Neural Computing and Applications
https://doi.org/10.1007/s00521-022-07034-6

ORIGINAL ARTICLE

FP-SMA: an adaptive, fluctuant population strategy for slime mould algorithm

Jassim Alfadhli[1] · Ali Jaragh[1] · Mohammad Gh. Alfailakawi[1] · Imtiaz Ahmad[1]

Received: 16 August 2021 / Accepted: 30 January 2022

Abstract
In this paper, an adaptive Fluctuant Population size Slime Mould Algorithm (FP-SMA) is proposed. Unlike the original SMA where population size is fixed in every epoch, FP-SMA will adaptively change population size in order to effectively balance exploitation and exploration characteristics of SMA's different phases. Experimental results on 13 standard and 30 IEEE CEC2014 benchmark functions have shown that FP-SMA can achieve significant reduction in run time while maintaining good solution quality when compared to the original SMA. Typical saving in terms of function evaluations for all benchmarks was between 20 and 30% on average with a maximum being as high as 60% in some cases. Therefore, with its higher computation efficiency, FP-SMA is much more favorable choice as compared to SMA in time stringent applications.

Keywords Fluctuant population (FP) · Slime mould algorithm (SMA) · Metaheuristic algorithm (MA) · Population diversity · Population adaptation

1 Introduction

Population-based meta-heuristics have been the dominant methods to find optimal or good solutions to many complex optimization problems in reasonable time [22]. The popularity of meta-heuristics has increased considerably due to their key role in learning and knowledge discovery within the emerging fields of big data and machine learning. These meta-heuristics derive their inspiration from mimicking intelligent processes arising in nature, and are commonly classified into two categories: evolutionary (EA) and swarm intelligence algorithms [13]. The most popular EA algorithms are genetic (GA) [17] and differential evolution (DE) [47]. As for the swarm intelligence category, this includes particle swarm (PSO) [47], cuckoo search (CS) [58], whale optimization [32], monarch butterfly optimization (MBO) [53], moth search (MSA) [52], and Harris hawks optimization (HHO) [19] among others.

Although the development of meta-heuristics has witnessed tremendous progress in recent years, there is still much room for improvement as no single algorithm can solve all problems efficiently as per the "No Free Lunch" theorem [55]. Recently, a new population-based meta-heuristic called slime mould algorithm (SMA) has been proposed by Li et al. [27]. SMA is inspired by a unique slime mould, i.e., physarum polycephalum, which is an organism that can live freely as a single cell but can also form communicating aggregates when foraging food sources. In order to find food, slime mould starts the search process with a randomly distributed population. Once having identified food concentration during the random search, the slime mould will approach and wrap the food and secrete enzyme to digest it, while retaining certain exploration capability to search for better food sources. In order to imitate slime mould's exploration and exploitation behaviors, authors in [27] proposed a bio-oscillating policy that separates the population into two groups according to their fitness. The first group, called positive group, is the group of individuals from the population with the best fitness whereas the other one, labeled as negative group, is the one consisting of those with the lowest fitness. The better fitness group will be exploited to find the best

✉ Mohammad Gh. Alfailakawi
alfailakawi.m@ku.edu.kw

[1] Computer Engineering Department, College of Engineering and Petroleum, Kuwait University, Safat 13060, Kuwait

Published online: 06 March 2022

possible solution whereas the lower fitness one will be used to explore outward regions. In addition, a vibration parameter based on the sigmoid function is introduced to simulate food-grabbing behavior of slime mould.

Despite SMA being a recent algorithm, it has demonstrated excellent performance compared to state-of-the-art meta-heuristics in many fields. However, one of the key disadvantages of SMA lies in its relatively long run time and high computational cost. Being applied successfully in multiple fields, in this work we investigate the enhancement of the algorithm by augmenting it with a population size adaptation method that can reduce its prohibitively long run time. Population size plays a very important role in both run-time efficiency and optimization effectiveness of meta-heuristics and thus by balancing exploration and exploitation characteristics of the SMA algorithm, its performance and computational cost can be improved [27]. In order to balance exploration and exploitation phases of an algorithm, population size adaptation schemes can automatically adjust population size according to population diversity during the search process thus enhancing performance and reducing run time. Population size adaptation has been widely studied in genetic algorithms [5, 25], differential evolution [40, 50], artificial bee colony optimization [9], swarm intelligence [7, 12, 41] and recently to sine cosine algorithm [3]. However, to the best of the author's knowledge, no such work has been reported for SMA.

To fill this research gap, we propose herein an adaptive fluctuant population size SMA algorithm called FP-SMA. Unlike the original SMA where population size is fixed in every epoch, FP-SMA will adaptively change population size to enhance run time characteristics of both exploitation and exploration phases of the algorithm. Population adaptation concept used in the proposed algorithm is a cluster-based approach borrowed from K-mean clustering algorithm. The threshold used to start the adaptation process is a scaled sigmoid function that decreases smoothly initially, then dramatically midway, and again smoothly near the end of algorithm execution. Once population diversity is out of the threshold, population size increases or decreases using a sine wave pattern (for randomization). Therefore, key contributions of this study can be summarized as the following:

- Propose an adaptive fluctuant population size slime mould algorithm (FP-SMA) that automatically adjust population size during the search process according to population diversity to effectively balance exploitation and exploration characteristics of conventional SMA algorithm.
- FP-SMA performance is analyzed over several benchmark functions, including 13 standard and 30 IEEE CEC2014 benchmark functions.
- Simulation results revealed that FP-SMA can achieve significant reductions in the number of function evaluations as compared to conventional SMA without impacting solution quality.

The remainder of the paper is organized as follows. Section 2 highlights SMA algorithm, literature review, and population diversity adaptation techniques. Section 3 provides details of the proposed algorithm. Experimental results are reported and analyzed in Sect. 4. Finally, conclusions and some future directions are presented in Sect. 5.

2 Background

In this section, we introduce SMA algorithm's mathematical models followed by the short literature review of SMA, and finally brief discussion of population diversity-based adaptation techniques [56] that motivated this work.

2.1 SMA introduction

In [27], a new stochastic optimizer called slime mould algorithm (SMA) was proposed. The algorithm models the morphological changes of slime mould, namely Physarum polycephalum, when foraging. Slime mould is eukaryote where its organic matter utilizes a process called Plasmodium as its main process to seek food. Once a food source is identified, the slime mould surrounds it and secrete enzymes to digest it. The foraging process of the Slime mould is divided into three phases, where the first process is finding food source, followed by wrapping, and then food grabble. The mathematical model for the various processes of the slime mould is described as follows [27]:

$$\vec{X}(t+1) = \begin{cases} rand \cdot (UB - LB) + LB & rand < z \\ \overrightarrow{X_b}(t) + \vec{vb} \cdot \left(\vec{W} \cdot \overrightarrow{X_A}(t) - \overrightarrow{X_B}(t) \right) & r < p \\ \vec{vc} \cdot \vec{X}(t) & r \geq p \end{cases} \quad (1)$$

with *rand* and *r* are random values $\in$ [0,1], *UB*/*LB* representing the upper/lower bound of the search space, and value z is used to balance exploration and exploitation characteristics. As for $\vec{X}$ and $\overrightarrow{X_b}$, they represent the locations of current and best fitness individuals at iteration t, respectively, where best fitness here represents the individual with the highest odor concentration. $\overrightarrow{X_A}$ and $\overrightarrow{X_B}$ are

two randomly selected individuals from the slime mould population. Parameter $\overrightarrow{vc}$ is a linearity decreasing value from 1 to 0 whereas $\overrightarrow{vb}$ is a variable $\in$ [a,-a] where a is calculated using [27]:

$$a = arctanh(-(\frac{t}{T}) + 1) \tag{2}$$

where T represents maximum number of iterations. Parameter $\overrightarrow{W}$ is a vector representing the slime mould weight and is described mathematically as [27]:

$$\overrightarrow{W}(SmellIndex(i)) \begin{cases} 1 + r \cdot \log\left(\frac{F_b - S(i)}{F_b - F_w} + 1\right), condition \\ 1 - r \cdot \log\left(\frac{F_b - S(i)}{F_b - F_w} + 1\right), otherwise \end{cases} \tag{3}$$

$$SmellIndex = sort(S) \tag{4}$$

where F_b/F_w represents best/worst fitness value at the current iteration and r being a random number $\in$ [0,1]. Moreover, $S(i)$ represents the individual's fitness, *condition* indicates the rank of $S(i)$ in the first half of the population (i.e., the positive group). In Eq. (4), the term *SmellIndex* denotes the result of sorting S in an ascending order. Parameter p in Eq. (1) is calculated using [27]:

$$p = tanh(|S(i) - DF)|) \tag{5}$$

where DF is the overall global best fitness and $S(i)$ represents the individual's fitness.

