.00E+00**	**0.00E+00**	**0.00E+00**
	0.00E+00	**0.00E+00**	**0.00E+00**
f_8	**4.44E–16**	**4.44E–16**	**4.44E–16**
	4.44E–16	**4.44E–16**	**4.44E–16**
f_9	**0.00E+00**	**0.00E+00**	**0.00E+00**
	0.00E+00	**0.00E+00**	**0.00E+00**
f_{10}	**– 1.0310**	– 1.0301	– 1.0276
	– 1.0313	– 1.0310	– 1.0311
f_{11}	– 4.45E+02	**– 4.47E+02**	– 4.39E+02
	– 4.50E+02	**– 4.50E+02**	**– 4.50E+02**
f_{12}	**– 3.69E+02**	– 3.47E+02	– 2.80E+02
	– 4.50E+02	**– 4.50E+02**	**– 4.50E+02**
f_{13}	3.93E+02	**3.92E+02**	8.92E+02
	3.90E+02	**3.90E+02**	**3.90E+02**
f_{14}	**– 3.30E+02**	– 3.299E+02	– 3.281E+02
	– 3.30E+02	**– 3.30E+02**	**– 3.30E+02**
f_{15}	– 9.46E+02	**– 1.12E+02**	– 1.11E+02
	– 1.11E+01	**– 1.15E+02**	– 1.12E+02

Abed-Alguni et al. 2019; Al-Betar et al. 2019; Al-Betar and Awadallah 2018). Therefore, $s = 12$ was selected for the next six scenarios.

Table 5 shows the simulation results of the next three experimental scenarios (Scenario4, Scenario5 and Scenario6), which were designed to show the effect of different values of M_f on the convergence behavior of *i*CSPM and *i*CSPM2. The rank of the scenarios in Table 5 were: Scenario4 ($M_f = 100$), Scenario5 ($M_f = 50$) and finally Scenario6 ($M_f = 500$). This means that low and medium migration frequencies generate better results than large-migration frequencies. This may be because large-scale migrations have more effects on the diversity of populations in the islands compared to low-scale migrations. The value of $M_f = 100$, which achieved the best results in Table 5, was used in the last three experimental scenarios.

Table 6 shows the simulation results for the last three experimental scenarios (Scenario7, Scenario8 and Scenario9). There is no clear indication in Table 6 on the effects of low and high values of M_r on the performance of *i*CSPM and *i*CSPM2. This is maybe because of the small dimension of the benchmark functions.

The results in Tables 4, 5 and 6 show that *i*CSPM and *i*CSPM2 achieved similar results for 7 benchmark functions (f_1, f_2, f_3, f_5, f_7, f_8, f_9). However, *i*CSPM2 outperformed *i*CSPM for 8 out of 15 functions. These observations suggest that *i*CSPM2 has better performance than *i*CSPM.

Table 6 The effect of various values of M_r on the performance of *i*CSPM and *i*CSPM2. D = 10, runs = 50 and number of iterations = 10,000 and number of runs is 50

Function	Scenario7 $M_r = 10\%$	Scenario8 $M_r = 20\%$	Scenario9 $M_r = 30\%$
f_1	**0.00E+00**	**0.00E+00**	**0.00E+00**
	0.00E+00	**0.00E+00**	**0.00E+00**
f_2	**0.00E+00**	**0.00E+00**	**0.00E+00**
	0.00E+00	**0.00E+00**	**0.00E+00**
f_3	**0.00E+00**	**0.00E+00**	**0.00E+00**
	0.00E+00	**0.00E+00**	**0.00E+00**
f_4	**8.29E+02**	**8.29E+02**	**8.29E+02**
	1.12E–02	**2.055E–03**	9.10E–03
f_5	**0.00E+00**	**0.00E+00**	**0.00E+00**
	0.00E+00	**0.00E+00**	**0.00E+00**
f_6	**4.124E+04**	4.1566E+04	4.1867E+04
	– 2.690E+03	**– 2.690E+03**	– 2.541+03
f_7	**0.00E+00**	**0.00E+00**	**0.00E+00**
	0.00E+00	**0.00E+00**	**0.00E+00**
f_8	**4.44E–16**	**4.44E–16**	**4.44E–16**
	4.44E–16	**4.44E–16**	**4.44E–16**
f_9	**0.00E+00**	**0.00E+00**	**0.00E+00**
	0.00E+00	**0.00E+00**	**0.00E+00**
f_{10}	– 1.0301	**– 1.0302**	– 1.0266
	– 1.0290	– 1.0310	**– 1.0330**
f_{11}	– 4.46E+02	**– 4.47E+02**	– 4.42E+02
	– 4.50E+02	**– 4.50E+02**	**– 4.50E+02**
f_{12}	– 3.68E+02	**– 3.69E+02**	– 2.79E+02
	– 4.50E+02	**– 4.50E+02**	**– 4.50E+02**
f_{13}	3.94E+02	**3.9E+02**	8.96E+02
	3.90E+02	**3.90E+02**	**3.90E+02**
f_{14}	**– 3.30E+02**	– 3.299E+02	– 3.281E+02
	– 3.30E+02	**– 3.30E+02**	**– 3.30E+02**
f_{15}	– 9.49E+01	– 1.04E+02	**– 1.12E+02**
	– 1.12E+01	**– 1.15E+02**	**– 1.15E+02**

5.5 Comparison between *i*CSPM2 and other well-known optimization algorithms

We used the 30 single-objective, real-parameter, optimization, functions of CEC2014, described in Sect. 5.2, to compare the simulation results of *i*CSPM2 (Scenario 8) to the reported simulation results in Abed-alguni and Barhoush (2018) for four popular optimization algorithms (DGWO Abed-alguni and Barhoush 2018, L-SHADE Tanabe and Fukunaga 2014, FWA-DM Yu et al. 2014 and MHDA Ks and Murugan 2017).

