December 19th, 2023

A Report on the Shuffled Frog Leaping Algorithm

Kevin Chan
k.chan@yale.edu
S&DS 431
Yale University
New Haven, CT 06511, USA

Editor: Me, Myself, and I

Abstract

The Shuffled Frog Leaping Algorithm (SFLA) was developed by Muzaffar Eusuff and Kevin Lansey in 2006 to solve combinatorial optimization problems [1]. The technique is a combination of the Shuffled Complex Evolution and Particle Swarm Optimization algorithms [2]. In this report, I will first provide a brief overview of the Shuffled Complex Evolution and Particle Swarm Optimization algorithms. I will then explain the Shuffled Frog Leaping Algorithm and contextualize the process in respect to the first two processes. After, I will explain why the SFLA is valid, analyze the global convergence of the algorithm using Markov chains, and compare the superior performance against other established algorithms in the context of neural network training.

1 Introduction

The Shuffled Frog Leaping Algorithm (SFLA) follows a trend of algorithms that have been inspired by the observation of biological animal behavior. The SFLA, evident by its name, aims to imitate the behavior patterns of frogs, where each frog takes into account a crowd of other frogs leaping in a swamp. The swamp itself has multiple points that allow for traversal through the space. While exploring, the frogs are each on the lookout for the location with the highest food quantity available. The frog metaphor naturally mirrors a classic optimization problem where the swamp is the feasible set of solutions, the frogs are candidates for the optimal solution, the location with the highest food quantity available is the optimal solution, and the traversal through the swamp represents the exploration of the candidates to find the optimal solution.

The SFLA has a variety of real world applications along with growing popularity in several research fields. In 2020 alone, there were over 140 published papers utilizing the algorithm, totaling close to 1200 unique papers between the years 2006 to 2020 featuring the algorithm [1]. One area of application for the SFLA is in hydrology, the study of the distribution and movement of water [3]. A notable instance would be the first implementation of the algorithm in 2003, for the design of a water distribution network in an optimized way [2]. The algorithm has also been used for groundwater model calibration problems where such models track water, sediment, and generic pollutants in order to simulate the movement of water from precipitation and snow melt. Water may evaporate, run off on the surface, or infiltrate and move through the soil or ground, making the groundwater models extremely complex in variables and interactions.

2 Shuffled Complex Evolution Method

The Shuffled Complex Evolution Method, also known as the SCE-UA global optimization method, was proposed by Duan et al. from the University of Arizona in 1994. The method offers an effective and efficient optimization technique for calibrating watershed models, provided that the user defines the right algorithmic parameters [4]. The large number of parameters within the model are not directly measurable, thus motivating the need for an optimization algorithm to aid in calibration. Calibration criterion serve as measures of fit between the simulated and observed outputs and are subsequently what algorithms must optimize. At the time, Duan et al. found that contemporary optimization techniques commonly employed were not powerful enough to deal with the response surface conditions encountered during model calibration. The five major characteristics complicating the optimization problem in CRR (conceptual rainfall-runoff) models are:

1. Regions of Attraction - There lies more than one main convergence region
2. Minor local optima - There are many small "pits" in each of the regions
3. Roughness - The problem has a rough response surface with discontinuous derivatives
4. Sensitivity - There is poor and varying sensitivity of response surface in region of optimum and non-linear parameter interaction
5. Shape - The problem is non-convex with long curved ridges

As a result, Duan et al. created the SCE-UA algorithm as a synthesis of four standard concepts: 1) a combination of deterministic and probabilistic approaches, 2) a systematic evolution of a "complex" of points spanning the parameter space in the direction of global improvement, 3) competitive evolution, and 4) complex shuffling. The steps for the SCE-UA are as follows:

1. Generate a sample of s points randomly in the feasible parameter space and compute the criterion value at each of the points. The random generation can be done using a uniform distribution across the space, but if there exists information on the approximate location of the global optimum, one may use a more appropriate probability distribution to reflect the information.
2. Rank the s-many points in order of increasing criterion value such that the first point represents the smallest criterion value and the last point represents the largest (assuming that the goal is to minimize criterion value, otherwise if the goal is to maximize, one can simply reverse the order).
3. Partition the s points into p complexes, with each complex containing m points. Note that the points are partitioned so that the first complex contains every $p(k-1)+1$ ranked point, the second complex contains every $p(k-1)+2$ ranked point, and so on where $k = 1, 2, ..., m$.

4. Evolve each complex according to the competitive evolution (CCE) algorithm which we will discuss later.

5. Shuffle the complexes by combining the points in the evolved complexes into a single sample population. Sort the sample population in order of increasing criterion value, then shuffle the same population into p complexes according to the procedure specified in step 3.