A flowchart for SMA algorithm is depicted in Fig. 1 where it starts with initializing parameters D, P, T, LB, UB, z, and a random population $\overrightarrow{X_i}(t = 0)$. In each iteration, SMA will calculate individuals' fitness and find the best one in the current iteration. The next population is then updated according to Eq. (1) and conditional weighting parameter W. This iterative process is repeated until maximum number of iteration is reached where the best fitness and the solution are stored as the final result.

In Eq. (1), the exploration capability is guaranteed with a probability of at least z while exploitation is performed with a probability of at least $1 - p$. When *rand* is less than z, SMA will take a random walk within the boundaries defined by LB and UB. However, when another random number r is larger than p, SMA will perform exploitation and search in the neighbourhood of the current location. When r is less than p, SMA will wrap around the current best position, $\overrightarrow{X_b}(t)$, with wrapping direction and radius depending on the current position's fitness. Such behaviors can be much more evident when plugging the definition of W in Eq. (3) back to Eq. (1) when $r < p$ resulting in [27]:

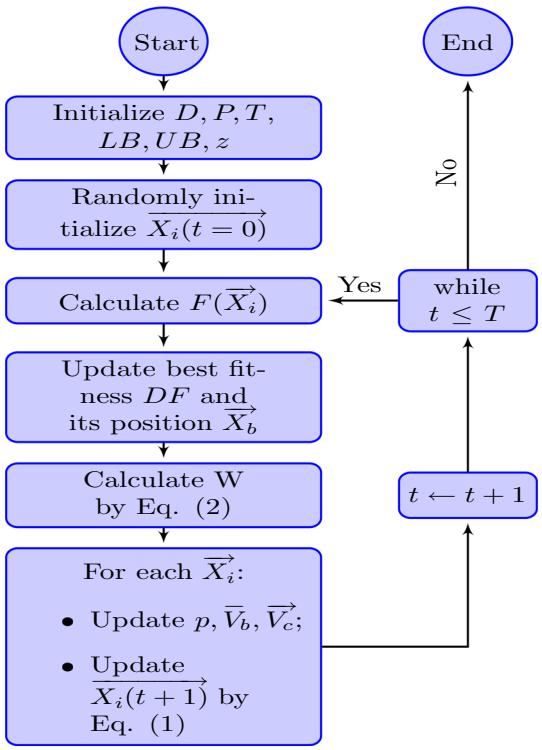


Fig. 1 SMA flowchart

$$\overrightarrow{X}(t+1) = \overrightarrow{X_b}(t) + \overrightarrow{vb} \cdot \left(\overrightarrow{X_A}(t) - \overrightarrow{X_B}(t) \pm r \cdot \log\left(\frac{F_b - S(i)}{F_b - F_w} + 1\right)\right) \tag{6}$$

The SMA algorithm will wrap the food in two directions depending on the fitness of the current position's with the radius depending on the amplitude of $\overrightarrow{vb}$ and population variance. In Eq. (1), $\overrightarrow{vb}$ and $\overrightarrow{vc}$ are two tuning parameters oscillating towards 0 with iterations imitating food-grabbing behaviour and hence exploitation around the best food source.

2.2 Literature review

SMA and its variants were successfully applied to many problems such as image segmentation [34, 61], breast cancer detection [29, 36], COVID-19 early detection [4, 46], parameters estimation of photovoltaic cells [14, 31, 33], medical data classification [54], feature selection [23], proportional-integral-derivative (PID) motor speed controllers [11, 43], power systems [38, 45], bearings defect identification [51], travelling salesman problem [30], and machine learning models parameter tuning for support vector machine [8] and artificial neural network [63] to name a few.

However, SMA may suffer from some shortcomings such as slow convergence rate because of being trapped in local minima and having an unbalanced exploitation and exploration phases. Therefore, to further improve SMA performance, researchers have reported a variety of specific stochastic operators such as Levy distribution [4, 10], cosine function for controlling parameters [16, 18], quantum rotation gate [59], opposition-based learning [35, 45, 54], and chaotic operator [31]. In addition, SMA has been hybridized with other metaheuristics such as Harris hawk optimization [60], whale optimization [1], particle swarm [15], differential evolution [20, 29], and arithmetic optimization algorithm [62] to efficiently solve specific optimization problems. Furthermore, variants of SMA to solve discrete binary [2, 26] and multi-objective optimization problems [21, 24, 44] have been proposed. However, none of these works have considered population size adaptation to enhance the performance of SMA.

2.3 Population adaptation

Population size adaption has become prevalent in many population-based metaheuristic algorithms (MAs). The choice of a proper population size can substantially enhance the efficiency of many meta-heuristic algorithms including SMA. In a typical linear population size reduction algorithm, there is a large number of individuals in the population initially to enhance exploration capability. During population evolution, population size is decreased linearly until reaching smallest population size at the end of algorithm execution in order to allow for proper exploitation. However, for more complex objective functions, it is also possible to increase population size towards the end of the search process to avoid premature convergence or stagnation. A common criteria to control population size direction is to use population diversity as a metric (i.e., population distribution). For example, in [6, 39, 42, 56, 57], the authors proposed to use population diversity to start and stop population adaption process in differential evolution. The following is a review of population diversity adaptation technique based on the work presented in [56]. Parameters and variables associated with this technique are listed in Table 1.<Table ID="

Population diversity is measured by mean of the Euclidean distances in each feature described as follows:

$$DI_t = \sqrt{\frac{1}{P_t}\sum_{i=1}^{P_t}\sum_{j=1}^{D}(x_{i,j} - \bar{x_j})^2} \tag{7}$$

where value $\bar{x_j}$ is calculated as:

Table 1 Population diversity parameters and variables [56]

$x_{i,j}$	Value of j-th feature in i-th individual at t-th iteration
$\bar{x_j}$	Center of j-th feature at t-th iteration
P_t	Population size at t-th iteration
DI_t	Population diversity at t-th iteration
RD_t	Relative population diversity at t-th iteration
$R_{FES,t}$	Relative ratio of the depleted function evaluations
γ	Scaling factor

$$\bar{x_j} = \frac{1}{P_t}\sum_{i=1}^{P_t} x_{i,j} \tag{8}$$

During the evolution process, a relative measure of population diversity is calculated using:

$$RD_t = \frac{DI_t}{DI_{t=0}} \tag{9}$$

where the lower and upper bound for $RD_t \in [0.9 \times \gamma R_{FES,t}, 1.1 \times \gamma R_{FES,t}]$ where γ is a scaling factor and $R_{FES,t}$ is the relative number of depleted function evaluations given by:

$$R_{FES,t} = \frac{\text{current number of function evaluations}}{\text{number of function evaluation allowed}} \tag{10}$$

When RD_t drops below the lower bound (i.e., $0.9 \times \gamma R_{FES,t}$), P_t will increase by 1 whereas when it exceeds the upper bound (i.e., $1.1 \times \gamma R_{FES,t}$) it will decreased by 1. Such a technique results in a linearly fluctuant population that utilizes population diversity based on Euclidean distance.

