A Simplified Fish School Search Algorithm for Continuous Single Objective Optimisation

Elliackin Figueiredo[1], Clodomir Santana[2], Hugo Valadares Siqueira[3],
Mariana Macedo[4], Attilio Converti[5], Anuradha Gokhale[6], Carmelo Bastos-Filho[1]*

Abstract

The Fish School Search (FSS) algorithm is a metaheuristic known for its distinctive exploration and exploitation operators and cumulative success representation approach. Despite its success across various problem domains, FSS presents issues due to its high number of parameters, making its performance susceptible to improper parameterisation. Additionally, the interplay between its operators requires a sequential execution in a specific order, requiring two fitness evaluations per iteration for each individual. This operator's intricacy and the number of fitness evaluations pose the issue of costly fitness functions and inhibit parallelisation. To address these challenges, this paper proposes a Simplified Fish School Search (SFSS) algorithm that preserves the core features of the original FSS while redesigning the fish movement operators and introducing a new turbulence mechanism to enhance population diversity and robustness against stagnation. The SFSS also reduces the number of fitness evaluations per iteration and minimises the algorithm's parameter set. Computational experiments were conducted using a benchmark suite from the CEC 2017 competition to compare the SFSS with the traditional FSS and five other well-known metaheuristics. The SFSS outperformed the FSS in 84% of the problems and achieved the best results among all algorithms in 10 of the 26 problems.

Introduction

In computational intelligence, swarm and evolutionary metaheuristics have garnered significant attention for their ability to solve complex optimisation problems. Leveraging nature's process to evolve efficient and effective solutions to many challenges, techniques in this domain seek to apply these principles to the design metaheuristics. Among these metaheuristics are Genetic Algorithms (GA) holland1992genetic, Particle Swarm Optimisation (PSO) Kennedy:1995, Artificial Bee Colony (ABC) Karaboga:2007, and Fish School Search (FSS) bastos2021fish. Besides drawing inspiration from unique biological and evolutionary principles in their design, each one of these methods possesses unique characteristics related to their behaviour, operators, and capabilities.

For example, Genetic algorithms are a class of optimisation algorithms inspired by natural selection and belong to the broader category of evolutionary algorithms (EAs) holland1992genetic. GA leverages selection, crossover, and mutation operators to evolve a population of potential solutions over successive generations schmitt2001theory. GAs typically encode solutions as strings or chromosomes, often using binary representation. This encoding scheme allows for easy manipulation and combination of solutions, making it an excellent choice for combinatorial optimisation problems kumar2013encoding.

Unlike the GA, particle swarm optimisation is in the swarm intelligence (SI) field. PSO is based on the social behaviour of birds flocking and fish schooling Kennedy:1995. It comprises a population of candidate solutions (particles) that move through the search space to find the optimal solution. Unlike other metaheuristics, PSO uses velocity and position vectors to guide the search process. The velocity vector determines the direction and speed of a particle's movement, while the position vector represents the particle's current solution sousa2017review. The updates are governed by equations that incorporate both the particle's best-known position and the best position discovered by the swarm.

Another example of SI metaheuristic is the artificial bee colony, which simulates the foraging behaviour of honey bees Karaboga:2007. In ABC, there are three types of bees: employed bees, onlooker bees, and scout bees, representing phases of the algorithm. These phases allow for a balanced exploration and exploitation of the search space. The ABC differs from the GA and most swarm-based algorithms because the candidate solutions are not encoded as part of the agents (bees). Instead, the ABC used the analogy of food sources to represent

the candidate solutions. The bee searchers exploit these food sources to find better solutions to the optimisation problem Karaboga:2007.

A third algorithm from the SI family is the fish school search. The Fish School Search Algorithm (FSS) is inspired by fish swarms' collective and individual behaviour bastos2008novel. In the FSS, individual fish represent potential solutions, and local and global behaviours influence their movements. Individual, collective-instinct, and collective-volitive components govern the fish movement bastos2021fish. Individual movement allows each fish to explore the search space based on its own experiences. In contrast, the collective movements direct the fish school towards promising regions in the search space, influenced by the overall school's behaviour bastos2021fish. While the ABC and the PSO use current or previous best positional information to estimate success, the FSS employs a cumulative success representation for its candidate solutions bastos2021fish. The success accumulation is represented as the fish's weight. Over iterations, fish can gain weight when they improve their solution, and the populations will tend to move to regions with the heaviest fish. FSS is known for its unique strategy to balance exploration and exploitation base on the fish movements and the feeding operator bastos2008novel.

While the ABC and the PSO use current or previous best positional information to estimate success, the FSS employs a cumulative success representation for its candidate solutions de2016solving. The success accumulation is represented as the fish's weight. Over iterations, fish can gain weight when they improve their solution, and the populations will tend to move to regions with the heaviest fish. FSS is known for its unique strategy to balance exploration and exploitation based on the fish movements and the feeding operator bastos2008novel.

Despite their success and widespread application, these metaheuristics exhibit drawbacks. For example, GA can present imitations linked to the definition of a proper solution encoding strategy ronald1997robust and premature convergence pandey2014comparative, PSO has issues maintaining swarm diversity and avoiding premature convergence jordehi2015enhanced, abdel2018improved, tang2009particle, and ABC has weak in exploration ye2022artificial, makas2016balancing, yu2018artificial.

It is worth noting the significant advancements made in the field of metaheuristics, which has not only new advanced methods and applications such as Salp Swarm Algorithm knypinski2024application, khajehzadeh2022adaptive but also novel ways to model and study fitness landscapes as hierarchical structure of solutions based on dominance relationships, where a solution is considered dominant if it outperforms others in multiple objectives or constraints hao2019domination.

Regarding the FSS, it is more complex to implement and has greater algorithmic complexity than the GA and PSO, and performance issues due to improper parametrisation can occur. Also, the interplay between the movements requires them to be executed sequentially with two fitness evaluations per individual per iteration: one evaluation after the individual movement used to signal the individual guiding the collective movements and another after the collective movements to update the population's fitness. Reducing the number of fitness evaluations benefits computationally expensive fitness calculations and allows for parallel execution of the movements.

