Hybrid Moth-Flame and Salp-Swarm Optimization Algorithm

Orachun Udomkasemsub, Khajonpong Akkarajitsakul, and Tiranee Achalakul

***Abstract*—In this paper, the hybrid moth-flame, and salp-swarm algorithm is proposed to enhance the performance of the original moth-flame algorithm. As a moth moves spirally around a flame, the chance of investigating the distance between the moth and the flame is reduced. In this study, a salp chain was attached to each moth to investigate the distance. The proposed algorithm was evaluated on benchmark functions and compared with the original moth-flame optimization algorithm and particle swarm optimization algorithm. Results show that the proposed algorithm can generate a better minimum fitness value up to almost 100% for most functions compared with other algorithms. Moreover, the convergence rate of the proposed algorithms converged to a global optimum faster for most functions compared with other algorithms.**

***Index Terms*—Moth-flame optimization, salp-swarm algorithm, hybrid moth-flame, and salp-swarm algorithm.**

I. Introduction

Optimization is a process of identifying values of variables or configurations to support objectives. Objectives can be to minimize the time of production in manufacturing or the cost of server allocation. Meta-heuristic techniques are well adapted to these optimization problems because of the performance of techniques and flexibility to solve a wide area of problems.

Swarm intelligence (SI) [1] is one of the categories of optimization techniques. The basic concept of SI is to have collective simple search agents follow basic rules to emerge intelligent swarm capabilities. They are inspired by animals. For example, particle swarm optimization (PSO) [2] was inspired by a flock of birds that collaboratively fly to a food source without crashing one another, and ant colony optimization (ACO) [3] was inspired by an ant colony that collaboratively find a food location without any leader.

The common process in SI is to have a set of feasible solutions improved over time using the knowledge of other solutions. The process should support exploration and exploitation to get deep into a global optimum without being trapped by any local optima.

The moth-flame optimization (MFO) [4] and salp-swarm algorithm (SSA) [5] are other two remarkable meta-heuristic optimization algorithms published recently. The researchers showed that the performance of these algorithms can outperform other algorithms in optimization problems. The motivation of this research is to enhance the MFO algorithm with SSA to get a better quality of solutions and faster convergence. The rest of this paper is organized as follows.

Section II presents the adoption of SSA in the MFO algorithm. Section III explains an experiment for the performance evaluation using a variety of benchmark functions. The results of the experiment and discussion are also discussed in this section. Finally, Section IV summarizes the work done in this paper, remarks, and future research.

Manuscript received June 29, 2019; revised May 1, 2019. This work was supported by the Thailand Research Fund through the Royal Golden Jubilee Ph.D. Program (Grant No. PHD/0065/2554).

The authors are with the Computer Engineering Department, King Mongkut's University of Technology Thonburi, Bangkok, 10140, Thailand (e-mail: {orachun.udo, khajonpong.akk, tiranee.ach}@mail.kmutt.ac.th).

II. Methodology

This section describes the processes of MFO, SSA, and the proposed hybrid between both.

A. MFO Algorithm

The algorithm uses the position of each moth as a solution representation and a matrix of flames to represent the best solutions found so far. The dimensions of a position represent each variable in an optimization problem.

At the beginning of the algorithm, a population of moths is assigned to a random position based on the upper and lower bounds. Their fitness value is calculated, and the flame matrix is updated.

At each iteration, each moth will update its position with respect to the corresponding flame. The spiral function used to update the moth position is (1), where

M_i is the i^{th} moth,
F_j is the j^{th} flame,
$D_{i,j}$ is the distance between the i^{th} moth and j^{th} flame,
b is a constant to define the shape of the spiral function,
t is a random number between r and 1,
and r is a number linearly decreasing from −1 to −2.

$$S(M_i, F_j) = D_{i,j} \cos(2\pi t) + F_j \quad t \in [r, 1). \tag{1}$$

The decrement value of r increases a chance of exploitation after iterations.

After moths are updated, their fitness value is evaluated again for the updated position. Then, the best solutions are updated to the flame matrix.