6. Check to see if any of the pre-specified convergence criteria are satisfied. If so, then stop and if not, then continue to step 7.

7. Check the reduction in the number of complexes. If the minimum number of complexes required in the population, p_{min} is less than p, remove the complex with the lowest ranked points; set $p = p - 1$ and $s = p * m$; and return to step 4. If $p_{min} = p$, then return to step 4 as well.

The algorithm is quite simple on a global level. We will now elaborate on step 4 and how the CCE algorithm within each complex works. The CCE algorithm does the following:

1. Construct a sub-complex within a complex by randomly selecting q points using a trapezoidal probability distribution on the complex itself. The distribution will be defined such that the best point has the highest chance of being chosen to form the sub-complex and the worst point has the least chance.

2. Identify the worst point of the sub-complex and compute the centroid of the sub-complex without the worst point.

3. Attempt a reflection step by reflecting the worst point through the centroid. If the newly generated point is within the feasible space, go to step 4. Otherwise, randomly generate a point within the feasible space and go to step 5.

4. If the newly generated point is better than the worst point, replace the worst point by the new point. Go to step 7. Otherwise go to step 5.

5. Attempt a contraction step by computing a point halfway between the centroid and the worst point. If the contraction point is better than the worst point, replace the worst point by the contraction point and go to step 7. Otherwise go to step 6.

6. Since the contraction point is not an improvement, randomly generate a point within the feasible space. Replace the worst point by the randomly generated point.

7. Repeat steps 2-6 α-times, where $\alpha >= 1$ is the number of consecutive offspring generated by the same sub-complex.

8. Repeat steps 1-7 β times, where $\beta >= 1$ is the number of evolution steps taken by each complex before the complexes are shuffled.

One caveat of the CCE-UA algorithm is that the number of points in each complex, m, may take any value greater than or equal to 2. However, if there are too few points within all the complexes, the search algorithm would proceed in a manner similar to the ordinary Simplex procedure, undermining the global search capability of the algorithm. Conversely, if m is set to be too large, the algorithm could result in the excessive use of the computer processing time (CPU) with no certain gain in effectiveness. Previous testing by other researchers have found that the algorithm provides better overall performance in coping with a wide range of optimization problems by setting $m = 2n + 1$ where n is the number of parameters to be optimized on and by varying the number of complexes p. Additionally, these hyperparameters were found to be more effective for CCE-UA performance than just increasing the value of m alone [5].

3 Particle Swarm Optimization Algorithm

The Particle Swarm Optimization Algorithm, created by James Kennedy and Russell Eberhart in 1942, is a method for the optimization of continuous nonlinear functions [6]. The algorithm also serves as a prime example of an optimization method created through the observation of nature, specifically of fish schooling and swarming theory in particular. Sociobiologist E. O. Wilson stated in reference to fish schooling:

"In theory at least, individual members of the school can profit from the discoveries and previous experience of all other members of the school during the search for food. This advantage can become decisive, outweighing the disadvantages of competition for food items, whenever the resource is unpredictably distributed in patches."

The quote aptly describes the core idea of the Particle Swarm Optimization (PSO) algorithm, where the social sharing of information provides an evolutionary advantage. We will discuss the development of the PSO to contextualize how the present version of the algorithm works. Let us first imagine a scenario in which we define a population of "birds" by using points randomly initialized on a torus pixel graph. Each bird will have its own X and Y velocities. The first initial implementation of the PSO algorithm featured a loop where with each iteration/time step, each bird would identify its nearest neighbor and receive the neighbor's X and Y velocities. The result was the flock quickly adapting a unanimous and unchanging direction. To remedy this, a stochastic variable called *craziness* was introduced to add randomness to each bird's velocities. The consequent algorithm created an interesting and "lifelike" simulation appearance.

A later installment, the Cornfield Vector simulation developed by Frank Heppner [7] introduced the idea of a dynamic force. The birds flocked around a "roost", a position on the pixel screen that attracted said birds until they landed on the defined point. The roost eliminated the need of a variable like craziness and allowed for the simulation to visually improve. However, the model only had one roost whereas in reality, birds land on any building or structure that immediately suit their needs. The imperfection of Heppner's algorithm prompted another variation where each bird/agent "remembers" the best value it has achieved and the XY position to achieve such value. We can denote the value as $pbest[]$ and the positions as $pbestx[]$ and $pbesty[]$. Again, we are indexing a 2D array at the moment. Additionally, each agent keeps a record of the globally best position found by any member of the flock, mimicking communication between animals. The implementation was accomplished by assigning the array index of the agent which found the best value to a variable called $gbest$. Thus, $pbestx[gbest]$ and $pbesty[gbest]$ represent the best global X and Y positions respectively. Each bird's $vx[]$ and $vy[]$ are consistently updated as follows with a defined $g_increment$ term:

