Expert Systems With Applications 99 (2018) 56–70

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

ASCA-PSO: Adaptive sine cosine optimization algorithm integrated with particle swarm for pairwise local sequence alignment

Mohamed Issa[a,1], Aboul Ella Hassanien[b,1,*], Diego Oliva[c,1], Ahmed Helmi[a], Ibrahim Ziedan[a], Ahmed Alzohairy[d]

[a] Computer Engineering and Systems Department, Faculty of Engineering, Zagazig University, Egypt
[b] Faculty of Computers and Information, Cairo University, Egypt
[c] Departamento de Ciencias Computacionales, Universidad de Guadalajara, Mexico
[d] Genetic Department, Faculty of Agriculture, Zagazig University, Egypt

ARTICLE INFO

Article history:
Received 16 September 2017
Revised 12 January 2018
Accepted 13 January 2018
Available online 3 February 2018

Keywords:
Sine cosine algorithm (SCA)
Particle swarm optimization (PSO)
Meta-heuristics algorithms
Pairwise local alignment
Longest consecutive substrings
Smith-Waterman alignment algorithm

ABSTRACT

The sine cosine algorithm (SCA), a recently proposed population-based optimization algorithm, is based on the use of sine and cosine trigonometric functions as operators to update the movements of the search agents. To optimize performance, different parameters on the SCA must be appropriately tuned. Setting such parameters is challenging because they permit the algorithm to escape from local optima and avoid premature convergence. The main drawback of the SCA is that the parameter setting only affects the exploitation of the prominent regions. However, the SCA has good exploration capabilities. This article presents an enhanced version of the SCA by merging it with particle swarm optimization (PSO). PSO exploits the search space better than the operators of the standard SCA. The proposed algorithm, called ASCA-PSO, has been tested over several unimodal and multimodal benchmark functions, which show its superiority over the SCA and other recent and standard meta-heuristic algorithms. Moreover, to verify the capabilities of the SCA, the SCA has been used to solve the real-world problem of a pairwise local alignment algorithm that tends to find the longest consecutive substrings between two biological sequences. Experimental results provide evidence of the good performance of the ASCA-PSO solutions in terms of accuracy and computational time.

1. Introduction

In the past two decades, nature-inspired optimization methods, also called meta-heuristic algorithms (MAs), have attracted the attention of researchers from a variety of fields (Boussaïd, Lepagnot, & Siarry, 2013). MAs search for optimal solutions based on a search strategy that imitates a natural behavior. In this sense, different metaphors are created in MAs, such as genetic algorithms (GAs) (Holland, 1992) and differential evolution (DE) (Storn & Price, 1997), which are based on evolutionary theory. Additionally, physically based algorithms include methods such as the sine cosine algorithm (SCA) (Mirjalili, 2016), the ions motion optimization (IMO) (Javidy, Hatamlou, & Mirjalili, 2015) and the gravitational search algorithm (GSA) (Rashedi, Nezamabadi-Pour, & Saryazdi, 2009). Another group of methods is based on insects and other animals and includes particle swarm optimization (PSO) (Kennedy, 1995), artificial bee colony (ABC) (Karaboga & Akay, 2009) and moth-flame optimization (MFO) (Mirjalili, 2015). Other algorithms imitate human concepts (or creations), such as the mine blast algorithm (MBA) (Sadollah, Bahreininejad, Eskandar, & Hamdi, 2013) and teaching learning-based algorithm (TLBO) (Rao, Savsani, & Vakharia, 2011).

Two contradictory factors must be considered in designing new MAs: the exploration of the search space (diversification) and the exploitation of prominent regions (intensification). Exploration is used to diversify the regions of the search space to ensure that all regions of the search space are evenly explored and that the search is not confined to a limited number of regions and, in addition, to avoid becoming trapped in local minima. Exploitation is the process of analyzing the bounded search area around the best solution in order to improve it. Balancing between exploration and exploitation is essential to enhancing the efficiency of a meta-heuristic algorithm.

In the related literature, a substantial number of meta-heuristics can be attributed to the no-free-lunch (NFL) theorem (Wolpert & Macready, 1997), which states that the success of an optimization technique in addressing a specific problem

* Corresponding author.
E-mail addresses: aboitcairo@fci-cu.edu.eg (A.E. Hassanien), doliva@ucm.es (D. Oliva).
[1] Scientific Research Group in Egypt, (SRGE) http://www.egyptscience.net.

https://doi.org/10.1016/j.eswa.2018.01.019
0957-4174/

does not guarantee success in different optimization problems with different natures and types. Hence, according to NFL, the research in meta-heuristics has three main directions: (1) the improvement of current methods, (2) the creation of new algorithms, and (3) the combination of different meta-heuristics. The first direction modifies the operators to enhance the performance of the existing approaches, such as chaotic maps (Wang, Guo, Gandomi, Hao, & Wang, 2014; Petrović, Mitić, Vuković, & Miljković, 2016), local searches (Cao, Li, & Chaovalitwongse, 2017; Premalatha & Natarajan, 2008) and evolutionary operators (Wang, Guo, Duan, Liu, & Wang, 2012; Wang, Deb, Gandomi, & Alavi, 2016). The second direction proposes new optimization mechanisms inspired by different behaviors, such as the slap swarm algorithm (SSA) (Mirjalili et al., 2017), whale optimization algorithm (WOA) (Mirjalili & Hatamlou, 2016) and multi-verse optimizer (MVO) (Mirjalili & Lewis, 2016). The most recent direction for meta-heuristics is hybridizing different optimization algorithms to benefit from each of their advantages (Garg, 2016; Güçyetmez & Çam 2016; Santra, Mukherjee, Sarker, & Chatterjee, 2016; Yang, Wang, Lin, & Chen, 2016).

One of the main problems of MAs is that they commonly have parameters that must be tuned according to the problems to be solved. In this context, the SCA possesses several parameters that must be tuned to maximize performance in the optimization process. Tuning these parameters is challenging, and if they are not correctly selected, the algorithms can become trapping in local optima or premature convergence. However, the main advantage of the SCA is its power of exploration of the search space. SCA has been used in applications and has efficient performance on problems such as handwritten Arabic text (Mudhsh, Xiong, El Aziz, Hassanien, & Duan, 2017), photovoltaic systems (Kumar, Hussain, Singh, & Panigrahi, 2017) and detection of galaxies using image retrieval (Abd ElAziz, Xiong, & Selim, 2017). However, according to the NFL theory, the SCA would not perform well on all optimization problems, including finding the longest consecutive substrings between two biological sequences. This article introduces an enhanced version of the SCA and merges it with PSO, a traditional optimization algorithm inspired by the behavior of flocking birds or schooling fish. The main advantages of PSO are robustness, the need for few parameters and the efficient exploitation of the search space.

Considering the above, performance of the SCA is improved by integrating the features of PSO to exploit the optimal solutions with the capabilities of the SCA to explore the entire search space. This hybridization provides a good balance between exploration and exploitation throughout the iterative process.

The proposed algorithm is called ASCA-PSO and consists of two layers: the bottom layer is responsible for exploration the search space, which is performed by the search agents of the SCA; the top layer is responsible for the exploitation of the best solution found by the bottom layer based on the search agents of the PSO. The proposed approach enables a good diversification of the population and preserves the best information on the position at each iteration.

To verify the performance of the proposed algorithm, it was tested over a set of mathematical optimization problems with different degrees of difficulty. Moreover, the local sequence alignment (LSA) problem is used as a case of study for testing the improved ASCA-PSO. The experimental results on both the mathematical and LSA problem provide evidence of the accuracy of the proposed method in complex optimization problems and show that it maintains balance with regard to the computational time.

The remainder of this paper is organized as follows. Section 2 describes the preliminaries for the standard SCA and PSO. The proposed technique is presented in Section 3. Section 4 describes the results of the proposed algorithm compared with the testing benchmark functions. Section 5 provides the fragmentation local sequence alignment technique based on a proposed algorithm for comparison with the SCA. Finally, Section 6 presents the conclusions.

2. Preliminaries

This section provides a brief explanation of the basic framework of PSO and the SCA along with some of the fundamental concepts.

2.1. Particle swarm optimization (PSO)

Swarm optimization algorithms are stochastic population-based search methods that mimic the behavior of fish schooling, birds flocking and other grouping behaviors. The search strategy of these algorithms is mainly based on global communications among the individuals of the population, where the particles adapt their movements toward the particle that finds the best solution. PSO generates a swarm of particles, and each particle has a position (a solution to the problem) in the search space. All particles tune their movements according to Eqs. (1) and (2) toward the particle (P^{gbest}) that has the best position (the global best solution) and the best personal position (P_i^{best}) for each particle pass during the previous iterations.

$$v_i(t+1) = w * v_i(t) + c_1\, rand\, (P^{\text{best}}{}_i - P_i(t)) + c_2\, rand\, (P^{\text{gbest}} - P_i(t)) \quad (1)$$

$$P_i(t+1) = P_i(t) + v_i(t+1) \quad (2)$$

Here, v_i is the velocity of the ith particle, c_1 is the best local position weight coefficient, and c_2 is the global best position weight coefficient. w is the inertia coefficient that controls the effect of the previous velocity on the new velocity. P_i is the position of particle i, t is the iteration number, and *rand* is a uniformly distributed random variable in the range (0–1). P_i^{best} is the best local position (solution) found by particle P_i, and P^{gbest} is the best solution found in the whole swarm.

PSO has several advantages, such as robustness and information interchange among particles, which provides a high probability of achieving a near-optimal solution with a reasonable convergence speed. PSO has been used to solve many optimization problems, such as the optimization of the parameters of electrical motors (Calvini, Carpita, Formentini, & Marchesoni, 2015), solar cell design (Khanna, Das, Bisht, & Singh, 2015) and surgical robot applications (Tuvayanond & Parnichkun, 2017). The steps of PSO are summarized in Algorithm 1.

The time complexity of a PSO is $O(T{*}n{*}c_{\text{pso}})$, where n is the number of particles, c_{pso} is the time cost of updating the position of one particle and T is the number of iterations.

2.2. Sine-cosine optimization algorithm (SCA)

SCA is a population-based optimization algorithm that depends on sine and cosine operators for updating the movement of the

Algorithm 1 Particle swarm optimization.