This algorithm also decreases the number of flames over iterations to ensure convergence. The number of flames in each iteration can be calculated using (2), where

N is the maximum number of flames,
l is the current iteration number,
and T is the maximum number of iterations.

$$flames(N, l, T) = \text{round}(N - l(N-1)/T). \tag{2}$$

DOI: 10.7763/IJMO.2019.V9.713

To summarize, the algorithm can be represented as follows:

MFO (*F*, *G*, *H*)

flameCount ← *N* ← max number of flames
for *i* ←1 **to** *n*
  for *j* ← 1 **to** *d*
  $Moths_{ij}$ ← random between lb_j and ub_j
  $Fits_i$ ← F(M_i, *G*, *H*)
Flames ← sort(*Moths*, *Fits*, *flameCount*)
for *l* ← 1 **to** *T*
  flameCount ← flames(*N*, *l*, *T*) from (2)
  for *i* ← 1 **to** *flameCount*
  for *j* ← 1 **to** *d*
  $Moths_{ij}$ ← S($Moths_{ij}$, F_{ij}) from (1)
  $Fits_i$ ← F($Moths_i$, *G*, *H*)
  Flames ← sort(*Moths*, *Fits*, *flameCount*)
return $Flames_0$

B. SSA

This algorithm represents a solution through a salp position where its dimensions are the problem variable, similar with MFO.

At the beginning of the algorithm, each salp in the population is assigned to a chain, and assigned to a random position based on upper and lower bounds. Their fitness value is calculated, and the best solution is set to the food source.

During the iterations, leaders' position will be updated to move toward the food source using (3), where

$x_{1,j}$ is the position of the leader salp at the j^{th} dimension,
F_j is the position of the food source at the j^{th} dimension,
ub_j is an upper bound of the variable in the j^{th} dimension,
lb_j is a lower bound of the variable in the j^{th} dimension,
and c_1, c_2, and c_3 are random numbers.

$$x_{1,j} = \begin{cases} F_j + c_1((ub_j - lb_j)c_2 + lb_j) & c_3 \geq 0 \\ F_j - c_1((ub_j - lb_j)c_2 + lb_j) & c_3 < 0 \end{cases}. \quad (3)$$

While the leaders chase the food source, the followers move toward the salp in front of them, which will, directly, and indirectly, follow the leader. The followers' position will be updated by (4), where

$x'_{i,j}$ is the new position of the i^{th} salp at the j^{th} dimension,
and $x_{i,j}$ is the previous position of the i^{th} salp at the j^{th} dimension.

$$x'_{i,j} = (x_{i.j} + x_{i-1,j}) / 2. \quad (4)$$

The process of this algorithm can be summarized as follows:

SALP-SWARM OPTIMIZATION (*F*, *G*, *H*)

cn ← a number of chains
cl ← chain length
for *i* ←1 **to** *cn*
  for *j* ←1 **to** *cl*
  for *k* ← 1 **to** *d*
  $Salps_{ijk}$ ← random between lb_j and ub_j
  $Fits_{ij}$ ← F($Salps_{ij}$, *G*, *H*)
foodSource ← $Salps_{ij}$ with minimum $Fits_{ij}$
for *l* ← 1 **to** *N*
  for *i* ←1 **to** *cn*
  for *j* ←1 **to** *cl*
  for *k* ← 1 **to** *d*
  if j = 1 **then**
  $Salps_{ijk}$ ← $x_{1,j}$ from (3)
  else
  $Salps_{ijk}$ ← $x'_{i,j}$ from (4)
  $Fits_{ij}$ ← F($Salps_{ij}$, *G*, *H*)
  foodSource ← $Salps_{ij}$ with minimum Fit_{ij}
return *foodSource*

C. Proposed Hybrid Moth-Flame Salp-Swarm (MFSS) Algorithm

This paper proposed a method to increase the exploration and exploitation balance to the MFO by appending each moth a chain of follower salps. In addition, the salp will explore, and exploit the distance between the moth and the flame, while the moth only examines the surrounding of the flame.

The proposed algorithm can be illustrated as follows.