if $presentx[] > pbestx[gbest]$, then $vx[] = vx[] - rand() * g_{increment}$
if $presentx[] < pbestx[gbest]$, then $vx[] = vx[] + rand() * g_{increment}$
if $presenty[] > pbesty[gbest]$, then $vy[] = vy[] - rand() * g_{increment}$
if $presenty[] < pbesty[gbest]$, then $vy[] = vy[] + rand() * g_{increment}$

Note that when $p\_increment$ and $g\_increment$ are set relatively high, the flock will "be sucked violently into the cornfield", which intuitively makes sense since each point's distance between the optimal point will be more aggressively reduced. When the two are set relatively low, the flock swirls around the goal, realistically approaching the point until all birds land on the "target". When the ability of the PSO to optimize two-dimensional, linear algorithms was confirmed, researchers were able to discover that craziness was an unnecessary component and that the optimization occurs faster without the nearest neighbor velocity matching. The resulting algorithm became visually more akin to a swarm rather than a flock. $pbest$ and $gbest$ are still necessary as $pbest$ resembles autobiographical memory with each bird remembering its own best value and $gbest$ represents publicized knowledge or a group norm that other birds seek to attain [6]. In simulations, a high value of $p\_increment$ relative to $g\_increment$ results in excessive wandering since the birds prefer their personal optimum rather than the global group optimum. Conversely, a high value of $g\_increment$ as opposed to $p\_increment$ leads to the birds flocking to a local optimum prematurely since the birds prioritize the current global value over their own exploration.

Naturally, the next generalization of the PSO would be to a higher dimensional feasible space, where most of the interesting optimization problems are neither two-dimensional nor linear. A logical step to convert the 2D algorithm into a multidimensional algorithm is to turn the one-dimensional arrays in $vx[]$ and $vy[]$ into DxN matrices where D is the number of dimensions and N is the number of agents or candidate solutions. The current version of the PSO algorithm is the following adjustment to velocity:

$$vx[][] = vx[][] \\ +2 * rand() * (pbest[][] - presentx[][]) + 2 * rand() * (pbest[][gbest] - presentx[][])$$

$$vy[][] = vy[][] \\ +2 * rand() * (pbest[][] - presenty[][]) + 2 * rand() * (pbest[][gbest] - presenty[][])$$

Note the absence of $p-$ and $g-$increment due to the inability to determine whether the two terms should be larger or smaller. The stochastic factor in the current PSO is multiplied by 2 to yield a mean of 1, allowing agents to "overshoot" their targets about 50% of the time.

An interesting use case of the PSO algorithm is for feed-forward multi-layer perceptron neural network training [6]. An example is a three-layer neural network for an XOR problem, meaning that we have a function that is not linearly separable [8]. The problem requires two input and one output processing elements (PEs), along with a number of hidden PEs. The 2,3,1 neural network consequently, requires the optimization of 13 parameters. The problem was approached by using the PSO to explore the 13-dimensional space until an average sum-squared error per PE criterion was met. The algorithm performed well and allowed the XOR network to be trained to an $e < 0.05$ criterion, in an average of 30.7 iterations with 20 agents.

4 Shuffled Frog Leaping Algorithm

With an understanding of the SCE-UA and the PSO algorithms, we can now discuss the SFLA. The algorithm is population-based meta-heuristic, meaning that it is a search procedure designed to find a good solution to an optimization problem that is complex and difficult to solve to optimality [9]. The population of candidate solutions can be seen as metaphorical frogs and is split up into subgroups of complexes or memeplexes. The population of frogs can be denoted as:

$$U_1, U_2, ..., U_F \tag{1}$$

where frog i stands for a decision vector variable such that:

$$U_i = U_i^1, U_i^2, ..., U_i^d \tag{2}$$

and d signifies the number of decision variables for each frog. A local search is conducted in each memeplex and after a certain number of evolution stages, the frogs are shuffled across memeplexes to transfer ideas. Both the local and global movements continue until the convergence criteria defined are satisfied. The two phases of the Shuffled Frog Leaping Algorithm are:

1. The global exploration consisting of the following steps:
   (a) Initialize the population by specifying the number of memeplexes m, the number of frogs in each memeplex n, and then generating the population with $F = m * n$.
   (b) When generating the sample F, we must have the frogs with positions:
   $$U_1, U_2, ..., U_F \in \Omega \tag{3}$$
   with Ω being the achievable/feasible space and
   $$\Omega \in S^d \tag{4}$$
   where d signifies the number of decision variables for each frog. We then calculate the value of each frog's position using the fitness function f.
   (c) We rank the F many frogs in descending order based on the fitness function f value and place them in a list X:
   $$X = \{U_i, f_{(i)}, i = 1, ..., F\} \tag{5}$$
   where i = 1 is the index of the best global frog, with which we will denote as P_x, such that $P_x = U_1$.

