Lion Optimization Algorithm

Tyrel Dogup, Deo Fetalvero

December 17, 2018

1 Overview

Solving complex optimization problems has been a hot topic the past decade and has garnered large followings most of which are practitioners and researchers. This attention has brought upon the development of new metaheuristic algorithms, many of which are inspired by various phenomena demonstrated by nature. The paper proposes a new population based algorithm, the Lion Optimization Algorithm (LOA) [5]. The distinct lifestyle of lions and their characteristics of utilizing cooperation was made as the motivational basis for the development of this optimization algorithm. The algorithm is also tested against benchmark problems sourced from literature and whose primary solutions was compared with the results of the test. The results also confirm the performance of this algorithm alongside other algorithms used in the paper.

2 Optimization

Basically, optimization is searching for the best solutions out of all possible solutions. A best solution can be defined regarding either the most of some measure of success (e.g. revenue) or the least of another measure (e.g. cost). We can be looking at a group of answers or one answer from a set. In solving optimization problems, one's goal is to minimize or maximize a result variable of a function by trying out different parameters for input.

3 Optimization Algorithms

Optimization algorithms can be divided into two major categories, as exact and approximate [1]. Exact algorithms guarantee that the optimal solution to the problem will be found in a finite amount of time. There are, however, harder optimization problems that requires the searching of very large solution sets and thus making it impractical to use exact algorithms. An example of this is the travelling salesman problem, whereby a salesman is tasked to plan a path that visits a series of cities exactly once and return to his starting point with minimum distance travelled. In this problem, the number of possible paths that the salesman can take grows factorially as the number of cities increases. It is possible to solve this problem using brute-force method but it would take a lot of time. As such, the usage of approximate algorithms are necessitated. These algorithms do not guarantee that the optimal solution will be found, but it can find an approximate (sometimes exact) solution to the problem in a relatively short amount of time, sometimes using a less computationally intensive method. Approximate algorithms can be further divided into two major categories as heuristic and metaheuristic algorithms [1].
Heuristic algorithms are problem-dependent techniques that approximate the solution to a problem using readily available information. Meaning, they try to take advantage of the particularities of the problem to find a solution. Applications of these algorithms include finding the best move in a chess game,

solving a tic-tac-toe puzzle, and pathfinding. In these examples, the underlying concepts of the problem are first analyzed and then used to guide the algorithm in searching for a solution.
Metaheuristic algorithms, on the other hand, requires minimal or no assumptions about the problem being solved. They can be tailored to optimize a specific problem which makes them applicable to a wide variety of problems. A metaheuristic algorithm optimizes a problem by iteratively improving a candidate solution until a desired quality is achieved. Metaheuristic algorithms often employ mechanisms to escape from being stuck in a local optimum and thus making them more likely to obtain the global optimum solution.
An example of an algorithm that can often get stuck in a local optimum is the hill climbing algorithm. The hill-climbing algorithm starts with an arbitrary solution and makes iterative changes to it. If, after an iteration, the current solution state is found to be worse than its previous state, then it is reverted back. Otherwise, the solution is retained and is changed again. This process is repeated until no further improvements can be done to the solution. The problem with this method is that it is greedy, meaning that the solution achieved can be the best solution out of all its neighbouring solution, but not out of the entire solution space. This can be visualized by considering all possible states of the solution laid out on the surface of a landscape, wherein the height of any point on the landscape represents the optimality of the solution state at that point. Say we have two hills on that landscape, one of them higher than the other. The higher hill represents the global optimum of the solution space, while the lower hill represents the local optimum. If the initial solution state is near the smaller hill, it is more likely that the algorithm will "climb" the smaller hill. And since, it only accepts a state that is better than the previous one, it will not be able to go down the hill and climb the higher hill. Thus, it can get stuck in that local optimum. Modern algorithms have mechanisms that

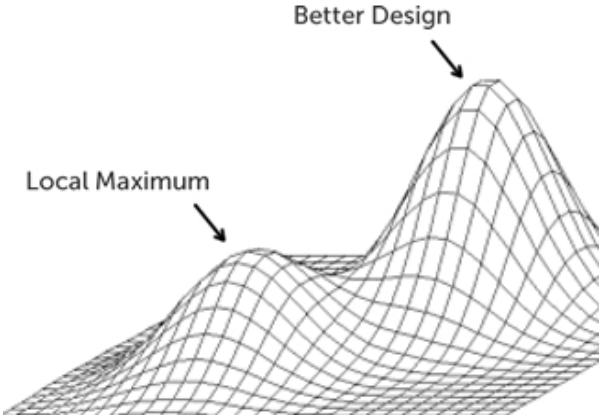


Figure 1: Hill-climbing algorithm local optimum

tackle this problem. An example of an algorithm that employ such mechanisms is simulated annealing. The simulated annealing is very similar to hill-climbing except that it occassionally accepts solutions that are worse than the current. Over time, the probability of accepting such solutions decreases. This allows for the avoidance of getting stuck in a local optimum.
The simulated annealing algorithm is modeled after the cooling process of a molten substance, and is one of many algorithms based on natural processes. Others include the genetic algorithm, ant colony optimization, and particle swarm optimization. For centuries humans have relied on nature to find the most appropriate solutions to problems. That's why in the past decades, computer scientists has turned to nature to develop novel algorithms.

4 No Free Lunch Theorems

Since there are already so many optimization algorithms, what's the point of making a new one? The No Free Lunch Theorems were introduced in 1997 by Wolpert and Macready to address the need for newer optimization algorithms. The first of their theorems states that for any pair of algorithms a_1 and a_2, iterated m times,

$$\sum_f P(d_m^y|f, m, a_1) = \sum_f P(d_m^y|f, m, a_2),$$

where d_m^y denotes the set of size m of the cost values $y \in Y$ associated to the input values $x \in X, f : X \to Y$ is the function being optimized and $P(d_m^y|f, m, a)$ is the conditional probability of obtaining a given sequence of cost values from algorithm a run m times on function f. Essentially, this says that when all functions f are equally likely to be used, the probability of observing an arbitrary sequence of cost values over the course of optimization is independent of the algorithm. This theorem indicates that if an algorithm performs better than another algorithm on some class of problems, then it must perform worse on the remaining problems. Consequently, this implies that there is no general-purpose optimization algorithm that is universally superior than the rest. Hence, the development of newer optimization algorithms is still needed.

5 Optimization: Genetic Algorithm

The genetic algorithm [3] is modeled after Darwin's theory of evolution. An initial population is initialized in which each individual represents a solution to the problem. Through a series of selection, crossover, and mutation, successive populations are generated until an individual with a desired fitness is obtained. Fitness is defined as how well an individual can solve the given problem.
Selection pertains to the process in which individuals with the best fitness are selected and allowed to pass their genes (or properties. E.g. bits) to the next

generation. The individuals with the highest fitness have a higher chance of being selected.
Crossover is when genes of the selected individuals are interchanged with each other. The results are called offspring and are added to the next generation.
Mutation refers to when genes of the offspring are subject to mutate or change (e.g. Binary value from 1 to 0). This further diversifies the sample space and consequently prevents the algorithm from converging early.

6 Optimization: Particle Swarm Optimization

The Particle Swarm Optimization [2] algorithm is modeled after the movement of a swarm as a whole (e.g. A flock of birds collectively foraging for food). In this algorithm, an initial set of particles is first initialized with each particle having a random initial velocity. These particles are then flung through hyperspace where each of their position represents a solution to the problem at hand. Each of these solutions are evaluated regarding their fitness and a particle's current most fit solution is stored as its 'pbest'. The current best solution attained by the entire set particles is also stored; this is called 'gbest'. For each time step of the algorithm, each particle is accelerated towards its 'pbest' and the 'gbest'. Eventually, these particles will converge around an optimal solution.

7 Optimization: Artificial Immune Systems

Artificial immune systems are a classification of rule based machine learning systems inspired by the vertebrate immune systems. This systems model the learning and memory for use in problem solving. These systems adapt to what they learn in the environment to become better at solving problems. Compared to GA these methods use less mutation per generation whenever the fitness becomes better.

8 Optimization: Ant Colony Optimization

Ants can find food faster by utilizing shorter paths found by the other ants in its colony. An ant would leave more pheromones when it has reached the food faster so other ants would be more inclined to use the path. This also applies to optimization by using memory and prioritizing directions with more incentives.

9 Optimization: Marriage in Honey Bee Optimization

The model simulates the evolution of honey-bees starting with a solitary colony (single queen without a family) to the emergence of an eusocial colony (one or more queens with a family).

10 Optimization: Lion Pride Optimizer

Previous works such as the Lion Pride Optimizer was inspired by this brutal competition of male lions whom also plays an important role for the persistence of the pride. In the work, the optimization chooses two of the best points in a "pride" and each "female" with a mating coefficient will create 4 offsprings only to choose one at last based on the best male in the pride. The optimization also uses safeguards to prevent stagnation in the pride by either replacing all members in the pride or resetting the search space.

11 Optimization: Lion's Algorithm

The lion's algorithm by Rajakumar [4] is another inspiration for the LOA. This algorithm is modeled after the territorial behavior of a lion pride, where the pride represents the solution space and a lion represents a solution. The pride is first initialized with one male and one female lion. Through mating, four cubs are generated as a result of single point crossover with dual possibilities. Four more cubs are generated from the mutation of these cubs, totalling eight cubs. These cubs are then grouped according to gender, and the weakest cubs are killed. A cub needs 2-4 years to reach maturity and so the territorial lions must defend the territory for the same number of years. During this time, nomadic lions may invade the pride. For each year, a nomadic lion is generated to test the strength of the pride. If the nomadic lion is found to be stronger than the territorial lions, the nomadic lion takes over the pride and kills the territorial lions' cubs. If the cubs survive and they mature, the best male and female lions take over the entire pride while the rest are killed.

12 Applications

Researchers have made good use of optimization problems such as scheduling problems, data clustering, image and video processing, tuning of neural networks, and pattern recognition.

13 Application: Scheduling Problems

Scheduling problems are problems where different difficulty on jobs would take different amount of time that will be processed by different nodes. The problem is to minimize the time that all the nodes would simultaneously get to be finished. Optimization helps find the best job to node assignment to minimize the mean time.

14 Application: Data Clustering

Data clustering or cluster analysis is the way to group a set of objects such that objects with the most similarity a group together in clusters. The objects may have one or more properties to identify and the groups may have smaller subgroups that one can also classify. Optimization helps identify the best clustering of an object based on its parameters.

15 Application: Image and Video Processing

Image and video processing is the process of adding metadata to an image or video based to what its visual content actually is. An image or video in a computer is represented as pixels or boxes of colors that is displayed on the screen of the viewer. These pixels individually cannot determine the actual content, which is significant to the viewer, of the image and video. Optimization algorithms help computers identify what pixels in an image or video actually represents to the viewer. Such algorithms help with edge detection, segmentation, representation and description of the parts of an image or video.

16 Application: Tuning of neural networks

Neural networks can be tuned by its parameters. These parameters talked about are parameters that are constant throughout the run of the network. This parameters may be tuned to get the best performance out of neural network which may also drive the network to learn faster, slower or not at all. Optimization helps to find the best parameters that will drive the system's performance.