In [48], the authors proposed using a pseudo randomization technique to change population size where

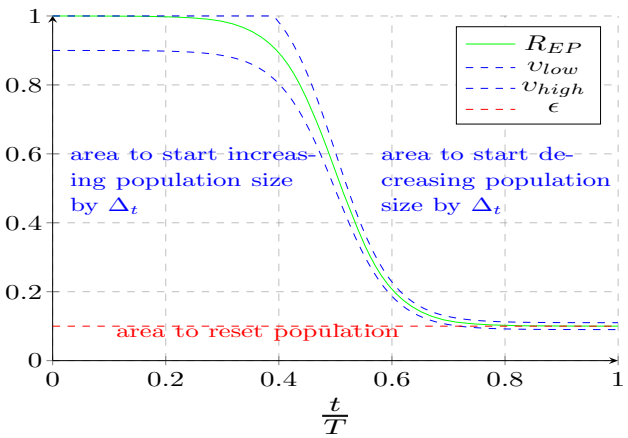


Fig. 2 $R_{EP}, \upsilon_{low}, \upsilon_{high}$ and ϵ against $\frac{t}{T}$

population size P_t in the t-th function evaluation is calculated using:

$$P_t = P_{min} + \left\lfloor \frac{M_t \times D - P_{min}}{2} \times \left(\cos\left(\frac{t \times \pi}{M_t \times D^2} \right) + 1 \right) \right\rfloor \tag{11}$$

where "$\lfloor\rfloor$" is a rounding operator, P_{min} is minimum population size, D is the dimension of the function to be optimized, and M_t is a linear reduction parameter defined as:

$$M_t = \frac{(M-1) \times (T-t)}{T} + 1 \tag{12}$$

with T being maximum function evaluations and t being the index of current function evaluation. Parameter M is a function of initial population size and problem dimension which is calculated using:

$$M = \frac{P_{init}}{D} \tag{13}$$

In this paper, we propose to use population diversity to start and stop population adaptation similar to Poláková et al. [39] but cluster the population by applying K-mean algorithm. As pointed out in [39], in the SMA process when best food sources are gradually grabbed, it is expected that populations are gradually contracted around food sources with the best fitness, and hence DI_t will gradually decreases toward 0. By tracking the value of DI_t, SMA convergence rate can be envisioned and population size can be adapted accordingly. If DI_t is high, then the population is too diverse and population size can be reduced. If DI_t is low, the population is contracted and if that happens during the initial phase of SMA, more population should be added to enhance exploration capability. However, if SMA is approaching the last few iterations, population size should be reduced to save computation time. If DI_t is decreasing dramatically to a small value during the evolution process, resetting the population can also be considered to avoid being stuck at local optimal. The proposed algorithm herein is based on this concept and its details are described in the next section.

3 FP-SMA and analysis

In this section, the proposed algorithm, called FP-SMA, is described in details. Suppose at the t-th epoch, there are $x_i, i = 1, ..., P_t$ individuals in the population. By applying the K-mean algorithm, each individual x_i is associated to a group center x_{c_i} resulting in population diversity at iteration t-th as:

$$DI_t = \frac{1}{P_t} \sum_{i=1}^{P_t} (x_i - x_{c_i})^2 \tag{14}$$

The initial population diversity, $DI_{init} = DI_{t=0}$, is stored as a reference to define relative diversity (RD_t) during the evolution process as defined in Eq. (9) with relative expected population diversity R_{EP} calculated as:

$$R_{EP} = 1 - \gamma \frac{1 + arctanh(\beta(x - 0.5))}{2} \text{ where } x = \frac{t}{T} \in [0, 1] \tag{15}$$

where t is current iteration index, T is the total number of iterations, and β is a scaling factor with a default value of 10. $\gamma \in [0, 1]$ is the fraction of relative population diversity expected to be consumed during the SMA process and hence R_{EP} can be treated as the expected relative population diversity at the current iteration. Initially, it is expected that $R_{EP} \approx 1$ but then towards the end of the evolution process becomes $R_{EP} \approx 1 - \gamma$. It is also expected that R_{EP} decreases smoothly when $R_{EP} \approx 1$, then abruptly halfway through the process, and then smoothly again until $R_{EP} \approx 1 - \gamma$.

Value R_{EP} is used as a reference to trigger population adaptation. During the optimization, if current relative population diversity RD_t is less than $\upsilon_{low} = 0.9 \times R_{EP}$ or larger than $\upsilon_{high} = 1.1 \times R_{EP}$, then population adaptation will start. Optionally, if $R_{EP} < \epsilon$ and $P_t < P_{min}$ then the population is reset to its initial size. The term P_{min} is the required minimum population size to guarantee minimum amount of exploration. Typically values for $P_{min} = \frac{P_{init}}{2}$ and $\epsilon = 0.1$.

Once population adaptation is started, Δ_t points are added/removed to/from the current population with Δ_t defined as

$$\Delta_t = \max\left(\left\lfloor \frac{M_t \times D}{2} \times \left(\sin^2\left(\frac{t \times \pi}{\Lambda} \right) + 1 \right) \right\rfloor, 1 \right) \tag{16}$$

The definition of Δ_t is similar to that of P_t in Eq. (11) except for parameter Λ, which is a problem-specific parameter to control fluctuation period. In the original definition of P_t, the period of fluctuation is $2 \times M_t \times D^2$ which may fluctuate too slowly for practical engineering design problems. Note that if $\Lambda = \infty$, then Δ_t is fixed to be one and thus is rolling back to the approach proposed in [48]. Therefore, population size change can be summarized as:

$$P_{t+1} = \begin{cases} P_{init} & if \quad RD_t < \epsilon \text{ and } P_t < P_{min} \\ P_t - \Delta_t & if \quad RD_t < \epsilon \text{ and } P_t > P_{min} \\ P_t + \Delta_t & if \quad \epsilon < RD_t < \upsilon_{low} \text{ and } P_t < P_{init} \\ P_t - \Delta_t & if \quad RD_t > \upsilon_{high} \text{ and } P_t > P_{min} \\ P_t & otherwise \end{cases} \tag{17}$$

As depicted in Fig. 2, the solid green line shows the expected R_{EP} as a function of t. When the actual RD_t is moving outside the region covered by dashed blue lines, population adaptation will change by Δ_t. Furthermore, if RD_t is always dropping below ϵ when population size is already at its minimum level (i.e., $P_t = P_{min}$), population size can be reset.

The FP-SMA algorithm is illustrated in Algorithm 1. The input to the algorithm is the fitness function to evaluate $\mathcal{F}$. In lines 2–6, parameters and the population are initialized and population diversity are calculated using Eq. (14) before starting the iterations. In lines 8–13, the fitness of each individual in the population will be evaluated and then sorted accordingly before being divided into two groups, positive and negative group. Each individual then will have its position updated according to Eq. (1). Line 14 through 15 are the newly proposed steps where DI_t, RD_t and R_{EP} are updated accordingly. Finally in line 16, population size for next epoch is updated as defined by the fluctuation strategy described in Eq. (17).