1: Initialize a set of population solutions (P_i), initial velocity (v_i) and algorithm's parameters (c_1, c_2 and w)
2: **Repeat**
3: Evaluate the objective function based on population solutions
4: Update the best local solution for each particle (P_i^{best})
5: Update the best global solution over all particles (P_{gbest})
6: Update the next position of population solutions using Eqs. (1) and (2)
7: **Until** ($T <$ maximum number of iterations)
8: Return the best solution (P^{gbest}) obtained as the global optimum

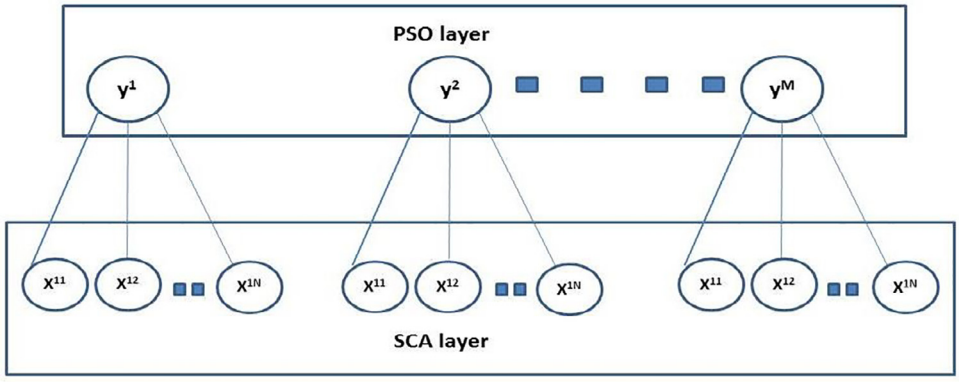

Fig. 1. The structure of the proposed adaptive SCA-PSO.

Algorithm 2 Sine-cosine optimization algorithm.

1: Initialize an N search agents (P_i) where $1 < i < N$, algorithm parameters (r_1, r_2, r_3, and r_4)
2: **Repeat**
3: Evaluate the objective function based on search agents
4: Update the best solution obtained so far (P_{gbest})
5: Update r_1, r_2, r_3 and r_4
6: Update the next position of each search agent solutions using Eq. (3)
7: **Until** ($T <$ maximum number of iterations)
8: Return the best solution (P^{gbest}) obtained as the global optimum solution found

search agents toward the best solution found according to Eq. (3).

$$P_i^{t+1} = \begin{cases} P_i^t + r_1 \sin(r_2) \mid r_3 P^{\text{gbest}} - P_i^t \mid r_4 < 0.5 \\ P_i^t + r_1 \cos(r_2) \mid r_3 P^{\text{gbest}} - P_i^t \mid r_4 \geq 0.5 \end{cases} \quad (3)$$

where r_1 is the parameter responsible for determining the next region of the search and increasing the exploration of the search space for a higher value of it, r_2 defines the direction of movement toward or away from P_{gbest} and how far the movement should go, and r_3 controls the effect of the destination on the current movement.

The values of r_1, r_2, and r_3 are updated at each iteration to increase the diversity of the solutions. r_4 is used to switch between the sine and cosine functions, as in Eq. (3). Eq. (4) is used to balance between exploration and exploitation, where t is the current iteration, T is the maximum number of iterations, and a is a constant that should be set by the designer.

$$r_1 = a\left(1 - \frac{t}{T}\right) \quad (4)$$

The steps of the SCA are presented in Algorithm 2. The time complexity of the SCA is $O(T*n*c_{\text{sca}})$, where n is the number of search agents, c_{sca} is the time cost of updating the positions of the search agent and T is the number of iterations.

3. Adaptive sine cosine integrated with particle swarm (ASCA-PSO)

In this section, the enhanced ASCA-PSO is presented. This algorithm improves the convergence and quality of solutions produced by the standard SCA. The basic SCA has drawbacks, such as slow convergence and the tendency to become trapped in local optima solutions in optimization problems such as finding the longest consecutive substrings (Smith & Waterman, 1981). Such problems occur because the SCA has parameters that are difficult to properly tune, and a poor configuration is reflected in a weak exploitation. However, the exploration of the search space is not affected by these parameters.

The structure of the proposed approach considers layers as in Fig. 1, where the top layer contains M particles and their movement is performed by the PSO operators. The bottom layer separates the population into M groups, and each group contains N search agents; the new positions are computed using the SCA. The best solution found by each group is kept by a particle in the top layer. Hence, the bottom layer focuses on exploring the search space, while the top layer focuses on exploiting the best solutions found by the bottom layer. This hybridization scheme belongs to the classification of high-level and co-evolutionary hybrid metaheuristics (Talbi, 2009).

Each search agent in the bottom layer is described as (x_{ij}), where $i = 1, 2, 3, \ldots, M$, and $j = 1, 2, 3, \ldots, N$. i and j represent the index of solutions in the top and bottom layer, respectively. Such elements of the population are updated based on the SCA according to Eq. (5) toward the best solution found by (y_i), and the parameters of the SCA are tuned for the exploration phase more than for exploitation.

$$x_{ij}^{t+1} = \begin{cases} x_{ij}^t + r_1 \sin(r_2) \mid r_3 y_i^t - x_{ij}^t \mid r_4 < 0.5 \\ x_{ij}^t + r_1 \cos(r_2) \mid r_3 y_i^t - x_{ij}^t \mid r_4 \geq 0.5 \end{cases} \quad (5)$$

In the same context, each solution from the top layer is represented by (y^i), where $i = 1, 2, 3, \ldots, M$, and y^i also represents the best solution found by each i group in the bottom layer. The elements of the top layer update their positions in the local region of the best solution found from the bottom layer based on the PSO and move toward the best solution (y^{gbest}) from all search agents of the top and bottom layers.

This process is performed using Eqs. (6) and (7), where (y_i^{pbest}) represents the best single solution for y_i over all previous iterations. Here, y^{gbest} is the best global solution for all individuals and is determined as the best solution in the top layer.

$$v_i^{t+1} = w * v_i^t + c_1 \, rand \, (y_i^{\text{pbest}} - y_i^t) + c_2 \, rand \, (y^{\text{gbest}} - y_i^t) \quad (6)$$

$$y_i^{t+1} = y_i^t + v_i^t \quad (7)$$

Hence, the individuals in the bottom layer (x_{ij}) are influenced by the best solution of the group in the top layer (y_i). Moreover, (y_i) is influenced by the best solution found between the whole set of individuals in the top layer (y^{gbest}). This fact increases the diversity of the search space. Additionally, performing exploration and exploitation together in each iteration enhances the chance of finding the global optimum solution. Algorithm 3 shows the proposed ASCA-PSO optimization algorithm.

Algorithm 3 ASCA-PSO optimization.

1: Initialize $M \times N$ search agents (x_{ij}) in the bottom layer and M particles (y_i) in the top layer, SCA parameters (r_1, r_2, r_3 *and* r_4), PSO parameters (w, c_1, c_2).
2: Evaluate the objective function based on search agents (x).
3: Update each particle (y_i) by the best solution found by the related group in bottom layer
4: Update y^{gbest} by the solution produce best fit from the particles in the top layer
5: **Repeat**
6: *for* $i = 1$: M
7: *for* $j = 1$: N
8: *Update* (x_{ij}) according to Eq. (5)
9: If ($x_{ij} < y_i$) Then $y_i = x_{ij}$
10: Update r_1, r_2, r_3 and r_4
11: *end for j*
12: *Update* (y_i) according to Eqs. (6) and (7)
13: Update y^{gbest} by the solution produce best fit from the particles in the top layer
14: *end for i*
15: **Until** ($T <$ maximum number of iterations)
16: Return the best solution (y^{gbest}) obtained as the global minimum solution

The time complexity of the ASCA-PSO algorithm is $O(TM(Nc_{sca} + c_{pso}))$, where M and N are the sizes of the search agents in the top and bottom layers, respectively, and c_{sca} and c_{pso} are the time costs of updating all of the search agents per iteration for the SCA and PSO, respectively, while T is the number of iterations.

4. Experimental results and discussion

ASCA-PSO has been tested on several unimodal and multimodal mathematical benchmark functions, listed in Table 1 (Jamil & Yang, 2013), to measure its performance compared to the SCA with regard to the quality of solution and convergence speed.

4.1. Evaluation criteria

In each run of the individual optimizer, the following measures are calculated to test the mathematical benchmark functions:

- **Statistical mean** is the average of the solutions (S_i) that are produced by executing the optimization algorithm M times and is calculated according to Eq. (8).

$$Mean = \frac{1}{M}\sum_{i=1}^{M} S_i \tag{8}$$

where S_i is the obtained solution of the run time i.
- **Statistical standard deviation (Std)** is an indicator of the variation of the best fitness values found when running the optimization algorithm for M run times. Additionally, it represents the robustness and stability, and it is computed as shown in Eq. (9):

$$\text{Std} = \sqrt{\frac{1}{M-1}\sum_{i=1}^{M}(S_i - \text{Mean})^2} \tag{9}$$

- **Wilcoxon rank sum test** (Higgins, 2003) is a non-parametric test description of the t-test for two independent groups and is used to test the null hypothesis that two data points, x and y vectors, are acquired from continuous distributions with equal medians against the alternative that they are not.

Table 1
Benchmark of test functions.

Function		Dimension	Range of bounds	F_{min}
F_1	$-20\ e^{-0.2\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}} - e^{\frac{1}{d}\sum_{i=1}^{d}\cos(2\pi x_i)} + 20 + e^1$	30	$[-32,\ 32]$	0
F_2	$10\ d + \sum_{i=1}^{d}[x_i^2 - 10\cos(2\ \pi\ x_i)]$	30	$[-5.12,\ 5.12]$	0
F_3	$\sum_{i=1}^{d}\frac{x_i^2}{4000} - \prod_{i=1}^{d}\cos(\frac{x_i}{\sqrt{i}}) + 1$	30	$[-600,\ 600]$	0
F_4	$\sum_{i=1}^{d} i\ x_i^4$	30	$[-1.28,\ 1.28]$	0
F_5	$\sum_{i=1}^{d} x_i^2$	30	$[-5.12,\ 5.12]$	0
F_6	$\sum_{i=1}^{D}\lfloor x_i + 0.5\rfloor^2$	30	$[-100,\ 100]$	0
F_7	$\frac{-1-\cos(12\sqrt{x_1^2+x_2^2})}{0.5(x_1^2+x_2^2)+2}$	2	$[-5.12,\ 5.12]$	−1.00
F_8	$0.5 + \frac{\sin^2(x_1^2-x_2^2)-0.5}{[1+0.001(x_1^2+x_2^2)]^2}$	2	$[-100,\ 100]$	0
F_9	$\sum_{i=1}^{d}[\sum_{j}^{i}(x_i)]^2$	30	$[-65,\ 65]$	0
F_{10}	$(x_1 - 1)^2 + \sum_{i=2}^{d} i\ (2x_i^2 - x_{i-1})^2$	30	$[-10,\ 10]$	0
F_{11}	$\sum_{i=1}^{d} x_i + \prod_{i=1}^{d} x_i$	30	$[-10,\ 10]$	0
F_{12}	$4x_1^2 - 2.1\ x_1^2 + 0.33\ x_1^6 + x_1\ x_2 + 4\ x_2^2 - 4\ x_2^4$	2	$[-5,\ 5]$	−1.03
F_{13}	$max_i\{\lvert x_i\rvert, 1 \le i \le D\}$	30	$[-100,100]$	0
F_{14}	$\sum_{i=1}^{D}\lvert x_i \sin(x_i) + 0.1\ x_i\rvert$	30	$[-10,10]$	0
F_{15}	$\sum_{i=1}^{D}\sin(x_i) * (\sin(\frac{i\ x_i^2}{\pi}))^{20}$	30	$[0,\pi]$	0
F_{16}	$[1 + (x_1 + x_2 + 1)^2\ (19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)]$ $X\ [30 + (2x_1 - 3x_1)(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)]$	2	$[-2,2]$	3.00
F_{17}	$(1.5 - x_1 + x_1\ x_2)^2 + (2.25 - x_1 + x_1\ x_2^2)^2 + (2.625 - x_1 + x_1\ x_2^3)^2$	2	$[-4.5,\ 4.5]$	0
F_{18}	$(\sum_{i=1}^{D} x_i^2)^2$	30	$[-100,100]$	0
F_{19}	$\left(\sum_{i=1}^{D}\lvert x_i\rvert\right)\left(e^{\sum_{i=1}^{D}-\sin(x_i^2)}\right)$	30	$[-2\pi,\ 2\pi]$	0
F_{20}	$\left[\sum_{i=1}^{D} sin^2(x_i) - e^{-\sum_{i=1}^{D} x_i^2}\right]\left(e^{-\sum_{i=1}^{D} sin^2\sqrt{\lvert x_i\rvert}}\right)$	30	$[-10,10]$	0

Table 2
The parameters of the algorithm and their values.