HYBRID MFSS ALGORITHM (*F*, *G*, *H*)

flameCount ← *N* ← max number of flames
for *i* ←1 **to** *n*
  for *j* ← 1 **to** *d*
  $Moths_{ij}$ ← random between lb_j and ub_j
  $MothsFits_i$ ← F($Moths_i$, *G*, *H*)
  for *j* ←1 **to** *cl*
  for *k* ← 1 **to** *d*
  $Salps_{ijk}$ ← random between lb_j and ub_j
  $SalpsFits_{ij}$ ← F($Salps_{ij}$, *G*, *H*)
Flames ← sort(*Moths* + *Salps*, *MothsFits*+*SalpsFits*, *flameCount*)
for *l* ← 1 **to** *T*
  flameCount ← flames(*N*, *l*, *T*) from (2)
  for *i* ← 1 **to** *flameCount*
  for *j* ← 1 **to** *d*
  $Moths_{ij}$ ← S($Moths_{ij}$, F_{ij}) from (1)
  $MothsFits_i$ ← F($Moths_i$, *G*, *H*)
  for *j* ← 1 **to** *cl*
  for *k* ← 1 **to** *d*
  $Salps_{ijk}$ ← $x'_{i,j}$ from (4)
  $SalpsFits_{ij}$ ← F($Salps_{ij}$, *G*, *H*)
  Flames ← sort(*Moths* + *Salps*, *MothsFits*+*SalpsFits*, *flameCount*)
return $Flames_0$

This paper has a hypothesis that the hybrid MFSS algorithm can provide a better quality of solutions and faster convergence. To support this hypothesis, an experiment was designed, and executed. The details of the experiment, results, and discussion of the results will be discussed in the next section.

III. RESULTS AND DISCUSSION

To evaluate the performance of the hybrid MFSS algorithm, 12 benchmark functions for the optimization algorithm from [6] were used as objective functions. That is, the algorithm must find variable values to get the minimum function value. These functions can be divided into two groups: unimodal and multimodal.

The unimodal function contains only one global optimum without any local optima, whereas the multimodal function can have more than one local, and global optima. The list of benchmark functions is described in Appendix B. First, the unimodal functions were used to evaluate the exploitation

property of the algorithm. Second, the multimodal functions were used to evaluate the capability to avoid the local optima.

The experiment compared the results with those from the PSO algorithm, genetic algorithm (GA), MFO algorithm, and SSA. The parameter settings of each experiment treatment are described in Table I. The experiment also compared the different numbers of salps and moths in the algorithm settings. However, the total number of search agents was equal in all settings to make a fair comparison. In addition, the constant number of a spiral shape, *b*, was set to 0.1 for all runs. This number was experimentally selected. For the PSO algorithm, the number of particles, and value of each weight required in the velocity updating function was specified.

In addition to the parameter settings of each algorithm, the experiment setting in Appendix B also defines all the runs used.

During the experiment, the average fitness value of the best solution found so far was recorded. Table II and Fig. 1 show the solutions and their fitness value and convergence from each algorithm setting, respectively. The bold number in Table II is the lowest fitness value among all algorithm settings. The results showed that MFSS has a potential to obtain a better quality of solutions compared with the original MFO (M500), SSA (S500), GA (GA500), and PSO (PSO500) up to almost 100%. Specifically, 250 moths, and 2-salp-length chain can perform a bit better than 100 moths, and 5-salp-length chain relatively from 19% up to almost 95%. The possible reason is that excessively reducing the number of moths and increasing the number of salps may reduce the algorithm's exploration.

Fig. 1 shows a convergence of each algorithm setting for each function at the first 500 iterations. The x-axis represents the iteration number in the logarithmic scale with base 2, and the y-axis represents the average normalized fitness value using the maximum, and minimum values obtained for each function. The plots support that both M250S2 and M100S5 (yellow lines with triangle marks and pink lines with start marks) have the potential to move faster toward convergence in most benchmark functions.