17 Application: Pattern Recognition

Patterns can be also found in data. These patterns can help add to the metadata of that data. Pattern recognition finds those patterns that can be found off the data. Optimization helps identify patterns that best identify a given data.

18 Inspiration for the algorithm

Lions have displayed cooperation and antagonism especially in hunting. Lions are also socially inclined meaning that they also organize information that other lions have collected and use them for their benefit. Male lions have radically different social behavior and appearance than the female lions and v.v. The lions can also be classified if they're residents or nomads. Resident lions create groups called prides, establish their territories and flourish there while nomads take what they need in an area then finds another area to pillage not establishing territories. A pride typically would include five females have cubs of both sexes

and one or more adult male lions. As young males would grow they would separate from their birth pride and establish their own prides. Nomads, who doesn't establish territories, would move about sporadically (whenever they want) and either in pairs or singularly. Lions usually hunt together in prides. Female lions would work together to surround and swiftly catch the prey. There could also be a hunter female lion who would go out of territory to hunt on their own while the other members of the pride would wait for the lioness to return. But still, coordinated group hunting would bring greater success in prey hunts. Lions do go mate anytime around the year and females can have more than one reproductive cycle each year. A lioness can also mate with more than one lion when in heat. Additionally, to mark their territory the pride would place urine all over the place to drive away others who would intrude.

19 Idea for the algorithm

The initially proposed algorithm started an initial population formed by a set of solutions randomly generated labelled as Lions.A percentage $\%N$ of the initial population of solutions are selected as 'Nomad Lions' while the rest are the 'Resident Lions'. While the nomad lions are individually grouped, the resident lions are then further divided into partitions called 'Prides' where a percentage $\%S$ is percentage of the females in the group but in nomad lions, this percentage is reversed, where $\%S$ will be used to identify the males in the nomad lions. Each lion will have a variable pertaining to the best obtained solution for every passing iteration that will be called best visited position and will be updated regularly for every iteration. In each pride, a few random females will be selected to go hunting. These females will encircle the prey and catch it. The males in the pride will roam the territory. The females may mate with one or more resident males then a young male is created. These males may establish their own prides and territory later or may become a nomad. The nomad lions roams around the search space to find better (places) solutions. A nomad lion may invade and replace a resident male in a pride, driving out that resident male (replacement). Also, a female lion may also migrate to another pride or become a nomad herself. Weak lions, who have not found better prey (solutions) where there is no competition, will die or be killed (stagnation). The process will go on until the stopping condition is satisfied.

20 Opposition Based Learning

Opposition based learning is a novel idea of using opposite entities to arrive to better solutions or to arrive at the best solution faster. OBL is used to arrive to the best solution faster than the naive method. In Genetic Algorithm, OBL is also used to get better solutions by allowing multiple best solutions determined using the cost or revenue function to try predict better solutions influenced by the selected solutions' genes. A basic example of OBL is finding a solution X in

a one dimensional solution set. For the solution $\hat{X}$, we have an estimate $\hat{X}$ that approaches X from the left. As we optimize to the solution X, the difference between the solution X and the estimate $\hat{X}$ will get less and less as $\hat{X}$ gets near to the solution X. Suppose we get to X at a certain time, which is the time from the start of the optimization to whenever the estimate $\hat{X}$ will be equal to the solution X. If we use OBL and add another estimate $\hat{X}_2$ that would approach X from the right then the time it takes for the optimization to arrive at solution X would be the maximum between the time it takes for $\hat{X}$ or $\hat{X}_2$ to reach solution X.

21 Proposed Algorithm: Initialization

The first step of the algorithm is to randomly generate solutions called Lions with a population of N. In a Nvar dimensional search space optimization problem a Lion is represented as

$$Lion = [x_1, x_2, ..., x_{Nvar}]$$

As most optimization algorithms, there will be a given cost function, where the fitness value of a solution can be gauged.

$$\text{fitness} = f(\text{Lion}) = f(x_1, x_2, ..., x_{Nvar})$$

Along with generating the solutions, a percentage of N will be selected as nomad lions and the rest would be divided into a number P of prides. The solutions in the pride will have a specific gender which will identify their role in finding solutions. A percentage S of the prides in the population are labeled as females (others are males) while in nomads will have the ratio reversed where $1 - S$ will be the percentage of females in the nomads. The percentage $\%S$ is typically chosen between 75 to 90 percent.
The positions of every lion (solution) is random of which is uniformally distributed across the entire search space. Until the percentage of nomad reaches the percentage threshold, lions would be labeled as nomads. The remaining lions would then be labeled as pride lions and would be divided using a specific group size. Each pride lion would be added to the group until the desired group sized would be reached, proceeding to the next group if filled. Until the gender percentage threshold is reached the lions in the pride will be labeled as female otherwise as a male. The same shall be done in labeling the gender of the nomads.

22 Proposed Algorithm: Hunting

In a pride, females would look for a prey to provide food for the pride. The females would fill specific roles to execute certain strategies to encircle the prey and catch it. In general, lions would approximately follow a pattern in hunting. Stander divided lions into seven different stalking roles which would be grouped

into by the Left Wings, Centers and Right Wings. Left Wings and Right Wings both attack the prey from opposite directions in which the idea of Opposition Based Learning is utilized which also is proven to effectively solve optimization problems. The group with members of the highest finesses (highest revenue, lowest cost) is considered as the Centers while the other two groups are considered as Left and Right Wings. In the algorithm, the hunters are divided into three groups: Left, Right and Center. The group with highest cumulative finesses is considered to be the Center and the other groups would be Left and Right Wings. A dummy prey would move to a new position and escape as

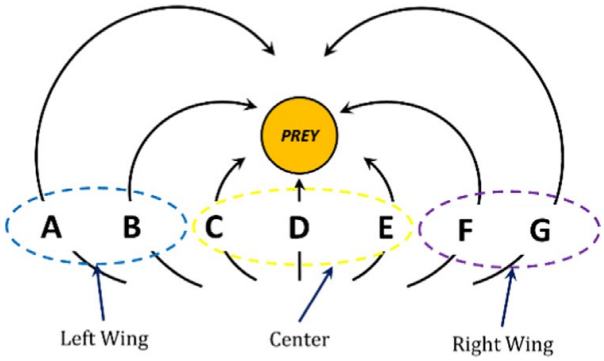


Figure 2: Generalized lion hunting behavior

follows:

$$\text{PREY}' = \text{PREY} + \text{rand}(0,1) \times \text{PI} \times (\text{PREY} - \text{Hunter})$$

where PREY is the current position of the prey, PREY′ is the new position and PI is the percentage of improvement of the finesse of the hunter. For the left and right wings, they approach the prey as follows:

$$\text{Hunter}' = \begin{cases} \text{rand}((2 \times \text{PREY} - \text{Hunter}), \text{PREY}) & (2 \times \text{PREY} - \text{Hunter}) < \text{PREY} \\ \text{rand}(\text{PREY}, (2 \times \text{PREY} - \text{Hunter})) & (2 \times \text{PREY} - \text{Hunter}) > \text{PREY} \end{cases}$$

where Hunter is the current position of the hunter and Hunter' is the new position of the hunter. As for the Center hunters, their new position is as follows:

$$\text{Hunter}' = \begin{cases} \text{rand}(\text{Hunter}, \text{PREY}) & \text{Hunter} < \text{PREY} \\ \text{rand}(\text{PREY}, \text{Hunter}) & \text{Hunter} > \text{PREY} \end{cases}$$

In all of these equations, $\text{rand}(a, b)$ generates a random number between a and b, where a and b are upper and lower bounds, respectively.
The following figures, figure 3 and figure 4, visualizes how the wing hunter's vector and center hunter's vector changes based on the prey's changing position, respectively. It also shows how the prey moves upon this change of position by

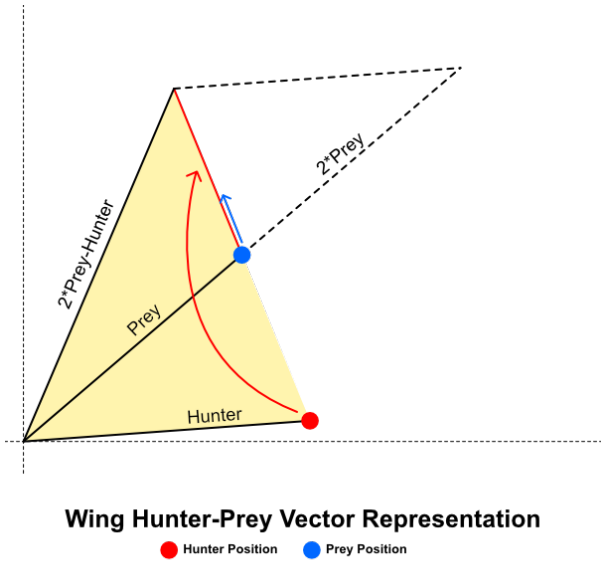


Figure 3: Wing Hunt Vector Visualization

the hunter that is currently attacking it in the current iteration. The figure gives an instance of a prey and hunter vector and shows the derivation of positions. In the algorithm, this prey would be a dummy prey put at the start of the phase where PREY would be the midpoint between all the Hunter positions. Each of the hunters would be iterated to attack the dummy prey and the prey would escape to the direction opposite to the new position of the attacking Hunter from the Prey's current position. This mechanism allows the hunters to create a circle shaped neighborhood around the prey, approach it from different directions and catch it. This also allows better solutions to be found (escape local optima) because some hunters use opposite positions. Therefore hunting in each pride can be stated in the following Pseudo-code:
At the beginning of the hunting phase, a group of female lions would be chosen from the pride. These lions would be ranked by their fitness and a number of these lions would be marked as centers while others would be marked as left and right wings.
The number of centers chosen for each pride is configurable by a percentage. The remaining pride members would be defaulted to be the left and right wings. For ever hunting phase executed, a prey would be set by the average positions of the female lions chosen for hunting. All hunters would advance to the prey once and then hunting phase is done.
The following example shows optimization of the lions to the solution when advancements of the lions is repeated. The fitness function used in this example

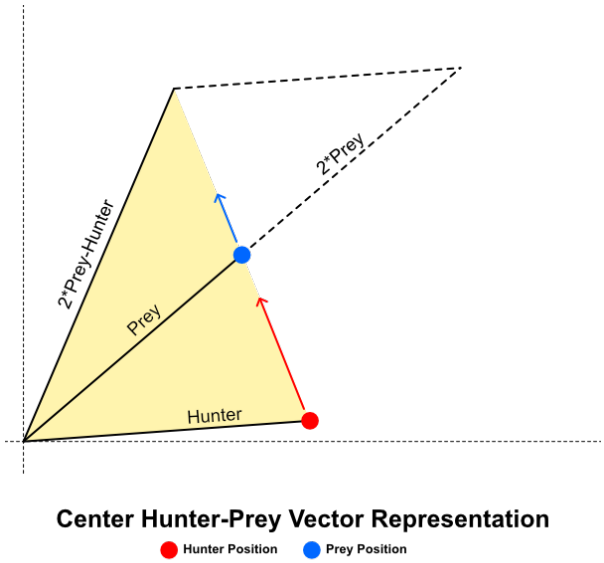


Figure 4: Center Hunt Vector Visualization

is $f(x, y) = (x - 1)^2 + (y - 1)^2$ where the solutions are minimized for their cost, where in this function is toward the center (1,1) at cost zero. The prey is not reconfigured each iteration so that it can simulate real catch and escape situation.
As shown in the previous figures, the prey acts as a marker to indicate the position where the all the hunting lions would move next. As the prey approaches the minimal solution in the correct direction, the hunting lions would also approach the ideal solution. In this example, the iterations stops when all the lions have shown no improvement in their advancement to the prey. In the figures, the lions have visibly reached each other at about 10 iterations but hasn't stopped its pursuit because of very small adjustments that isn't visible in the figures.
This part of the algorithm helps in searching local spaces for one solution as proven in the example.