This paper proposes a novel simplified version of the Fish School Search algorithm. Our approach aims to retain the core advantages of FSS—such as the balance between exploration and exploitation, adaptability, and robustness—while reducing the number of parameters and fitness evaluations per iteration. The main challenges involve identifying and preserving the essential characteristics contributing to the algorithm's success while eliminating redundancies and minimising computational overhead. The simplification process aims to:

- Analyse the original FSS to identify critical elements that drive its performance and refine or eliminate non-essential components.
- Reduce the number of fitness evaluations per iteration by redesigning the interplay between the fish movement operators.
- Decrease the number of parameters to make it less susceptible to performance issues due to improper parametrisation.

The remainder of the article is organised as follows: Section S1 describes the original FSS and other proposed versions of it, Section S2 presents the new version of the FSS algorithm, Section S3 shows the computational results achieved using databases from the CEC '2017 competition and a discussion about them. Finally, Section S4 presents the conclusions.

S1 Fish School Search (FSS)

The Fish School Search has four operators: individual movement, feed operator, collective instinctive movement, and collective volitive movement santana2019sbfss. The collective movements are unique to all schools. It considers the following variables: N is the total number of fish or the school size, $\vec{z}_i^t$ is the current position of the fish i at the iteration t. The operators are described below bastos2021fish:

- Individual movement ($\vec{n}_i^{t+1}$): random search in which each fish randomly chooses a new position in its neighbourhood. It causes diversity and triggers the other operators. It is executed according to Equation S1 Eq:

$$\vec{n}_i^{t+1} = \vec{z}_i^t + step_{ind}.rand[-1, 1] \tag{S1 Eq}$$

  where $\vec{n}_i^{t+1}$ is a new position (temporary), $step_{ind}$ is an individual step set by the user (this decay is often linear, although other variants of the FSS proposed other methods such as exponential decay demidova2021application) and $rand[-1, 1]$ is a random value generated by a uniform probability density function in the interval [-1,1]. Observe that $rand[-1, 1]$ must be drawn for each dimension $d = 1, ..., D$ separately, while $step_{ind}$ is constant in the current iteration. In this movement, the fish goes to the new position only if there is more food than the current position.

- Feeding operator (w_i^t): updates the fish weight and occurs after the individual movement. Firstly, the new position $\vec{n}_i^t$ is evaluated according to the fitness $f[\vec{n}_i^t]$ obtained in the previous movement and compared to the fitness of the current position $\vec{z}_i^t$, according to Equation S2 Eq:

$$\Delta f_i^{t+1} = |f[\vec{n}_i^{t+1}] - f[\vec{z}_i^t]| \tag{S2 Eq}$$

  The value Δf_i^{t+1} is used to update the fish weight, as shown in Equation S3 Eq

$$w_i^{t+1} = w_i^t + \left(\frac{\Delta f_i^{t+1}}{max[\Delta f_i^{t+1}]}\right) \tag{S3 Eq}$$

  Equation S3 Eq shows that the weight of the fish increases according to the success rate achieved by the individual movement. The fish will move to the new position $\vec{n}_i^{t+1}$ if the movement elevates its fitness or, in other words, if the new position is better than the current (greedy search).

- Instinctive Collective Movement ($\vec{m}^t$): This movement is influenced by the fish who successfully updated their fitness from the individual movement. All the fish perform this movement, calculated via Equation S4 Eq

$$\vec{m}^{t+1} = \frac{\sum_{i=1}^{N} \Delta \vec{z}_i^{t+1} \Delta f_i^{t+1}}{\sum_{i=1}^{N} \Delta f_i^{t+1}} \tag{S4 Eq}$$

  where $\vec{z}_i^t$ is the displacement if the fish i caused by the individual movement and Δf_i^t is calculated by S2 Eq.

  So, all the school has its position updated by Equation S5 Eq

$$\vec{z}_i^{t+1} = \vec{z}_i^t + \vec{m}^{t+1} \tag{S5 Eq}$$

- Volitive Collective Movement ($\vec{B}^t$): This second collective movement is performed according to the overall success rate of the fish school, measured by the sum of the fish weights. If the total school weight has increased ($w^{t+1} > w^t$), this means that the current search was successful. So, the school should contract to improve the exploitation behaviour. However, if the school weight has decreased ($w^{t+1} < w^t$), it should expand to increase the exploration of the search space. This movement is executed according to the school

barycenter calculated via S6 Eq

$$\vec{B}^{t+1} = \frac{\sum_{i=1}^{N} \vec{z}_i^{t+1} w_i^{t+1}}{\sum_{i=1}^{N} w_i^{t+1}} \tag{S6 Eq}$$

If the weight of the school grows ($w^{t+1} > w^t$), the fish' positions are updated according to Equation S7 Eq:

$$\vec{z}_i^{t+1} = \vec{z}_i^{t+1} - step_{vol}.rand[0,1]\left(\vec{z}_i^{t+1} - \vec{B}^{t+1}\right) \tag{S7 Eq}$$

If not ($w^{t+1} < w^t$), perform the new positions by S8 Eq:

$$\vec{z}_i^{t+1} = \vec{z}_i^{t+1} + step_{vol}.rand[0,1]\left(\vec{z}_i^{t+1} - \vec{B}^{t+1}\right) \tag{S8 Eq}$$

where the $step_{vol} = 2.step_{ind}$ (previously defined in the individual movement), and $rand[0,1]$ is a random value generated by a uniform probability density function in the interval [0,1]. Observe that $rand[0,1]$ must be drawn separately for each dimension $d = 1, ..., D$, while $step_{vol}$ is constant in the current iteration.

Algorithm 1 presents the pseudocode of the original FSS.