TABLE I: ALGORITHMS' PARAMETER SETTINGS

Setting Name	Parameter	Value
M500	Algorithm	MFO
	Number of moths	500
	Number of salps in a chain	0
	b	0.7
M250S2	Algorithm	MFSS
	Number of moths	250
	Number of salps in a chain	2
	b	0.7
M100S5	Algorithm	MFSS
	Number of moths	100
	Number of salps in a chain	5
	b	0.7
PSO500	Algorithm	PSO
	Number of particles	500
	Inertia weight	0.5
	Weight of particle best	2
	Weight of global best	2
GA500	Algorithm	GA
	Number of Individuals	500
	Mutation rate	0.01
	Cross-over rate	0.9
	Reproduction rate	0.9
S500	Algorithm	SSA
	Number of Salps	500

TABLE II: OPTIMIZATION SOLUTIONS FROM EACH ALGORITHM SETTING

Unimodal Functions			Multimodal Functions		
Fn	Setting	Solution	Fn	Setting	Solution
f01	GA500	3.77E+00	f07	GA500	−4.87E+03
f01	M500	3.56E−05	f07	M500	−1.06E+04
f01	PSO500	***0***	f07	PSO500	−9.28E+03
f01	S500	***0***	f07	S500	−5.49E+03
f01	M100S5	***0***	f07	M100S5	−9.13E+03
f01	M250S2	***0***	f07	M250S2	***−1.09E+04***
f02	GA500	4.57E−01	f08	GA500	1.96E+00
f02	M500	4.82E−04	f08	M500	3.78E+00
f02	PSO500	1.10E+01	f08	PSO500	4.21E+01
f02	S500	1.15E−05	f08	S500	1.04E+02
f02	M100S5	***0***	f08	M100S5	***0***
f02	M250S2	***0***	f08	M250S2	***0***
f03	GA500	4.22E+02	f09	GA500	6.87E−01
f03	M500	4.05E−05	f09	M500	1.20E+01
f03	PSO500	3.09E+05	f09	PSO500	***0***
f03	S500	***0***	f09	S500	1.13E−05
f03	M100S5	***0***	f09	M100S5	***0***
f03	M250S2	***0***	f09	M250S2	***0***
f04	GA500	9.92E−01	f10	GA500	8.25E−01
f04	M500	2.99E−04	f10	M500	3.44E−05
f04	PSO500	***0***	f10	PSO500	3.70E−03
f04	S500	1.64E−05	f10	S500	1.33E−01
f04	M100S5	***0***	f10	M100S5	***0***
f04	M250S2	***0***	f10	M250S2	***0***
f05	GA500	6.28E+01	f11	GA500	9.71E−02
f05	M500	1.44E+01	f11	M500	***−1.02E+00***
f05	PSO500	1.84E+04	f11	PSO500	***−1.02E+00***
f05	S500	2.86E+01	f11	S500	−6.55E−01
f05	M100S5	2.20E−01	f11	M100S5	***−1.02E+00***
f05	M250S2	***1.03E−02***	f11	M250S2	***−1.02E+00***

f06	GA500	9.60E+00	f12	GA500	3.92E+00
f06	M500	3.66E−05	f12	M500	4.12E−05
f06	PSO500	***0***	f12	PSO500	***0***
f06	S500	2.05E−01	f12	S500	1.75E−01
f06	M100S5	***0***	f12	M100S5	***0***
f06	M250S2	***0***	f12	M250S2	***0***