Algorithm 1 Pseudo code FP-SMA algorithm

1: **procedure** FPSMA($\mathcal{F}$) ▷ $\mathcal{F}$: fitness function
2: Initialize parameters D, P_{init}, T, z, Λ, ϵ, γ,
3: Initialize population, $\{X_i : i = 0, ..., P\}$
4: Calculate $DI_{init} = DI_{t=0}$ using Eq. (14),
5: Calculate multiplier, $M = \frac{P_{init}}{D}$
6: $t \leftarrow 0$, $P_t \leftarrow P$
7: **while** $t < T$ **do**
8: Sort population $\{X_i\}$ by fitness $\mathcal{F}(X_i)$
9: Calculate $\{W_i\}$ using Eq. (3)
10: Save best fitness, $(X_b, \mathcal{F}(X_b))$
11: **for** each slime mould at X_i **do**
12: Update position using Eq. (1)
13: **end for**
14: Calculate DI_t using Eq. (14)
15: Calculate R_{EP}, RD_t using Eq. (15, **??**)
16: Update P_{t+1} using Eq. (17)
17: **end while**
18: **end procedure**

The time complexity of FP-SMA depends on number of iterations (T), population size (P), function dimension (D) and is bounded by the computation performed within the while loop (lines 7–17). Therefore, based on simple analysis of the main compute intensive processes executed during the while loop, one can compute FP-SMA's time complexity. For each iteration, computational complexity depends on fitness evaluation and sorting (line-8) which can be performed in $\mathcal{O}(P \log P)$, weight update (line-9), and position update (lines 11-13) where both can be performed in $\mathcal{O}(P * D)$. K-means clustering (lines 14–16) for population size adaptation can be performed in $\mathcal{O}(P * K)$ for a fixed number of iterations and attributes, where K is the number of clusters. Therefore, the final time complexity of FP-SMA is: $\mathcal{O}(T * ((P \log P) + (P * D) + (P * K))$ which is comparable with the original SMA. However, in our case, the average value for P is less due to the adaptive nature at each epoch as compared to SMA which has fixed value for all iterations.

4 Experiment results

In this section, we apply FP-SMA on *Ackley* benchmark function to validate the relationship between relative population diversity and population size in addition to convergence characteristics. Moreover, the performance of FP-SMA as compared to original SMA using a set of 13 benchmarks and CEC2014 functions [27, 28, 37] will be discussed. The performance metrics used to compare solution quality will be fitness values, whereas for run time, the number of functional evaluations will be used. All results have been obtained using Python 3.7 running on Intel® Core™2 Quad CPU Q8400 @ 2.66 GHz with a 8 GB RAM and a 64-bit OS.

4.1 Convergence analysis

In convergence analysis experiment, FP-SMA was applied to *Ackley* benchmark function which is widely used as a multivariate test function for optimization problems [49]. It is described mathematically by Eq. (18) and plotted in Fig. 3.

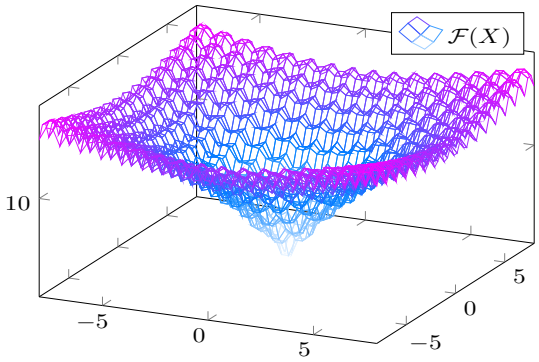


Fig. 3 *Ackley* function with $a = 20, b = 0.2, c = 2\pi, D = 2$

$$\mathcal{F}(X) = -a\exp\left(-b\sqrt{\frac{1}{D}\sum_{i=1}^{D}X_i^2}\right) - \exp\left(\frac{1}{D}\sum_{i=1}^{D}cos(X_i)\right) + a + \exp(1) \tag{18}$$

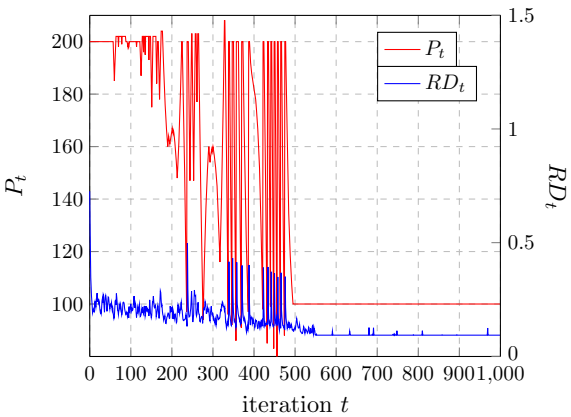
The *Ackley* function is characterized by its nearly flat outer region and a global optimal at the center ($X^* = 0$) with many local optimal close by. Recommended variable values for *Ackley* benchmark are $a = 20, b = 0.2, c = 2\pi$. As for the parameters used in FP-SMA implementation they are: D=100, P_{init}=200, T=1000, z=0.03, Λ=100, ϵ=0.1, γ=1. These parameters are used throughout the remainder of this discussion.

Figure 4a shows population size P_t and relative population diversity RD_t during FP-SMA execution whereas Fig. 4b shows best fitness evolution. In the first tens of epochs, RD_t is decreasing dramatically from 1 to below 0.5 with best fitness improved to around 10^0 indicating convergence toward a local optimal. However, to continue the exploration process, population size is still fixed at its initial value of $P_{init} = 200$. During the first 500 epochs, population size is slightly changing due to RD_t fluctuation to maintain the balance between exploration and exploitation. After 500 iterations, R_{EP} drops considerably as are seen in Fig. 4a leading to a sharp decrease in population size down to $P_{min} = 100$. Keeping this minimum population size is sufficient to reach the global minimum at around the 540th iteration.

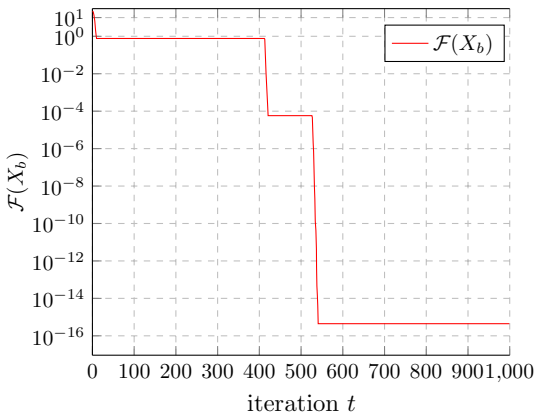
Figure 5 shows population distribution only in the first two dimensions during the optimization process. At the beginning when $t = 0$, the population is randomly distributed between the lower and upper bound; however, as execution continues, the population is gradually concentrated around multiple centers. As a result, population size can be decreased gradually to save computation without effecting exploitation characteristics. At $t = 750$, the algorithm converges toward two centers indicating that minimum population size is sufficient for exploitation.

4.2 Benchmarks comparisons

FP-SMA was also evaluated using 13 standard benchmark function (see Table 5 in appendix) commonly used to evaluate optimization algorithms and additional 30 benchmarks from CEC2014 [28]. The results are the average values for 30 independent runs of the algorithms on each benchmark. Tables 2 and 3 show the best fitness on both standard and CEC2014 benchmarks, respectively. In the tables, column f_{min_s} specifies the fitness value followed by standard deviation (σ). Column Υ indicates the fitness of SMA whether it is better, equal or worst than FP-SMA using symbols "−", "=" or "+," respectively. Moreover, a comparison between SMA and FP-SMA performance in terms of function evaluations are represented in column $\delta(\%)$ which represents the percentile decrease in the number of function evaluations calculated using:



(a) Population size vs. diversity



(b) Best fitness evolution

Fig. 4 FP-SMA Performance on *Ackley* benchmark

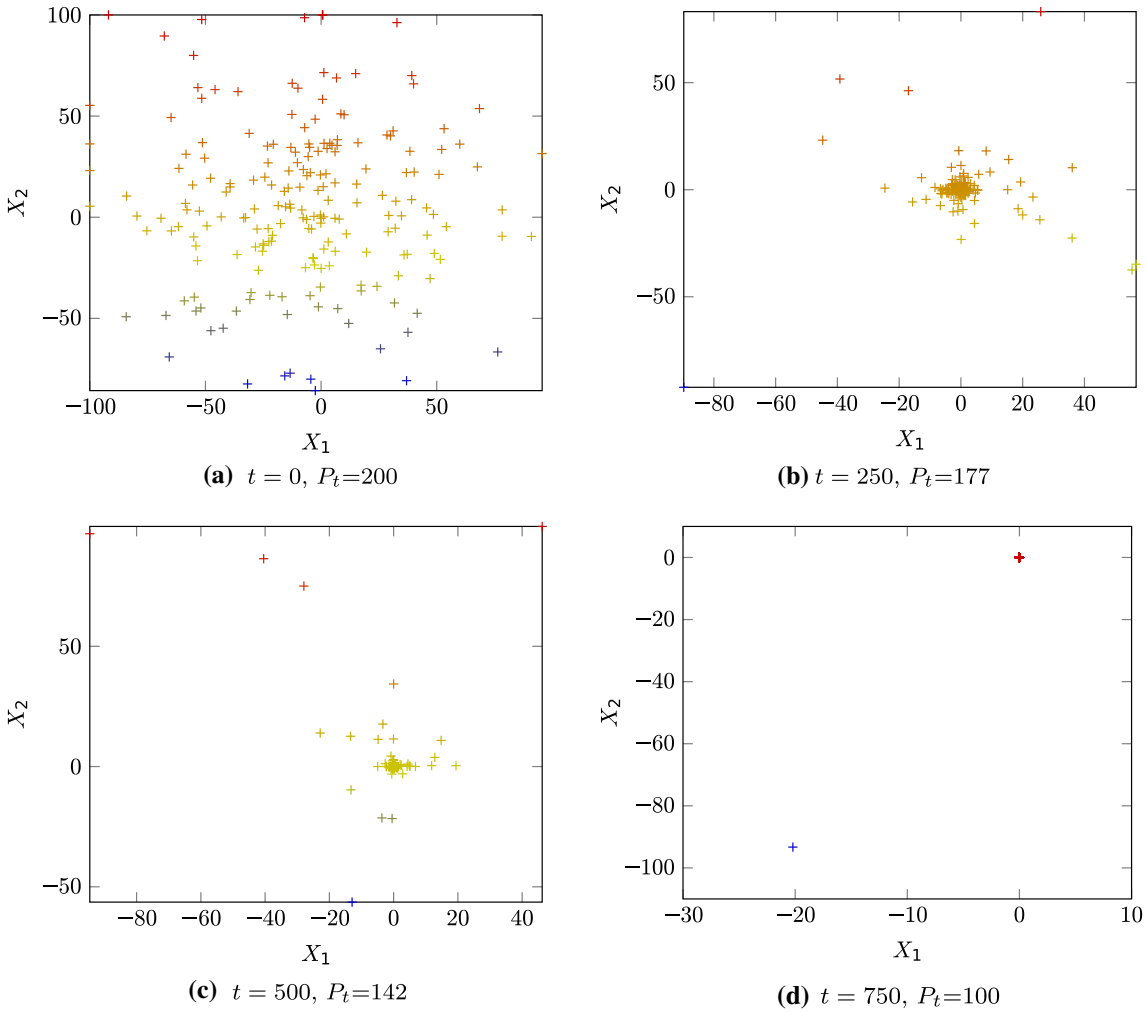


Fig. 5 Population distribution

Table 2 Results for standard benchmark

Fn.	f_{min}	SMA		FP-SMA		Υ	δ (%)
		f_{min}	σ	f_{min}	σ		
f_0	0.0	0.000000	(0.000000)	0.000000	(0.000000)	=	↓28.09
f_1	0.0	0.000000	(0.000000)	0.000000	(0.000000)	=	↓28.96
f_2	0.0	0.000000	(0.000000)	0.000000	(0.000000)	=	↓25.99
f_3	0.0	0.000000	(0.000000)	0.000000	(0.000000)	=	↓29.90
f_4	0.0	0.000000	(0.000000)	0.000000	(0.000000)	=	↓30.77
f_5	0.0	0.001636	(0.000320)	0.009851	(0.006180)	−	↓29.05
f_6	0.0	0.000071	(0.000054)	0.000123	(0.000091)	−	↓28.67
f_7	$-419 * n$	−26553.12	(1029.06)	−23658.74	(1394.06)	−	↓22.35
f_8	0.0	0.000000	(0.000000)	0.000000	(0.000000)	=	↓29.19
f_9	0.0	0.000000	(0.000000)	0.000000	(0.000000)	=	↓28.54
f_{10}	0.0	0.000000	(0.000000)	0.000000	(0.000000)	=	↓30.29
f_{11}	0.0	−0.270612	(0.0081)	−0.261275	(0.0082)	−	↓33.33
f_{12}	0.0	0.365256	(0.346215)	0.727546	(0.361334)	−	↓26.21
Average (δ)							28.56%

$$\delta = \frac{\#\text{function evaluations(SMA)} - \#\text{function evaluations(FP - SMA)}}{\#\text{function evulations (SMA)}}$$

Tables 2 and 3 compares that attained best fitness for SMA and FP-SMA algorithms on standard and CEC2014 benchmarks, respectively. The tables show that FP-SMA was able to reduce the number of function evaluation for standard benchmark on average by approximately 28% and CEC2014 ones by 24%. In 8 out of 13 standard benchmarks and more than half of CEC2014 benchmarks, FP-SMA was able to achieve equivalent or better fitness than the original SMA. FP-SMA was able to achieve the same fitness in 5 CEC2014 benchmarks while having worst fitness in 10 benchmarks with an average fitness loss of about 9% which is much less than the 26% fitness improvement found in the other benchmarks. As a matter of fact, if f_8 and f_{19} are excluded from fitness results, then overall loss in

Table 3 Results for CEC2014 benchmarks

Fn.	SMA		FP-SMA		ϒ	δ (%)
	f_{min}	σ	f_{min}	σ		
1	101779108.2	(23005654.76)	72392092.94	(15456620.21)	+	↓15.02
2	54068.22766	(32827.35102)	33888.22483	(10398.97004)	+	↓23.73
3	15771.81289	(1937.027374)	12969.05761	(3055.875967)	+	↓26.22
4	765.8050354	(38.33549)	743.462391	(28.69199231)	+	↓24.93
5	521.3523084	(0.025410748)	521.3536462	(0.02636567)	−	↓28.74
6	689.7700054	(4.495182553)	702.4456421	(8.778099803)	−	↓11.34
7	701.2252342	(0.019830644)	701.3457094	(0.085363385)	−	↓6.02
8	1591.485312	(81.15187989)	1850.510688	(119.2295506)	−	↓23.46
9	900.1435128	(0.042795493)	900.1265529	(0.038929931)	+	↓20.75
10	15683.92405	(1005.930421)	16346.75008	(943.8132163)	−	↓26.48
11	17194.68012	(1069.398267)	16451.43246	(840.0234629)	+	↓24.11
12	1201.439864	(0.198065608)	1201.509781	(0.318894703)	−	↓14.43
13	1300.862725	(0.040562908)	1300.87667	(0.06223831)	−	↓21.81
14	1400.783038	(0.284020258)	1400.761513	(0.272489399)	+	↓25
15	1577.920191	(10.31230952)	1580.754892	(10.6801292)	−	↓17.37
16	1644.25172	(0.278229155)	1643.969788	(0.532866376)	+	↓28.38
17	9512024.724	(2280058.593)	7026197.112	(2059096.534)	+	↓25.34
18	430718.1946	(91400.58899)	304522.4289	(40505.38438)	+	↓22.8
19	2188.738236	(258.0159205)	3676.819488	(2689.284593)	−	↓25.58
20	1.89872E+14	(1.9095E+11)	1.9012E+14	(5.48002E+11)	−	↓32.78
21	7740706.982	(1910878.222)	4949377.984	(1919460.462)	+	↓23.36
22	9340.247418	(1287.694449)	7587.324668	(691.5903953)	+	↓26.18
23	2567.439243	(3.558234589)	2563.896174	(4.562597725)	+	↓12.27
24	2600.5	(0)	2600.5	(0)	=	↓31.33
25	2700	(0)	2700	(0)	=	↓12.89
26	2800	(0)	2800	(0)	=	↓64.3
27	2900.003182	(3.45898E−12)	2900.003182	(3.45898E−12)	=	↓29.81
28	3000.003182	(3.45898E−12)	3000.003182	(3.45898E−12)	=	↓27.49
29	109322190.1	(89077824.93)	1632038.666	(1338227.485)	+	↓23.93
30	38576257.21	(22524047.63)	4019890.212	(1611603.934)	+	↓25.52
Average (δ)						24.05%