Algorithm	Parameter	Value
GA	Cross over percentage	0.7
	Mutation percentage	0.3
	Mutation rate	0.1
SCA	A	2.0
	r_1	10
	r_2	random $(0, 2\pi)$
	r_3	1.5
PSO	Maximum velocity (V_{max})	1.0
	Minimum velocity (V_{min})	−1.0
	Cognitive coefficient (C1)	0.5
	Cognitive coefficient (C2)	0.5
MFO	B	1.0
	L	[−1,1]
ABC	The maximum cycle number	100
	modification rate (MR)	0.8
WOA	$\bar{a}$	2
	$\bar{r}$	1
MVO	Minimum of wormhole existence probability	0.2
	Maximum of wormhole existence probability	1.0
	Maximum inertia weight (w_{max})	0.9
	Minimum inertia weight (w_{min})	0.4
SSA	c_1	2
	c_2, c_3	Random (0.1)
GWO	$\bar{a}$	2
	$\bar{r}$	[−1,1]

4.2. Comparison between ASCA-PSO, SCA and other similar algorithms

ASCA-PSO has also been compared with the standard SCA and nine recent and standard optimization algorithms, namely, moth-flame optimization (MFO) (Mirjalili, 2015), the slap swarm algorithm (SSA) (Mirjalili et al., 2017), artificial bee colony (ABC), PSO, the genetic algorithm (GA), the whale optimization algorithm (WOA) (Mirjalili & Hatamlou, 2016), the multi-verse optimizer (MVO) (Mirjalili & Lewis, 2016) and grey wolf optimization (GWO) (Mirjalili, Mirjalili, & Lewis, 2014).

Table 2 shows the parameter settings of these algorithms in the experimental test over the benchmark functions. The number of populations was 500 with 300 iterations. All tests were done on Matlab R2015a on a computer machine has the following specification: Intel processor with I3 core each one has speed 2.27 GHz with 4 GB RAM. The experimental tests were implemented on a CPU Core I3 with 4 GB RAM on Matlab 2015b software. All the algorithms were executed with 50 independent runs over each benchmark function for statistical analysis. Tables 3 and 4 show the average optimal values of the test functions and standard deviations for all of the algorithms over 50 independent runs. As shown in Table 3, ASCA-PSO outperformed all of the other algorithms with regard to the quality of the solutions for all functions.

In contrast, the other algorithms produced accurate results on certain functions and poor results on others. Each method produced poor results on a set of functions, as follows: GA (F_1, F_2, F_6, F_{10}, F_{13}), SCA (F_2, F_3, F_{10}, F_{14}), PSO (F_1-F_3, F_{10}, F_{11}, F_{13}, F_{14}, F_{18}), MFO (F_1-F_6, F_9-F_{11}, F_{13}, F_{14}, F_{18}), MVO (F_1-F_4, F_6, F_9-F_{11}, F_{13}, F_{14}, F_{18}) and SSA (F_1-F_2, F_6, F_9, F_{10}, F_{13}). ABC and WOA produced accurate results on most of the functions except for (F_{13}, F_{16}) and (F_{13}), respectively. This finding reflects the efficient performance of the proposed ASCA-PSO in comparison with the other algorithms. Hence, the order of algorithms for producing high-quality means of solutions is as follows: (1) ASCA-PSO, (2) WOA, (3) ABC, (4) SCA, and (5) other algorithms.

Table 4 shows the standard deviation values of 50 independent runs, and ASCA-PSO provides the minimum standard deviation for almost all of the functions, except for F_{19} and F_{20}. After ASCA-PSO is GA, which provides a minimum standard deviation for the functions (F_8, F_{12}, F_{15}), followed by SCA (F_4), PSO (F_{12}, F_{16}), MFO (F_{15}, F_{16}, F_{19}), ABC (F_2, F_{12}). WOA (F_1, F_5, F_9, F_{11}, F_{18}), MVO (F_{19}), SSA (F_7, F_{15}, F_{19}, F_{20}) and GWO (F_1, F_{19}). Thus, the order of the algorithms for producing accurate quality means of the solutions is as follows: (1) ASCA-PSO, (2) WOA, (3) SCA, (4) GA and MFO, and (5) other algorithms.

Table 5 provides the average execution time for 50 independent runs of each algorithm for all of the functions. SCA produces the minimum execution time, followed by ASCA-PSO, PSO, GA, and ABC.

In the Wilcoxon rank test, as the null hypothesis, it is assumed that there is no difference among the algorithms that are compared. The alternative hypothesis considers an actual difference between the values from both approaches. The results obtained by the Wilcoxon test indicate that data cannot be assumed to be occurring by coincidence (i.e., due to the typical noise contained in the process). Thus, this test is used as proof that the results of ASCA-PSO are different from those using the other methods. The analysis of the values of Table 6 considers a 5% significance over the averaged best values of the functions. Each algorithm was run for 50 single experiments, after which the Wilcoxon rank test was performed over the best solutions found in each experiment.

Table 3
The average minimum values for all algorithms.

F	ASCA-PSO	GA	SCA	PSO	MFO	ABC	WOA	MVO	SSA	GWO
F_1	8.88E−16	0.21	1.71E−07	3.62	5.36	6.80E−14	4.50E−15	0.84	0.51	3.90E−14
F_2	0.00	1.11	67.6	44.24	113.74	4.80E−07	0.00	93.41	25.84	1.04
F_3	0.00	0.03	0.22	1.11	1.67	0.00	1.30E−03	0.58	0.00	0.00
F_4	7.8E−264	2.00E−12	5.17E−29	0.00	1.697	3.60E−16	2.00E−03	0.56	0.00	0.00
F_5	3.8E−131	8.0E−06	9.91E−15	0.05	0.191	8.90E−16	1.50E−71	0	1.90E−11	1.00E−35
F_6	0.00	0.82	0.00	0.00	85.98	0.00	0.00	3.78	4.92	0.00
F_7	−1	−0.98	−0.99	−1	−1	−1	−1	−0.99	−1	−1
F_8	0	2.60E−17	0	0	0	2.40E−08	0	4.20E−06	2.90E−13	0.00
F_9	8.8E−128	0.05	5.20E−11	5.9E−178	5260.6	8.90E−16	7.80E−69	8.49	4.08	2.80E−32
F_{10}	0.00	1.88	0.66	20.38	3729.6	0.00	6.50E−01	1.7	0.8	0.66
F_{11}	4.94E−65	0.04	4.55E−08	1.48	27.57	1.90E−15	1.30E−39	0.31	0.09	2.30E−19
F_{12}	−1.03	−1.03	−1.03	−1.03	−1.031	−1.03	−1.03	−1.03	−1.03	−1.03
F_{13}	4.33E−65	0.7	3.18E−07	11.69	31.9	44.5	1.14E	0.48	0.36	5.50E−09
F_{14}	4.14E−66	0.00	0.31	53	2.792	1.40E−05	2.50E−01	2.63	0.00	0.00
F_{15}	0.00	1.40E−88	0.05	2.52E−19	1.00E−48	3.20E−19	0.00	1.80E−07	6.70E−34	0.00
F_{16}	3	3	3	3	3	3.21	3	3	3	3
F_{17}	2.00E−25	5.60E−05	0.00	0.00	0.00	0.00	7.40E−14	1.80E−08	1.40E−15	2.80E−08
F_{18}	2.6E−256	1.90E−05	1.70E−22	370.99	7216.1	1.80E−17	2.70E−90	0.12	6.20E−17	1.80E−64
F_{19}	1.94E−06	4.10E−12	3.50E−06	5.92E−12	3.30E−11	3.60E−12	3.30E−11	3.30E−11	3.30E−11	3.30E−11
F_{20}	3.98E−11	6.60E−18	4.50E−11	3.45E−14	5.20E−14	1.90E−15	2.40E−01	6.50E−16	1.40E−23	2.60E−16

Table 4
Standard deviation of testing functions for all algorithms. The bold values provides the minimum standard deviation for almost all of the functions.

F	ASCA-PSO	GA	SCA	PSO	MFO	ABC	WOA	MVO	SSA	GWO
F_1	**0**	0.37	3.20E−07	1.21	3.41	1.50E−14	2.30E−15	0.62	0.65	3.70E−15
F_2	**0**	0.68	82.5	11.69	30.6	2.30E−06	**0**	26.61	10.64	2.76
F_3	0.09	0.03	0.28	0.26	0.37	0	**0**	0.08	**0**	**0**
F_4	**0**	2.80E−12	1.49E−28	**0**	0.34	8.50E−17	0	0.06	**0**	**0**
F_5	1.6E−131	7.00E−06	1.85E−14	0.08	0.07	1.90E−16	7.30E−71	0	3.12E−12	1.30E−35
F_6	**0**	0.71	**0**	**0**	41.4	**0**	**0**	2.34	2.63	**0**
F_7	**0**	0.02	0.0175	**0**	0.48	**0**	**0**	6.10E−09	6.20E−16	**0**
F_8	**0**	1.80E−16	**0**	**0**	0	6.10E−08	**0**	4.50E−06	2.80E−13	**0**
F_9	4.5E−128	0.08	**0**	120.8	2916.32	2.30E−16	4.10E−68	5.73	11.07	3.30E−32
F_{10}	0.02	3.05E−04	**0**	18.71	18,300.8	0.06	0.08	1.77	0.29	7.80E−06
F_{11}	**2.20E−65**	0.08	6.38E−08	0.88	18.18	2.50E−16	4.10E−39	0.06	0.18	1.90E−19
F_{12}	**0**	4.40E−16	**0**	4.49E−16	6.60E−16	4.40E−16	3.70E−14	7.10E−08	9.20E−15	2.60E−09
F_{13}	**2.64E−65**	0.13	2.65E−07	2.63	7.78	6.5	20.373	0.12	0.48	3.20E−09
F_{14}	2.19E−66	**0**	1.03	4.31	5.13	3.10E−05	1.75	1.4	0.02	**0**
F_{15}	**0**	1.00E−87	0.06	9.67E−19	7.30E−48	4.60E−19	**0**	8.30E−07	4.36E−33	**0**
F_{16}	0.25	3.21E−15	**0**	3.32E−15	2.60E−15	0.17	3.40E−08	6.70E−07	5.90E−14	4.60E−07
F_{17}	9.20E−41	**0**	**0**	**0**	**0**	**0**	2.05E−13	1.50E−08	1.30E−15	2.50E−08
F_{18}	**0**	3.65E−05	8.84E−22	1200	6544.15	1.60E−17	1.6E−137	0.05	2.10E−17	6.30E−64
F_{19}	1.95E−06	3.40E−13	3.98E−06	1.35E−12	**4.50E−26**	1.30E−13	**4.54E−26**	**4.50E−26**	**4.50E−26**	**4.50E−26**
F_{20}	1.90E−11	7.60E−18	1.45E−11	3.48E−14	4.80E−14	4.00E−16	0.42	1.90E−16	**3.50E−24**	4.80E−17

Table 5
The elapsed time of computing functions for all algorithms. The bold values means, SCA produces the minimum execution time.