Over the last decade, many improvements have been made to the FSS algorithm. The Density Based Fish School Search (dFSS) was developed to solve multimodal hyper-dimensional problems, adding new operators as memory and partition bastos2021fish. The Weight-based Fish School Search (wFSS) modifies the barycenter by adding the Link Formation Rule, which causes the formation of niches bastos2021fish. Some versions are proposed to tackle premature convergence and stagnation bastos2021fish. Most recently, the FSS family was expanded to cover multi-objective problems for continuous and binary spaces bastos2021fish. Lastly, a simplified version of the FSS was also proposed for problems in the binary domain bastos2021fish, which demonstrated that it was possible to reduce the complexity of the FSS while improving its performance.

Algorithm 1: FSS Pseudocode

```
Initialize randomly all fish positions z_i^0, according to Equation S1 Eq;
Initialize randomly all fish weights W_i^0;
while stop criterion is not reached do
    foreach fish do
        Find neighbor position according to Equation S1 Eq;
        if Δ(f_i^{t+1}) < 0 (minimization) then
            Evaluate the neighbor position
            Perform greedy search and calculate the displacement using S2 Eq
        else
            Stands in the same position;
        end
    end
    Feed the fish using S3 Eq;
    foreach fish do
        Calculate the instinctive collective movement via Equation S4 Eq;
        Execute the instinctive movement using S5 Eq;
    end
    Calculate barycenter using S6 Eq;
    foreach fish do
        Execute volitive movement using either S7 Eq or S8 Eq
    end
    Calculate the instinctive collective movement via Equation S4 Eq;
    Update s_ind and s_vol;
end
Return the best solution found;
```

S2 The Proposed Fish School Search

The Simplified Fish School Search (SFSS) algorithm follows the structure and inspiration of the FSS (movements and operators). The main goal was to reduce the use of fitness functions while maintaining its generation of diversity and its automatic balance of exploitation and exploration mechanisms. Moreover, another objective was to minimize the number of parameters the user needs to initialize and define, such as initial and final step sizes, initial weight, and weight limits. The only parameter preserved is the number of individuals in the swarm.

In the original FSS, the swarm evaluates twice: one time after the individual movement and another time after the volitive movement. To reduce to only one evaluation per iteration, instead of updating the individual's position after each movement, the movements generate displacements based on the fish's current position. After all displacements are calculated, the fish position is updated by combining all three displacement values. More than reducing the number of fitness evaluations, this strategy also allows the three displacements to be calculated in parallel, which reduces the execution time.

- Individual Displacement ($\vec{Ind}_i^t$): For each fish in the school, it is drawn a random value generated by a uniform distribution in the interval [0,1]. If the probability of the fish i is greater than the value generated, then the displacement is calculated using the Equation S9 Eq. Otherwise, the fish do not perform an individual displacement.

$$\vec{Ind}_{i,d}^{t+1} = rand[-1,1].(\vec{x}_{i,d}^{t-1} - \vec{x}_{j,d}^{t-1}) \tag{S9 Eq}$$

  where $\vec{Ind}_{i,d}^t$ is the displacement for fish i, j is a random fish selected from the swarm (S10 Eq), $rand[-1,1]$ is a random value generated by a uniform probability density function in the interval [-1,1] and d is a random dimension selected from the number of problem dimensions.

  Equation S10 Eq calculates the fish selection probability.

$$P_i^{t+1} = \frac{w_i^{t+1}}{max[W^{t+1}]} \tag{S10 Eq}$$

  where w_i is the weight of the fish i and $max[W^{t+1}]$ returns weight of the heaviest fish in the school (i.e. the current best solution).

- Instinctive Displacement ($\vec{Ins}_i^t$): For each fish that had improved in the school, the new position is calculated using the Equation S11 Eq:

$$\vec{Ins}_{i,d}^{t+1} = \frac{select(\vec{x}_{i,d}^{t-1} - \vec{x}_{i,d}^{t})}{\sum_{i=1}^{N} w_i^t} \tag{S11 Eq}$$

  where $\vec{Ins}_i^t$ represents the instinctive displacement of fish i in dimension d, $select([-1,1])$ is a function which selects and returns 1 or -1 and w_i^t is the weight of fish i at time t.

- Volitive Collective Displacement ($\vec{Vol}_i^t$): For each fish in the school, the displacements are generated by Equation S12 Eq:

$$\vec{Vol}_i^{t+1} = sign\left(\vec{x}_i^t - \vec{x}_j^t\right) \tag{S12 Eq}$$

  where $\vec{Vol}_i^{t+1}$ is the volitive displacement for fish i, j is a fish selected from the swarm using a binary tournament process, $sign$ is a function which returns a random value generated by a uniform probability density function in the interval [-1,0], if the weight of the fish j is greater than the weight of fish i, or [1, 0] otherwise. This means that the fish i will move towards the fish j if the fish j is heavier.

The new position of the fish is generated by combining the three displacements, as can be seen in Equation S13 Eq

$$\vec{x}_i^{t+1} = \vec{Ind}_i^{t+1} + \vec{Ins}_i^{t+1} + \vec{Vol}_i^{t+1} \tag{S13 Eq}$$

Another modification made is related to the feeding operator (W_i^t). The fish's weight reflects how good the solution is found, and it determines the degree of influence a fish has on the swarm. Initially, when a fish moves to a better region, it gains weight; if it cannot improve, its weight remains the same. Even though it will be punished by not having the instinctive movement and its influence will decrease as other fish get heavier, this process might be slow. In SFSS, weight loss was introduced to penalize even more fish that do not improve. First, the (Δf_i^{t+1}) is calculated using Equation S2 Eq, if $\Delta f_i^{t+1} > 0$, fish weight is updated with the FSS weight gain (Equation S3 Eq), otherwise use (S14 Eq).

$$w_i^{t+1} = w_i^t . e^{-\left(\left|\frac{\Delta f_{t+1}^t}{max[\Delta f_i^{t+1}]}\right|\right)} \tag{S14 Eq}$$

where w_i is the weight of fish i, e is the exponential function and $max[\Delta f_i^{t+1}]$ return the maximum variation of fitness in the school.