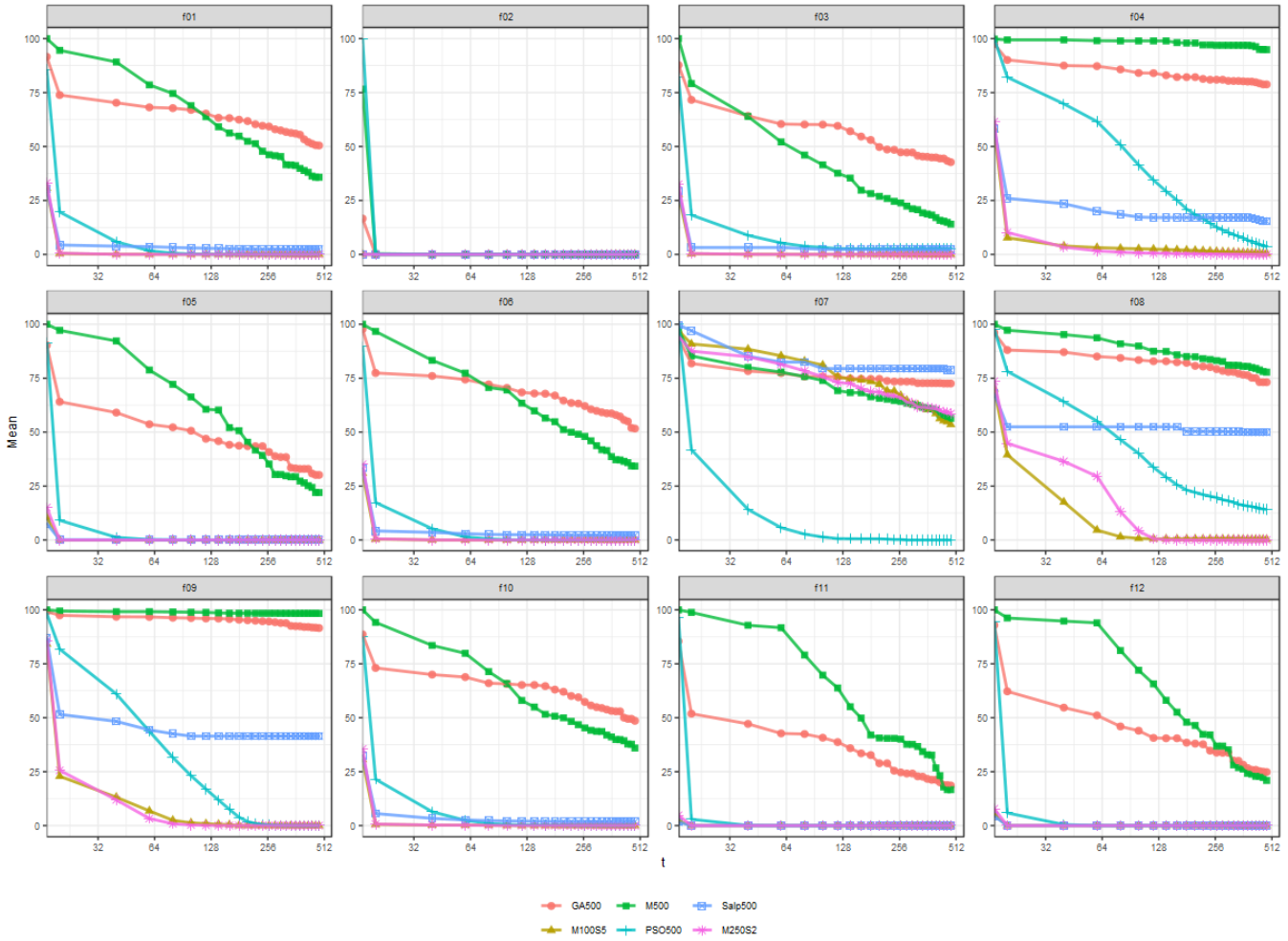

Fig. 1. Convergence plot of each algorithm setting.

IV. Conclusions

Optimization problem solving has been a very popular research area in the previous decades. Researchers have come up with new algorithms and techniques to address many challenges. The MFO algorithm and salp-swarm optimization algorithm are nature-inspired algorithms with high-performance from balanced exploration and exploitation. This research combined these algorithms and expected the better performance and quality of solutions. The experiment results support that the hybrid MFSS optimization algorithm with the linear movement of salps can outperform the original MFO algorithm and the SI and PSO algorithms.

However, the results from the proposed algorithm depend on the appropriate parameter settings, and the characteristic of the search space and problem. Adjusting parameter settings for the better balance of exploration and exploitation can produce a better quality of the solutions and vice versa. In addition, a different hybrid approach can be further investigated, for example, the flame selection method for a moth and the position update function for a salp.