Table 4 Standard benchmarks run-time comparison

Benchmark	T_{SMA}(s)	T_{FP-SMA}(s)	$\delta(\%)$
0	1.61	1.16	27.9
1	2.54	2.09	17.7
2	108.73	74.69	31.3
3	2.65	2.17	18.1
4	49.83	36.68	26.4
5	1.83	1.3	28.9
6	21.05	14.46	31.3
7	73.52	57.92	21.2
8	60.19	48.73	19.0
9	5.73	3.83	33.1
10	59.86	45.08	24.7
11	122.84	100.51	18.2
12	108.49	80.69	25.6
Average			↓24.9

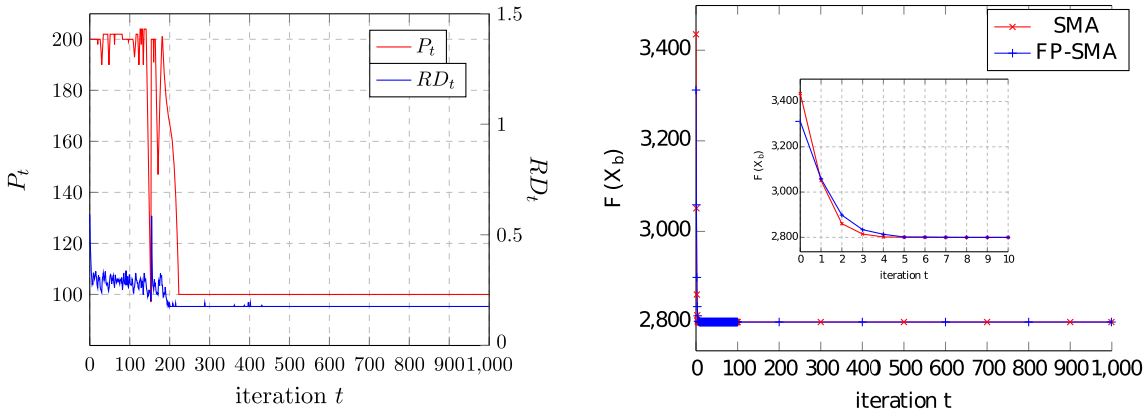
fitness will become negligible (i.e., 0.8%) for the benchmarks with worst fitness.

To get a sense of run time enhancement as compared to simply the reduction in number of function evaluations, Table 4 shows run-time characteristics for both SMA and FP-SMA algorithms on standard benchmarks where the values are given in seconds. Column δ gives percentile reduction in run-time when using FP-SMA as compared to SMA. It is apparent that on average, FP-SMA provide a reduction of 25% in run-time. The same characteristics are observed for CEC2014 benchmarks (i.e., 22% reduction in run-time) but not shown to keep the discussion concise.

Surprisingly, FP-SMA was able to save 64.3% of computational cost associated with f_{26} as compared to original SMA. By plotting population size (P_t) and relative diversity (RD_t) in Fig. 6a and best fitness in Fig. 6b, it can be observed that the relative diversity has dramatically decreased after about 100 epochs, indicating algorithm stagnation. Population size fluctuation can no longer help the algorithm escape its stagnation and after 250 epochs, the relative diversity RD_t drops to zero and population size is set to its minimum value. This results in a considerable reduction in the number of function evaluations as depicted in Table 3. Since the algorithm was able to identify this condition, it is possible to utilize such an approach to trigger early algorithm termination resulting in further reduction in function evaluation. Note that in these tables, we only presented the saving in the number of function evaluation without changing the terminating condition (i.e., maximum number of iteration). A possible further enhancement to the proposed algorithm is to consider this case and terminate the algorithm to boost reduction in overall execution time.

In summary, the following observations can be made from the experimental results:

- Population size adaptation based on population diversity played an important role in both run-time efficiency and optimization effectiveness of FP-SMA as compared with SMA.



(a) Population size vs. diversity

(b) Best fitness evolution

Fig. 6 FP-SMA Performance on f_{26} benchmark

- Experimental results on 13 standard and 30 CEC2014 benchmark functions in Tables 2 and 3 have revealed that FP-SMA can achieve 20–30% savings in function evaluations on average while maintaining good solution quality when compared to SMA.
- As depicted in Table 4, the FP-SMA showed a 25% reduction in run-time on standard benchmarks on average when compared to SMA and thus demonstrating its balanced exploration and exploitation capabilities.

5 Conclusion

This paper proposed an acceleration strategy for SMA algorithm named FP-SMA that adaptively change population size during algorithm execution. A cluster-based population diversity from K-mean is used as an indicator to change population size to balance exploration and exploitation phases of the algorithm. Thresholds for population diversity that trigger population size change are dynamically fine-tuned to appropriate levels during different stages of the algorithm. Once population diversity exceeds the threshold, population size is modified using sine wave function pattern. Simulation results on 13 standard and 30 IEEE CEC2014 benchmark functions have revealed that FP-SMA algorithm can achieve approximately 20% reduction in computation cost while maintaining good solution quality. The performance gain can be attributed to the flexibility offered by FP-SMA to switch between exploration and exploitation phases and the adaptable population size. The proposed algorithm can be found on Github using the link https://github.com/e6la3banoh/FP-SMA. As future work, we would like to study the parallelization of FP-SMA and its extension for multiple-objective SMA [21, 24, 44].

Appendix A: Standard benchmarks

See Appendix Table 5.

Table 5 Definitions of standard benchmarks

Function	Range
$f_1(x) = \sum_{i=1}^{n} x_i^2$	$[-100, 100]$
$f_2(x) = \sum_{i=1}^{n} \lvert x_i \rvert + \prod_{i=1}^{n} \lvert x_i \rvert$	$[-10, 10]$
$f_3(x) = \sum_{i=1}^{n} (\sum_{j=1}^{i} x_j)^2$	$[-100, 100]$
$f_4(x) = \max_{i:1 \le i \le n} \lvert x_i \rvert$	$[-100, 100]$
$f_5(x) = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$	$[-30, 30]$
$f_6(x) = \sum_{i=1}^{n} [x_i + 0.5]^2$	$[-100, 100]$
$f_7(x) = \sum_{i=1}^{n} i x_i^4 + \mathbf{U}_{random}[0, 1]$	$[-128, 128]$
$f_8(x) = \sum_{i=1}^{n} -x_i \sin \sqrt{\lvert x_i \rvert}$	$[-500, 500]$
$f_9(x) = \sum_{i=1}^{n} [x_i^2 - 10\cos(2\pi x_i) + 10]$	$[-5.12, 5.12]$
$f_{10}(x) = -20exp(-0.2(\frac{1}{n}\sum_{i=1}^{n} x_i^2)^{0.5}) - exp(\frac{1}{n}\sum_{i=1}^{n} cos(2\pi x_i)) + 20 + e$	$[-32, 32]$
$f_{11}(x) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos(\frac{x_i}{\sqrt{i}}) + 1$	$[-600, 600]$
$f_{12}(x) = \frac{\pi}{n}[10\sin(\pi y_1) + \sum_{i=1}^{n-1}(y_i - 1)^2[1 + 10\sin^2(\pi y_{i+1})] + (y_n - 1)^2] + \sum_{i=1}^{n} u(x_i, 10, 100, 4)$ where $y_i = 1 + \frac{x_i + 1}{4}$ and $u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m & x_i > a \\ 0 & -a < x_i < a \\ k(-x_i - a)^m & x_i < a \end{cases}$	$[-50, 50]$
$f_{13}(x) = 0.1[\sin^2(3\pi x_i) + \sum_{i=1}^{n-1}(x_i - 1)^2[1 + \sin^2(3\pi x_i + 1)] + (x_n - 1)^2[1 + \sin^2(2\pi x_n)]] \sum_{i=1}^{n} u(x_i, 5, 100, 4)$	$[-50, 50]$