Table 7 shows the function error values (FEV) for the 30 CEC 2014 functions. FEV is the distance between the best objective value calculated over multiple runs by an optimization algorithm and the actual optimal value. In the table, the best FEV for each function is highlighted in **Bold**. It is obvious that *i*CSPM2 outperforms all the algorithms in Table 7 by achieving the lowest FEV values for 14 out of the 30 functions. We suggest three possible reasons for the superior performance of *i*CSPM2. Firstly, it partitions the population of n candidate solutions for a given optimization between s independent islands and then it equally divides them among 4 efficient variations of CS: CS1, CS10, CSJ and CS11. Secondly, the island model and its migration process provide a suitable environment for unfitted solutions to evolve to better solutions. Thirdly, it uses EOBL to explore the neighborhood of the elite solutions in the population of each island.

In addition, L-SHADE is the algorithm with the second best performance in Table 7. It achieved the lowest FEV for almost a third of the 30 CEC2014 functions, which may be because L-SHADE dynamically adjusts its internal parameters and population size during its optimization process.

Figure 5 shows the convergence charts of *i*CSPM2, DGWO and L-SHADE over the first 1000 iterations for selected functions from CEC2014 (F_2, F_5, F_{11}, F_{15}, F_{20}, F_{27}). The charts clearly show that *i*CSPM2 converges to good solutions faster than DGWO and L-SHADE. This is because *i*CSPM2 is based on the island model and obtains the advantages of the multiple mutation methods it uses with CS. The charts also show that L-SHADE is the second fastest algorithm followed by DGWO.

5.6 Results of statistical tests

The results of Friedman's test (Table 8 and Fig. 4) are statistical information about the ranks of *i*CSPM2, DGWO, L-SHADE, MHDA and FWA-DM. In this table, the best rank is marked with **bold** font. The order of ranks of the algorithms was as follows: *i*CSPM2, L-SHADE, DGWO, MHDA and FWA-DM, respectively. This means that *i*CSPM2 performed best amongst the tested algorithms (Fig. 5).

5.7 Performance analysis of *i*CSPM2 in comparison to GAIbH and MASC using Taillard's benchmark for PFSSP

We compared the performance of *i*CSPM2 to the performance of GAIbH and MASC using 12 instances of Taillard's benchmark (described in Sect. 5.2). Table 9 shows the mean makespans over 50 runs and the ARD (average relative difference) values. An ARD value shows the relative difference between the obtained mean value using an optimization algorithm to the best-known value for a given benchmark function. The ARD values were computed using the follow-

Table 7 Simulation results for *i*CSPM2, DGWO, L-SHADE, FWA-DM and MHDA

Function	iCSPM2	DGWO	L-SHADE	FWA-DM	MHDA
F_1	**19.20E–16**	4.36E+00	9.00E–15	4.91E+05	3.59E+03
F_2	**1.45E–20**	2.36E+00	8.50E–11	2.50E–16	3.82E+03
F_3	**1.88E–17**	2.54E–04	5.83E–10	1.88E–16	5.80E–07
F_4	**1.73E–10**	1.63E–09	2.58E–09	2.23E+01	1.42E–08
F_5	4.67E+00	2.00E+02	2.00E+01	2.11E+01	**2.36E**+00
F_6	**7.15E–16**	1.21E+00	1.25E–06	1.82E+01	8.52E–14
F_7	2.26E–10	8.53E–10	7.25E–09	2.53E–03	**2.25E–11**
F_8	8.55E–17	**1.51E–19**	1.25E–09	9.53E–15	2.20E–19
F_9	**1.03E+00**	**1.03E+00**	8.96E+00	6.54E+01	5.30E+00
F_{10}	**1.08E–03**	3.20E–03	2.36E–02	1.13E+01	1.22E+03
F_{11}	**1.20E+02**	2.95E+03	2.30E+03	2.19E+03	1.52E+02
F_{12}	**6.29E–02**	6.30E–02	9.00E–01	3.25E–01	1.42E–01
F_{13}	3.14E–01	4.59E–01	6.50E–01	**3.11E–01**	4.78E–01
F_{14}	2.01E–01	**1.99E–01**	8.60E–01	2.99E–01	5.43E–01
F_{15}	1.69E+00	7.23E+01	**1.60E+00**	8.36E+00	3.25E+00
F_{16}	**9.20E+00**	9.53E+00	1.02E+01	1.10E+01	1.06E+01
F_{17}	3.55E+00	4.55E+03	**2.20E+00**	6.59E+03	4.53E+02
F_{18}	3.96E+00	3.94E+01	**1.90E+00**	7.24E+01	3.69E+00
F_{19}	5.61E+00	1.22E+02	**5.30E+00**	1.04E+01	3.78E+02
F_{20}	**4.19E+00**	4.73E+02	4.30E+00	4.37E+01	7.09E+02
F_{21}	2.82E+02	7.09E+02	3.69E+02	8.75E+02	**2.57E+02**
F_{22}	**1.32E+02**	2.73E+02	**1.32E+02**	1.62E+02	2.73E+02
F_{23}	**3.68E+01**	3.69E+0	3.26E+02	3.16E+02	3.10E+03
F_{24}	1.94E+02	2.25E+02	**1.93E+02**	2.96E+02	2.26E+02
F_{25}	2.01E+02	2.11E+02	**2.00E+02**	2.09E+02	2.11E+02
F_{26}	0.89E+02	2.10E+02	2.69E+02	**9.93E+01**	1.00E+02
F_{27}	**1.26E+02**	4.09E+02	**1.26E+02**	4.10E+02	4.05E+02
F_{28}	3.71E+02	1.65E+03	**3.62E+02**	4.22E+02	1.54E+03
F_{29}	2.30E+02	**2.29E+02**	7.33E+02	2.78E+02	7.86E+02
F_{30}	2.91E+00	**2.83E+00**	6.99E+02	4.69E+02	2.63E+03