In future works, the hybrid MFSS algorithm can be experimented with a variety of real optimization problems, including job-shop scheduling, vehicle routing problem, and workflow scheduling.

Appendix

A. Related Works

This section reviews the optimization problem. There are some challenges that require algorithms to tackle. Optimization algorithms are categorized and described. Some of the high-performance collective-based optimization algorithms are reviewed.

1) Optimization problem

An optimization problem is a problem to minimize or maximize one or more objectives subject to a set of equality and/or inequality constraints. An optimization problem can be represented in (5), where

$\vec{x}$ is a solution candidate,

$F(\vec{x})$ is an objective function,

n is a number of objectives,

$g_i(\vec{x})$ is an unequal constraint,

m is a number of unequal constraints,

$h_i(\vec{x})$ is an equal constraint,

p is a number of equal constraints,

lb_i is a lower bound for x_i,

ub_i is an upper bound for x_i, and

d is a number of dimensions.

$$\begin{aligned} &\text{Minimize: } F(\vec{x}) = \{f_1(\vec{x}), f_2(\vec{x}), \ldots, f_n(\vec{x})\} \\ &\text{Subject to: } G = \left\{ g_i \mid g_i(\vec{x}) \geq 0 \right\} \quad i = 1, 2, ..., m \\ &\qquad\qquad H = \left\{ h_i \mid h_i(\vec{x}) = 0 \right\} \quad i = 1, 2, ..., p \\ &\qquad\qquad lb_i \leq x_i \leq ub_i \quad\quad i = 1, 2, ..., d \end{aligned} \tag{5}$$

There are many challenges in solving the optimization problem. First, constraints create areas of infeasible solutions in the search space and potentially cause an optimization algorithm ineffective, while having a good performance in the search space without constraints. Moreover, a significant obstacle of solving the practical optimization problem is the local optima existence. An algorithm to solve this kind of problem should have techniques to avoid local optima and incorrectly taking it as the global optimum. However, having a more intensive exploration process to avoid local optima may not lead to convergence and, in consequence, may not be able to obtain the global optimum.

2) Optimization algorithms

Optimization algorithms are designed to solve optimization problems and their challenges. There are two main groups of optimization algorithms: deterministic, which always obtain the same solution for a specific problem, and stochastic, which can obtain the different solutions due to the randomness component. With randomness, stochastic algorithms can avoid local optima but cause less reliability. However, they can be improved by balancing exploration, and exploitation, and increasing the number of runs.

Stochastic optimization algorithms also can be divided into two more groups: individual based and collective-based. The collective-based algorithm generates multiple random solutions and improves them over iterations, whereas the individual based algorithm works with only a single solution. The collection can increase a chance to get a true global optimum, but it generates more computational cost because there are more solutions to evaluate their fitness. However, the collective-based algorithms are very popular because of the high potential to avoid local optima, and their drawback can be relieved by computers with higher performance or concurrent computing. Their advantages also include simplicity of algorithms and flexibility work in different types of problems.

Due to these advantages and relievable drawback, collective-based stochastic optimization algorithms can definitely interest researchers to solve optimization problems.

3) PSO algorithm

The PSO algorithm [2] is one of the collective stochastic optimization algorithms proposed by Kennedy and Eberhart. It was inspired by the movement of a flock of birds. [7] shows that this algorithm has the highest number of publications among several swarm-intelligence-based optimization algorithms, such as artificial bee colony [8] and ACO [3].

The PSO algorithm solves the optimization problem by having collaborated particles move to a better position by knowledge of each particle and the swarm. Each particle's velocity and position are iteratively updated based on its inertia, its best position, and the swarm's best position. The weights for each component are used to balance the exploration and exploitation of the algorithm.

4) MFO algorithm

The MFO algorithm was proposed by Mirjalili [4]. It was inspired by the movement of a moth toward a flame. Normally, in a night, a moth moves to a path and constantly keep the angle with respect to the very faraway moon. This method makes the moth move at an almost-straight path. However, the flame or artificial lights confuse a moth because they are very close to the moth compared with the moon. The moth will fly spirally around the light to preserve the angle and eventually hit the light [9].