F	ASCA-PSO	GA	SCA	PSO	MFO	ABC	WOA	MVO	SSA	GWO
F_1	3.35	3.48	**1.98**	3.59	7.05	3.22	8.67	18.35	17.42	6.73
F_2	2.11	2.23	**1.72**	2.32	4.51	3.39	6.05	15.90	17.39	6.75
F_3	3.40	3.69	**1.97**	3.61	6.52	4.22	7.83	17.55	17.44	6.77
F_4	3.07	2.99	**1.83**	3.30	6.88	1.88	7.80	17.40	17.36	6.80
F_5	1.85	2.93	**1.54**	2.02	5.72	1.63	7.71	17.73	17.61	6.72
F_6	1.88	2.01	**1.55**	1.96	5.58	2.39	7.52	17.72	19.37	6.83
F_7	1.71	2.56	**0.47**	1.70	3.85	1.98	4.81	5.18	17.51	6.81
F_8	0.53	1.82	**0.23**	0.53	3.75	1.09	4.78	4.87	17.48	6.73
F_9	5.91	3.33	**2.36**	6.11	9.48	3.65	10.8	22.20	17.45	6.73
F_{10}	2.06	2.65	**1.58**	2.22	5.49	2.79	7.18	17.57	17.34	6.73
F_{11}	1.78	2.40	**1.54**	1.90	6.47	3.45	7.92	17.74	17.33	6.73
F_{12}	0.69	1.66	**0.26**	0.70	4.04	1.01	4.89	4.95	18.59	6.75
F_{13}	3.53	3.88	**1.83**	3.67	5.90	2.98	7.48	18.31	18.50	7.23
F_{14}	2.02	2.13	**1.57**	2.23	5.67	2.12	6.98	18.09	17.38	6.95
F_{15}	3.90	3.95	**1.86**	4.18	7.41	4.45	7.30	19.3	17.34	6.80
F_{16}	0.58	0.99	**0.25**	0.59	3.73	1.39	4.66	4.75	17.51	6.78
F_{17}	0.57	2.33	**0.24**	0.56	3.82	2.56	4.60	4.69	17.58	6.70
F_{18}	1.94	1.22	**1.61**	2.11	5.58	3.24	6.98	17.21	17.40	6.77
F_{19}	2.18	2.56	**1.63**	2.44	6.50	4.56	7.33	17.81	19.42	6.75
F_{20}	3.99	3.65	**2.00**	4.25	6.86	6.97	7.91	18.56	17.53	6.75

Table 6
P-value of Wilcoxon rank sum test of comparison between ASCA-PSO and other algorithms ($P > .05$ is underlined).

F	ASCA-PSO vs.								
	GA	SCA	PSO	MFO	ABC	WOA	MVO	SSA	GWO
F_1	**3.30E−20**	**3.30E−20**	**3.30E−20**	**3.30E−20**	**2.90E−20**	2.50E−15	**3.30E−20**	**3.30E−20**	9.00E−21
F_2	**3.30E−20**	2.30E−15	**3.30E−20**	**3.30E−20**	**3.30E−20**	**2.90E−20**	**3.30E−20**	**3.20E−20**	4.00E−14
F_3	0.94	0.23	2.30E−17	**7.00E−18**	3.50E−12	**2.50E−18**	7.00E−16	3.40E−06	2.60E−15
F_4	**7.00E−18**	**7.00E−18**	**7.00E−18**	**7.00E−18**	**7.00E−18**	7.70E−14	**7.00E−18**	**7.00E−18**	2.30E−08
F_5	7.00E−18	7.00E−18	7.00E−18	7.00E−18	7.00E−18	7.00E−18	7.00E−18	7.00E−18	7.00E−18
F_6	5.70E−12	**3.30E−20**	**3.30E−20**	**3.30E−20**	**2.90E−20**	3.50E−12	**2.60E−20**	**2.90E−20**	3.50E−12
F_7	6.20E−19	**3.30E−20**	**3.30E−20**	0.01	**3.30E−20**	**3.30E−20**	6.80E−18	**3.30E−20**	**3.30E−20**
F_8	0.32	**3.30E−20**	**3.30E−20**	**3.30E−20**	1.80E−10	**3.30E−20**	**3.30E−20**	**3.30E−20**	**3.30E−20**
F_9	7.00E−18	7.00E−18	7.00E−18	7.00E−18	7.00E−18	7.00E−18	7.00E−18	7.00E−18	7.00E−18
F_{10}	1.10E−10	1.10E−17	**7.00E−18**	**7.00E−18**	**7.00E−18**	**7.00E−18**	4.70E−17	0.99	**7.00E−18**
F_{11}	7.00E−18	7.00E−18	7.00E−18	7.00E−18	7.00E−18	7.00E−18	7.00E−18	7.00E−18	7.00E−18
F_{12}	**3.30E−20**	7.00E−18	**3.30E−20**	**3.30E−20**	**3.30E−20**	**3.30E−20**	7.00E−18	**3.30E−20**	6.10E−18
F_{13}	7.00E−18	7.00E−18	7.00E−18	7.00E−18	7.00E−18	7.00E−18	7.00E−18	7.00E−18	7.00E−18
F_{14}	7.00E−18	7.00E−18	7.00E−18	7.00E−18	7.00E−18	7.00E−18	7.00E−18	7.00E−18	7.00E−18
F_{15}	4.70E−13	**1.70E−14**	4.90E−13	4.90E−13	4.90E−13	0.02	4.90E−13	4.80E−13	4.90E−13
F_{16}	**1.20E−19**	1.50E−16	**1.20E−19**	**1.20E−19**	0.8	3.70E−17	1.30E−16	**1.20E−19**	1.20E−16
F_{17}	1.00E−17	3.80E−13	**3.30E−20**	**3.30E−20**	1.10E−07	7.00E−18	7.00E−18	7.00E−18	7.00E−18
F_{18}	7.00E−18	7.00E−18	7.00E−18	7.00E−18	7.00E−18	7.00E−18	7.00E−18	7.00E−18	7.00E−18
F_{19}	6.90E−18	0.19	7.00E−18	**3.30E−20**	6.80E−18	**3.30E−20**	**3.30E−20**	**3.30E−20**	**3.30E−20**
F_{20}	7.00E−18	0.28	7.00E−18	7.00E−18	7.00E−18	6.60E−18	7.00E−18	7.00E−18	7.00E−18

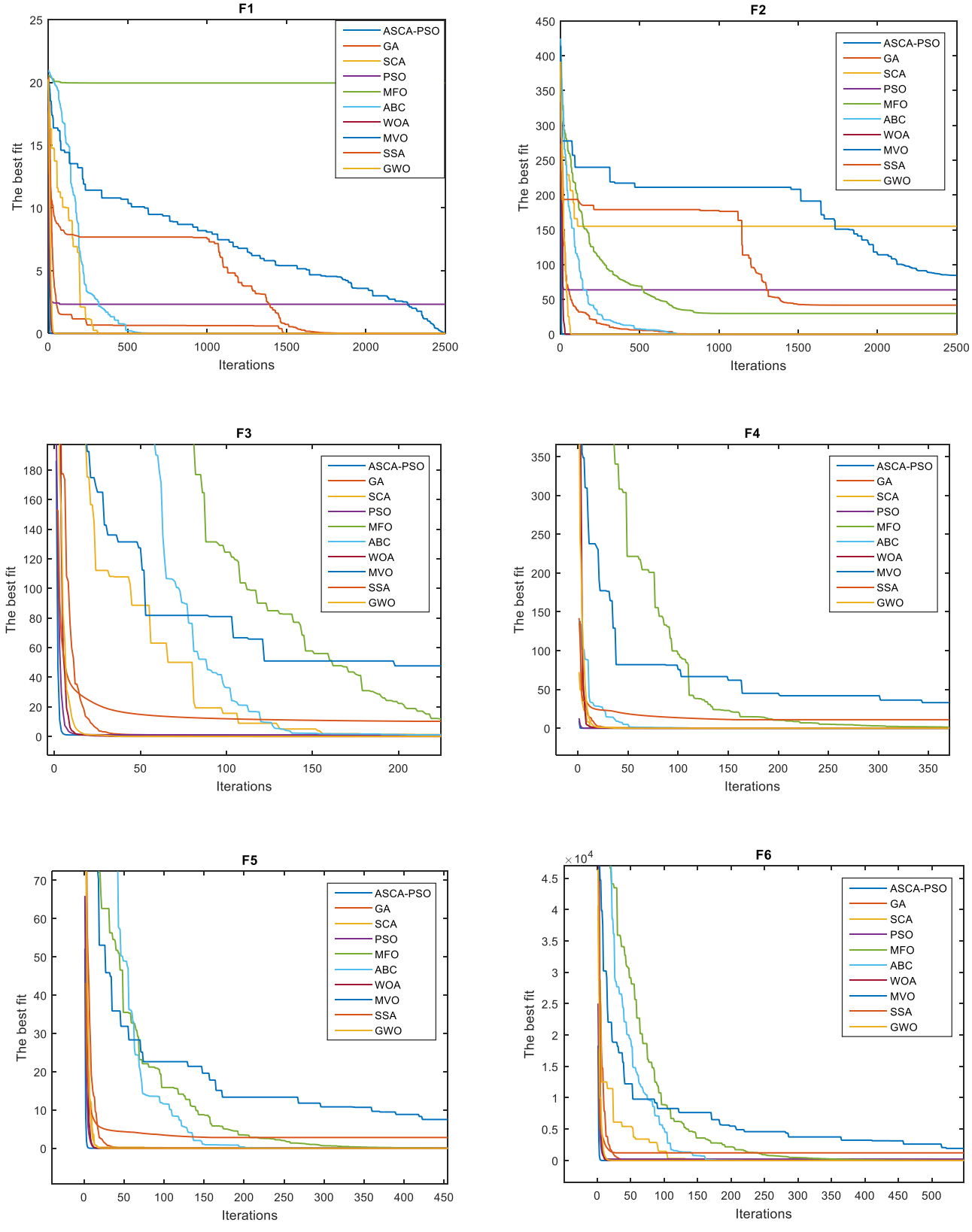

Fig. 2. Convergence curves for fitness function from F_1 to F_6.