(d) Divide the frogs into memeplexes where memeplex 1 receives the rank 1 frog, memeplex 2 gets the rank 2 frog, memeplex m gets the rank m frog. Once we have gone through the top m frogs, we assign the $m+1$ frog to memeplex 1, the $m+2$ frog to memeplex 2, and keep partitioning the frogs in this fashion. Note that we are essentially dividing up the optimal value frogs across the memeplexes such that no memeplex will have the all of the highest value frogs.

(e) Evolve the memetic, or the propagation of information, inside each memeplex by entering the phase of local exploration. We will explain the local exploration shortly.

(f) Shuffle the memeplexes.

(g) Verify convergence. Terminate if the convergence criterion has been met, otherwise go back to step (c).

2. The local search strategy plays a crucial role in the algorithm. Each memeplex progression is to continue autonomously N times in step 1d of the global search. The algorithm performs the global shuffling after the memeplexes have been progressed through their local exploration. The steps of the local exploration for each memeplex are as follows:

   (a) Set the counters for the number of memeplexes (m) and the number of evolutionary steps for each memeplex N such that:

   $$im = m + 1 \tag{6}$$

   $$iN = N + 1 \tag{7}$$

   (b) Form a sub-memeplex by choosing frogs within each memeplex. The frogs that will be selected for the sub-memeplex are obtained through a triangular probability distribution where greater weights are assigned to frogs with greater performance (high value) and fewer weights to frogs with lower performance. The weights can be allocated as:

   $$p_n = \frac{2(n+1-j)}{n(n+1)} \quad \text{where} \quad j = 1, 2, ..., n \tag{8}$$

   We form the sub-memeplex by selecting q-many distinct frogs from the n-many frogs within each memeplex and arrange them in descending order such that $iq = 1$ is the index of the best fitness value-yielding position out of the frogs in the sub-memeplex (position is defined as P_B) and $iq = q$ is the index of the frog with the worst fitness-value position P_W.

   (c) Calculate the step size S with the following set of equations:

   $$step\ size\ S = min\{int[rand(P_B - P_W)], S_{max}\} \quad \text{for a positive step} \tag{9}$$

   $$step\ size\ S = max\{int[rand(P_B - P_W)], S_{max}\} \quad \text{for a negative step} \tag{10}$$

where $rand$ represents a random number between the ranges of zero to one and S_{max} represents the maximum step size allowed for a frog to metaphorically "jump". We then perform the update:

$$U_{(q)} = P_W + S \tag{11}$$

to the position of the worst frog where $U_{(q)}$ is a new position for the next iteration.

(d) If the new $U_{(q)}$ exceeds the previous $U_{(q)}$, then we perform the local exploration again, otherwise we progress to the next step.

(e) We now take the global best frog's position P_X and perform the updates:

$$step\ size\ S = min\{int[rand(P_X - P_W)], S_{max}\} \quad \text{for a positive step} \tag{12}$$

$$step\ size\ S = max\{int[rand(P_X - P_W)], S_{max}\} \quad \text{for a negative step} \tag{13}$$

So we are using the best frog's position to obtain a step size in reference to the worst frog's position within the memeplex. We then use equation 11 to calculate the worst frog's new position.

(f) We must now deal with the cases of performing the adjustment on the worst frog.

   i. If $U_{(q)}$ could not resolve inside viable space, then calculate the new performance $f_{(q)}$, or else, move to step f.

   ii. If the value of $f_{(q)}$ was better, then switch the old frog with the new one and move to the next step g, otherwise move to step f.

(g) Generate a new frog randomly.

(h) Upgrade the memeplex

(i) Move to step 2b if $iN < N$ since we need to perform more memeplex evolutions.

(j) Move to step 2a if $im < m$ since we have created more memeplexes. Otherwise, move back the global search to shuffle memeplexes.