23 Proposed Algorithm: Moving towards safe place

As mentioned earlier, some of the females go hunting, not all. So, remaining females go toward one of the areas of the territory. In the algorithm, the territories of each pride would consist of personal best solutions so far, which would

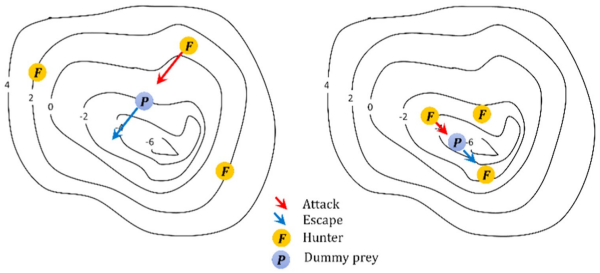


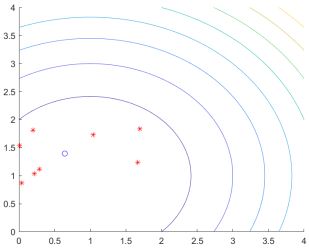
Figure 5: Attack and Escape Example

Divide hunters into three sub groups randomly
Generate a prey
***For** i=1:H (H is number of hunters)*
Move ith hunter toward prey according to its relevant group
***If** new place of ith hunter is better than its last position*
Prey escapes from hunter
End
End

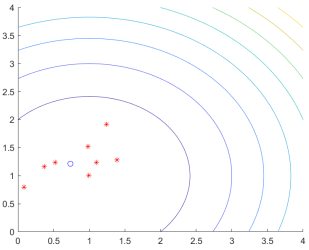
assist the algorithm to save the best solutions obtained so far over the course of iteration. Therefore the new position for a female lion is given as:

$$\text{Lion}' = \text{Lion} + 2D \times rand(0,1)R1 + U(-1,1) \times \tan(\theta) \times D \times R2$$
$$\text{where } R1 \cdot R2 = 0, ||R2|| = 1$$

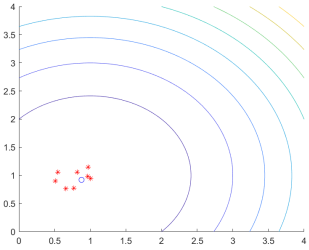
where Lion and Lion' is the previous and next position of the female lion, respectively, and D is the distance between the female lion's position and the selected point chosen by tournament selection in the pride's territory. The following figure shows the range of possible next positions of the lion.
R1 is a vector which its start point is the previous location of the female lion and its direction is toward the selected position. R2 is perpendicular to R1. θ is an angle that is selected by uniform distribution among $-\pi/6$ and $\pi/6$ and U is a function selecting a random number with uniform distribution.
In the following example (which uses the same fitness function and method as the previous section), multiple lions are created in the space; one of which lions is selected to move (source), tournament selection is done without replacement and each selected lion would be evaluated which of them have the best fitness (in this case: least cost).
In the algorithm, the tournament selection's size is not set but is computed according to the pride's number of successes in updating their best position in the last iteration. This is computed using the following equations.



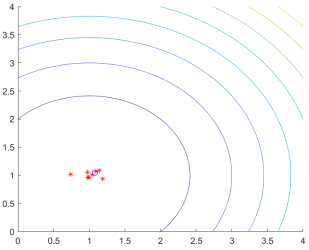
(a) Iteration 0



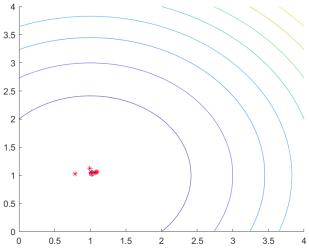
(b) Iteration 1



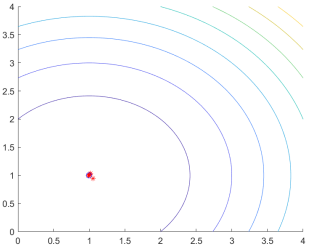
(c) Iteration 2



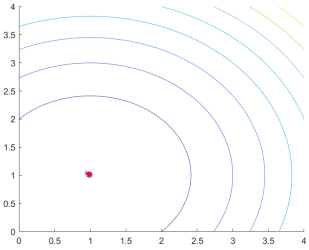
(d) Iteration 3



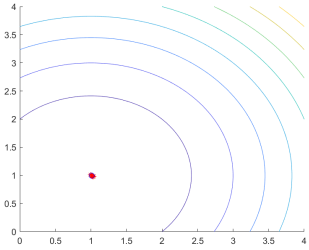
(e) Iteration 4



(f) Iteration 5



(g) Iteration 6



(h) Iteration 7

Figure 6: Hunting Simulation

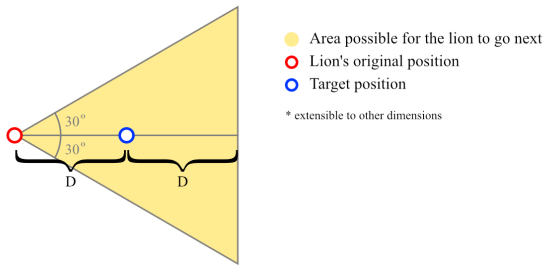


Figure 7: Range of next positions relative to original

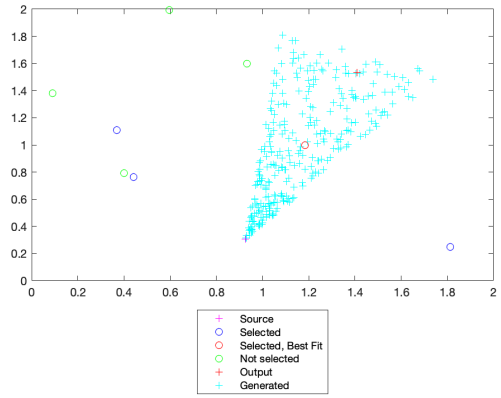


Figure 8: Trajectory visualization by generating 300 outputs

$$K_j(P) = \Sigma_{i=1}^{N} S(i,t)$$
$$T_j^{Size} = \max\left(2, ceil\left(\frac{K_j(P)}{2}\right)\right)$$

where, K_j is the number of successes in a pride and T_j is the tournament size of that pride for the next iteration.
This mechanism, which would also be done by the resident males, is used in the roaming part of the algorithm.

24 Proposed Algorithm: Roaming

Roaming in territory is something lions in a pride usually do. This also helps in getting used to the territory for the lions. For nomads, this is done to find

better places to get resources from. In optimization, this can be used to further optimize the local search space for each pride and is used by nomads for searching solutions.
Along with roaming, if a resident male visits a new position better than its current position, update its current best position and best visited solution.
For pride lions, roaming would mean to use each others' position to find better positions (solutions). Similar to the previous section, in this part of the algorithm, the lions would go towards each other using the same formula. The algorithm for resident male roaming is done by the following pseudo-code:
For nomad lions, however, roaming would be utilizing random movement to find better positions (solutions). In the algorithm, nomad lions have an advantage to not get stuck in local optima. A nomad lion would randomly roam around the search if space especially when its fitness among the nomads is worse. These nomad lions are only required to check their standing among peer nomads unlike resident lions whom tries to optimize with each others position, thus, giving the nomad lions an advantage to not get stuck. The new position of these nomads are generated using the following equation:

$$\text{Lion'}_{ij} = \begin{cases} \text{Lion}_{ij} & \text{if rand}(0,1) > pr_i \\ \text{RAND}_j & \text{otherwise} \end{cases}$$

where Lion_i is the position of the lion, j is the dimension, rand is a uniformally distributed random number between 0 and 1 and RAND_j is a random position in the search space. Now, pr_i (i for every nomad lion) is represented by:

$$pr_i = 0.1 + \min\left(0.5, \frac{(\text{Nomad}_i - \text{Best}_{nomad})}{\text{Best}_{nomad}}\right)$$

This shows that the nomad lion, instead of moving relative to other positions, would move in a sporadic manner proving its 'randomness'.
Similar to the previous examples, the following example simulates the roaming mechanism of the algorithm (using the same parameters) to show how this part of the algorithm helps in optimizing solutions. The mini-algorithm runs until the maximum set iterations of 200. (Iterations on next page.)
The example shows how fast the optimization technique used in pride lions compared to the slow but sure optimization technique of nomads. Though after 200 iterations, the nomad lions still hasn't optimized enough to the global minima of $(1, 1)$.

25 Proposed Algorithm: Mating and Equilibrium

Mating of the lions in the population refers to the mixing of their genes to allow exchanging of information. In this algorithm, for every pride, $\%Ma$ of the females mate with one or more males. For the nomads, $\%Ma$ of the females mate

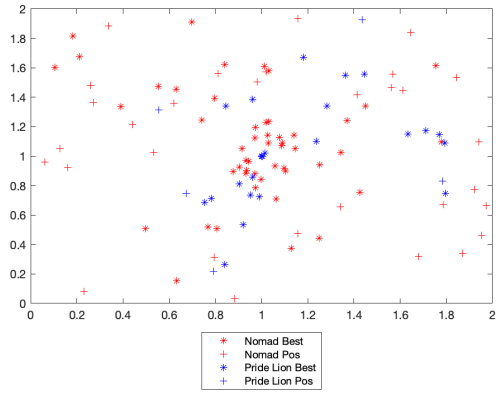


Figure 9: Initial positions and best of nomad and resident lions

with only one male. These individuals are selected randomly. After mating, two offsprings are produced using the equations:

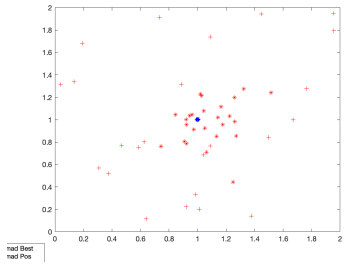
$$\text{Offspring}_j 1 = \beta \times \text{Female Lion}_j + \sum \frac{1-\beta}{\sum_{i=1}^{NR} S_i} \times \text{MaleLion}_j^i \times S_i,$$

$$\text{Offspring}_j 2 = (1-\beta) \times \text{Female Lion}_j + \sum \frac{\beta}{\sum_{i=1}^{NR} S_i} \times \text{MaleLion}_j^i \times S_i$$

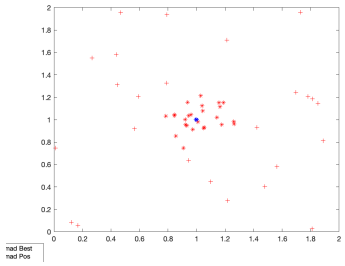
,
where j is dimension; S_i is 1 if male i is selected for mating, 0 otherwise; NR is the number of resident males in a pride; β is a randomly generated number with a normal distribution with mean value 0.5 and standard deviation 0.1.
One of these offsprings is set as male, and the other as female. A mutation is applied on each gene of one of the produced offspring with probability Mu. Through mutation, a random number replaces the value of the gene.
The purpose of β in the equation is to find a value somewhere in between the values of the female and male genes; and the purpose of mutation is to further diversify the solution generated.
For example:
Given the population:

	1	2	3	4
Position	(0.5, 0.5)	(0.5, 1)	(1.5, 0.5)	(1, 1.5)

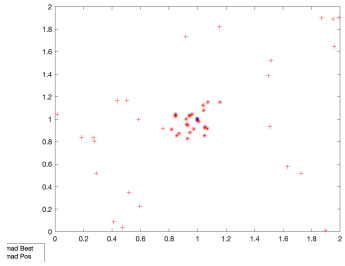
Table 1: Mating Demo Female Population



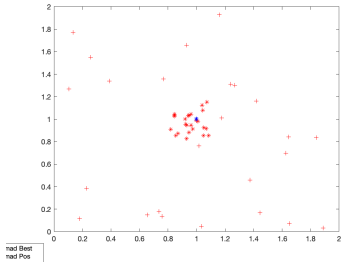
(a) Iteration 25



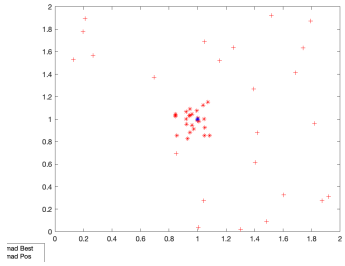
(b) Iteration 50



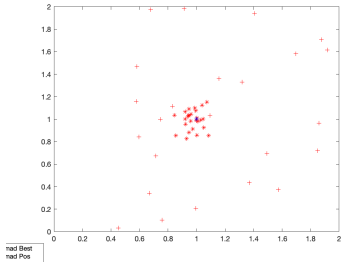
(c) Iteration 75



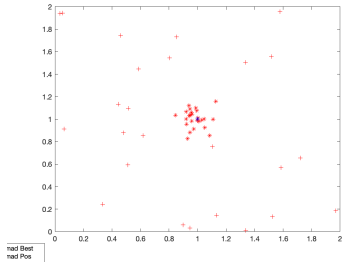
(d) Iteration 100



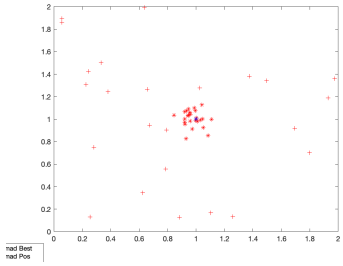
(e) Iteration 125



(f) Iteration 150



(g) Iteration 175



(h) Iteration 200

Figure 10: Roaming Simulation

	1	2	3	4
Position	(0.75, 0.5)	(1, 0.5)	(0.5, 1.5)	(1, 1)

Table 2: Mating Demo Male Population

Selecting Female 2 (0.5, 1.0) and Males 2 (0.75, 0.5) and 3 (0.5, 1.5) and β as 0.4, the genes are computed as:

For j= 1:

Female Gene Contribution: 0.5

Male Gene Contribution:

$$\sum MaleLion_1^i \times S_i = \frac{0.75 \times 0 + 1 \times 1 + 0.5 \times}{2} = 0.75$$

Interpolate:

$$Offspring_1 1 = (0.4) \times 0.5 + (1 - 0.4) \times = 0.65$$

$$Offspring_1 2 = (1 - 0.4) \times 0.5 + 0.4 \times 0 = 0.6$$

(a) Mating Demo Offspring Gene 1

For j=2:

Female Gene Contribution: 1

Male Gene Contribution:

$$\sum MaleLion_2^i \times S_i = \frac{0.5 \times 0 + 0.5 \times 1 + 1.5 \times 1 + 1 \times 0}{2} = 1$$

Interpolate:

$$Offspring_2 1 = (0.4) \times 1 + (1 - 0.4) \times 1 = 1$$

$$Offspring_2 2 = (1 - 0.4) \times 1 + 0.4 \times 1 = 1$$

(b) Mating Demo Offspring Gene 2

The result offspring are Offspring 1 = (0.65, 1) and Offspring 2 = (0.6, 1). Plotting this example yields:

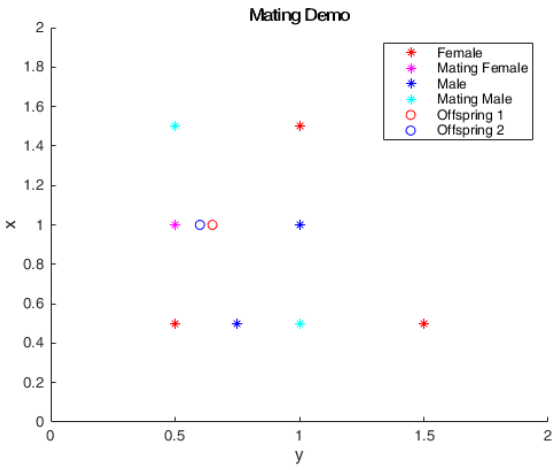


Figure 12: Mating Demo Plot

26 Proposed Algorithm: Defense

When they mature, male cubs in a pride fight with the existing males. The weaker males are driven out of the pride and become nomads. This is done in the algorithm by sorting all the males in the pride according to their fitness. The number of males to be driven out is determined by the sex rate of the pride. Similarly, nomad males can fight existing males in a pride, and if they prove to be stronger than the resident male, they replace the beaten male in the pride. For each nomad male in the population, it will attack a single or multiple prides determined randomly. Once a pride is selected, the fitness of the nomad male is compared to each resident male of the pride. If the nomad is determined to be stronger, it will replace the resident male, and the resident male becomes a nomad. This behaviour allows the retention of strong lions/solutions in the pride.

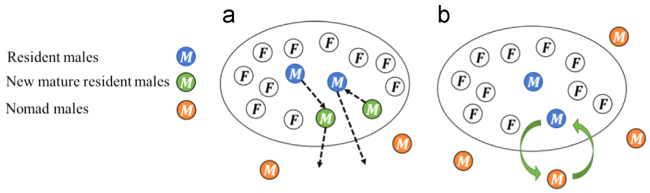


Figure 13: (a) Defense against newly matured cubs and (b) Defense against nomads[5]

27 Proposed Algorithm: Migration

Some females in a pride migrate to another pride or become nomads. Through migration, nomad females can also become residents of a pride. This enhances the diversity of the pride by the best position attained by the migrating females. For each pride, the maximum number of females is determined by the sex rate of the pride. The surplus females plus $\%I$ of the maximum number of females in a pride migrate and become nomads. The nomad females are then sorted according to their fitness and the stronger ones are distributed to random prides to fill the empty place of the migrated females. The number of females to be distributed is determined by the amount of empty female slots of all prides. This procedure allows information sharing among the different prides.

28 Proposed Algorithm: Equilibrium

At the end of each iteration of LOA, the number of nomad lions will be controlled with respect to the permitted number of each gender determined by the sex rate. The ones with the least fitness value will be removed from the population. This is to maintain equilibrium in the lion's population.

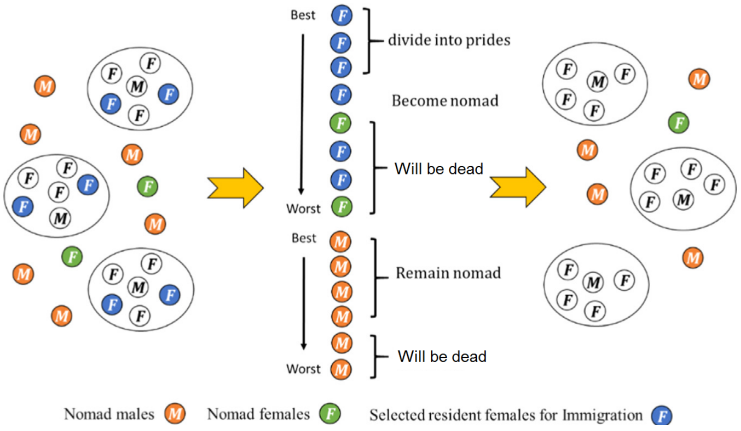


Figure 14: Migration and Population Equilibrium [5]

29 Mating, Defense, Migration, and Equilibrium Algorithm Demo

In this section, the mating, defense, migration, and equilibrium algorithm is demonstrated using an example. The function

$$f(x, y) = (x - 1)^2 + (y - 1)^2$$

is being minimized and the following figure displays the initial positions of the lions.

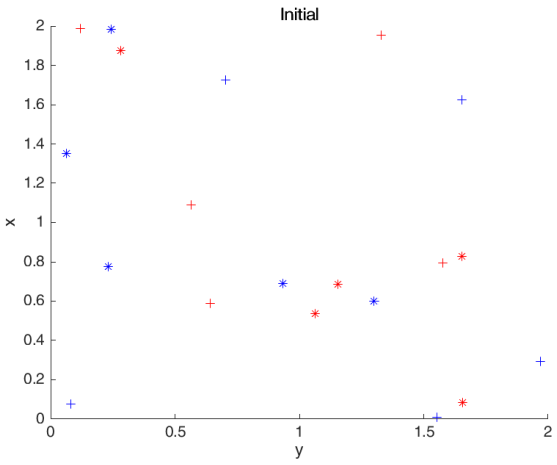


Figure 15: Mating, Defense, Migration, and Equilibrium Algorithm Demo: Initial positions[5]

The asterisks represent the residential lions and the plus signs represent the nomad lions. The red color means its a female, while the blue color means its a male. The first iteration is as follows:
Through mating, four new lions are generated. The positions of these lions are found in between the positions of their parents. No mutation has occured in this particular example.

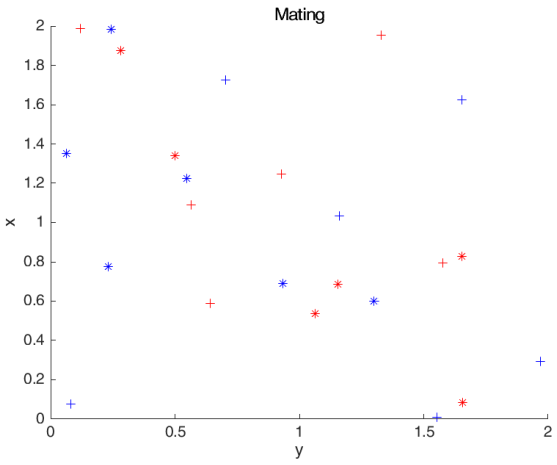


Figure 16: Iteration 1: Mating - Four new points (lions) are generated

The newly generated cubs mature and compete with other residential lions to see who will retain in the pride. In this iteration, a residential male becomes a nomad (indicated by a blue asterisk becoming a blue plus).