In Table 6, values less than 0.05 demonstrate that the methods are substantially different and provide results that differ. Values higher than 0.05 provide evidence that the methods are similar in this specific implementation.

Figs. 2–5 show a comparison between ASCA-PSO and all the other algorithms for the convergence rate for the fitness versus the iterations for all of the functions. These figures show that ASCA-

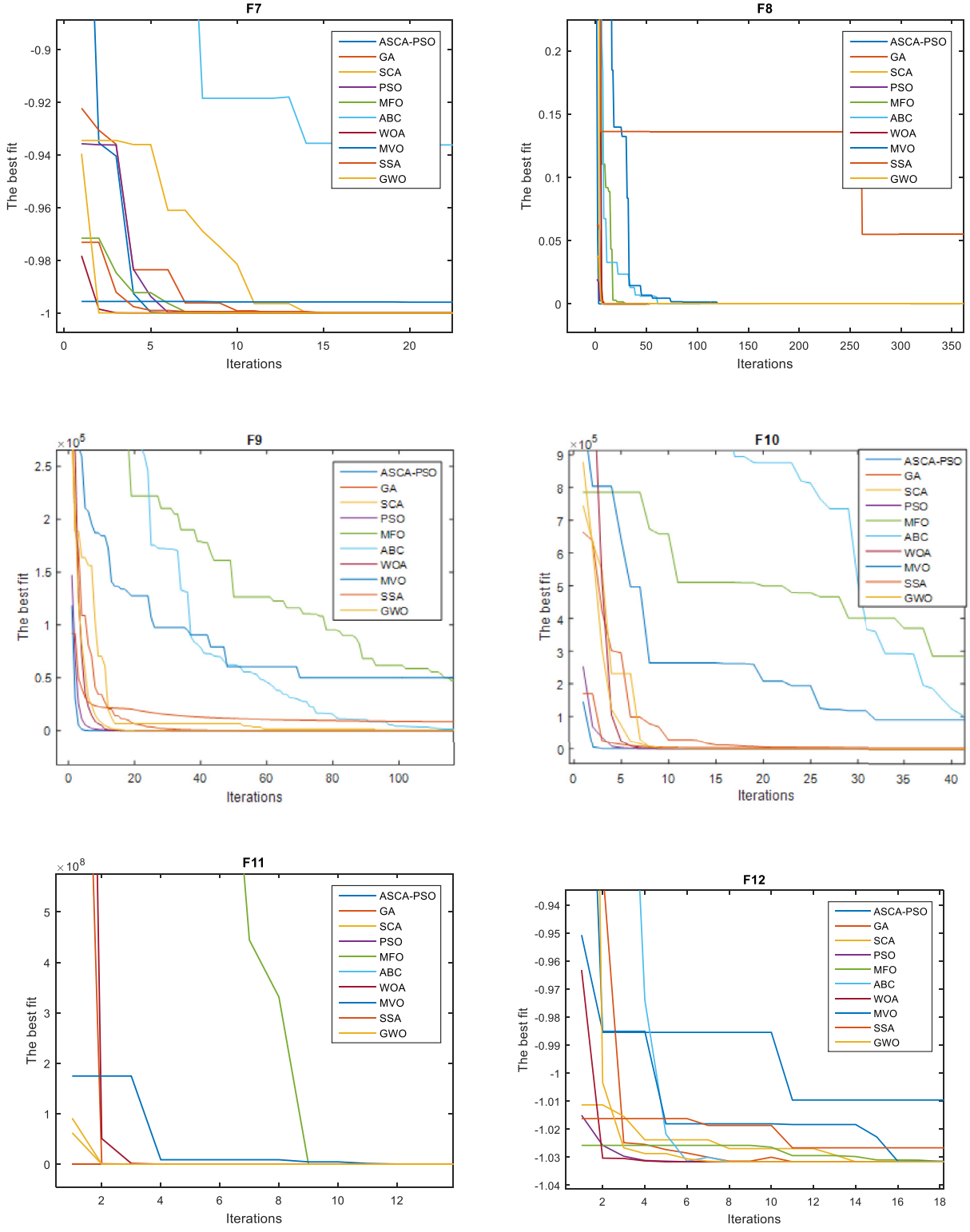

Fig. 3. Convergence curves for fitness function from F_7 to F_{12}.

PSO outperforms all the other algorithms in terms of the convergence speed with an accurate solution. In addition, Tables 3–6 show that ASCA-PSO provides an efficient strategy for finding the optimal global solution of an optimization problem with a fast convergence rate.

5. Case study: biological pairwise local sequence alignment algorithm

Alignment is one of the main processes in bioinformatics and is used to measure the similarity between biological sequences to provide indications about evolutionary and functional relations among RNA, DNA and protein sequences (Berger & Rozener, 1998;

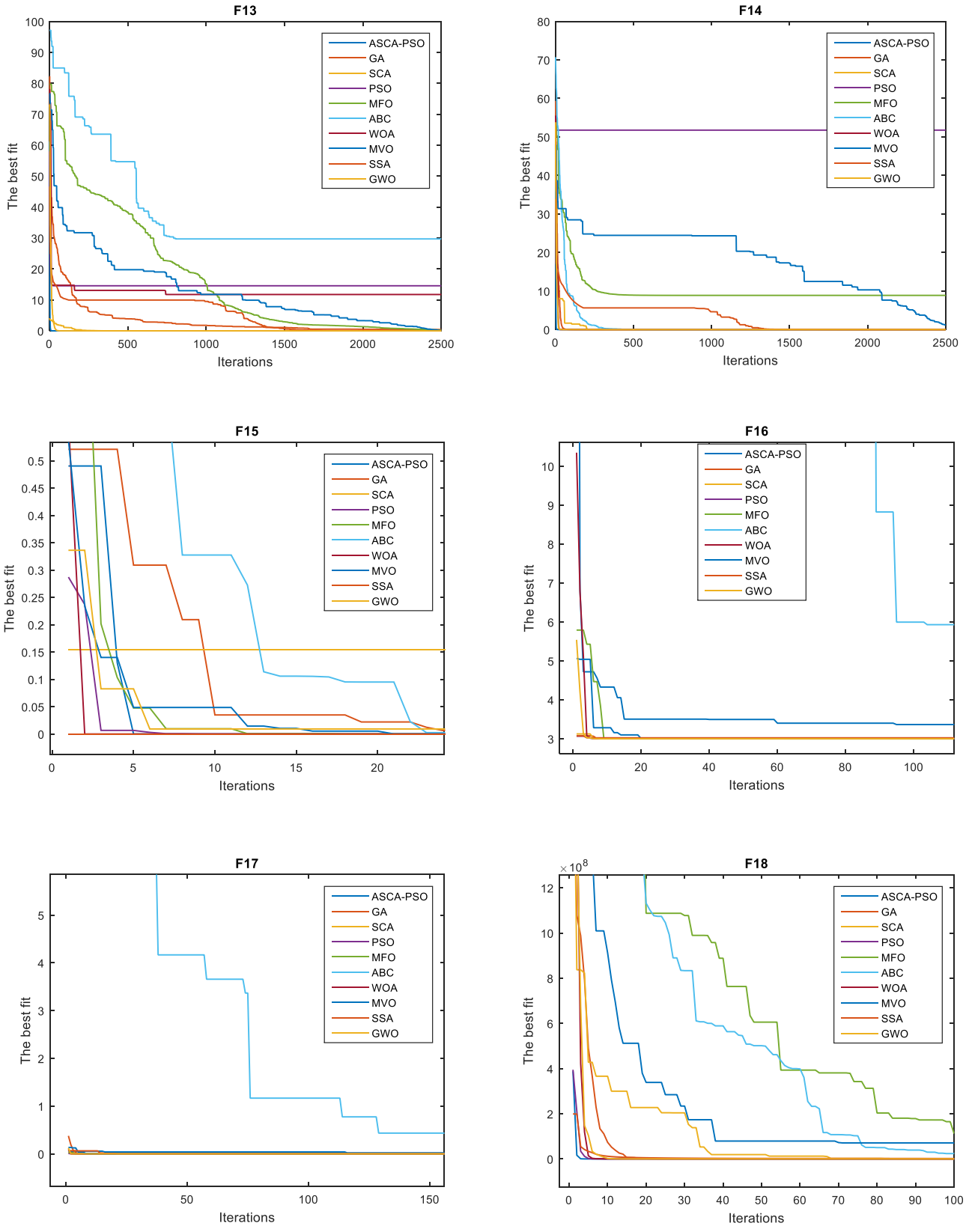

Fig. 4. Convergence curves for fitness function from F_{13} to F_{18}.

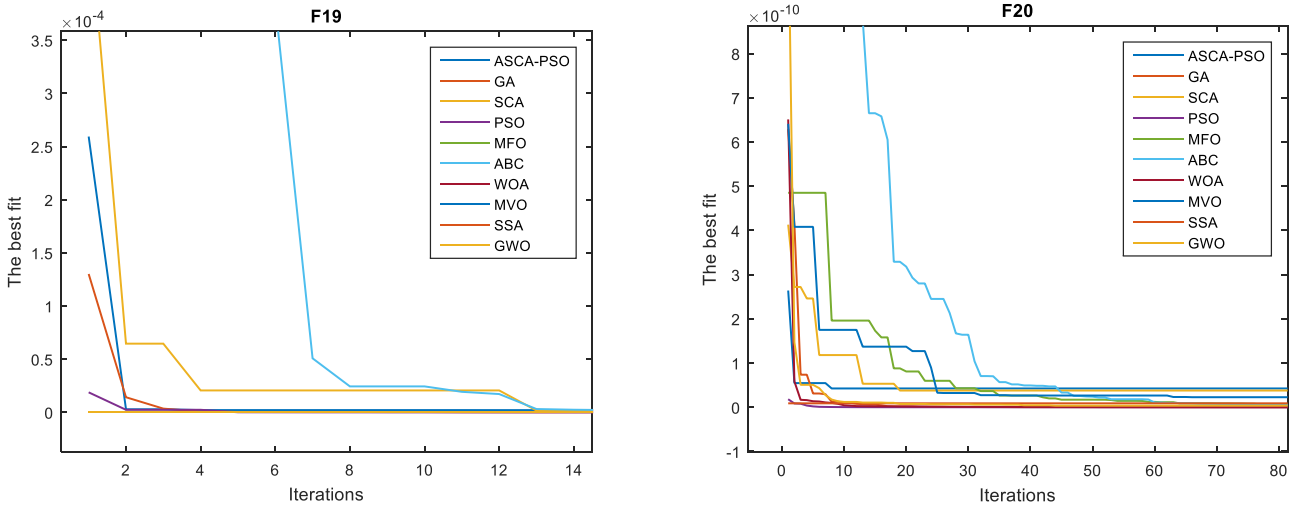

Fig. 5. Convergence curves for fitness function from F_{19} to F_{20}.