D = 30, runs = 30 and number of iterations = 10,000

ing equation Wang et al. (2017):

$$ARD = \frac{100 \times (C^{\text{opt}} - C^{A})}{C^{\text{opt}}} \tag{18}$$

where C^A is the makespan achieved by the tested algorithm and C^{opt} is the upper bound with the minimum value for the tested instance of Taillard's benchmark.

The simulation results in Table 9 suggest that *i*CSPM2 outperforms GAIbH and MASC. *i*CSPM2 achieved the best known results (lowest Mean and ARD) for 10 instances out of the 12 instances of Taillard's benchmark and it also scored the lowest total average ARD (0.23%). This may be because *i*CSPM2 incorporates four powerful variations of CS into the island model, which increases the diversity of its candidates solutions and improves its convergence behavior.

6 Discussion

The simulation results in Table 7 showed that *i*CSPM2 outperforms all the algorithms by achieving the lowest FEV values for 14 out of the 30 CEC 2014 functions (F_1-F_4, F_6, F_9-F_{12}, F_{16}, F_{20}, F_{23}, F_{27}). *i*CSPM2 also achieved the second lowest FEV values for 12 out of the 30 CEC 2014 functions (F_5, F_7, F_{13}, F_{14}, F_{17}, F_{19}, F_{21}, F_{25}, F_{26}, F_{28}-F_{30}). L-SHADE provides competitive results (second best performing algorithm with the lowest FEV values for 9 out of the 30 CEC 2014 functions) compared to *i*CSPM2. This is mainly because it follows a dynamic adjustment procedure for its internal parameters and population size during its optimization process. Besides, it is based on the DE algorithm, which is one of the most efficient basic evolutionary algorithms (Tanabe and Fukunaga 2014). According to Table 7, DGWO is the third best performing algorithm and it provides

Table 8 Results of Friedman's test

vskip Function	Ranks of the algorithms				
	*i*CSPM2	DGWO	L-SHADE	MHDA	FWA-DM
F_1	**1**	3	2	4	5
F_2	**1**	4	3	5	2
F_3	**1**	5	3	4	2
F_4	**1**	2	3	4	5
F_5	**2**	5	3	1	4
F_6	**1**	4	3	2	5
F_7	2	3	4	**1**	5
F_8	**3**	1	5	2	4
F_9	**1.5**	**1.5**	4	3	5
F_{10}	**1**	2	3	5	4
F_{11}	**1**	5	4	2	3
F_{12}	**1**	2	5	3	4
F_{13}	2	3	5	4	**1**
F_{14}	2	**1**	5	4	3
F_{15}	2	5	**1**	3	4
F_{16}	**1**	2	3	4	5
F_{17}	2	4	**1**	3	5
F_{18}	3	4	**1**	2	5
F_{19}	2	4	**1**	5	3
F_{20}	**1**	4	2	5	3
F_{21}	2	4	3	**1**	5
F_{22}	**1.5**	4.5	**1.5**	4.5	3
F_{23}	2	**1**	4	5	3
F_{24}	2	3	**1**	4	5
F_{25}	2	4.5	**1**	4.5	3
F_{26}	**1**	4	5	3	2
F_{27}	**1.5**	4	**1.5**	3	5
F_{28}	2	5	**1**	4	3
F_{29}	2	**1**	4	5	3
F_{30}	2	**1**	4	5	3
Sum of ranks	**49.5**	96.5	87	105	112
Sum of ranks squared	**2450.25**	9312.25	7569	11,025	12,544
Average of ranks	**1.65**	3.2166	2.9	3.5	3.733

competitive results to the results of *i*CSPM2 and L-SHADE. This is because DGWO is a variation of GWO that is based on the island model that improves the diversity and performance of GWO. The performance of FWA-DM and MHDA are the worst among all the algorithms in Table 7. This is maybe because the exploration operators of these algorithms are not efficient as the ones of *i*CSPM2, L-SHADE and DGWO. Further, the results of Friedman's test in Table 8 showed that *i*CSPM2 achieved the highest rank among all tested algorithms, which means that the simulation results of *i*CSPM2 are significant and provide sufficient evidence that *i*CSPM2 is the best performing algorithm. The results of Friedman's test also confirm that DGWO is the third best performing algorithms and strongly competitive to L-SHADE.