We observed the algorithm's performance in different problems in preliminary experiments by analysing the population weight over the iterations. In these experiments, we noticed that in some cases, the population weight could reach values below one after several iterations without improvements and losing diversity, causing stagnation issues. To address this issue, the SFSS features a turbulence mechanism to promote population diversity and increase the probability of improvements in the population. This mechanism is triggered only in stagnation situations (e.g., swarm weight below one) and for a limited number of fish in the population. In preliminary experiments, we noticed that applying the perturbation to 10% of the worst individuals in the school was enough to produce satisfactory results. This perturbation is not used on consecutive iterations to prevent the adverse effects of introducing too much diversity. Also, we chose the Gaussian perturbation as our turbulence operator as it is simple to implement, has a low computational cost and produces the expected results.

The SFSS is described in the Algorithm 2. It is important to mention that in line 5 the turbulence will only be applied on the worst ten percent fish of the school.

Algorithm 2: SFSS Pseudocode

```
1  Initialize randomly all fish positions z_i^0, according to Equation S1 Eq;
2  Initialize all fish weights (W_i^0) as 0 and initial probability as 1/(school size);
3  while stop criterion is not reached do
4      if (school weight < 1) and (turbulence was not used in last iteration) then
5          Apply Gaussian turbulence;
6      else
7          foreach fish do
8              Calculate the displacements using equations S9 Eq, S11 Eq and S12 Eq;
9              Generate the new position using Equation S13 Eq and evaluate it;
10             Move to new position only if cost of new position is greater than the current cost
11         end
12         foreach fish do
13             Feed the fish with Equation S3 Eq if fish improved, otherwise use Equation S14 Eq;
14             Update swarm weight and the probability applying Equation S10 Eq;
15         end
16     end
17 end
```

Figure 1 compares the execution flow of the FSS and the SFSS. We highlighted the components where the algorithms perform fitness evaluations of their populations in orange.

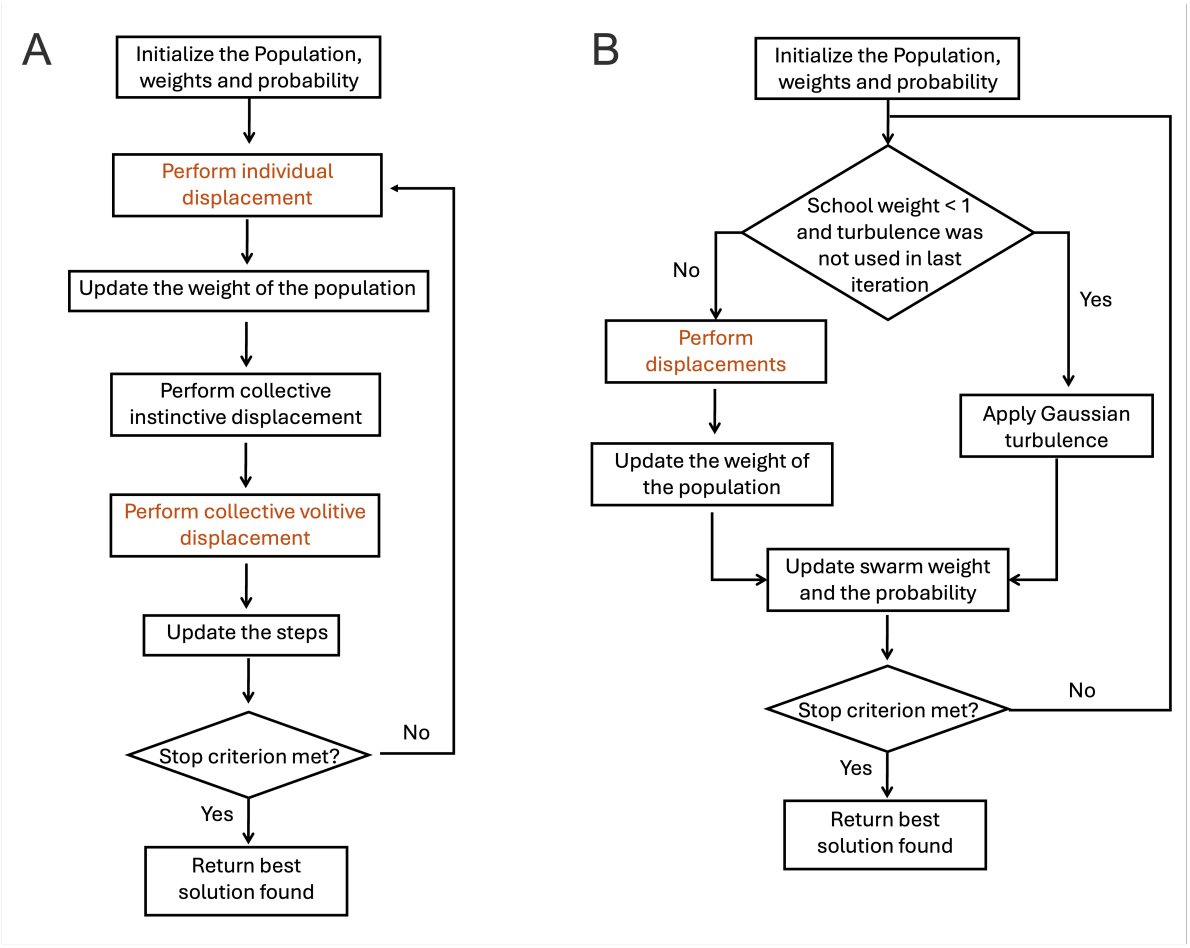

Figure 1: Comparison between the execution flow of the FSS(A) and SFSS(B). The steps highlighted represent the points the algorithm needs to evaluate the population's fitness.

S2.1 SFSS: Trials

During the development of the proposal, the following ideas were also considered as candidates to replace the movements and operators of the FSS. However, they were not used in the final version because they did not improve the algorithm performance, and another simpler solution revealed similar or better results than these.