The MFO algorithm solves the optimization problem by having a swarm of moths spirally move around the flames represented by a specified number of best positions found so far.

Similar to the PSO, the position of each moth represents a solution. A logarithmic spiral function used to update the position of each moth allows the moth to not only move between the current position and the targeted flame but also move around the flame to establish exploration on a search space. However, the convergence parameter is defined to reduce the distance between the result from the logarithmic spiral function and the flame, so exploration, and exploitation can be balanced. That is, exploration is high, but exploitation is low at the beginning, whereas exploration is slightly reduced but exploitation is slightly decreased over iterations.

In addition, the number of flames is reduced over iterations until there is only one flame for all moths to follow. This also increases exploitation and guarantees algorithm convergence.

5) SSA

SSA was proposed by Mirajalili et al. [5]. A salp is a barrel-shaped planktonic tunicate. It was inspired by how a swarm of salps forms salp chains for believably better foraging. For each salp chain, salps can be divided into a leader and followers. SSA solves the optimization problem by having a leader of each chain move toward a food source, and followers follow their leader directly and indirectly. The position of each salp represents a solution. As the food source in the real optimization problem is unknown, the best solution found so far will be used as a food source to be chased.

The exploration process is fulfilled by the followers that move toward the leader, not only the food source. However, followers move closer to their leader over iterations. The process reduces exploration and increases exploitation.

In addition, the exploitation process is fulfilled by the leaders, which always move toward the best solutions found so far.

From the survey, MFO and SSA both have the potential to solve optimization problems, and challenges because of their

strong exploration and exploitation process. This paper combines MFO and SSA for the better quality of solutions and faster convergence. The next section discusses the MFO algorithm and SSA and how to combine them into the proposed hybrid algorithm.

B. Benchmark Functions And The Experiment's Parameter Settings

Tables III and IV present a list of benchmark functions and environment parameters used in the experiment, respectively.

TABLE III: BENCHMARK FUNCTIONS

Function	Range	f_{min}
Unimodal Functions		
$F_{01}(X)=\sum_{i=1}^{n}x_i^2$	[−100, 100]	0
$F_{02}(X)=\sum_{i=1}^{n}\lvert x_i\rvert+\prod_{i=1}^{n}\lvert x_i\rvert$	[−10, 10]	0
$F_{03}(X)=\sum_{i=1}^{n}(\sum_{j=0}^{i}x_j^2)$	[−100, 100]	0
$F_{04}(X)=\max\left(\lvert x_i\rvert\right)\quad 1\le i\le n$	[−100, 100]	0
$F_{05}(X)=\sum_{i=1}^{n}\left[100(x_{i+1}-x_i^2)^2+(x_i-1)^2\right]q$	[−30, 30]	0
$F_{06}(X)=\sum_{i=1}^{n}(x_i+0.5)^2$	[−100, 100]	0
Multimodal Functions		
$F_{07}(X)=\sum_{i=1}^{n}-x_i\sin\left(\sqrt{\lvert x_i\rvert}\right)$	[−500, 500]	−418.9829 *n
$F_{08}(X)=\sum_{i=1}^{n}\left(x_i^2-10\cos(2\pi x_i)+10\right)$	[−2π, 2π]	0
$F_{09}(X)=\sum_{i=1}^{n}-20\exp\left(-0.2\sqrt{\left(\sum_{i=1}^{n}nx_i^2\right)/n}\right)-\exp\left(\left[\sum_{i=1}^{n}\cos(2\pi x_i)\right]/n\right)+20+e$	[−32, 32]	0
$F_{10}(X)=(1/4000)\sum_{i=1}^{n}x_i^2-\prod_{i=1}^{n}\left[\cos\left(x_i/\sqrt{i}\right)+1\right]$	[−600, 600]	0
$F_{11}(X)=(\pi/n)\{10\sin(\pi y_1)+\sum_{i=1}^{n-1}(y_i-1)^2\left[1+10\sin^2(\pi y_{i+1}\right]+(y_n+1)^2\}+\sum_{i=1}^{n}u(x_i,10,1000,4)$, $y_i=1+(x_i+1)/4$, $u(x_i,a,k,m)=\begin{cases}k(x_i-a)^m & x_i>a\\ 0 & -a<x_i<a\\ k(-x_i-a)^m & x_i<a\end{cases}$	[−50, 50]	0
$F_{12}(X)=0.1\{0.1\sin^2(3\pi x_i)+\sum_{i=1}^{n}(x_i-1)^2\left[1+\sin^2(3\pi x_i+1)\right]+(x_n-1)^2\left[1+\sin^2(2\pi x_n)\right]\}$	[−50, 50]	0