The results in Table 9 also showed that *i*CSPM2 is the best performing algorithm by achieving the lowest ARD values for 10 out of the 12 Taillard's benchmark instances for the PFSSP. Note that *i*CSPM2 provided the lowest ARD values for the most complex instances of Taillard's benchmark (Ta110 and Ta120). The results also showed that *i*CSPM2 has the lowest average ARD value (0.23) for the 12 instances of Taillard's benchmark over 50 independent runs. This means that *i*CSPM2, in general, provides better schedules for the 12 instances of Taillard's benchmark than other tested algorithms. On the other hand, MASC in Table 9 is the second

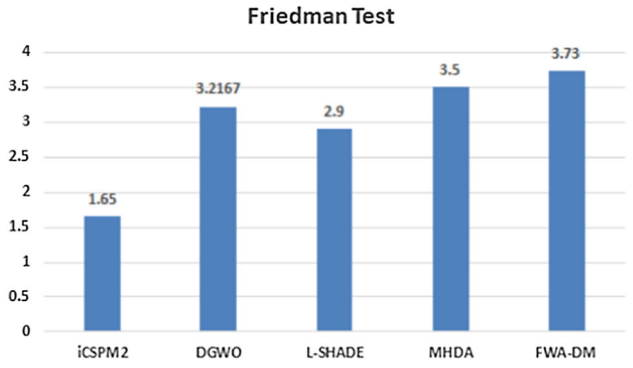

Fig. 4 The average ranks of the algorithms according to Friedman's Test

best performing algorithm and also provides competitive results compared to the results of *i*CSPM2. This may be because it combines the strengths of the simulated annealing and Nawaz–Enscore–Ham algorithms into the GA algorithm. Note that GAIbH is the worst performing algorithm in Table 9. This may be because GAIbH is only based on a heuristic procedure that calculates the completion times of all jobs in a job permutation based on the job position in other existing partial job permutations.

There are three main explanations for the superior performance of *i*CSPM2 in solving the CEC 2014 suit and 12 instances of Taillard's benchmark for the PFSSP.

Firstly, *i*CSPM2 is based on the island model, while the other tested algorithms (DGWO, L-SHADE, MHDA, FWA-DM, GAIbH and MASC) are not based on the island model. The island model provides several advantages when integrated with an evolutionary algorithm (Abed-alguni and Barhoush 2018; Al-Betar et al. 2019; Al-Betar and Awadallah 2018; Awadallah et al. 2020). It increases the chances of candidate solutions with low objective values evolving into better ones. It can also be run in parallel on several processing devices or a device that supports parallel processing. The migration mechanism in the island model regulates population diversity and lowers the possibility of early convergence.

Secondly, the four variations of CS used in *i*CSPM2 have unique advantages. CS1 utilizes the Lévy flight mutation, which has a better exploration capability than the random mutation method (Yang and Deb 2009). CS10 uses the HDP mutation method that can sample the entire search space between the lower and upper bounds of a decision variable regardless of its value (Abed-Alguni and Paul 2018; Abed-alguni 2019; Alawad and Abed-alguni 2020). CSJ uses the Jaya mutation method that mutates the candidate solutions using stochastic moves based on the best and worst candidate solutions (Rao 2016). CS11 uses the pitch adjustment mutation, which is known for its quick convergence (Abed-Alguni and Paul 2018; Al-Betar et al. 2015). In summary, we conclude that the mutation methods in *i*CSPM2 collectively provide it with strong exploration and exploitation capabilities.

Finally, *i*CSPM2 utilizes two well-known opposition learning approaches (OBL and EOBL), while the other tested algorithms in Sect. 5 do not utilize any opposition learning approach. The OBL approach is used in the initialization step of *i*CSPM2 to widen the search are by considering the opposite solutions of randomly generated solutions. This helps in increasing the diversity of initial population. The EOBL approach speeds up the convergence of *i*CSPM2 by exploring the opposite solutions of the best-known candidate solutions (Paiva et al. 2017; Zhang et al. 2017).

7 Conclusions and future work

This paper introduced an improved variation of the island-based Cuckoo Search algorithm by the name island-based Cuckoo Search with elite opposition-based learning and multiple mutation methods (*i*CSPM2). The source code of *i*CSPM2 is publicly available at https://github.com/bilalh2021/iCSPM2. The new algorithm distributes the population of candidate solutions for a given optimization among s independent islands and then it evenly divides the islands among 4 variations of Cuckoo Search (Cuckoo search via Lévy flights, Cuckoo Search with polynomial mutation, Cuckoo Search with Jaya mutation, Cuckoo Search with pitch adjustment mutation). This means that each variation of Cuckoo Search is applied to $n/4$ islands. Besides, *i*CSPM2 uses elite opposition-based learning to explore the neighborhood of elite solutions in the population of each island. *i*CSPM2 is capable of solving complex, scheduling problems as well as continuous optimization problems. This is because it uses the smallest position value method with complex scheduling problems, such as the permutation flow shop scheduling problem, to convert the continuous candidate solutions into discrete ones.