S2.1.1 Movements Trials

- Uses a roulette wheel to select a fish that will try to move and another roulette to choose a fish that will attract fish. The fish moves if the new location is better than the previous one.
- Similar to the previous one, but instead of a roulette wheel to select a fish that will try to move, all fish try to move.
- All fish try to move, and selecting a random fish from the school will attract fish.
- Generating a new position in all dimensions VS modifying only one random dimension.

S2.1.2 Feeding Operator Trials

The variations described in this section aimed to find a more appropriate form to penalize or reward the fish when necessary. The weight loss means that a fish could not improve in the current iteration, and when the school weight decreases, it might indicate that the swarm converged or is trapped in a local minimal.

- Exponential weight gain and loss:

$$w_i^t = w_i^{t-1} e^{\frac{\Delta f_i^{t+1}}{max[\Delta f_i^{t+1}]}} \quad \text{(S15 Eq)}$$

- Nonlinear weight gain attempt 1:

$$w_i^t = w_i^{t-1} + \left(\frac{|\Delta f_i^{t+1} - \Delta f_i^t|}{max[\Delta f_i^{t+1} - \Delta f_i^t]}\right)\left(\frac{-1}{\left|\frac{\Delta f_i^{t+1} - \Delta f_i^t}{max[\Delta f_i^{t+1} - \Delta f_i^t]}\right| - 1} - 1\right) \quad \text{(S16 Eq)}$$

- Nonlinear weight gain attempt 2:
  It is similar to the previous one but with an addition operation instead of multiplication between the normalized variation of the delta cost with the last term.

- Nonlinear weight gain attempt 3:

$$w_i^t = w_i^{t-1} + \left(\frac{|\Delta f_i^{t+1} - \Delta f_i^t|}{max[\Delta f_i^{t+1} - \Delta f_i^t]}\right)\left(\frac{|\Delta f_i^{t+1} - \Delta f_i^t|}{max[\Delta f_i^{t+1} - \Delta f_i^t]}\right)^5 \quad \text{(S17 Eq)}$$

- Nonlinear weight gain attempt 4:

$$w_i^t = w_i^{t-1}\left(\frac{|\Delta f_i^{t+1} - \Delta f_i^t|}{max[\Delta f_i^{t+1} - \Delta f_i^t]}\right)^5 \quad \text{(S18 Eq)}$$

- Nonlinear weight gain attempt 5:

$$w_i^t = w_i^{t-1} + \left(\frac{|\Delta f_i^{t+1} - \Delta f_i^t|}{max[\Delta f_i^{t+1} - \Delta f_i^t]}\right).100^{\left(\frac{|\Delta f_i^{t+1} - \Delta f_i^t|}{max[\Delta f_i^{t+1} - \Delta f_i^t]}\right)-1} \quad \text{(S19 Eq)}$$

All the exponential weight gain attempts produced similar results. For this reason, the criteria for selecting one of the approaches were simplicity and computational cost.

S3 Case Study

We tested the algorithms using 26 optimization problems from the IEEE Congress on Evolutionary Computation (CEC) 2017 test suit wu2016problem. Although the test suit has 28 problems, we excluded the F17 and F21 because they show unstable behaviour possibly caused by the source code. The Python code for the CEC'17 test suite can be downloaded from the GitHub page [1]. All the functions are tested in 30 dimensions, and among this problem, we have unimodal, multimodal, shifted, rotated, and composed functions. The experiments were conducted in a Apple M4 Pro, 36GB of RAM, 1TB of hard drive, and running macOS Sequoia 15.3.2 operating system.

Since the ABC and the FSS algorithms have more than one fitness evaluation per individual per iteration, to provide a fair comparison, we decided to use the number of fitness evaluations as the stop criteria of the execution. Furthermore, considering that the CEC functions can be challenging, it was decided, after previous experiments, that the number of fitness evaluations adopted would be five hundred thousand. This means that the best result recorded for a given execution will be the one achieved by the 500.000 fitness evaluation. At this point, the algorithm will cease its execution. All the algorithms were executed 30 times for each function and had a population of 30 individuals. Also, all the algorithms employed the same population initialisation strategy. At the beginning of each execution, the population is randomly distributed across the search space. Although some algorithms can benefit from different initialisation strategies, we used the same one to provide a more fair comparison.

The PSO was implemented with a global best topology and used $w0 = 0.72984$, C_1, and C_2 equal to $(2.05w)$ and a maximum velocity of 100. ABC algorithm employed the trial limit of 100. The GA algorithm was configured with a mutation rate of 0.05 and a crossover constant equal to 0.9. The FSS has initial and final individual steps, respectively, equal to 0.1 and 0.0001, the initial volitive step of 0.01, and a final volitive step of 0.001. Moreover, the initial weight and weight scale of FSS were one and (number of fitness evaluation)/4.0, respectively. The SFSS and FA do not have additional parameters to set aside from the population size.

[1] more information available at `https://github.com/tilleyd/cec2017-py/tree/master`

Figure 2 shows an example of the convergence curve of all seven algorithms in six CEC problems, while Table 1 and Figure 3 compare their performance in all 26 problems. As seen in Table 1 and Figure 3, the SFSS overcame the FSS algorithm in 23 of the 26 CEC problems. The SFSS performed best in 10 of the 26 problems compared to the other algorithms. These results suggest that the SFSS reduced the number of fitness evaluations per iteration and presented performance gains.