Declarations

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

References

1. Abdel-Basset M, Chang V, Mohamed R (2020) Hsma_woa: a hybrid novel slime mould algorithm with whale optimization algorithm for tackling the image segmentation problem of chest x-ray images. Appl Soft Comput 95:106642. https://doi.org/10.1016/j.asoc.2020.106642
2. Abdel-Basset M, Mohamed R, Chakrabortty RK, Ryan MJ, Mirjalili S (2021) An efficient binary slime mould algorithm integrated with a novel attacking-feeding strategy for feature selection. Comput Ind Eng 153:107078
3. Al-Faisal HR, Ahmad I, Salman AA, Alfailakawi MG (2021) Adaptation of population size in sine cosine algorithm. IEEE Access 9:25258–25277. https://doi.org/10.1109/ACCESS.2021.3056520
4. Anter AM, Oliva D, Thakare A, Zhang Z (2021) Afcm-lsma: new intelligent model based on lévy slime mould algorithm and adaptive fuzzy c-means for identification of covid-19 infection from chest x-ray images. Adv Eng Inform 49:101317
5. Arabas J, Michalewicz Z, Mulawka J (1994) Gavaps-a genetic algorithm with varying population size. In: Proceedings of the 1st IEEE conference on evolutionary computation. IEEE world congress on computational intelligence. IEEE, pp 73–78
6. Brest J, Greiner S, Bošković B, Mernik M, Žumer V (2006) Self-adapting control parameters in differential evolution: a comparative study on numerical benchmark problems. IEEE Trans Evol Comput 10:646–657
7. Chen D, Zhao C (2009) Particle swarm optimization with adaptive population size and its application. Appl Soft Comput 9(1):39–48
8. Chen Z, Liu W (2020) An efficient parameter adaptive support vector regression using k-means clustering and chaotic slime mould algorithm. IEEE Access 8:156851–156862
9. Cui L, Li G, Zhu Z, Lin Q, Wen Z, Lu N, Wong KC, Chen J (2017) A novel artificial bee colony algorithm with an adaptive population size for numerical function optimization. Inf Sci 414:53–67
10. Cui Z, Hou X, Zhou H, Lian W, Wu J (2020) Modified slime mould algorithm via levy flight. In: 2020 13th international congress on image and signal processing, biomedical engineering and informatics (CISP-BMEI). IEEE, pp 1109–1113 (2020)
11. İzci D ES (2021) Comparative performance analysis of slime mould algorithm for efficient design of proportional–integral–derivative controller. Electrica 21:151–159
12. Dhal KG, Das A, Sahoo S, Das R, Das S (2019) Measuring the curse of population size over swarm intelligence based algorithms. Evol Syst 12:1–48
13. Dokeroglu T, Sevinc E, Kucukyilmaz T, Cosar A (2019) A survey on new generation metaheuristic algorithms. Comput Ind Eng 137:106040
14. El-Fergany AA (2021) Parameters identification of pv model using improved slime mould optimizer and lambert w-function. Energy Rep 7:875–887
15. Gao Z, Zhao J, Li S (2020) The hybridized slime mould and particle swarm optimization algorithms. In: 2020 IEEE 3rd international conference on automation, electronics and electrical engineering (AUTEEE). IEEE, pp 304–308 (2020)
16. Gao ZM, Zhao J, Li SR (2020) The improved slime mould algorithm with cosine controlling parameters. J Phys: Confer Ser 1631:012083. https://doi.org/10.1088/1742-6596/1631/1/012083
17. Goldberg DE, Holland JH (1988) Genetic algorithms and machine learning. Mach Learn 3:95–99
18. Hassan MH, Kamel S, Abualigah L, Eid A (2021) Development and application of slime mould algorithm for optimal economic emission dispatch. Expert Syst Appl 182:115205
19. Heidari AA, Mirjalili S, Faris H, Aljarah I, Mafarja M, Chen H (2019) Harris hawks optimization: algorithm and applications. Fut Gener Comput Syst 97:849–872. https://doi.org/10.1016/j.future.2019.02.028
20. Houssein EH, Mahdy MA, Blondin MJ, Shebl D, Mohamed WM (2021) Hybrid slime mould algorithm with adaptive guided differential evolution algorithm for combinatorial and global optimization problems. Expert Syst Appl 174:114689
21. Houssein EH, Mahdy MA, Shebl D, Manzoor A, Sarkar R, Mohamed WM (2022) An efficient slime mould algorithm for solving multi-objective optimization problems. Expert Syst Appl 187:115870
22. Hussain K, Salleh MNM, Cheng S, Shi Y (2019) Metaheuristic research: a comprehensive survey. Artif Intell Rev 52(4):2191–2233
23. Ibrahim RA, Yousri D, Abd Elaziz M, Alshathri S, Attiya I (2021) Fractional calculus-based slime mould algorithm for feature selection using rough set. IEEE Access 9:131625–131636
24. Khunkitti S, Siritaratiwat A, Premrudeepreechacharn S (2021) Multi-objective optimal power flow problems based on slime mould algorithm. Sustainability 13(13):7448
25. Koumousis VK, Katsaras CP (2006) A saw-tooth genetic algorithm combining the effects of variable population size and reinitialization to enhance performance. IEEE Trans Evol Comput 10(1):19–28
26. Li L, Pan TS, Sun XX, Chu SC, Pan JS (2021) A novel binary slime mould algorithm with au strategy for cognitive radio spectrum allocation. Int J Comput Intell Syst 14(1):1–18
27. Li S, Chen H, Wang M, Mirjalili AAHS (2020) Slime mould algorithm: a new method forstochastic optimization. Futur Gener Comput Syst. https://doi.org/10.1016/j.future.2020.03.055
28. 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, vol 635. Technical report Zhengzhou, China
29. Liu L, Zhao D, Yu F, Heidari AA, Ru J, Chen H, Mafarja M, Turabieh H, Pan Z (2021) Performance optimization of differential evolution with slime mould algorithm for multilevel breast cancer image segmentation. Comput Biol Med 138:104910
30. Liu M, Li Y, Huo Q, Li A, Zhu M, Qu N, Chen L, Xia M (2020) A two-way parallel slime mold algorithm by flow and distance for the travelling salesman problem. Appl Sci 10(18):6180
31. Liu Y, Heidari AA, Ye X, Liang G, Chen H, He C (2021) Boosting slime mould algorithm for parameter identification of photovoltaic models. Energy 234:121164
32. Mirjalili S, Lewis A (2016) The whale optimization algorithm. Adv Eng Softw 95:51–67. https://www.sciencedirect.com/science/article/pii/S0965997816300163
33. Mostafa M, Rezk H, Aly M, Ahmed EM (2020) A new strategy based on slime mould algorithm to extract the optimal model parameters of solar pv panel. Sustain Energy Technol Assess 42:100849
34. Naik MK, Panda R, Abraham A (2020) Normalized square difference based multilevel thresholding technique for multispectral images using leader slime mould algorithm. J King Saud Univer-Comput Inf Sci. https://doi.org/10.1016/j.jksuci.2020.10.030
35. Naik MK, Panda R, Abraham A (2021) Adaptive opposition slime mould algorithm. Soft Comput 25(22):14297–14313