*i*CSPM2 has been shown to perform better than a number of other well-known optimization algorithms through several experiments conducted in order to test its performance using popular benchmark suites. Firstly, the sensitivity of *i*CSPM and *i*CSPM2 to the number of islands, s, the migration frequency, M_f, and the migration rate, M_r, were studied and analyzed based on different experimental scenarios. Overall results indicate that higher values of s and lower values of M_f give better performance for both *i*CSPM and *i*CSPM2. However, it remains unclear whether high or low values of M_r improve either algorithm's performance. In any case, the experimental results indicate that *i*CSPM2 outperforms *i*CSPM. Secondly, performance of *i*CSPM2 was compared against four well-known swarm optimization algorithms: DGWO (Abed-alguni and Barhoush 2018), L-SHADE (Tanabe and Fukunaga 2014), MHDA (Ks

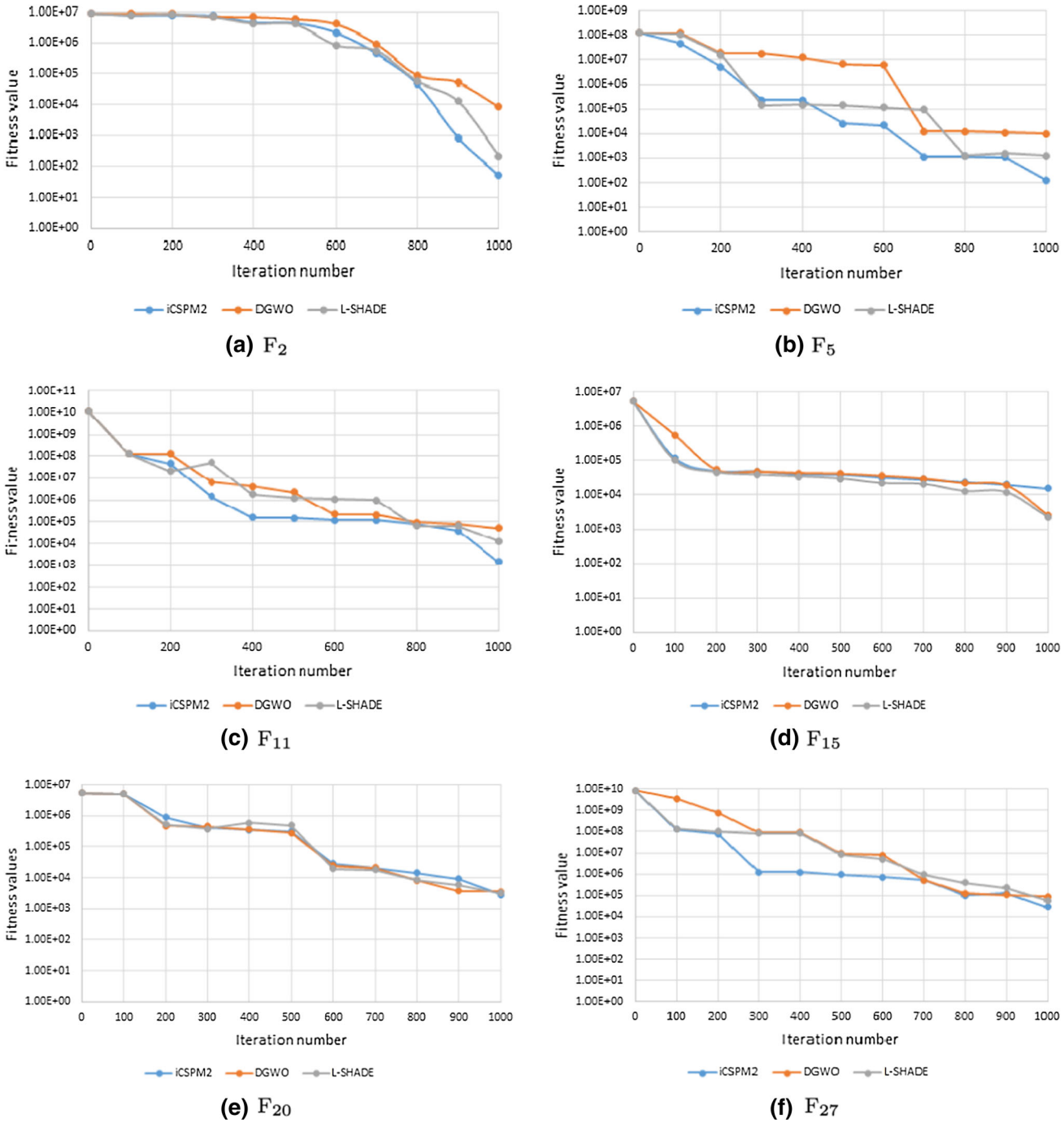

Fig. 5 The convergence charts of *i*CSPM2, DGWO and L-SHADE over the first 1000 iterations for selected functions from CEC2014 (F_2, F_5, F_{11}, F_{15}, F_{20}, F_{27})

and Murugan 2017) and FWA-DM (Yu et al. 2014) using the single-objective IEEE-CEC 2014 functions. The overall results show that *i*CSPM2 performs better than the other well-known swarm optimization algorithms on these functions. Finally, we conducted experiments using 12 instances of Taillard's benchmark for the PFSSP to show that *i*CSPM2 is an efficient algorithm for scheduling problems. In these experiments, *i*CSPM2 was compared to two efficient discrete evolutionary algorithms (GAIbH Fernandez-Viagas et al. 2020 and MASC Kurdi 2020). The results indicate that *i*CSPM2 is a better scheduling algorithm than GAIbH and MASC.