The establishment of the Shuffled Complex Evolution-UA and the Particle Swarm Optimization algorithms were necessary since both algorithms' ideas were contained within the SFLA (also, it helped me understand what the SFLA algorithm was even doing in the first place). We observe how the general sample solutions are partitioned into smaller groups (memeplexes vs. complexes) and within the sub-groups, an algorithm is performed (local search vs. CCE). The influence of the PSO algorithm comes from the linear update of the worst frog after each local search step. The linear updates serve to make the worst point within the sub-complex more optimal and as a result, with enough iterations, will push the general population towards optimality. The memetic updates are also parallels of the SCE-UA updates. The most evident critique of the SFLA algorithm however, is how the algorithm does not consider the influence of evolutionary algebra on the frog's moving step size due to the calculation using only the best and worst positions of the frogs within the sub-memeplex. Additionally, only the worst individual frogs in each sub-memeplex are updated and the positions of other frogs are not changed [10]. Work has been done to improve the classic SFLA to achieve better precision and convergence speed, but for the scope of this project, we will not discuss the more recently-developed algorithms.

Like with any other heuristic algorithm, selecting parameters is crucial for the SFLA to succeed. The SFLA has the following hyperparameters that the user must tune:

1. m, signifying the number of memeplexes we define
2. n, signifying the number of frogs within a memeplex
3. q, signifying the number of frogs in a sub-memeplex
4. N, signifying the number of evolution steps between two consecutive shuffles for a memeplex
5. S_{max}, signifying the size of the maximum step permitted in the course of an evolutionary step

A smaller value of F, where $F = m*n$, will increase the probability of locating the global optimum or near optimum. A larger value of F will lead to a more computationally intensive model. If m, n, and q, which are the number of memeplexes, the number of elements in a memeplex, and the number of elements per sub-memeplex respectively, are small, then the advantage of the local memetic evolution strategy is lost. Additionally, if q is small, then there would be a slower exchange of information since a lower number of candidate solutions would be chosen in a sub-memeplex, resulting in longer solution computation. If m, n, and q are increased though, the searching time of the algorithm increases. For N, the number of evolutionary steps per memeplex before shuffling, a lower value means the memeplexes will be shuffled regularly, thus decreasing the exchange of ideas on a local scale or within memeplexes. If N is too large, then each memeplex can shrink into a local optimum. The N parameter is thus a balance between the global and local information. If S_{max} , the step size, is small, then global exploration is decreased, leading to what is essentially a local searching algorithm. If S_{max} is too large, then the true optimum could be lacking [2].

Thus, the Shuffled Frog Leap Algorithm works because of how it divides a large feasible set into memeplexes, each with the same number of candidate solutions, divides these memeplexes further into sub-memeplexes with only some candidate solutions from the original memeplex, improves the values of the solutions within the sub-memeplex and repeatedly does this for N times. Once the memeplex has locally evolved, the constraints of the memeplexes disappear, and the candidate solutions are reordered into new memeplexes. This process occurs until convergence criterion are met and the beauty of the algorithm comes from how the memeplexes and sub-memeplexes optimize themselves first and how the shuffling allows the candidate solutions to explore and not get caught in local minima while gradually optimizing the entire space.

5 SFLA Convergence

We will now mathematically analyze the convergence of the SFLA by using the proof from the paper "Convergence and Parameters Analysis of Shuffled Frog Leaping Algorithm" by

Lianguo Wang and Yaxing Gong in 2013 [11]. We begin with the definition of matrix P:

Let P be a transition probability matrix of size k by k, with $\{p_{i,j} : i, j = 1, 2, ..., k\}$.

The stochastic process $(X_0, X_1, ...)$ of a finite state space $S = s_1, s_2, ..., s_k$ is called a time-homogeneous Markov chain of P if for any given n, $i, j \in 1, 2, ..., k$, with $i_0, i_1, ..., i_{n-1} \in 1, 2, ..., k$:

$$P(X_{n+1} = s_j|X_0 = s_{i0}, X_1 = s_{i1}, ..., X_{n-1} = s_{i(n-1)}, X_n = s_{in}) = P(X_{n+1} = s_j|X_n = s_i) = P_{ij} \tag{14}$$

So the past iterations have no influence on the probability of the current iteration with the exception of the iteration directly before.

The time-homogeneous Markov chain $(X_0, X_1, ...)$ of transition probability matrix P is called an irreducible one, if for any given $s_i, s_j \in S$:

$$P(X_{m+n} = s_j|X_m = s_i) > 0 \tag{15}$$

is true. If the Markov chain $(X_0, X_1, ...)$ is time-homogeneous and irreducible, then:

$$P(X_{m+n} = s_j|X_m = s_i) = (P^n)_{i,j} > 0 \tag{16}$$

The Markov chain $(X_0, X_1, ...)$ is said to be nonperiodic if for any given $s_i \in S$ if:

$$d(s_i) = gcd\{n \geq 1 : (P^n)_{i,j} > 0\} = 1 \tag{17}$$

where $gcd\{a1, a2, ...\}$ is the greatest common divisor of $a1, a2, ...$.