Cohen, 2004). Sequence alignment compares sequences by matching their bases (amino acids for proteins and nucleotides for DNA) to produce the best alignment that provides a maximum score representing the degree of similarity. Many biological operations are based mainly on sequence alignment processes, such as phylogenetic tree constructions (Feng & Doolittle, 1990), where the alignment is used to evaluate the similarities of the sequences to construct the evolutionary trees containing the sequences. Protein secondary structure prediction and analysis use the alignments to improve the prediction quality (Di Francesco, Garnier, & Munson, 1996). Additionally, DNA fragment assembly uses sequence alignment to aid the arrangement of the short fragmented sequences to construct the original DNA sequence (Li & Khuri, 2004).

Sequence alignment is classified into pairwise alignment, which is used to align two sequences, and multiple alignment, which aligns more than two sequences (Xiong, 2006). There are two types of pairwise alignments: global and local alignment. Global alignment finds the alignment over the entire length of the sequences by aligning the matching regions of the two sequences. Local alignment aims to find the longest consecutive substrings between two sequences.

There is a difference between the longest consecutive substrings and the longest common subsequences. The longest consecutive substrings are the consecutive substrings that are common between two sequences (strings) but with the condition that the substring is one segment whose bases are in consecutive order (Gusfield, 1997). However, the longest common subsequences are the common subsequences between two sequences but are not in consecutive order (Maier, 1978).

Pairwise local alignment is computed using the Smith-Waterman (SW) alignment algorithm (Smith & Waterman, 1981), which is based mainly on a dynamic programming approach (Cormen, 2009), and thus, it produces the most accurate alignment results.

For computing an SW alignment, a matrix with size $(m+1)$ and $(n+1)$ for the row and column, respectively, is constructed, where m and n represent the lengths of the two sequences. Each cell saves a score of an alignment from all possible alignments. The alignment score is computed using Eq. (10) based on the previous cells on the same row and column and the diagonal cell.

$Score(i, j)$

$$= \max \begin{Bmatrix} Score(i-1, j-1) + Similarity\ (Seq_A\ (i),\ Seq_B(j)) \\ max_{k=1:i-1}\ (Score(i,k) + g_0 + k\, g_e) \\ max_{k=1:i-1}\ (Score(k,j) + g_0 + k\, g_e) \\ 0 \end{Bmatrix} \quad (10)$$

where Seq_A and Seq_B are the sequences to be aligned with lengths m and n, respectively, and i and j denote the row and column indices, respectively, with $1 < i < m$ and $1 < j < n$. Here, g_o and g_e are the open gap and extended gap penalties, where the insertion of gaps (k is the number of inserted gaps) represents the possibility of aligning the bases of sequences with nothing when shifting bases for matching. A linear gap penalty $(g_0 + k\, g_e)$ is chosen for penalizing sequential gaps instead of separated gaps to maximize the overall alignment score (Gotoh, 1982). In some cases, common consecutive substrings may require gaps '–' to be inserted within the substrings to match as many bases as possible in order to increase the lengths of the substrings, as shown in Fig. 6, in which the three consecutive gaps enable the longest substrings to be identified. *Similarity* () is a function that computes the similarity between two bases according to the scoring scheme used based on the matching or un-matching of bases. There are different schemes for measuring similarity, and for proteins, there are two common scoring schemes: (1) BLOcked SUbstitution Matrix (BLOSUM) (Mount, 2008) and (2) Point Accepted Mutation (PAM) (Henikoff & Henikoff, 1992). For DNA, a positive score is assigned for matching nucleotides, and for un-matching nucleotides, a negative score is assigned.

SW alignment produces the most accurate longest consecutive substrings between two biological sequences, but it requires a very long execution time, where the time complexity of SW alignment is $O(n^3)$ and the space complexity is $O(n^2)$, in which n is the length of the two sequences to be aligned.

Hence, this paper attempts to reduce the execution time of SW alignment to provide results that give a primary indication of common substrings between long sequences. Then, SW alignment can be used to re-align the sequences if the primary results are acceptable. The proposed approach is based on fragmentation of the sequences into short fragments, and thus, the execution time of the alignment is reduced due to aligning short fragments instead of the entire lengths of the sequences. The rule of using meta-heuristics algorithms is to update the positions of the fragments in the two sequences toward the position of a fragment that produces the best longest consecutive substrings found.

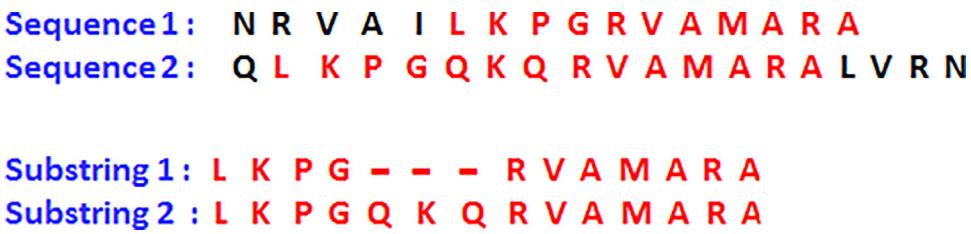

Fig. 6. Example of common longest consecutive substrings with gaps.

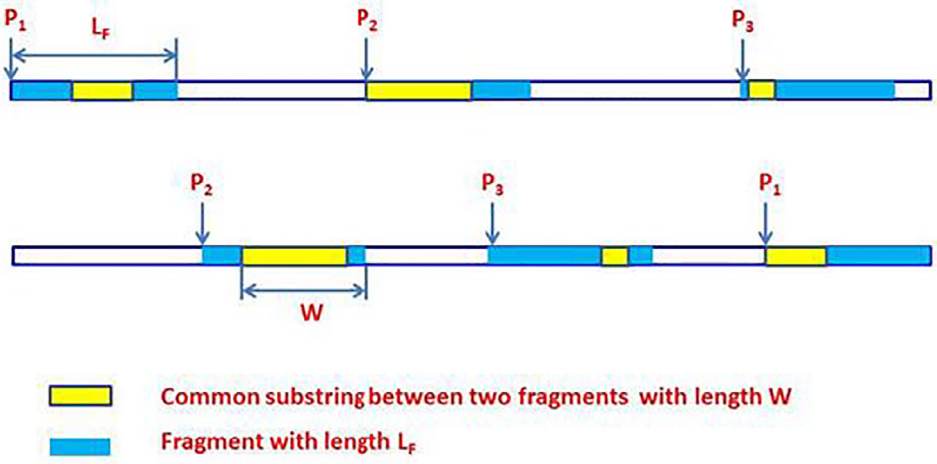

Fig. 7. Example of aligning fragments of sequences.

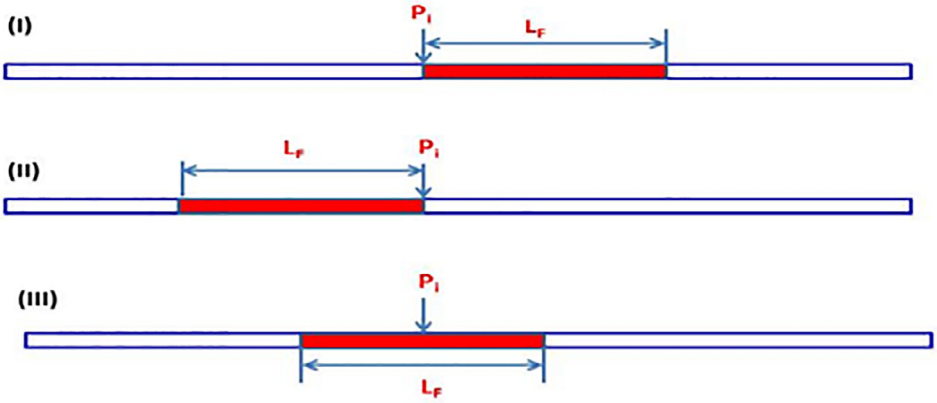

Fig. 8. Different ways to cut a fragment with length LF.

5.1. Formulating the problem as optimization problem

As noted above, the objective of the local sequence alignment algorithm is to find the longest consecutive substrings between two biological sequences. Thus, the fragmentation of sequences is used, in which short fragments of sequences are aligned instead of the entire sequence to overcome the problem of an extended execution time for the long sequences. The SW alignment algorithm aligns the short fragments to find the longest consecutive substrings between them. The sum of pair (SOP) objective function, as shown in Eq. (11), is used to evaluate the lengths of the consecutive substrings found.

$$Alignment_{Score} = \sum_{i=1}^{L} \left\{ \begin{array}{c} if\ A_i = B_i \quad penalize(+1)\ score \\ otherwise\ penalize\ zero \end{array} \right\} \tag{11}$$

where A and B are the aligned sequences.

However, there is a large number of trials for the fragmentation of sequences in their many positions. Thus, the idea of the proposed alignment scheme is to cut the number of fragments in each sequence at different positions and find the longest consecutive substrings between each pair. Then, we update the positions of the fragments to be cut in the subsequent iterations toward the positions of the fragments that obtained the maximum scores found (the longest consecutive substrings). The optimization search strategy of the meta-heuristic algorithms is used for updating the positions toward the fragments that obtained the best result.