36. Naik MK, Panda R, Abraham A (2021) An entropy minimization based multilevel colour thresholding technique for analysis of breast thermograms using equilibrium slime mould algorithm. Appl Soft Comput 113:107955
37. Nguyen T (2020) A framework of optimization functions using numpy (opfunu) for optimization problems (2020). https://doi.org/10.5281/zenodo.3620960
38. Nguyen TT, Wang HJ, Dao TK, Pan JS, Liu JH, Weng S (2020) An improved slime mold algorithm and its application for optimal operation of cascade hydropower stations. IEEE Access 8:226754–226772
39. Poláková R, Tvrdík J, Bujok P (2019) Differential evolution with adaptive mechanism of population size according to current population diversity. Swarm Evolut Comput 50:100519
40. Piotrowski A (2017) Review of differential evolution population size. Swarm Evol Comput 32:1–24
41. Piotrowski AP, Napiorkowski JJ, Piotrowska AE (2020) Population size in particle swarm optimization. Swarm Evol Comput 58:100718
42. Polakova R, Tvrdik J, Bujok P (2014) Controlled restart in differential evolution applied to CEC2014 benchmark functions. In: IEEE congress on evolutionary computation, pp 2230–2236
43. Precup RE, David RC, Roman RC, Petriu EM, Szedlak-Stinean AI (2021) Slime mould algorithm-based tuning of cost-effective fuzzy controllers for servo systems. Int J Comput Intell Syst 14(1):1042–1052
44. Premkumar M, Jangir P, Sowmya R, Alhelou HH, Heidari AA, Chen H (2021) Mosma: multi-objective slime mould algorithm based on elitist non-dominated sorting. IEEE Access 9:3229–3248
45. Rizk-Allah RM, Hassanien AE, Song D (2021) Chaos-opposition-enhanced slime mould algorithm for minimizing the cost of energy for the wind turbines on high-altitude sites. ISA Trans 121:191–205
46. Shi B, Ye H, Zheng J, Zhu Y, Heidari AA, Zheng L, Chen H, Wang L, Wu P (2021) Early recognition and discrimination of covid-19 severity using slime mould support vector machine for medical decision-making. IEEE Access 9:121996–122015
47. Storn R, Price K (1997) Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. J Glob Optim 11:341–359
48. Sun G, Xu G, Gao R, Liu J (2019) A fluctuant population strategy for differential evolution. Evolut Intell. https://doi.org/10.1007/s12065-019-00287-6
49. Back T (1996) Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms. Oxford University Press on Demand
50. Teo J (2006) Exploring dynamic self-adaptive populations in differential evolution. Softw Comput. 10:673–686
51. Vashishtha G, Chauhan S, Singh M, Kumar R (2021) Bearing defect identification by swarm decomposition considering permutation entropy measure and opposition-based slime mould algorithm. Measurement 178:109389
52. Wang GG (2018) Moth search algorithm: a bio-inspired metaheuristic algorithm for global optimization problems. Memet Comput. https://doi.org/10.1007/s12293-016-0212-3
53. Wang GG, Deb S, Cui Z (2015) Monarch butterfly optimization. Neural Comput Appl. https://doi.org/10.1007/s00521-015-1923-y
54. Wazery YM, Saber E, Houssein EH, Ali AA, Amer E (2021) An efficient slime mould algorithm combined with k-nearest neighbor for medical classification tasks. IEEE Access 9:113666–113682
55. Wolpert DH, Macready WG (1997) No free lunch theorems for optimization. IEEE Trans Evol Comput 1(1):67–82
56. Yang M, Cai Z, Li C, Guan J (2013) An improved adaptive differential evolution algorithm with population adaptation. In: GECCO '13 proceedings of the 15th annual conference on genetic and evolutionary computation, pp 145–152
57. Yang M, Li C, Cai Z, Guan J (2014) Differential evolution with auto-enhanced population diversity. IEEE Trans Cybern 45:302–315
58. Yang XS, Deb S (2014) Cuckoo search: recent advances and applications. Neural Comput Appl 24(1):169–174
59. Yu C, Heidari AA, Xue X, Zhang L, Chen H, Chen W (2021) Boosting quantum rotation gate embedded slime mould algorithm. Expert Syst Appl 181:115082
60. Zhao J, Gao ZM (2020) The hybridized Harris hawk optimization and slime mould algorithm. In: Journal of physics: conference series, vol 1682. IOP Publishing, pp 012029
61. Zhao S, Wang P, Heidari AA, Chen H, Turabieh H, Mafarja M, Li C (2021) Multilevel threshold image segmentation with diffusion association slime mould algorithm and Renyi's entropy for chronic obstructive pulmonary disease. Comput Biol Med 134:104427
62. Zheng R, Jia H, Abualigah L, Liu Q, Wang S (2021) Deep ensemble of slime mold algorithm and arithmetic optimization algorithm for global optimization. Processes 9(10):1774
63. Zubaidi SL, Abdulkareem IH, Hashim KS, Al-Bugharbee H, Ridha HM, Gharghan SK, Al-Qaim FF, Muradov M, Kot P, Al-Khaddar R (2020) Hybridised artificial neural network model with slime mould algorithm: a novel methodology for prediction of urban stochastic water demand. Water 12(10):2692

Terms and Conditions

Springer Nature journal content, brought to you courtesy of Springer Nature Customer Service Center GmbH (“Springer Nature”).
Springer Nature supports a reasonable amount of sharing of research papers by authors, subscribers and authorised users (“Users”), for small-scale personal, non-commercial use provided that all copyright, trade and service marks and other proprietary notices are maintained. By accessing, sharing, receiving or otherwise using the Springer Nature journal content you agree to these terms of use (“Terms”). For these purposes, Springer Nature considers academic use (by researchers and students) to be non-commercial.
These Terms are supplementary and will apply in addition to any applicable website terms and conditions, a relevant site licence or a personal subscription. These Terms will prevail over any conflict or ambiguity with regards to the relevant terms, a site licence or a personal subscription (to the extent of the conflict or ambiguity only). For Creative Commons-licensed articles, the terms of the Creative Commons license used will apply.
We collect and use personal data to provide access to the Springer Nature journal content. We may also use these personal data internally within ResearchGate and Springer Nature and as agreed share it, in an anonymised way, for purposes of tracking, analysis and reporting. We will not otherwise disclose your personal data outside the ResearchGate or the Springer Nature group of companies unless we have your permission as detailed in the Privacy Policy.
While Users may use the Springer Nature journal content for small scale, personal non-commercial use, it is important to note that Users may not:

1. use such content for the purpose of providing other users with access on a regular or large scale basis or as a means to circumvent access control;
2. use such content where to do so would be considered a criminal or statutory offence in any jurisdiction, or gives rise to civil liability, or is otherwise unlawful;
3. falsely or misleadingly imply or suggest endorsement, approval , sponsorship, or association unless explicitly agreed to by Springer Nature in writing;
4. use bots or other automated methods to access the content or redirect messages
5. override any security feature or exclusionary protocol; or
6. share the content in order to create substitute for Springer Nature products or services or a systematic database of Springer Nature journal content.

In line with the restriction against commercial use, Springer Nature does not permit the creation of a product or service that creates revenue, royalties, rent or income from our content or its inclusion as part of a paid for service or for other commercial gain. Springer Nature journal content cannot be used for inter-library loans and librarians may not upload Springer Nature journal content on a large scale into their, or any other, institutional repository.
These terms of use are reviewed regularly and may be amended at any time. Springer Nature is not obligated to publish any information or content on this website and may remove it or features or functionality at our sole discretion, at any time with or without notice. Springer Nature may revoke this licence to you at any time and remove access to any copies of the Springer Nature journal content which have been saved.
To the fullest extent permitted by law, Springer Nature makes no warranties, representations or guarantees to Users, either express or implied with respect to the Springer nature journal content and all parties disclaim and waive any implied warranties or warranties imposed by law, including merchantability or fitness for any particular purpose.
Please note that these rights do not automatically extend to content, data or other material published by Springer Nature that may be licensed from third parties.
If you would like to use or distribute our Springer Nature journal content to a wider audience or on a regular basis or in any other manner not expressly permitted by these Terms, please contact Springer Nature at

onlineservice@springernature.com