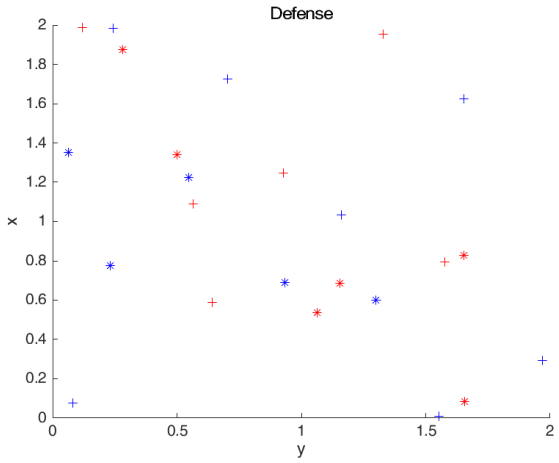


Figure 17: Iteration 1: Defense - A residential male becomes a nomad

Through the migration of residential females, some residential females become nomads, and some nomad females become included in prides (indicated by the red asterisks becoming red pluses and vice versa).

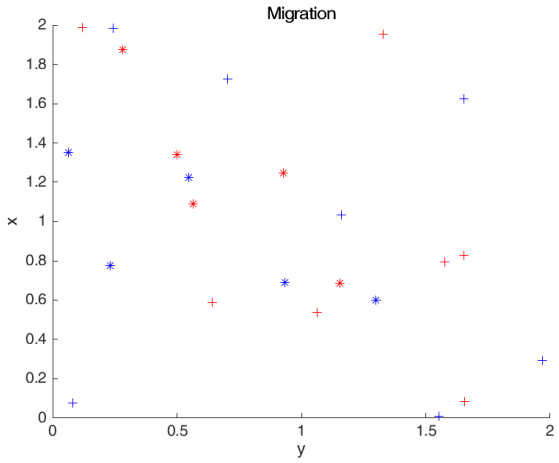


Figure 18: Iteration 1: Migration - Some residential females become nomads

At the end of the iteration, the population of the lions are controlled with respect to the given sex rate. In this example, two male and female nomads were removed from the population.

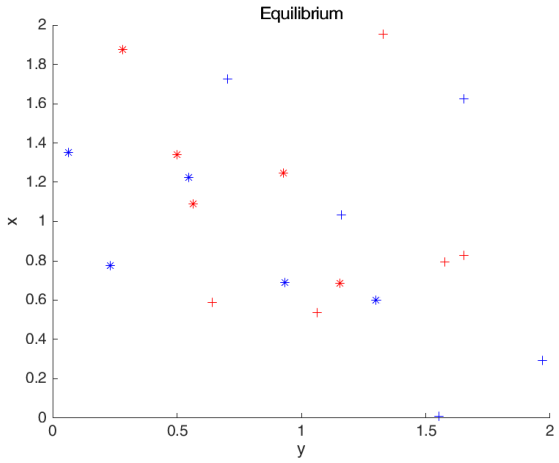


Figure 19: Iteration 1: Equilibrium - Four lions are killed.

The second iteration of the example is as follows:

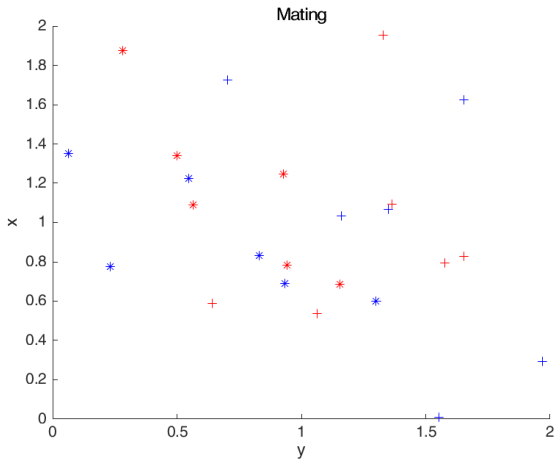


Figure 20: Iteration 2: Mating - Four new points (lions) are generated

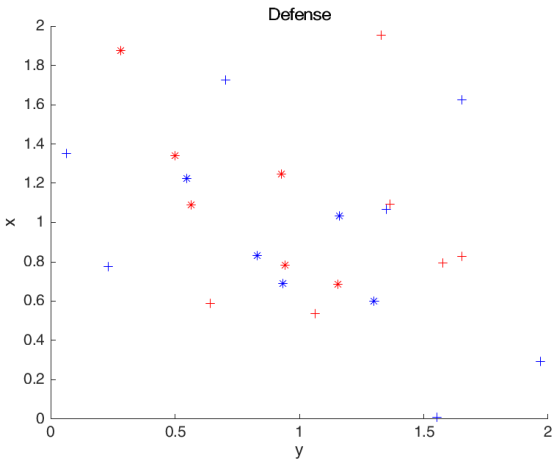


Figure 21: Iteration 2: Defense - A residential male becomes a nomad and a nomad becomes a residential lion

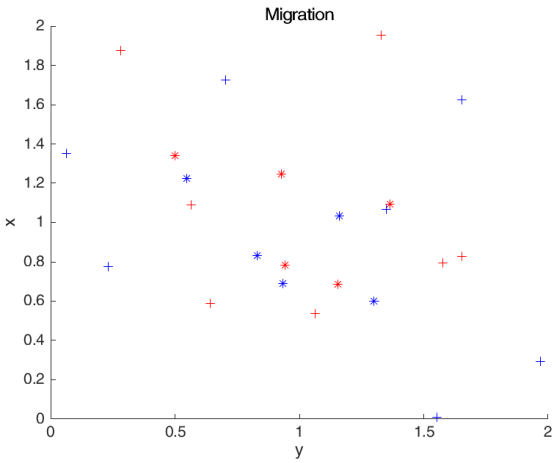


Figure 22: Iteration 2: Migration - Some residential females become nomads

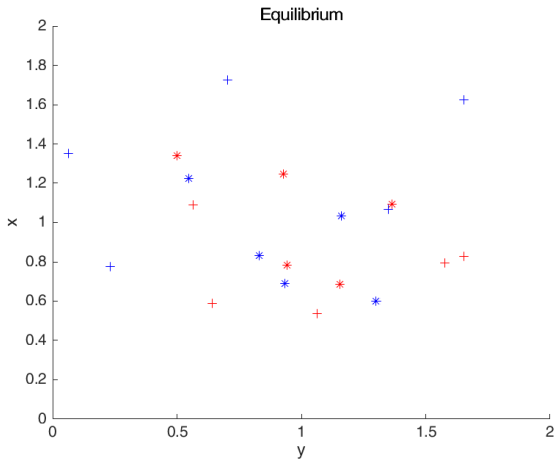
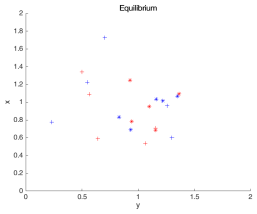
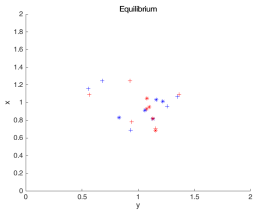


Figure 23: Iteration 2: Equilibrium - Four lions are killed

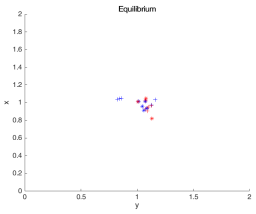
For the subsequent iterations we have the following figures:



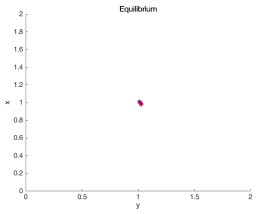
(a) Iteration 3: Equilibrium



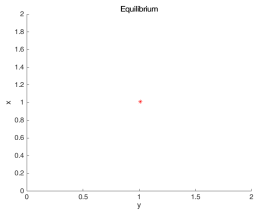
(b) Iteration 5: Equilibrium



(c) Iteration 10: Equilibrium



(d) Iteration 25: Equilibrium



(e) Iteration 50: Equilibrium

Figure 24: Iterations 3, 5, 10, 25, and 50

After each iteration, the algorithm converges towards the optima of the function. In this case, the minimum of the function is (1,1). In each iteration, the positions of the newly generated lions become progressively closer to this minimum. In the final iteration, the entire population converges at the minimum.

30 The algorithm

Using the previous ideas, the algorithm is then created. Using the concepts in the previous section, the pseudocode of the algorithm is presented:

Lion Optimization Algorithm pseudo code

1. Generate random sample of Lions N_{pop}(N_{pop} is number of initial population).
2. Initiate prides and nomad lions
   i. Randomly select %N (Percent of lions that are nomad) of initial population as nomad lion. Partition remained lions into P (P is number of prides) prides randomly, and formed each pride's territory.
   ii. In each pride %S (Sex rate) of entire population are known as females and the rest as males. This rate in nomad lions is inversed.
3. For each pride do
   i. Some randomly selected female lion go hunting.
   ii. Each of remained female lion in pride go toward one of the best selected position from territory.
   iii. In pride, for each resident male; %R (Roaming percent) of territory randomly are selected and checked. %Ma (Mating probability) of females in pride mate with one or several resident male. → *New cubs become mature.*
   iv. Weakest male drive out from pride and become nomad.
4. For Nomad do
   i. Nomad lion (both male and female) moving randomly in search space. %Ma (Mating probability) of nomad Female mate with one of the best nomad male. → *New cubs become mature.*
   ii. Prides randomly attacked by nomad male.
5. For each pride do
   i. Some female with I rate ((Immigrate rate)) immigrate from pride and become nomad.
6. Do
   i. First, based on their fitness value each gender of the nomad lions are sorted. After that, the best females among them are selected and distributed to prides filling empty places of migrated females.
   ii. With respect to the maximum permitted number of each gender, nomad lions with the least fitness value will be removed.

If termination criterion is not satisfied, then go to step 3

Figure 25: The psedocode of LOA

The algorithm starts with an initialization. It produces the random samples to be used as test solutions as positions to be improved upon. These samples are grouped in prides or by the nomads. They are also given a gender in these groups. The algorithm then proceeds to the iterative loop that begins with pride hunting, where select females would improve their own positions. The other females in the pride, lions that are not selected in the previous part, would try to find better positions by utilizing the best positions in their pride. The other resident lions, the males, would also do something similar. Better solutions are then made by mating the best lions in a pride. The children from this part is only added to the pride after the mating part has happened. The weakest males that are the least fit are driven out from prides and are put into the nomad group. After that the nomads are then checked and moved randomly in search space. The lions that are nomads are then mated one to one, also creating new solutions and improving on the other positions. The nomad males then try to attack prides by testing the resident male's fitness against their own. After this, the resident females, females in the pride, immigrate bringing them out of the pride and becoming a nomad. The nomad lions' population is then regulated by bringing the very best females back into prides filling the empty space that was once filled from the very initialization of the algorithm. To limit the population, the nomad lions with the least fitness are deleted based on the

limit placed using the original population size of the nomads.