Fig. 7 shows an example of cutting three fragments of the sequences at positions P_1, P_2 and P_3, where each position is represented by one search agent. The yellow part represents the common substring with length W between two fragments, and as shown, the P_2 cut fragments have the longer consecutive substrings than those of P_1 and P_3. Hence, the optimization search strategy must update positions P_1 and P_3 toward P_2 to find com-

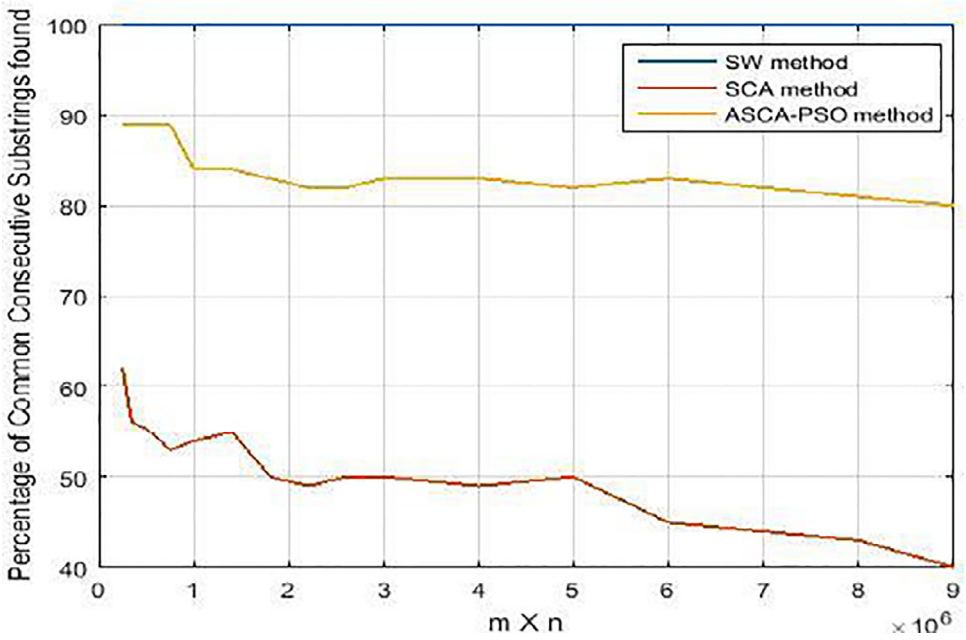

Fig. 9. Percentage of common consecutive substrings obtained using SW, SCA and ASCA-PSO methods.

Table 7
The parameters of ASCA-PSO local alignment technique.

Algorithm	Parameter	Value
SW alignment	Match	+1.0
	g_o	−1.0
	g_e	−0.5
SCA	a	2.0
ASCA-PSO	Maximum velocity (V_{max})	1.0
	Minimum velocity (V_{min})	−1.0
	Cognitive coefficient (C1)	0.5
	Cognitive coefficient (C2)	0.5
	a	2.0

mon substrings with a length better than that found by P_2. Hence, when repeating these steps with a large number of search agents, there is a high likelihood of finding the longest consecutive substrings or parts of them with a high percentage in less time than that consumed by the SW alignment.

In Fig. 7, the fragments are cut starting from position (P_i) to ($P_i + L_F$), but there are two other possibilities, as shown in Fig. 8:

- Cutting a fragment with length L_F starting from ($P_i - L_F$) to (P_i).
- Cutting a fragment with length L_F starting from ($P_i - (L_F/2)$) to ($P_i + (L_F/2)$).

In the experimental section, each method for cutting fragments (I, II and III) is studied separately to show the effect of each method on the results to choose the most suitable method of cutting.

5.2. The fragmentation local alignment proposed method

Step 1: Initialize N search agents (P_i, $i = 1, 2, 3, \ldots, N$) with random positions in the range from 1 to length ((Seq_A or Seq_B) – L_F), where L_F is the length of the fragments.

Step 2: Cut the fragments with length L_F in each sequence according to one of the three methods in Fig. 8.

Step 3: Each corresponding fragment determined by each search agent is aligned using the SW alignment algorithm, and the alignment score is computed using the objective function in Eq. (11).

Table 8
Comparison of different method for cut of fragments using ASCA-PSO.

$m \times n$	Percentage of longest consecutive substrings			
	I	II	III	Mix
250,000	56	89	84	95
350,000	56	73	84	95
550,000	67	84	84	95
750,000	56	78	73	95
1,000,000	62	89	95	89
1,400,000	73	84	89	89
1,800,000	62	73	84	84
2,200,000	56	78	89	84
2,600,000	62	73	84	84
3,000,000	67	73	89	84
4,000,000	62	78	84	84
5,000,000	56	78	78	84
6,000,000	62	67	73	84
7,000,000	50	67	73	84
8,000,000	56	78	78	84
9,000,000	56	67	84	84

Table 9
Wilcoxon rank sum test results between ASCA-PSO and SCA vs. product of sequences' lengths.

$m \times n$	P-value
250,000	3.29E−09
350,000	5.87E−09
550,000	7.66E−08
750,000	5.39E−10
1,000,000	1.45E−08
1,400,000	6.25E−09
1,800,000	6.01E−13
2,200,000	7.46E−08
2,600,000	3.60E−11
3,000,000	1.70E−10
4,000,000	4.68E−11
5,000,000	6.55E−12
6,000,000	2.31E−12
7,000,000	6.69E−09
8,000,000	5.17E−12
9,000,000	1.33E−07

Table 10
Comparison of SW alignment method with fragmentation alignment using ASCA-PSO and SCA on real protein sequences.

Protein ID (length)		Method	Score	Longest Consecutive Substrings Obtained
1	Q9NP78 (766) O95342 (1321)	SW	42	GEKGAQLSGGQKQRVAMARALVRNPPVLILDEATSALDAESE
		ASCA-PSO	18	SGGQKQRVAIARALIRNP
		SCA	13	LDEATSALDAESE
2	Q8K442 (1620) P41233 (2160)	SW	29	MDEADILADRKVFISKGKLKCAGSSLFLK
		ASCA-PSO	27	EADILADRKVFISKGKLKCAGSSLFLK
		SCA	13	QITAILGHSGAGK
3	Q8K441(1624) Q91V24(2159)	SW	35	ALKGLFLDIYESQITAILGHSGAGKSSLLNILSGL
		ASCA-PSO	21	TAILGHSGAGKSSLLNILSGL
		SCA	18	AIMVSGRLRCIGPIQHLK
4	Q5TCY1 (1141) Q96AE7(1082)	SW	34	AVGVRGRYELPPCSGPGWLLSLSALLSVAARGAF
		ASCA-PSO	26	YELPPCSGPGWLLSLSALLSVAARGA
		SCA	5	QARPG
5	A0A0A6YYL4 (2161) Q5VTQ0 (3423)	SW	65	GKIQQQVDSPMNLKHPHDLVILMRQEATVNYLKELEKQLVAQKIHIEENEDRDTGLEQRHNKEDP
		ASCA-PSO	29	GKIQQQVDSPMNLKHPHDLVILMRQEATV
		SCA	11	LVILMRQEATVNY
6	Q7TQI7 (1024) E9Q4Z2 (2734)	SW	25	CRSLSLSSSKSNSQALNSSAQQHRG
		ASCA-PSO	18	SSKSNSQALNSSAQQHRG
		SCA	8	SAGDGLRQ
7	E9PX95 (5200) P70170 (1546)	SW	51	PHVFLLFITFPILFIGWGSQSSKVQIHHNTWLHFPGHNLRWILTFALLFVH
		ASCA-PSO	16	PHVFLLFITFPILFIG
		SCA	9	LITLLEMLM
8	Q8BGM7 (1620) Q8R0P4 (2287)	SW	26	LKRDTFLEFLYTALILLSLILFLQLH
		ASCA-PSO	12	FLEFLYTALILL
		SCA	11	MDEADIL
9	Q01337 (2524) Q9Z0F8 (2616)	SW	24	PRGGPEHRAAWGEADSRANGYPHA
		ASCA-PSO	12	GPEHRAAWGEAD
		SCA	6	PRGGPE
10	Q7TQI7 (1024) E9Q4Z2 (2472)	SW	26	SGYGAGDSCRSLSLSSSKSNSQALN
		ASCA-PSO	14	SGYGAGDSCRSLS
		SCA	6	YLDIGA
11	Q61618 (2080) Q99LJ2 (2477)	SW	32	SCQRPPRNPLSSNDTWPSPELQTNWTAAPGPE
		ASCA-PSO	16	SCQRPPRNPLSSNDTW
		SCA	4	ANDL

Step 4: Update the global optimum search agents (P^{gbest}) to determine the positions of the fragments in the two sequences that contain the longest consecutive substrings found.

Step 5: Each search agent updates its position toward P^{gbest} based on the optimization updating strategy of the used meta-heuristic technique.

Step 6: Repeat steps 2–5 for a number of iterations (T).

The time complexity of the fragmentation pairwise local alignment is (TNL_F^2), where T, N and L_F are the number of iterations, number of particles and width of the fragments, respectively. The proposed technique has a time complexity that is lower than that of the SW alignment algorithm, which is (n^3), where n is the length of the two sequences. Thus, the fragmentation of the sequences guarantees executing the local alignment in a shorter amount of time than that required for the SW alignment, and the role of using meta-heuristics is only to keep the search process moving toward the region that contains the alignment with the maximum score, i.e., the longest consecutive substrings between two sequences.

5.3. The experimental results

The proposed fragmentation local alignment method was implemented using ASCA-PSO, and the SCA and was tested on biological protein sequences from the Swiss-Prot database (UniProt, 2017) with various lengths (the product of the lengths of the sequences up to 9000,000). The objective of the test was to measure the percentage of the length of the longest consecutive substrings found by ASCA-PSO and the SCA compared with that found by the SW alignment algorithm.

Table 7 shows the parameter settings that were used in the test with 50 independent runs, and each method of cutting the fragments (I, II and III) was tested separately. In addition, the fourth

Table 11
Execution time comparison of ASCA-PSO vs. SCA and SW alignment methods.

$m \times n$	Time (Sec)			Search Agents	
	SW	SCA	ASCA-PSO	SCA	ASCA-PSO
250,000	11	4.70	6.04	40	50
350,000	18	4.70	6.04	40	50
550,000	37	12.0	12.7	100	110
750,000	57	14.3	14.9	120	130
1,000,000	138	18.8	18.9	150	160
1,400,000	208	21.7	21.7	180	190
1,800,000	288	25.0	26.0	200	210
2,200,000	320	29.0	30.2	240	250
2,600,000	354	49.2	51.2	400	410
3,000,000	428	49.2	51.2	400	410
4,000,000	680	53.9	55.9	450	460
5,000,000	922	53.9	55.9	450	460
6,000,000	1275	53.9	55.9	450	460
7,000,000	1670	67.0	69.8	550	560
8,000,000	2134	78.9	83.7	650	660
9,000,000	2750	78.9	83.7	650	660

cutting of fragments method was applied, which is a combination of the three methods, in which at each cutting of fragments, one of the three methods was applied and was chosen at random. Table 8 shows the percentage of the longest consecutive substrings found by ASCA-PSO in comparison to that found by the SW alignment algorithm, and as shown, cutting methods II and III produced better results than method I, but using a combination of the three methods produced the best results.

Fig. 9 shows the comparison among the ASCA-PSO, SCA and SW alignment methods. The fragment-cutting method used was the combination of the three methods (I, II and III) due to its significant impact on the results. As shown in the figure, ASCA-PSO enhanced the performance of the SCA when finding the common longest consecutive substrings between two sequences with lengths m and n with a product of up to 9000,000. Table 9 presents the p values obtained after applying the Wilcoxon rank test over the results of SCA and ASCA-PSO. A set of 50 independent experiments was performed by each algorithm. In this context, the ASCA-PSO vs. SCA comparison using the Wilcoxon ranks test provides evidence that the results obtained by the two algorithms are sufficiently different.