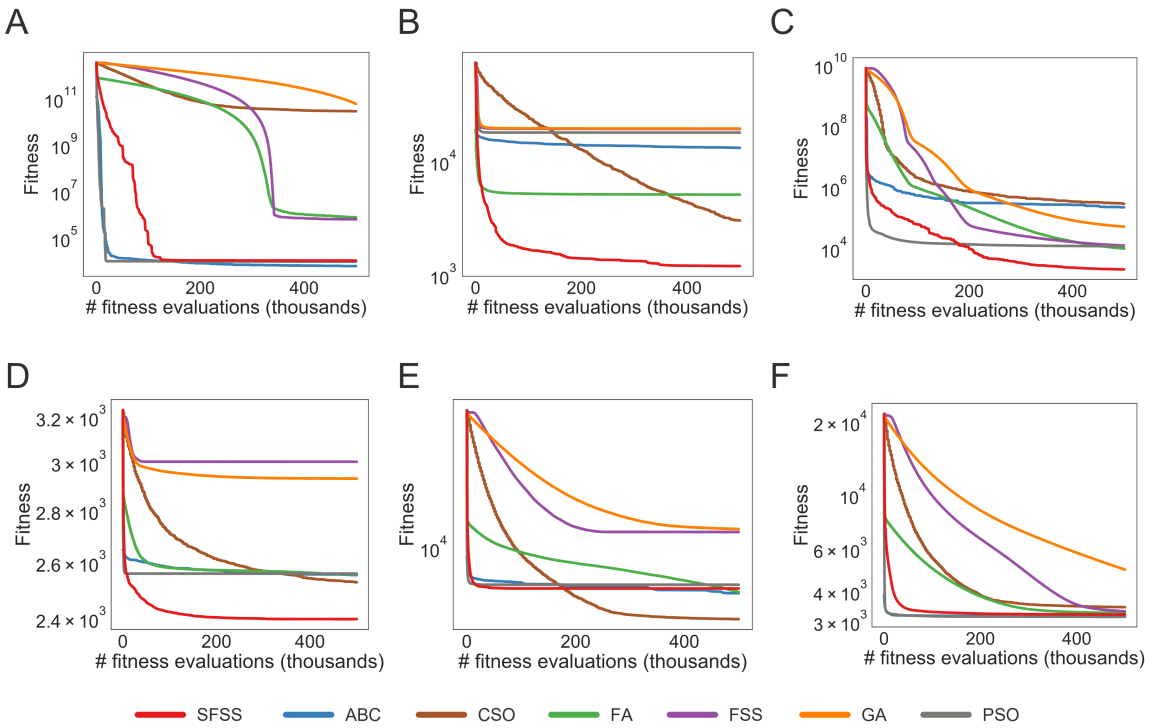

Figure 2: Example of convergence curves of the algorithms in (A) F1, (B) F9, (C) F14, (D) F21, (E) F26, and (F) F28. We present the results for 26 CEC problems with 30 dimensions. The algorithms were interrupted after 500 thousand fitness evaluations.

Figure 3 illustrates the results of the Wilcoxon test comparing the SFSS to the other algorithms. In this figures, the blue square means that the SFSS was superior, the red square denotes that the SFSS was inferior, and the grey square means that there is no statistical difference between them. The statistical results in Figure 3 reinforce the superiority of the SFSS over the FSS. Furthermore, comparing the SFSS to each algorithm reveals statistical significance in most of the results when the SFSS was better than the other algorithms.

Table 1: Performance evaluation in terms of average fitness value and (standard deviation) for all algorithms. The results for 26 CEC problems with 30 dimensions. The algorithms were interrupted after 500 thousand fitness evaluations. In bold, we have the best results.

Function	SFSS	ABC	CSO	FA	FSS	GA	PSO
F1	1.09E+04 (6.17E+03)	**6.30E+03** (**2.84E+03**)	2.87E+10 (4.46E+09)	7.87E+05 (7.86E+04)	6.46E+05 (8.34E+04)	5.95E+10 (1.64E+10)	9.71E+03 (5.43E+03)
F2	1.99E+32 (4.65E+32)	**1.03E+08** (**1.52E+08**)	5.46E+30 (1.39E+31)	2.74E+15 (3.02E+15)	1.47E+14 (1.13E+14)	2.06E+29 (2.90E+29)	3.95E+11 (2.12E+12)
F3	9.24E+04 (7.02E+04)	1.89E+05 (1.48E+04)	9.48E+04 (1.65E+04)	**9.99E+03** (**5.81E+02**)	1.09E+04 (2.04E+03)	1.38E+05 (1.33E+04)	5.02E+04 (2.90E+04)
F4	4.98E+02 (2.82E+01)	4.68E+02 (1.80E+01)	7.23E+02 (4.08E+01)	5.11E+02 (7.17E+00)	5.07E+02 (1.92E+01)	6.01E+02 (3.66E+01)	**4.21E+02** (**2.76E+01**)
F5	**6.25E+02** (**4.23E+01**)	7.11E+02 (1.89E+01)	7.39E+02 (1.58E+01)	7.60E+02 (1.45E+01)	1.59E+03 (1.33E+02)	1.48E+03 (5.23E+01)	7.79E+02 (7.30E+01)
F6	**6.22E+02** (**7.43E+00**)	6.93E+02 (2.50E+00)	6.40E+02 (6.09E+00)	6.72E+02 (3.86E+00)	7.20E+02 (9.24E+00)	7.22E+02 (5.51E+00)	7.10E+02 (1.28E+01)
F7	1.04E+03 (1.46E+02)	**8.60E+02** (**1.73E+01**)	1.02E+03 (1.73E+01)	1.30E+03 (1.49E+01)	6.63E+03 (9.01E+02)	6.32E+03 (2.17E+02)	1.25E+03 (1.66E+02)