However, *i*CSPM2 has two main limitations. Firstly, the computational complexity of *i*CSPM2 is $O(M_w.s.M_f.k)$,

Table 9 Results of *i*CSPM2, GAIbH and MASC

Problems vskip	Size	*i*CSPM2		GAIbH		MASC		C^{opt}
		Mean	ARD	Mean	ARD	Mean	ARD	
Ta010	20 × 5	**1108**	**0.00**	**1108**	**0.00**	**1108**	**0.00**	1108
Ta020	20 × 10	**1378**	**0.00**	**1378**	**0.00**	**1378**	**0.00**	1378
Ta030	20 × 20	**2100**	**0.00**	2123.6	0.05	**2100**	**0.00**	2100
Ta040	50 × 5	**2552**	**0.00**	**2552**	**0.00**	**2552**	**0.00**	2552
Ta050	50 × 10	**3006**	**0.00**	3024	0.03	**3006**	**0.00**	3006
Ta060	50 × 20	**3694.1**	**0.738**	3721.5	1.82	3696	0.74	3635
Ta070	100 × 5	**5014**	**0.00**	5018.7	0.02	5015.35	0.00	5014
Ta080	100 × 10	**5306**	**0.00**	5633	0.51	5306.75	0.02	5303
Ta090	100 × 20	6350	0.662	6482.7	1.94	**6241**	**0.65**	6241
Ta100	200 × 10	10,335.2	0.065	10,648	0.35	**10,331.7**	**0.04**	10,331
Ta110	200 × 20	**11,266.3**	**0.015**	11,347.9	1.89	11,292.35	0.66	11,294
Ta120	500 × 20	**25,931.7**	**0.906**	26,869.3	1.39	26,322.15	1.22	26,083
Total average			0.23		0.67		**0.28**	

which is more than the computational complexity of the basic CS algorithm (O($MaxItr.n$)). Hence, we recommend *i*CSPM2 be executed over s parallel devices, which will reduce its computational complexity to O($M_w.M_f.k$). Secondly, *i*CSPM2 can only be directly applied to single-objective optimization problems. Therefore, one of our future goals is to develop a variation of *i*CSPM2 that can solve some multi-objective optimization problems.

In the future, the following would be interesting research studies:

- Incorporating the mutation methods from Equilibrium optimizer (Faramarzi et al. 2020), Jaya (Rao 2016), Grasshopper (Saremi et al. 2017), L-SHADE and MHDA into *i*CSPM2.
- Applying *i*CSPM2 to multi-agent cooperative reinforcement learning (Abed-alguni et al. 2015a, b; Abed-alguni and Ottom 2018; Abed-Alguni et al. 2016) based on the models described in Abed-alguni (2018, 2017); Abed-Alguni (2014).
- Incorporating *i*CSPM into an intelligent distributed recommender system based on the problem model discussed in Alawad et al. (2016).
- Incorporating the island model with the arithmetic optimization algorithm (Abualigah et al. 2021) and Aquila optimizer (Abualigah et al. 2021) to solve the feature-section problem described in Abualigah et al. (2019).
- Finally, it could be possible to improve other heuristics and metaheurisics by enhancing them with different mutation methods and elite opposition-based learning in a similar manner to how island-based Cuckoo Search was improved in this paper.

Author contributions BHA contributed to conceptualization, methodology, investigation, validation, writing, experimentation, visualization, reviewing and editing. DP performed writing, reviewing and editing.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

Human and animal rights This article does not contain any studies with human participants or animals performed by any of the authors.

References

Abadlia H, Smairi N, Ghedira K (2017) Particle swarm optimization based on dynamic island model. In: 2017 IEEE 29th international conference on tools with artificial intelligence (ICTAI). IEEE, pp 709–716

Abed-alguni BH, Alawad NA, Barhoush M, Hammad R (2021) Exploratory cuckoo search for solving single-objective optimization problems. Soft Comput 1–14

Abed-alguni BH, Alkhateeb F (2018) Intelligent hybrid cuckoo search and β-hill climbing algorithm. J King Saud Uni Comput Inform Sci 1–43

Abed-Alguni BHK (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 BH, Klaib AF (2018) Hybrid whale optimisation and β-hill climbing algorithm. Int J Comput Sci Math 1–13

Abed-Alguni BH, Paul DJ, Chalup SK, Henskens FA (2016) A comparison study of cooperative Q-learning algorithms for independent learners. Int J Artif Intell 14(1):71–93

Abed-alguni BH (2017) Bat Q-learning algorithm. Jordanian J Comput Inform Technol 3(1):56–77

Abed-alguni BH (2018) Action-selection method for reinforcement learning based on cuckoo search algorithm. Arab J Sci Eng 43(12):6771–6785

Abed-alguni BH (2019) Island-based cuckoo search with highly disruptive polynomial mutation. Int J Artif Intell 17(1):57–82

Abed-alguni BH, Alawad NA (2021) Distributed grey wolf optimizer for scheduling of workflow applications in cloud environments. Appl Soft Comput 102:107113

Abed-alguni BH, Alkhateeb F (2017) Novel selection schemes for cuckoo search. Arab J Sci Eng 42(8):3635–3654

Abed-alguni BH, Barhoush M (2018) Distributed grey wolf optimizer for numerical optimization problems. Jordanian J Comput Inf Technol 4:130–149

Abed-alguni BH, Ottom MA (2018) Double delayed Q-learning. Int J Artif Intell 16(2):41–59

Abed-Alguni BH, Paul DJ (2018) Hybridizing the cuckoo search algorithm with different mutation operators for numerical optimization problems. J Intell Syst 29(1):1043–1062

Abed-alguni BH, Chalup SK, Henskens FA, Paul DJ (2015) Erratum to: a multi-agent cooperative reinforcement learning model using a hierarchy of consultants, tutors and workers. Vietnam J Comput Sci 2(4):227