31 Sample Runs using different Benchmarks

The algorithm is now tested upon different benchmarking functions. All benchmarks ran using the following parameters of the algorithm.

iterations = 300;

population = 100;

dimension = 2;

prides length = 5;

percent nomad = 0.2;
percent roam = 0.5;
percent sex = 0.8;

mating rate = 0.3;
mutation prob = 0.1;

immigration rate = 0.4;

The first benchmarking function is the function:

$$f(x) = x^2$$

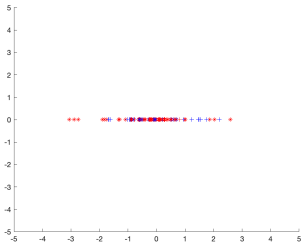
and figure 26 shows the plots of the positions on the x-axis after every other iterations and is stopped at iteration 25.
The second benchmarking function is the function:

$$f(x, y) = x^2 + y^2$$

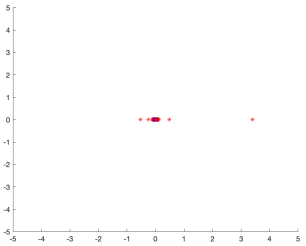
and figure 27 shows the plots of the positions on the xy-plane after every other iterations and is stopped at iteration 44.
The third benchmarking function is the function:

$$f(x, y, z) = x^2 + y^2 + z^2$$

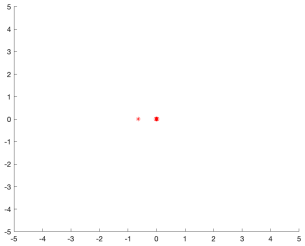
and figure 28 shows the plots of the positions on the 3D space after every other iterations and is stopped at iteration 22.
The fourth benchmarking function is the Rastrigin benchmarking function for 2 dimensions and figure 29 shows the plots of the positions on the xy-plane after every other iterations and is stopped at iteration 28.



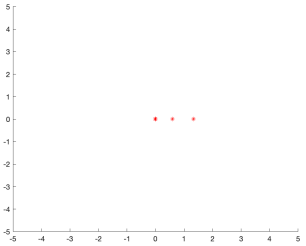
(a) Iteration 1



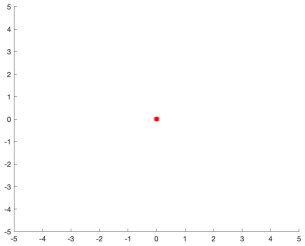
(b) Iteration 4



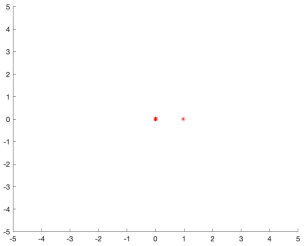
(c) Iteration 8



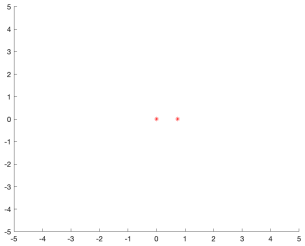
(d) Iteration 11



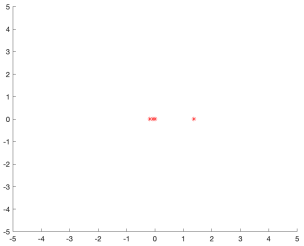
(e) Iteration 15



(f) Iteration 18

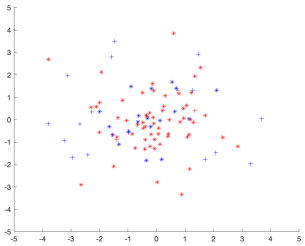


(g) Iteration 22

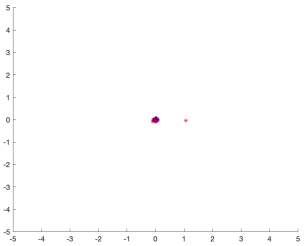


(h) Iteration 25

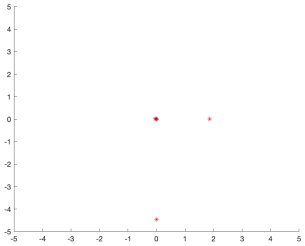
Figure 26: Function 1



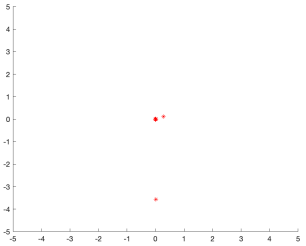
(a) Iteration 1



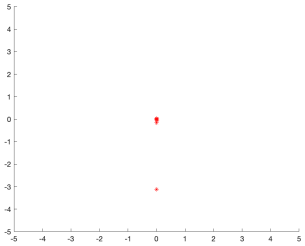
(b) Iteration 7



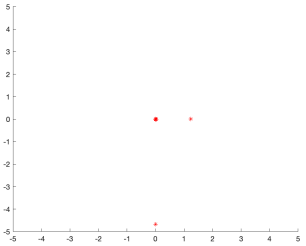
(c) Iteration 13



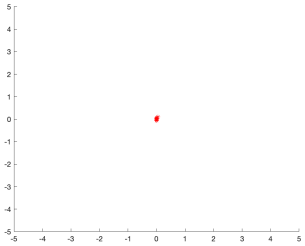
(d) Iteration 19



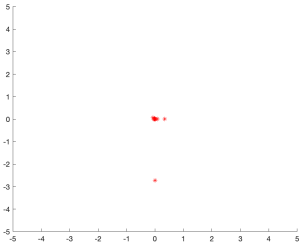
(e) Iteration 25



(f) Iteration 31

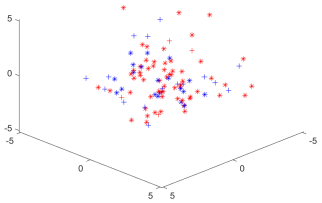


(g) Iteration 37

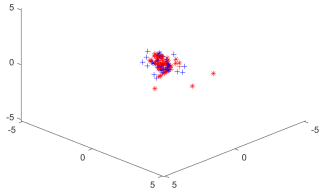


(h) Iteration 44

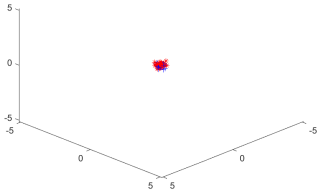
Figure 27: Function 2



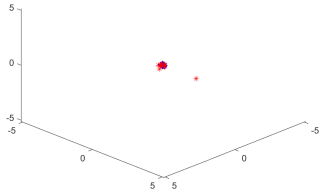
(a) Iteration 1



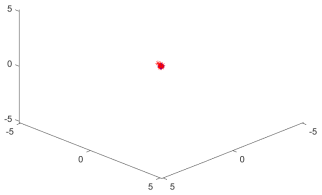
(b) Iteration 3



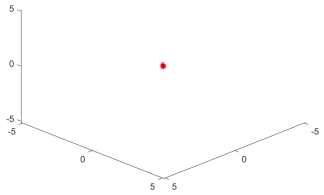
(c) Iteration 5



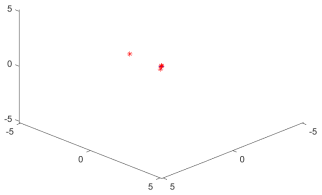
(d) Iteration 7



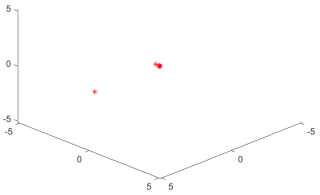
(e) Iteration 9



(f) Iteration 11

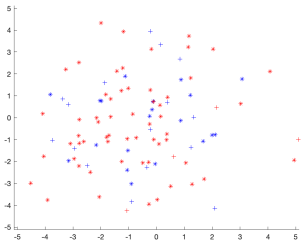


(g) Iteration 13

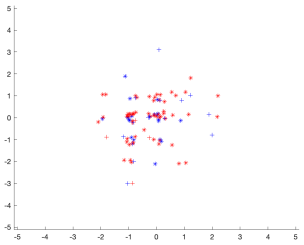


(h) Iteration 15

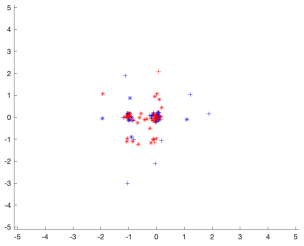
Figure 28: Function 3



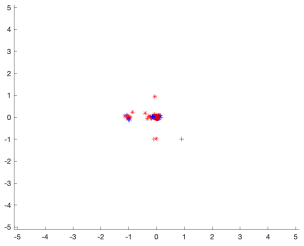
(a) Iteration 1



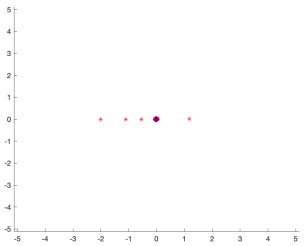
(b) Iteration 5



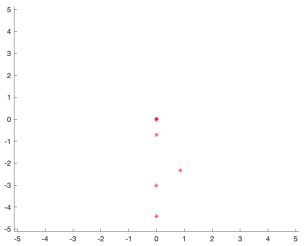
(c) Iteration 9



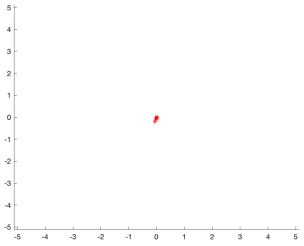
(d) Iteration 13



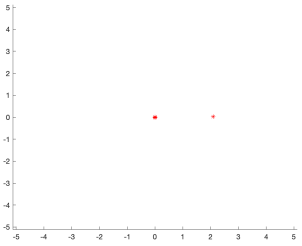
(e) Iteration 17



(f) Iteration 21



(g) Iteration 25



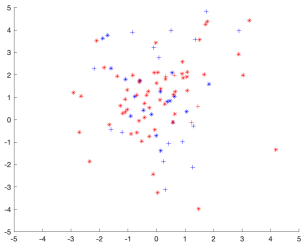
(h) Iteration 28

Figure 29: Function 4

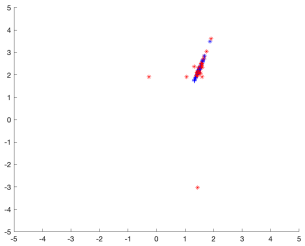
The fifth benchmarking function is the Rosenbrock benchmarking function for 2 dimensions with $a = 2$ and $b = 100$ and figure 30 shows the plots of the positions on the xy-plane after every other iterations and is capped at 300.
The sixth benchmarking function is the Rosenbrock benchmarking function for 2 dimensions with $a = 9$ and $b = 100$ and figure 31 shows the plots of the positions on the xy-plane after every other iterations and is capped at 300.
After 300 iterations, the global fitness achieved using each function is graphed against the algorithm's run time. This is displayed in figure 32. The best and worst fitness achieved, along with the standard deviation and median, of each function after 10 runs is also presented in table 3.