In addition, the proposed fragmentation SW alignment method implemented by ASCA-PSO and SCA was compared with the standard SW local alignment algorithm to find the common longest consecutive substrings between real protein sequences gathered from the mouse protein database (UniProt, 2017). Table 10 shows the comparison, in which the second column contains the protein ID for each sequence accompanied with the length of the sequence, and the results from each method are shown in the last column accompanied by the score. The score represents the length of the common substrings (matched bases).

As shown in the table, the SW alignment method finds the longest consecutive substrings between each pair of proteins, while SCA fragmentation alignment can find part of the longest consecutive substrings in cases 1, 5 and 9 but with a shorter length than that found by ASCA-PSO. In other cases, the SCA can find consecutive substrings but not the longest substrings; however, ASCA-PSO can find a longer segment of the longest consecutive substrings in all cases. Hence, this test proved successful for ASCA-PSO in enhancing the SCA for searching for the longest consecutive substrings based on a fragmentation local alignment method.

Table 11 lists the execution time of the fragmentation alignment methods using ASCA-PSO and the SCA in comparison with the SW alignment algorithm according to the number of search agents of the SCA and ASCA-PSO. For sequences with product lengths less than 250,000, the SW alignment method produced an accurate alignment in a smaller execution time than the proposed fragmentation local alignment method by the SCA or ASCA-PSO. Otherwise, ASCA-PSO sped up the alignment process for finding the longest consecutive substrings with reasonable results in comparison with the SW alignment method.

As listed in the table, the size of search agents used increased in proportion to the increase in the length of the sequences in order to cover the search space (length of sequences). However, the SCA alignment method was slightly faster than the ASCA-PSO alignment method, but the latter doubled the performance of the SCA alignment process.

6. Conclusions

In this paper, the SCA has the advantage of powerful exploration but poor exploitation. Thus, it was necessary to enhance the SCA by merging it with PSO, which has the advantage of powerful exploitation. The resulting hybrid technique was built in two layers, where the bottom layer explores the search space based on the search agents of SCA, while the top layer exploits the region around the best solution found by the bottom layer. Hence, the proposed technique is balanced between exploration and exploitation, which improves the quality of the solution while maintaining fast convergence. The enhancement of the SCA using ASCA-PSO was tested on finding the optimal solution for standard benchmark mathematical functions, and the results proved the enhancement of the quality of the solution and the convergence rate of the SCA. In addition, the problem of finding the longest consecutive substrings between two biological sequences was used as a case study for testing the proposed approach, the ASCA-PSO, compared with the SCA. This problem was formulated as an optimization problem, and ASCA-PSO and the SCA were used to find the longest common consecutive substrings in comparison with an accurate solution obtained by the SW alignment method. ASCA-PSO was successful for twice the percentage of common consecutive substrings obtained by the SCA, corresponding to those obtained by the SW alignment method, when testing on biological protein sequences. Hence, the results of the proposed enhancement approach (ASCA-PSO) in this work support its use for optimizing other engineering problems.

References

Abd ElAziz, M., Xiong, S., & Selim, I. M. (2017). Automatic detection of galaxy type from datasets of galaxies image based on image retrieval approach. *Science Reports, 7*, 4463.

Berger, B., & Rozener, M. (1998). Introduction to computational molecular biology. MIT Computer Science & Artificial Intelligence, 1998.

Boussaïd, I., Lepagnot, J., & Siarry, P. (2013). A survey on optimization metaheuristics. *Information Sciences, 237*, 82–117.

Calvini, M., Carpita, M., Formentini, A., & Marchesoni, M. (2015). PSO-based self-commissioning of electrical motor drives. *IEEE Transactions on Industrial Electronics, 62*(2), 768–776.

Cao, Y., Li, W., & Chaovalitwongse, W. A. (2017, July). Hybrid comprehensive learning particle swarm optimizer with adaptive starting local search. In *International conference in swarm intelligence* (pp. 148–157). Springer.

Cohen, J. (2004). Bioinformatics—an introduction for computer scientists. *ACM Computing Surveys (CSUR), 36*(2), 122–158.

Cormen, T. H. (2009). *Introduction to algorithms*. MIT Press.

Di Francesco, V., Garnier, J., & Munson, P. J. (1996). Improving protein secondary structure prediction with aligned homologous sequences. *Protein Science, 5*(1), 106–113.

Feng, D. F., & Doolittle, R. F. (1990). Progressive alignment and phylogenetic tree construction of protein sequences. *Methods in Enzymology, 183*, 375–387.

Garg, H. (2016). A hybrid PSO-GA algorithm for constrained optimization problems. *Applied Mathematics and Computation, 274*, 292–305.

Gotoh, O. (1982). An improved algorithm for matching biological sequences. *Journal of Molecular Biology, 162*(3), 705–708.

Gusfield, D. (1997). *Algorithms on strings, trees and sequences: Computer science and computational biology*. Cambridge University Press.

Güçyetmez, M., & Çam, E. (2016). A new hybrid algorithm with genetic-teaching learning optimization (G-TLBO) technique for optimizing of power flow in wind-thermal power systems. *Electrical Engineering, 98*(2), 145–157.

Henikoff, S., & Henikoff, J. G. (1992). Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences, 89*(22), 10915–10919.
Higgins, J. J. (2003). Introduction to modern nonparametric statistics.
Holland, J. H. (1992). Genetic algorithms. *Scientific American, 267*(1), 66–73.
Jamil, M., & Yang, X. S. (2013). A literature survey of benchmark functions for global optimisation problems. *International Journal of Mathematical Modelling and Numerical Optimisation, 4*(2), 150–194.
Javidy, B., Hatamlou, A., & Mirjalili, S. (2015). Ions motion algorithm for solving optimization problems. *Applied Soft Computing, 32*, 72–79.
Karaboga, D., & Akay, B. (2009). A comparative study of artificial bee colony algorithm. *Applied Mathematics and Computation, 214*(1), 108–132.
Kennedy, E. R. J. (1995). Particle swarm optimization. *Neural networks 1995. Proceedings*.
Khanna, V., Das, B. K., Bisht, D., & Singh, P. K. (2015). A three diode model for industrial solar cells and estimation of solar cell parameters using PSO algorithm. *Renewable Energy, 78*, 105–113.
Kumar, N., Hussain, I., Singh, B., & Panigrahi, B. (2017). Single sensor based MPPT of partially shaded PV system for battery charging by using Cauchy and Gaussian sine cosine optimization. *IEEE Transactions on Energy Conversion, 32*(3), 983–992.
Li, L., & Khuri, S. (2004). A Comparison of DNA fragment assembly algorithms. In *METMBS: Vol. 4* (pp. 329–335).
Maier, D. (1978). The complexity of some problems on subsequences and supersequences. *Journal of the ACM (JACM), 25*(2), 322–336.
Mirjalili, S. (2015). Moth-flame optimization algorithm: A novel nature-inspired heuristic paradigm. *Knowledge-Based Systems, 89*, 228–249.
Mirjalili, S. (2016). SCA: A sine cosine algorithm for solving optimization problems. *Knowledge-Based Systems, 96*, 120–133.
Mirjalili, S., Gandomi, A. H., Mirjalili, S. Z., Saremi, S., Faris, H., & Mirjalili, S. M. (2017). Salp swarm algorithm: A bio-inspired optimizer for engineering design problems. *Advances in Engineering Software*.
Mirjalili, S., & Lewis, A. (2016). The whale optimization algorithm. *Advances in Engineering Software, 95*, 51–67.
Mirjalili, S. M., & Hatamlou, A. (2016). Multi-verse optimizer: A nature-inspired algorithm for global optimization. *Neural Computing and Applications, 27*, 495–513.
Mirjalili, S., Mirjalili, S. M., & Lewis, A. (2014). Grey wolf optimizer. *Advances in Engineering Software, 69*, 46–61.
Mount, D. W. (2008). Comparison of the PAM and BLOSUM amino acid substitution matrices. *Cold Spring Harbor Protocols, 2008*(6) pdb-ip59.
Mudhsh, M., Xiong, S., El Aziz, M. A., Hassanien, A. E., & Duan, P. (2017). Hybrid swarm optimization for document image binarization based on Otsu function. *CASA2017*.
Petrović, M., Mitić, M., Vuković, N., & Miljković, Z. (2016). Chaotic particle swarm optimization algorithm for flexible process planning. *The International Journal of Advanced Manufacturing Technology, 85*(9-12), 2535–2555.
Premalatha, K., & Natarajan, A. M. (2008). A new approach for data clustering based on PSO with local search. *Computer and Information Science, 1*(4), 139.
Rao, R. V., Savsani, V. J., & Vakharia, D. P. (2011). Teaching–learning-based optimization: A novel method for constrained mechanical design optimization problems. *Computer-Aided Design, 43*(3), 303–315.
Rashedi, E., Nezamabadi-Pour, H., & Saryazdi, S. (2009). GSA: A gravitational search algorithm. *Information sciences, 179*(13), 2232–2248.
Sadollah, A., Bahreininejad, A., Eskandar, H., & Hamdi, M. (2013). Mine blast algorithm: A new population based algorithm for solving constrained engineering optimization problems. *Applied Soft Computing, 13*(5), 2592–2612.
Santra, D., Mukherjee, A., Sarker, K., & Chatterjee, D. (2016, October). Hybrid PSO-ACO algorithm to solve economic load dispatch problem with transmission loss for small scale power system. In *International conference on intelligent control power and instrumentation (ICICPI)* (pp. 226–230). IEEE.
Smith, T. F., & Waterman, M. S. (1981). Identification of common molecular subsequences. *Journal of Molecular Biology, 147*(1), 195–197.
Storn, R., & Price, K. (1997). Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization, 11*(4), 341–359.
Talbi, E. G. (2009). *Metaheuristics: from design to implementation*: Vol. 74. John Wiley & Sons.
Tuvayanond, W., & Parnichkun, M. (2017). Position control of a pneumatic surgical robot using PSO based 2-DOF H∞ loop shaping structured controller. *Mechatronics, 43*, 40–55.
UniProt: the universal protein knowledge base (2017). *Nucleic Acids Res., 45*, D158–D169.
Wang, G. G., Deb, S., Gandomi, A. H., & Alavi, A. H. (2016). Opposition-based krill herd algorithm with Cauchy mutation and position clamping. *Neurocomputing, 177*, 147–157.
Wang, G. G., Guo, L., Gandomi, A. H., Hao, G. S., & Wang, H. (2014). Chaotic krill herd algorithm. *Information Sciences, 274*, 17–34.
Wang, G., Guo, L., Duan, H., Liu, L., & Wang, H. (2012). A bat algorithm with mutation for UCAV path planning. *The Scientific World Journal, 2012*, 1–15.
Wolpert, D. H., & Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation, 1*(1), 67–82.
Xiong, J. (2006). *Essential bioinformatics*. Cambridge University Press.
Yang, S., Wang, W., Lin, Q., & Chen, J. (2016, December). A Novel PSO-DE Co-evolutionary Algorithm Based on Decomposition Framework. In *International conference on smart computing and communication* (pp. 381–389). Springer, Cham.