Abed-alguni BH, Chalup SK, Henskens FA, Paul DJ (2015) A multi-agent cooperative reinforcement learning model using a hierarchy of consultants, tutors and workers. Vietnam J Comput Sci 2(4):213–226

Abed-Alguni BH, Klaib AF, Nahar KM (2019) Island-based whale optimisation algorithm for continuous optimisation problems. Int J Reason Based Intell Syst 11(4):319–329

Abualigah LMQ et al (2019) Feature selection and enhanced krill herd algorithm for text document clustering. Springer

Abualigah L, Yousri D, Abd Elaziz M, Ewees AA, Al-qaness MA, Gandomi AH (2021) Aquila optimizer: a novel meta-heuristic optimization algorithm. Comput Indus Eng 157:107250

Abualigah L, Diabat A, Mirjalili S, Abd Elaziz M, Gandomi AH (2021) The arithmetic optimization algorithm. Comput Methods Appl Mech Eng 376:113609

Alawad NA, Abed-alguni BH (2020) Discrete island-based cuckoo search with highly disruptive polynomial mutation and opposition-based learning strategy for scheduling of workflow applications in cloud environments. Arab J Sci Eng 1–30

Alawad NA, Abed-alguni BH (2021) Discrete jaya with refraction learning and three mutation methods for the permutation flow shop scheduling problem. J Supercomput 1–17

Alawad NA, Abed-alguni BH (2021) Discrete jaya with refraction learning and three mutation methods for the permutation flow shop scheduling problem. J Supercomput 1–22

Alawad NA, Anagnostopoulos A, Leonardi S, Mele I, Silvestri F (2016) Network-aware recommendations of novel tweets. In: Proceedings of the 39th international ACM SIGIR conference on research and development in information retrieval. ACM, pp 913–916

Al-Betar MA (2021) Island-based harmony search algorithm for non-convex economic load dispatch problems. J Elect Eng Technol 1–31

Al-Betar MA, Awadallah MA (2018) Island bat algorithm for optimization. Expert Syst Appl 107:126–145

Al-Betar MA, Awadallah MA, Khader AT, Abdalkareem ZA (2015) Island-based harmony search for optimization problems. Expert Syst Appl 42(4):2026–2035

Al-Betar MA, Awadallah MA, Doush IA, Hammouri AI, Mafarja M, Alyasseri ZAA (2019) Island flower pollination algorithm for global optimization. J Supercomput 75(8):5280–5323

Ali IM, Essam D, Kasmarik K (2019) A novel differential evolution mapping technique for generic combinatorial optimization problems. Appl Soft Comput 80:297–309

Alkhateeb F, Abed-alguni BH, Al-rousan MH (2021) Discrete hybrid cuckoo search and simulated annealing algorithm for solving the job shop scheduling problem. J Supercomput 1–28

Alkhateeb F, Abed-Alguni BH (2017) A hybrid cuckoo search and simulated annealing algorithm. J Intell Syst

Awadallah MA, Al-Betar MA, Bolaji AL, Doush IA, Hammouri AI, Mafarja M (2020) Island artificial bee colony for global optimization. Soft Computing, pp 1–27

Casanova H, Giersch A, Legrand A, Quinson M, Suter F (2014) Versatile, scalable, and accurate simulation of distributed applications and platforms. J Parallel Distrib Comput 74(10):2899–2917

Chen H, Heidari AA, Chen H, Wang M, Pan Z, Gandomi AH (2020) Multi-population differential evolution-assisted harris hawks optimization: Framework and case studies. Futur Gener Comput Syst 111:175–198

Corcoran AL, Wainwright RL (1994) A parallel island model genetic algorithm for the multiprocessor scheduling problem. In: Proceedings of the 1994 ACM symposium on applied computing, Phoenix, Arizona, USA (New York, NY, USA). ACM, pp 483–487

Derrac J, García S, Molina D, Herrera F (2011) A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. Swarm Evol Comput 1(1):3–18

Doush I, Hasan B, Al-Betar M, AlMaghayreh E, Alkhateeb F (2014) Artificial bee colony with different mutation schemes: a comparative study. Comput Sci J Moldova 64(1):77–98

Faramarzi A, Heidarinejad M, Stephens B, Mirjalili S (2020) Equilibrium optimizer: a novel optimization algorithm. Knowl-Based Syst 191:105190

Fernandez-Viagas V, Molina-Pariente JM, Framinan JM (2020) Generalised accelerations for insertion-based heuristics in permutation flowshop scheduling. Eur J Oper Res 282(3):858–872

Friedman M (1940) A comparison of alternative tests of significance for the problem of m rankings. Ann Math Stat 11(1):86–92

Geem ZW, Kim JH, Loganathan G (2001) A new heuristic optimization algorithm: harmony search. Simulation 76(2):60–68

Guo S-S, Wang J-S, Ma X-X (2019) Improved bat algorithm based on multipopulation strategy of island model for solving global function optimization problem. Comput Intell Neurosci 2019

Hasan BHF, Doush IA, Al Maghayreh E, Alkhateeb F, Hamdan M (2014) Hybridizing harmony search algorithm with different mutation operators for continuous problems. Appl Math Comput 232:1166–1182

Karaboga D, Basturk B (2007) Artificial bee colony (abc) optimization algorithm for solving constrained optimization problems. In: International fuzzy systems association world congress. Springer, pp 789–798

Kennedy J, Eberhart R (1995) Particle swarm optimization. In: Proceedings of ICNN'95-international conference on neural networks, vol 4. IEEE, pp 1942–1948