A state $s_i \in S$ has period $d(s_i)$ if any return to state s_i must occur in multiples of $d(s_i)$ time steps.

We now state the following lemmas:

Lemma 1 Strictly positive Markov chains are irreducible and aperiodic.

Lemma 2 If P is a n order, reducible and stochastic matrix which can be switched, by the same row and column transformation:

$$\begin{bmatrix} C & 0 \\ R & T \end{bmatrix}$$

which we can define as $P^{'}$, where C is a m order primitive matrix, and $R, T \neq 0$. Thus, we can have:

$$P^{'\infty} = \lim_{k\to\infty} P^{'k} \tag{18}$$

and:

$$P^{'k} = \begin{bmatrix} C^k & 0 \\ \sum_{i=0}^{k-1} T^i R C^{k-i} & T^k \end{bmatrix}$$

and thus:

$$\lim_{k\to\infty} P'^k = \begin{bmatrix} C^\infty & 0 \\ R^\infty & 0 \end{bmatrix}$$

resulting in a stable stochastic matrix and $P^{'\infty} = 1'p^{'\infty}$. Additionally, $p^{'\infty} = p^{'0}$; and $P^{'\infty}$ is determined uniquely and independent of initial distribution. Lastly, $p_i^{'\infty} > 0, 1 \leq i \leq m; p_i^{'\infty} = 0, m \leq i \leq n$.

We now talk about the state space analysis. We can express component x_i of the frog $X = (x_1, x_2, ..., x_n)$ as M-bit binary series and we quantize $[\underline{x_i}, \overline{x_i}]$ to 2^M discrete values, where the precision can be expressed as $\epsilon = (\underline{x_i} - \overline{x_i})/2^M$. Thus, we can conclude that the convergence of SFLA can be analyzed by real-number encoding! If the required precision is ϵ, then the searching space S could be treated as a discrete space. Its size is:

$$|S| = \prod_{i=1}^{n} \lceil (\underline{x_i} - \overline{x_i})/\epsilon \rceil \tag{19}$$

We can define the fitness of each element $X_i \in S_i$ as $Fitness(X)$. Since $F = \{Fitness(X)|X \in S\}$ and since $|F| \leq |S|$, then F can be expressed as $F = \{F_1, F_2, ..., F_{|F|}\}$, where $F_1 \geq F_2 \geq ... \geq F_{|F|}$. S can be divided into several nonempty subsets $\{S_i|i = 1, 2, ..., |F|\}$ by fitness values and S_i has the form:

$$S_i = \{X|X \in S \text{ and } Fitness(X) = F_i\} \tag{20}$$

$$\sum_{i=0}^{|F|} |S_i| = |S|; S_i \neq \varnothing, \forall i \in \{1, 2, ..., |F|\}; S_i \cap S_j = \varnothing, \forall i \neq j; \bigcup_{i=1}^{|F|} S_i = S \tag{21}$$

For any given two elements $X_i \in S_i$ and $X_j \in S_j$, the following must also be true:

$$Fitness(X_i) > Fitness(X_j), \text{if } i < j \tag{22}$$

$$Fitness(X_i) = Fitness(X_j), \text{if } i = j \tag{23}$$

$$Fitness(X_i) < Fitness(X_j), \text{if } i > j \tag{24}$$

We know F_1 is the global optimum F^*, and set S_1 contains all individuals whose fitness is equal to F^*. For the entire length of the evolutionary process of SFLA, we note that the individual number N of population $p = \{X_1, X_2, ..., X_N\}$ remains unchanged.

Now let P be the whole swarm sets. Since the individuals in a swarm could be the same, the possible population number is:

$$|P| = \binom{|S| + N - 1}{N} \tag{25}$$

If we want to measure the population quality, we define the fitness of swarm P as:

$$Fitness(P) = max\{f(X_i)|i = 1, 2, ..., N\} \tag{26}$$

Thus, $F_{|F|} \leq Fitness(p) \leq F_1, \forall p \in P$. Thus, we can divide P into nonempty sets $\{P_i\}$:

$$P_i = \{p | p \in P \text{ and } Fitness(p) = F_i\}, i = 1, 2, ..., |F| \tag{27}$$