F8	**9.47E+02** (**4.98E+01**)	1.18E+03 (3.15E+01)	1.04E+03 (1.96E+01)	9.73E+02 (1.18E+01)	1.63E+03 (9.38E+01)	1.58E+03 (3.52E+01)	1.09E+03 (1.34E+02)
F9	**1.21E+03** (**4.42E+02**)	1.32E+04 (5.76E+02)	3.04E+03 (8.07E+02)	5.13E+03 (1.47E+02)	1.94E+04 (2.28E+03)	1.95E+04 (1.11E+03)	1.80E+04 (2.08E+03)
F10	5.75E+03 (8.61E+02)	**4.13E+03** (**2.17E+02**)	8.67E+03 (2.47E+02)	4.68E+03 (2.29E+02)	6.73E+03 (4.12E+02)	5.96E+03 (5.63E+02)	5.69E+03 (5.68E+02)
F11	1.29E+03 (2.89E+02)	2.30E+03 (6.97E+02)	2.37E+03 (3.42E+02)	**1.17E+03** (**4.17E+00**)	1.47E+03 (4.34E+01)	1.84E+03 (1.51E+02)	1.36E+03 (6.25E+01)
F12	2.80E+05 (2.33E+05)	2.30E+06 (5.79E+05)	2.99E+09 (6.01E+08)	1.17E+07 (2.52E+06)	5.28E+06 (5.62E+05)	7.88E+06 (2.32E+06)	**7.27E+04** (**9.55E+04**)
F13	**4.77E+03** (**3.43E+03**)	1.16E+04 (4.17E+03)	1.41E+09 (4.54E+08)	5.92E+04 (1.16E+04)	3.14E+05 (4.78E+04)	1.49E+04 (2.18E+03)	7.17E+03 (5.14E+03)
F14	**2.07E+03** (**2.36E+02**)	2.23E+05 (8.56E+04)	2.89E+05 (1.07E+05)	1.01E+04 (1.44E+03)	9.67E+03 (3.84E+03)	5.22E+04 (1.09E+04)	1.19E+04 (1.20E+04)
F15	1.08E+05 (1.30E+05)	6.12E+03 (3.93E+03)	8.55E+07 (4.76E+07)	1.60E+04 (1.53E+03)	1.20E+05 (1.78E+04)	**2.31E+03** (**4.47E+02**)	9.72E+03 (9.40E+03)
F16	2.32E+03 (3.64E+02)	**2.24E+03** (**1.06E+02**)	3.40E+03 (1.79E+02)	3.36E+03 (2.68E+02)	3.24E+03 (3.59E+02)	3.37E+03 (3.88E+02)	2.73E+03 (3.66E+02)
F18	**8.81E+04** (**3.76E+04**)	3.17E+05 (9.84E+04)	2.52E+06 (1.25E+06)	9.46E+04 (8.78E+03)	1.56E+05 (1.75E+04)	1.06E+06 (2.04E+05)	1.43E+05 (8.95E+04)
F19	7.10E+05 (2.23E+06)	2.56E+04 (1.43E+04)	1.64E+08 (5.89E+07)	9.96E+05 (2.62E+05)	1.30E+06 (2.41E+05)	**4.66E+03** (**9.24E+02**)	9.02E+03 (6.24E+03)
F21	**2.40E+03** (**3.40E+01**)	2.55E+03 (1.67E+01)	2.53E+03 (1.42E+01)	2.56E+03 (7.90E+01)	3.00E+03 (6.62E+01)	2.93E+03 (3.19E+01)	2.56E+03 (6.84E+01)
F22	7.04E+03 (1.39E+03)	**5.76E+03** (**2.45E+02**)	7.95E+03 (2.86E+03)	7.41E+03 (1.66E+02)	8.19E+03 (4.37E+02)	7.69E+03 (7.58E+02)	7.00E+03 (5.86E+02)
F23	**2.74E+03** (**4.10E+01**)	2.82E+03 (1.81E+01)	2.93E+03 (3.20E+01)	3.88E+03 (1.06E+02)	5.00E+03 (2.44E+02)	4.69E+03 (1.57E+02)	3.24E+03 (4.52E+02)
F24	**2.89E+03** (**3.13E+01**)	3.14E+03 (3.17E+01)	3.08E+03 (2.41E+01)	3.46E+03 (1.61E+02)	4.05E+03 (1.29E+02)	4.03E+03 (4.69E+01)	3.19E+03 (1.53E+02)
F25	2.90E+03 (1.30E+01)	2.88E+03 (5.62E-02)	3.03E+03 (1.80E+01)	2.90E+03 (2.02E+00)	**2.88E+03** (**1.72E+00**)	2.91E+03 (1.47E+01)	2.89E+03 (1.25E+01)
F26	5.76E+03 (1.03E+03)	5.40E+03 (1.32E+03)	**3.79E+03** (**9.98E+01**)	5.49E+03 (1.06E+03)	1.24E+04 (1.66E+03)	1.29E+04 (3.37E+03)	6.06E+03 (2.20E+03)
F27	3.24E+03 (1.90E+01)	**3.22E+03** (**3.66E+00**)	3.37E+03 (3.37E+01)	4.94E+03 (2.63E+02)	4.06E+03 (1.56E+02)	3.97E+03 (2.16E+02)	3.40E+03 (1.06E+02)
F28	3.21E+03 (2.44E+01)	3.16E+03 (4.08E+01)	3.45E+03 (3.76E+01)	3.26E+03 (2.91E+00)	3.32E+03 (4.03E+01)	4.93E+03 (3.77E+02)	**3.15E+03** (**6.30E+01**)

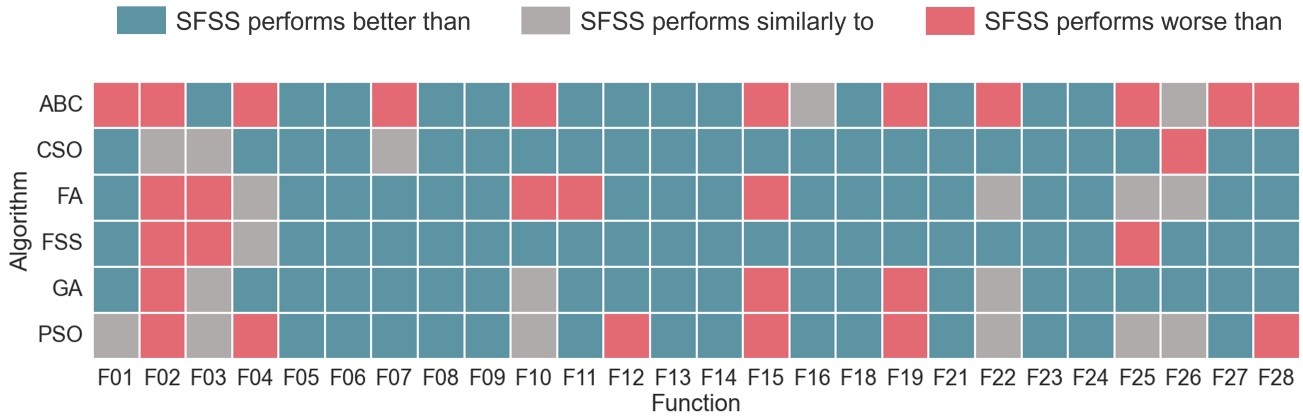

Figure 3: Wilcoxon test results comparing the proposed metaheuristic to others on 26 CEC problems (30D) after 500K fitness evaluations. Blue indicates SFSS superiority, red denotes inferiority, and grey shows no statistical difference.