Function	Worst Fitness	Best Fitness	Standard Deviation	Median
1	0.00	0.00	0.00	0.00
2	0.00	0.00	0.00	0.00
3	0.00	0.00	0.00	0.00
4	0.00	0.00	0.00	0.00
5	8.78E-4	3.19E-11	6.75E-4	2.77E-6
6	6.83E-4	4.72E-9	5.42E-4	1.09E-4

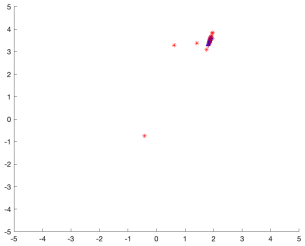
Table 3: Fitness Results of Functions After 10 Runs



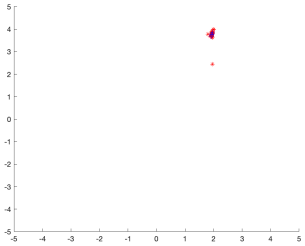
(a) Iteration 1



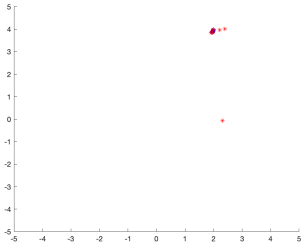
(b) Iteration 44



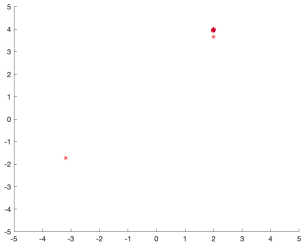
(c) Iteration 87



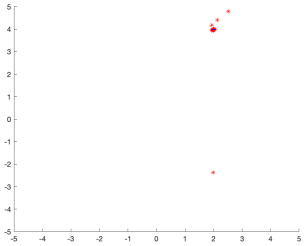
(d) Iteration 130



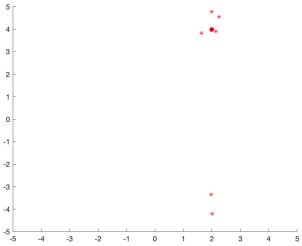
(e) Iteration 173



(f) Iteration 216

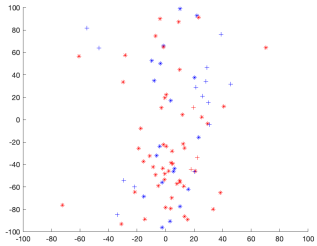


(g) Iteration 259

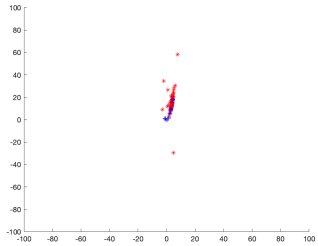


(h) Iteration 300

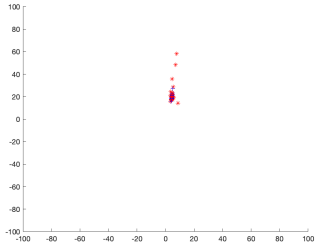
Figure 30: Function 5



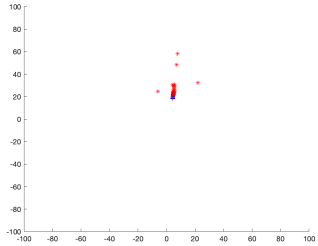
(a) Iteration 1



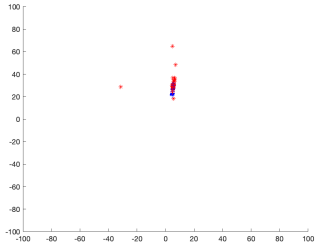
(b) Iteration 44



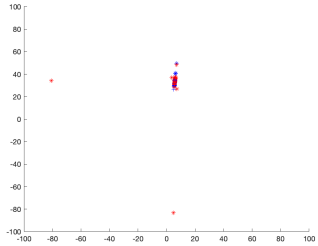
(c) Iteration 87



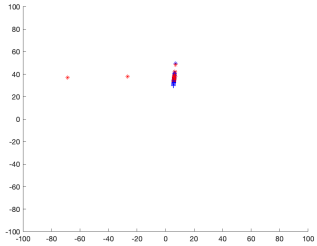
(d) Iteration 130



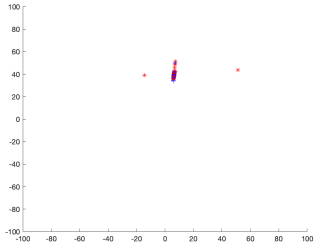
(e) Iteration 173



(f) Iteration 216

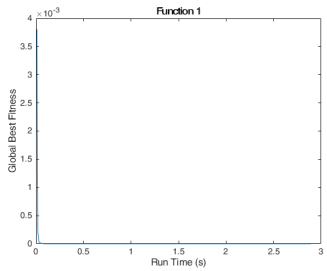


(g) Iteration 259

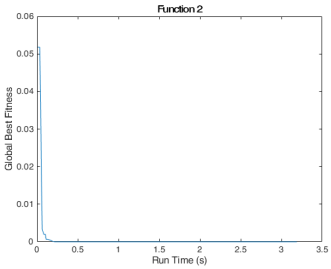


(h) Iteration 300

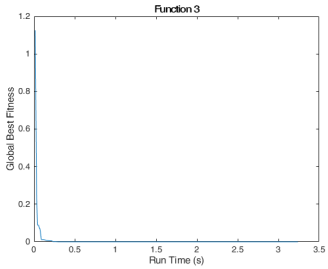
Figure 31: Function 6



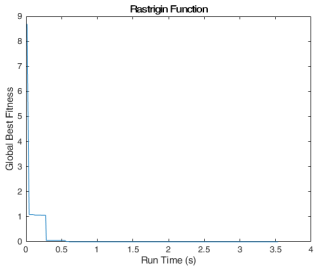
(a) Function 1



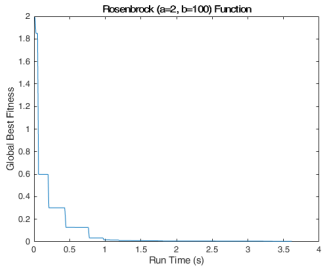
(b) Function 2



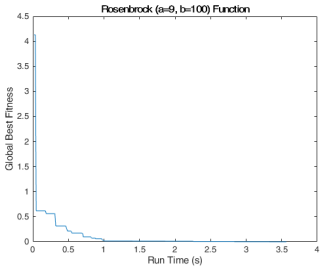
(c) Function 3



(d) Rastrigin Function



(e) Rosenbrock (a=2, b=100) Function



(f) Rosenbrock (a=9, b=100) Function

Figure 32: Run-Time vs Best Fitness Achieved of each function

32 CEC 2014 Benchmarks

CEC is a committee known for creating competitions for optimization algorithms. Each year CEC provides a set of functions that are known for testing optimization algorithms for scientists to submit their work and compare the effectiveness of their algorithms. Each test suite includes functions that are shifted and rotated which are to be run in black box tests though their code is reviewable and written in C++.
The following parameters are set and used in the algorithm.

$$\begin{aligned}
\text{Number of Prides} &= 4 \\
\text{Percent of Nomad Lions} &= 0.2 \\
\text{Roaming Percent} &= 0.2 \\
\text{Mutate Probability} &= 0.2 \\
\text{Sex Rate} &= 0.8 \\
\text{Mating Probability} &= 0.3 \\
\text{Immigrate Rate} &= 0.4
\end{aligned}$$

An overview of what functions are included in the test is seen in Table 3.

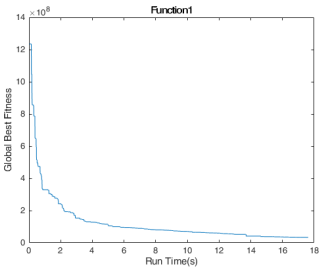
Type	ID	Function	f*
Unimodal	f1	Rotated high conditioned elliptic function	100
	f2	Rotated bent cigar function	200
	f3	Rotated discus function	300
Multimodal	f4	Shifted and rotated Rosenbrock function	400
	f5	Shifted and rotated Ackley's function	500
	f6	Shifted and rotated Weierstrass function	600
	f7	Shifted and rotated Griewank's function	700
	f8	Shifted Rastrigin function	800
	f9	Shifted and rotated Rastrigin's function	900
	f10	Shifted Schwefel function	1000
	f11	Shifted and rotated Schwefel's function	1100
	f12	Shifted and rotated Katsuura function	1200
	f13	Shifted and rotated HappyCat function	1300
	f14	Shifted and rotated HGBat function	1400
	f15	Shifted and rotated Expanded Griewank's plus Rosenbrock's function	1500
	f16	Shifted and rotated Expanded Scaffer's F6 function	1600
Hybrid	f17	Hybrid function1 (f 9, f 8, f 1)	1700
	f18	Hybrid function2 (f 2, f 12, f 8)	1800
	f19	Hybrid function3 (f 7, f 6, f 4, f 14)	1900
	f20	Hybrid function4 (f 12, f 3, f 13, f 8)	2000
	f21	Hybrid function5 (f 14, f 12, f 4, f 9, f 1)	2100
	f22	Hybrid function6 (f 10, f 11, f 13, f 9, f 5)	2200
Composition	f23	Composition function1 (f 4, f 1, f 2, f 3, f 1)	2300
	f24	Composition function2 (f 10, f 9, f 14)	2400
	f25	Composition function3 (f 11, f 9, f 1)	2500
	f26	Composition function4 (f 11, f 13, f 1, f 6, f 7)	2600
	f27	Composition function5 (f 14, f 9, f 11, f 6, f 1)	2700
	f28	Composition function6 (f 15, f 13, f 11, f 16, f 1)	2800
	f29	Composition function7 (f 17, f 18, f 19)	2900
	f30	Composition function8 (f 20, f 21, f 22)	3000

Table 4: Functions defined by CEC 2014 and used for testing

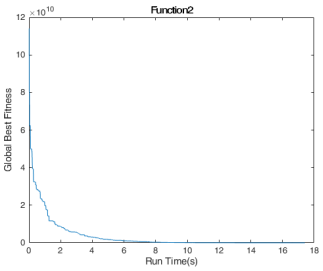
The following (Table 4) are results from running the reimplemented algorithm on the CEC 2014 Special Session and Competition on Real-Parameter Single Objective (Expensive) Optimization Benchmark[**?**], on a AMD Ryzen 5 2600 machine, parallelized on a 6 threaded workload to run 60 times of the algorithm on one function totalling to 30 functions. The limit placed to stop the algorithm is when it reaches 150,000 function evaluations. Each function ran for about 600 seconds on the machine. The best fitness for each run is recorded and computed for the Standard Deviation. The algorithm was written in MATLAB and the parallel computing toolbox was also used. The run times of the algorithm using each function were also graphed against the fitness values achieved at those times (Figure 33).

	min	**max**	**std**	**median**
f1	1.647E+07	6.730E+07	1.053E+07	3.844E+07
f2	2.645E+05	6.888E+06	1.386E+06	2.092E+06
f3	1.034E+04	3.092E+04	3.979E+03	1.670E+04
f4	4.738E+02	6.284E+02	3.272E+01	5.611E+02
f5	5.208E+02	5.210E+02	5.124E-02	5.210E+02
f6	6.195E+02	6.296E+02	2.167E+00	6.254E+02
f7	7.005E+02	7.011E+02	1.218E-01	7.009E+02
f8	8.547E+02	9.298E+02	1.350E+01	8.845E+02
f9	9.658E+02	1.028E+03	1.518E+01	9.971E+02
f10	2.586E+03	7.135E+03	1.273E+03	3.864E+03
f11	3.717E+03	8.245E+03	1.148E+03	7.470E+03
f12	1.202E+03	1.203E+03	3.312E-01	1.203E+03
f13	1.300E+03	1.301E+03	8.465E-02	1.300E+03
f14	1.400E+03	1.400E+03	3.087E-02	1.400E+03
f15	1.514E+03	1.549E+03	7.715E+00	1.525E+03
f16	1.611E+03	1.613E+03	3.086E-01	1.612E+03
f17	5.180E+05	3.363E+06	5.823E+05	1.612E+06
f18	1.991E+03	1.350E+04	2.131E+03	2.910E+03
f19	1.911E+03	1.923E+03	2.082E+00	1.919E+03
f20	4.630E+03	2.635E+04	4.406E+03	1.484E+04
f21	7.908E+04	7.593E+05	1.624E+05	2.698E+05
f22	2.473E+03	3.134E+03	1.545E+02	2.782E+03
f23	2.620E+03	2.633E+03	2.678E+00	2.624E+03
f24	2.618E+03	2.632E+03	3.211E+00	2.627E+03
f25	2.700E+03	2.709E+03	2.528E+00	2.707E+03
f26	2.700E+03	2.701E+03	1.122E-01	2.700E+03
f27	3.117E+03	3.739E+03	1.630E+02	3.536E+03
f28	3.255E+03	4.404E+03	1.954E+02	3.296E+03
f29	3.125E+03	4.300E+03	1.531E+02	3.131E+03
f30	3.813E+03	7.945E+04	1.646E+04	4.796E+03