$$\sum_{i=1}^{|F|} |P_i| = |P|; P_i \neq \varnothing, \forall i \in \{1, 2, ..., |F|\}; P_i \cap P_j \neq \varnothing, \forall i \neq j; \bigcup_{i=1}^{|F|} P_i = P \tag{28}$$

where the set P_1 contains all individuals whose fitness is F^*. Now let P_{ij} be the jth swarm of P_i, $i = 1, 2, ..., |F|, j = 1, 2, ..., |P_i|$. The state transitions from P_{ij} to P_{kl} can be expressed as $P_{ij} \to P_{kl}$ under the influence of the evolutionary operators of SFLA. Let $p_{ij.kl}$ be the transition probability from P_{ij} to P_{kl}, and $p_{ij.k}$ be the transition probability from P_{ij} to any swarm of P_k, $p_{i.k}$ be the transition probability from any swarm of P_i to any swarm of P_k. Thus, the following equation is true:

$$p_{ij.k} = \sum_{l=1}^{|P_k|} p_{ij.kl}, \sum_{l=1}^{|F|} p_{ij.k} = 1, p_{i.k} \geq p_{ij.k} \tag{29}$$

Now let us define an evolutionary algorithm converging to the global optimization solution if and only if:

$$\lim_{t \to \infty} Pr\{Fitness(P^t) = F^*\} = 1 \tag{30}$$

where Pr is probability; P^t is the t'th generation population.

Now let us state the following theorems, which we will not prove for brevity of this section.

Theorem 1 *The searching process of SFLA is a time-homogeneous Markov chain.*

Theorem 2 *In SFLA, $\forall i, k\{1, 2, ..., |F|\}$, the following is true:*

$$p_{i.k} > 0, k \leq i \tag{31}$$

$$p_{i.k} = 0, k > i \tag{32}$$

Now we can finally prove the theorem:

Theorem 3 *SFLA has global convergence.*

Proof With the result of Theorem 1, each $P_i, i = 1, 2, ..., |F|$, can be regarded as a state of the finite Markov chain. We obtain the transition matrix of the Markov chain as the following thanks to the conclusion of Theorem 2:

$$P' = \begin{bmatrix} p_{1.1} & p_{1.2} & \cdots & 0 \\ p_{2.1} & p_{2.2} & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ p_{|F|.1} & p_{|F|.2} & \cdots & p_{|F|.|F|} \end{bmatrix}$$

and we know the matrix is equivalent to:

$$\begin{bmatrix} C & 0 \\ R & T \end{bmatrix}$$

where $R = (p_{2.1}, p_{3.1}, ..., p_{|F|.1})^T > 0, \mathbf{T} \neq \mathbf{0}, C = (p_{1.1}) = (1) \neq 0$. From Lemma 2, we obtain:

$$P'^{\infty} = \lim_{k \to \infty} P'^{k} \tag{33}$$

and:

$$P'^{k} = \begin{bmatrix} C^k & 0 \\ \sum_{i=0}^{k-1} T^i R C^{k-i} & T^k \end{bmatrix}$$

and thus:

$$\lim_{k \to \infty} P'^{k} = \begin{bmatrix} C^{\infty} & 0 \\ R^{\infty} & 0 \end{bmatrix}$$

where in this instance, $C^{\infty} = 1, R^{\infty} = (1, 1, ..., 1)^T$. Since we have showed that P'^{∞} is a stable stochastic matrix, we know it is equal to:

$$P'^{\infty} = \begin{bmatrix} 1 & 0 & ... & 0 \\ 1 & 0 & ... & 0 \\ ... & ... & ... & ... \\ 1 & 0 & ... & 0 \end{bmatrix}$$

illustrating that the probability of an individual staying at a non-optimal solution approaches zero or in other words, when time t tends to infinity. Thus, SFLA will converge to the global optimum with probability of 1 and we can obtain the following formula from the definition of an evolutionary algorithm converging:

$$\lim_{t \to \infty} Pr\{Fitness(P^t) = F^*\} = 1 \tag{34}$$

completing the proof! Note that while global convergence is guaranteed, we are showing that it is guaranteed for time equals infinity. In application, we do not want to wait for time $= \infty$ as that is unreasonable. So while SFLA does eventually converge, it might take quite a long time for convergence to occur.