TABLE IV: EXPERIMENT'S PARAMETER SETTINGS

Parameter	Value
Number of iterations (stopping criteria)	10000
Number of runs	10
Dimensions of benchmark functions	30

REFERENCES

[1] C. Blum and X. Li, "Swarm intelligence in optimization," in *Swarm Intelligence*, C. Blum, D. Merkle, Springer, 2008, pp. 43-85.

[2] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. ICNN'95 - Int. Conf. Neural Networks*, Nov. 1995, pp. 1942-1948.

[3] M. Dorigo and C. Blum, "Ant colony optimization theory: A survey," *Theor. Comput. Sci.*, vol. 344, pp. 243-278, Nov. 2005.

[4] S. Mirjalili, "Moth-flame optimization algorithm: A novel nature-inspired heuristic paradigm," *Knowledge-Based Syst.*, vol. 89, pp. 228-249, Jul. 2015.

[5] S. Mirjalili, A. H. Gandomi, S. Z. Mirjalili, S. Saremi, H. Faris, and S. M. Mirjalili, "Salp swarm algorithm: A bio-inspired optimizer for engineering design problems," *Adv. Eng. Softw.*, vol. 114, pp. 163-191, 2017.

[6] M. Molga and C. Smutnicki. Test functions for optimization needs. [Online]. Available: https://www.robertmarks.org/Classes /ENGR5358/Papers/functions.pdf

[7] Y. Zhang, S. Wang, and G. Ji, "A comprehensive survey on particle swarm optimization algorithm and its applications," *Math. Probl. Eng.*, vol. 2015, pp. 1-38, Oct. 2015.

[8] D. Karaboga and B. Basturk, "A powerful and efficient algorithm for numerical function optimization: Artificial bee colony (ABC) algorithm," *J. Glob. Optim.*, vol. 39, pp. 459-471, Apr. 2007.
[9] K. D. Frank, "Effects of artificial night lighting on moths," in *Ecological Consequences of Artificial Night Lighting*, C. Rich and T. Longcore, Island Press, 2013, ch. 13, pp. 305-344.

Orachun Udomkasemsub was born in Thailand. He obtained his bachelor's degree in computer engineering from King Mongkut's University of Technology Thonburi (KMUTT), Thailand, in 2011.

He is currently a Ph.D. student of the computer engineering department of KMUTT. His research interests include area optimization and cloud computing.

Khajonpong Akkarajitsakul was born in Thailand. He obtained his philosophy degree in electrical and computer engineering from the University of Manitoba, Canada, in 2013.

He is currently a lecturer at the Department of Computer Engineering, Faculty of Engineering, KMUTT, Thailand. He has also been a data scientist at the Big Data Experience (BX) Center for several projects. His research interests include computational modeling, communications, and networking, distributed and mobile computing, and data analytics.

Tiranee Achalakul was born in Thailand. She obtained her philosophy degree in computer engineering from Syracuse University, New York, in 2000.

She has worked in the fields of data analytics, high-performance computing, and software development since 2000. She has extended experiences working in the IT industry and academia in both the United States and Thailand with two published textbooks and multiple journal and conference papers. During the past 14 years, she has been participating in many data analytics, and software development projects in both private and public sectors. She has also served as a committee of the National e-Science Infrastructure Consortium of Thailand. Her research interests include data analytics, high-performance computing, and software development.