Although the focus of this proposal was to reduce the complexity of the FSS by decreasing the number of user-defined parameters and fitness evaluations, the modifications introduced also suggest improvements in the algorithm's performance. These improvements could be due to minimising the risk of improved parametrisation and introducing the turbulence mechanism and the redesigned displacement operations. However, more experiments are necessary to evaluate the impact of the modifications proposed in the FSS behaviour.

Regarding the algorithm complexity, we employed a calculation similar to the one adopted at CEC'17 wu2016problem, which can be described as follows:

- Calculate the function complexity T_i by computing time of 10000 evaluations for problem i.
- Compute the algorithm complexity TA_i by computing the time of 10000 evaluations for problem i. To accommodate variations in performance due to the algorithms' stochastic nature, the TA_i is the average of 15 runs.
- The final complexity is given by $AC = (TA_i - T_i)/T_i$

The main difference between this definition and the CEC'17 is that here, we present the complexity per function, while the CEC'17 calculates the overall complexity in all problems in the test suit.

The results of the algorithm's complexity across the benchmark problems are presented in Table 2. As shown in Table 2, the PSO, the CSO, and GA are the algorithms with the lowest complexity, while the FA, FSS and SFSS presented the highest values. As this definition assesses the time required to execute a given number of fitness evaluations, we expect the SFSS to exhibit higher values than the FSS. This result arises from the SFSS combining its operators'complexity within a unique fitness evaluation per iteration. In contrast, the FSS has two evaluations per iteration. In this case, the gains made by reducing the fitness evaluation numbers would be more prominent in scenarios where the objective function has a very high computational cost (i.e. higher than the FSS operators). Future experiments are needed to compare FSS and SFSS in more complex problems, such as hyperparametization tunning of machine learning models.

Table 2: Complexity analysis on 26 benchmarks shows SFSS has higher complexity than FSS, as it condenses all operator computations into one evaluation per iteration, while FSS splits them into two.

Function	SFSS	ABC	CSO	FA	FSS	GA	PSO
F1	8.166	1.937	0.348	5.067	4.615	1.396	0.957
F2	5.416	1.828	0.315	4.874	4.367	1.336	0.937
F3	3.517	0.933	0.112	2.783	2.448	0.577	0.331
F4	5.283	1.091	0.037	3.239	2.913	0.704	0.463
F5	6.422	1.820	0.308	5.059	4.398	1.311	0.915
F6	4.109	1.090	0.155	3.253	2.782	0.741	0.456
F7	3.279	0.902	0.154	2.392	1.992	0.600	0.411
F8	4.869	1.544	0.441	3.937	3.637	1.070	0.774
F9	3.128	0.864	0.144	2.745	2.389	0.588	0.385
F10	2.649	0.780	0.154	2.399	1.962	0.508	0.358
F11	1.533	0.567	0.181	1.531	1.266	0.391	0.292
F12	1.880	0.561	0.197	1.315	1.204	0.370	0.303
F13	1.676	0.544	0.191	1.321	1.110	0.358	0.278
F14	1.305	0.489	0.184	1.252	1.032	0.333	0.244
F15	1.584	0.524	0.172	1.327	1.117	0.350	0.267
F16	1.228	0.413	0.153	1.046	0.897	0.262	0.198
F18	1.253	0.452	0.186	1.093	0.933	0.298	0.226
F19	0.968	0.343	0.162	0.845	0.718	0.227	0.174
F21	0.942	0.306	0.131	0.799	0.734	0.163	0.114
F22	0.850	0.341	0.196	0.741	0.748	0.204	0.161
F23	0.686	0.217	0.135	0.557	0.484	0.111	0.124
F24	0.782	0.287	0.162	0.650	0.576	0.175	0.139
F25	0.755	0.204	0.171	0.606	0.443	0.111	0.077
F26	0.672	0.210	0.159	0.595	0.448	0.119	0.090
F27	0.543	0.123	0.120	0.404	0.331	0.053	0.048
F28	0.661	0.189	0.179	0.484	0.396	0.120	0.098

S4 Conclusion

This paper presented the Simplified Fish School Search, a novel metaheuristic inspired by the Fish School Search (FSS) algorithm, designed to simplify FSS operators and parameters and enhance its performance. The proposed approach modifies the structure of FSS by consolidating the complexity of its operators into a single evaluation per iteration. We conducted experiments using the CEC 2017 benchmark suite to evaluate its effectiveness. The

results show that it could overcome the FSS in most problems analysed (22 of 26) and compete with well-known algorithms such as PSO, ABC, GA, CSO and FA.

The proposed algorithm's performance in unimodal, multimodal, and composition problems was satisfactory, showing the SFSS's versatility. Furthermore, the computational cost from the number of fitness evaluations per individual per iteration was reduced. Reducing the number of calls to the fitness function is essential when dealing with functions with elevated costs. Finally, we reduced the number of parameters, which led to a less user-dependent and problem-dependent algorithm with no parameter specification required besides the population size.

As the main limitations of our work, we list the necessity of in-depth experiments analysing the performance and behaviour changes introduced in the SFSS. Also, although we compared its performance against well-known algorithms, including more recent methods in the study would be desirable. We intend to address these issues in future works. To better evaluate the impact of the simplification proposed, we also plan to analyse the performance of the SFSS in more challenging and computationally expensive optimisation tasks.