6 SFLA Comparison to Other Justified Algorithms

For comparison with other optimization methods, we can look to Saadi et al.'s paper "Investigation of Effectiveness of Shuffled Frog-Leaping Optimizer in Training a Convolutional Neural Network". As the name suggests, Saadi et al. utilized the SFLA to optimize weight and biases of a CNN model during training. The researchers used four different data sets to test the SFLA performance and compared the algorithm to commonly used evolutionary trainers, namely Whale Optimization Algorithm (WO), Bacteria Swarm Foraging Optimization (BFSO), and Ant Colony Optimization (ACO). The four standard classification sets used in the study were the OxFord Flowers 17, the OxFord Flowers 102, Caltech/UCSD Birds, and Caltech 101 Airplanes. We list the training performances for the OxFord Flowers 17 data set below [12]:

Technique	MSE (AVE $\pm STD$)	Classification rate	Training time
LeNet 5	0.190425 ± 0.031687	88%	14 minutes
ACO-LeNet 5	0.121689 ± 0.011574	90%	16 minutes
BFSO-LeNet 5	0.085050 ± 0.034945	91%	25 minutes
WO-LeNet 5	0.032228 ± 0.039778	94%	18 minutes
SFLA-LeNet 5	0.009210 ± .039100	97%	23 minutes

The other data sets saw similar trend in performance where the SFLA-LeNet 5 would have a higher classification rate, longer training time, and lower MSE compared to the other models. We can clearly see the validity of the SFLA in real-world applications.

7 Conclusion

We have discussed the theory, motivation, and the implementation of the Shuffled Complex Evolution, Particle Swarm Optimization, and Shuffled Frog Leaping algorithms. The SFLA specifically, is an excellent optimization method for combinatorial optimization problems and, as previous research has shown, for model calibration problems such as specifying neural network parameters. We have shown the global convergence of SFLA through mathematical analysis and why, unsurprisingly, the SFLA is so widespread across different fields from machine learning to hydrology.

8 Special Thanks

Thank you to Professor Yang for his patience and his prioritization of our learning. Thank you to Jorge and Connor for their office hours. Without them, my pset grades would have been very non-optimal and not in my preferred feasible set. Thank you!

References

1) Eusuff, M., Lansey, K., and Pasha, F. 2004. Shuffled frog-leaping algorithm: a memetic meta-heuristic for discrete optimization. *Engineering Optimization* 38. 129-154. doi: 10.1080/03052150500384759

2) Maaroof, B. et al. 2022. Current Studies and Applications of Shuffled Frog Leaping Algorithm: A Review. *Archives of Computational Methods in Engineering* 29. 3459-3474. doi: 10.1007/s11831-021-09707-2Abs1

3) National Geographic Society. 2023. Hydrology. *National Geographic.*
Link: https://education.nationalgeographic.org/resource/hydrology/

4) Duan, Q., Sorooshian, S., and Gupta, V. 1994. Optimal use of the SCE-UA global optimization method for calibrating watershed models. *Journal of Hydrology* 158. 265-284. doi: 10.1016/0022-1694(94)90057-4

5) Duan, Q., Sorooshian, S., and Gupta, V. 1993. Shuffled complex evolution approach for effective and efficient global minimization. *Journal of Optimization Theory and Applications* 76. 501-521. doi: 10.1007/BF00939380

6) Kennedy, J. and Eberhart, R. 1995. Particle swarm optimization. *Proceedings of ICNN'95 - International Conference on Neural Networks.* doi: 10.1109/ICNN.1995.488968

7) Heppner, F. and Grenander, U. 1990. A Stochastic Nonlinear Model for Coordinate Bird Flocks. *American Association for the Advancement of Science.* 233-238. Link: https://www.researchgate.net/publication/216300775_A_Stochastic_Nonlinear_Model_for _Coordinate_Bird_Flocks

8) Quratulain Memon. 2023. XOR problem in neural network. *educative.*
Link: https://www.educative.io/answers/xor-problem-in-neural-network

9) Srinivasan Balan. Metaheuristics in Optimization: Algorithmic Perspective. *OR/MS Tomorrow.* Link: https://www.informs.org/Publications/OR-MS-Tomorrow/Metaheuristics-in-Optimization-Algorithmic-Perspective: :text=A

10) Wang, Z. et al. 2019. Research on Improved Strategy of Shuffled Frog Leaping Algorithm. Youth Academic Annual Conference of Chinese Association of Automation. doi: 10.1109/YAC.2019.8787721

11) Wang, L. and Gong, Y. 2013. Convergence and Parameters Analysis of Shuffled Frog Leaping Algorithm. International Conference on Artificial Intelligence and Software Engineering. doi: 10.2991/icaise.2013.17

12) Saadi, S. et al. 2022. Investigation of Effectiveness of Shuffled Frog-Leaping Optimizer in Training a Convolutional Neural Network. *Journal of Healthcare Engineering.* doi: 10.1155/2022/4703682