Komusiewicz C, Kratsch D et al (2020) Matching cut: kernelization, single-exponential time fpt, and exact exponential algorithms. Disc Appl Math 283:44–58

Ks SR, Murugan S (2017) Memory based hybrid dragonfly algorithm for numerical optimization problems. Exp Syst Appl 83:63–78

Kurdi M (2020) A memetic algorithm with novel semi-constructive evolution operators for permutation flowshop scheduling problem. Appl Soft Comput 106458

Kurdi M (2016) An effective new island model genetic algorithm for job shop scheduling problem. Comput Oper Res 67:132–142

Kushida J-i, Hara A, Takahama T, Kido A (2013) Island-based differential evolution with varying subpopulation size. In: 2013 IEEE 6th international workshop on computational intelligence and applications (IWCIA). IEEE, pp 119–124

Lardeux F, Goëffon A (2010) A dynamic island-based genetic algorithms framework. In: Asia-Pacific conference on simulated evolution and learning. Springer, pp 156–165

Liang JJ, Qu BY, Suganthan PN (2013) Problem definitions and evaluation criteria for the cec, 2014 special session and competition on single objective real-parameter numerical optimization. In: Computational intelligence laboratory, vol 635. Zhengzhou University, Zhengzhou China and Technical Report, Nanyang Technological University, Singapore, p 2013

Liu Y, Cao B, Li H (2020)Improving ant colony optimization algorithm with epsilon greedy and levy flight. Compl Intell Syst 1–12

Mehta S, Kaur P (2019) Scheduling data intensive scientific workflows in cloud environment using nature inspired algorithms. In: Nature-inspired algorithms for big data frameworks. IGI Global, pp 196–217

Michel R, Middendorf M (1998) An island model based ant system with lookahead for the shortest supersequence problem. In: International conference on parallel problem solving from nature, Amsterdam, The Netherlands. Springer, pp 692–701

Mirjalili S, Lewis A (2016) The whale optimization algorithm. Adv Eng Softw 95:51–67

Mirjalili S, Mirjalili SM, Lewis A (2014) Grey wolf optimizer. Adv Eng Softw 69:46–61

Mohammed SMZ, Khader AT, Al-Betar MA (2016) 3-sat using island-based genetic algorithm. IEEJ Trans Electron Inform Syst 136(12):1694–1698

Mugemanyi S, Qu Z, Rugema FX, Dong Y, Bananeza C, Wang L (2020) Optimal reactive power dispatch using chaotic bat algorithm. IEEE Access 8:65830–65867

Paiva FA, Silva CR, Leite IV, Marcone MH, Costa JA (2017) Modified bat algorithm with cauchy mutation and elite opposition-based learning. In: 2017 IEEE Latin American conference on computational intelligence (LA-CCI). IEEE, pp 1–6

Rakhshani H, Rahati A (2016) Intelligent multiple search strategy cuckoo algorithm for numerical and engineering optimization problems. Arabian J Sci Eng 1–27

Rao R (2016) Jaya: a simple and new optimization algorithm for solving constrained and unconstrained optimization problems. Int J Ind Eng Comput 7(1):19–34

Saremi S, Mirjalili S, Lewis A (2017) Grasshopper optimisation algorithm: theory and application. Adv Eng Softw 105:30–47

Sihwail R, Omar K, Ariffin KAZ, Tubishat M (2020) Improved harris hawks optimization using elite opposition-based learning and novel search mechanism for feature selection. IEEE Access 8:121127–121145

Skakovski A, Jedrzejowicz P (2019) An island-based differential evolution algorithm with the multi-size populations. Exp Syst Appl 126:308–320

Taillard E (1990) Some efficient heuristic methods for the flow shop sequencing problem. Eur J Oper Res 47(1):65–74

Tanabe R, Fukunaga AS (2014) Improving the search performance of shade using linear population size reduction. In: 2014 IEEE congress on evolutionary computation (CEC). IEEE, pp 1658–1665

Wang H, Wang W, Sun H, Cui Z, Rahnamayan S, Zeng S (2017) A new cuckoo search algorithm with hybrid strategies for flow shop scheduling problems. Soft Comput 21(15):4297–4307

Yang X-S, Deb S (2009) Cuckoo search via lévy flights. In: World congress on nature and biologically inspired computing. NaBIC. IEEE, pp 210–214

Yang X-S (2012) Flower pollination algorithm for global optimization. In: International conference on unconventional computing and natural computation. Springer, pp 240–249

Yu C, Kelley L, Zheng S, Tan Y (2014) Fireworks algorithm with differential mutation for solving the cec 2014 competition problems. In: 2014 IEEE congress on evolutionary computation (CEC). IEEE, pp 3238–3245

Yusta SC (2009) Different metaheuristic strategies to solve the feature selection problem. Pattern Recogn Lett 30(5):525–534

Zhang S, Luo Q, Zhou Y (2017) Hybrid grey wolf optimizer using elite opposition-based learning strategy and simplex method. Int J Comput Intell Appl 16(02):1750012

Zhou X, Wu Z, Wang H, Li K, Zhang H (2013) Elite opposition-based particle swarm optimization. Acta Electron Sin 41(8):1647–1652

Zhou Y, Wang R, Zhao C, Luo Q, Metwally MA (2019) Discrete greedy flower pollination algorithm for spherical traveling salesman problem. Neural Comput Appl 31(7):2155–2170