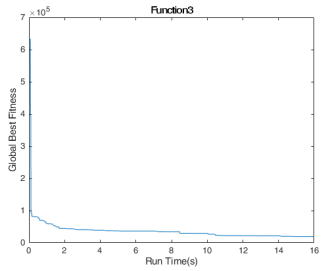
Table 5: Results for LOA under CEC 2014 Benchmark



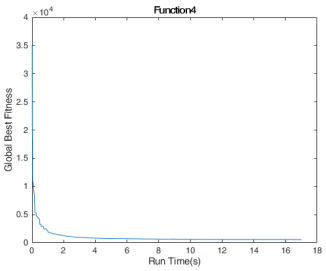
(a) Function 1



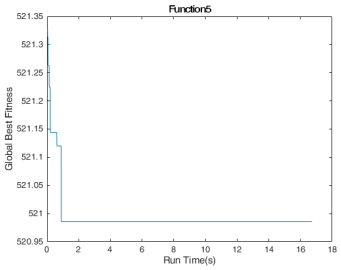
(b) Function 2



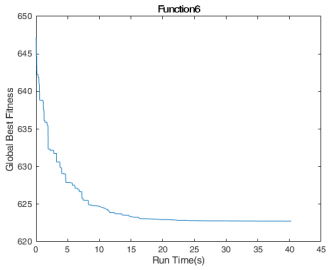
(c) Function 3



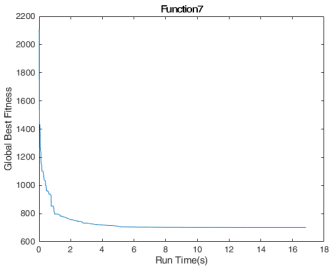
(d) Function 4



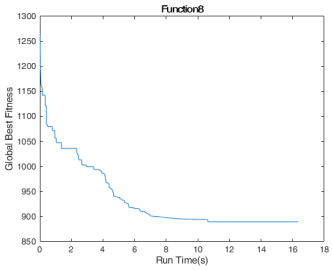
(e) Function 5



(f) Function 6

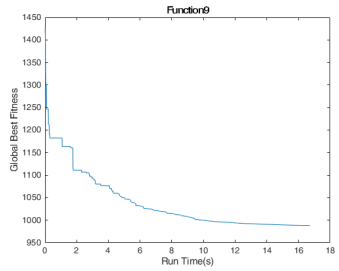


(g) Function 7

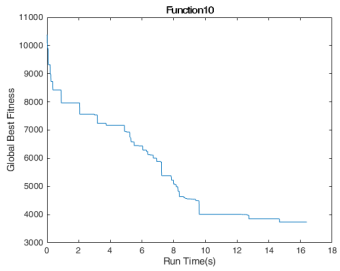


(h) Function 8

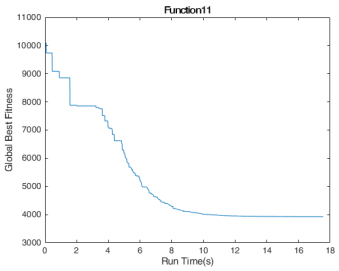
Figure 33: Run-Time vs Fitness of Functions 1-8



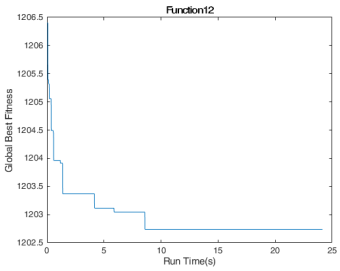
(a) Function 9



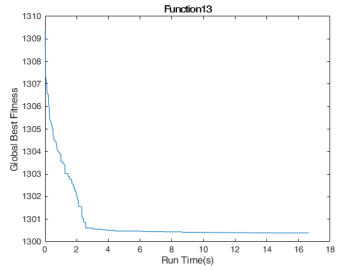
(b) Function 10



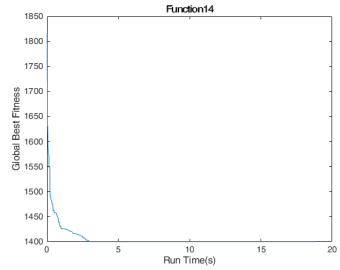
(c) Function 11



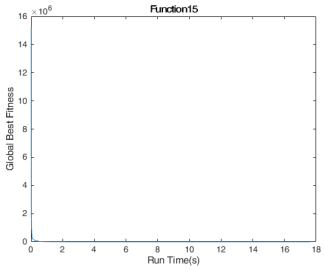
(d) Function 12



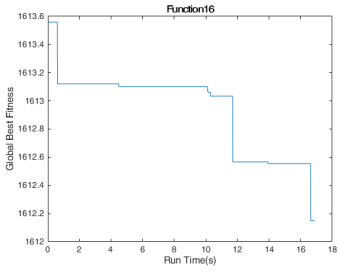
(e) Function 13



(f) Function 14

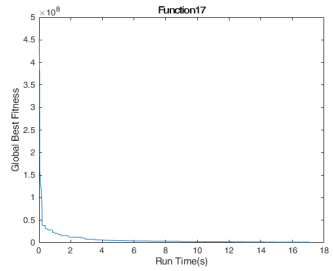


(g) Function 15

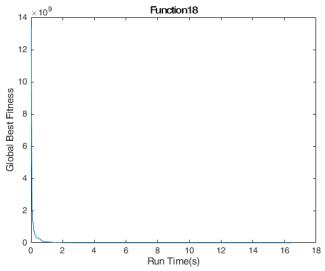


(h) Function 16

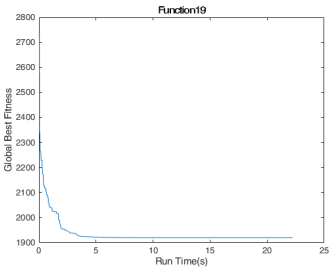
Figure 34: Run-Time vs Fitness of Functions 9-16



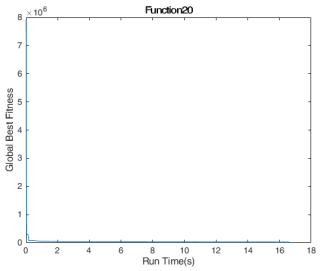
(a) Function 17



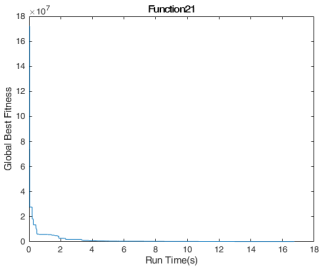
(b) Function 18



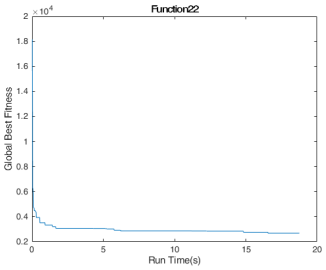
(c) Function 19



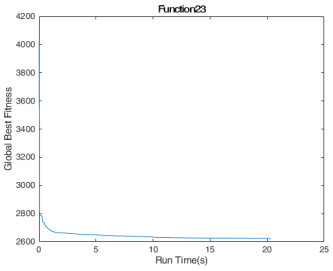
(d) Function 20



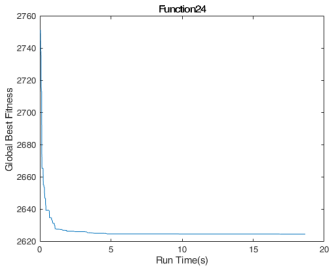
(e) Function 21



(f) Function 22

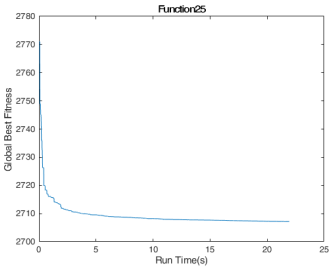


(g) Function 23

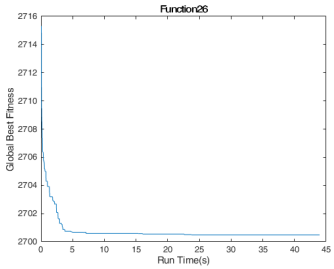


(h) Function 24

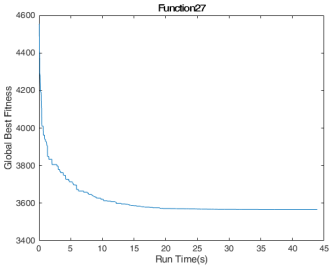
Figure 35: Run-Time vs Fitness of Functions 17-24



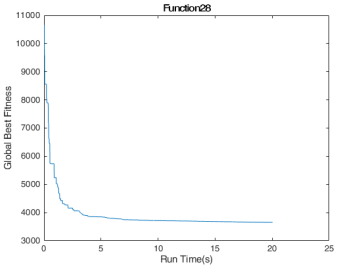
(a) Function 25



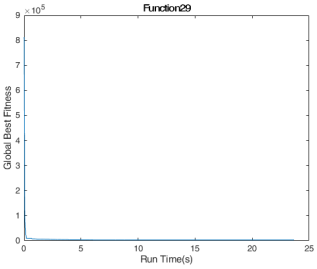
(b) Function 26



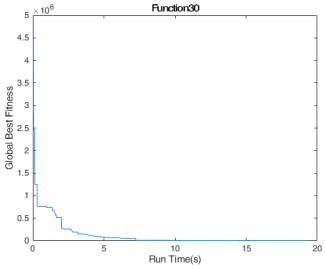
(c) Function 27



(d) Function 28



(e) Function 29



(f) Function 30

Figure 36: Run-Time vs Fitness of Functions 25-30

The original authors had results that are near and similar to the table except for f1, f2, f17 and 21 which had about 100 times or more fitter population than the reimplementations. The original paper was written such that there are gaps in logic that was filled by assumptions when the algorithm was being reimplemented. The original programming language used in the original implementation was not specified along with platform specifics which may result in test discrepancies. There also specifics that are key to how the algorithm works that were not defined by the original authors and tighter reimplementation may be needed. In another perspective, the reimplementation keeps up with some if not most algorithms that were used to compare with by the original authors. It also has had lesser deviation on some functions more than the original but also had some functions where its deviation was beat by the original.

33 Conclusion

As such, to conclude, the Lion Optimization Algorithm uses a Lion-Pride-Outcast behavior for a population to find solutions or what is thought of as a better territory or a safer place for the population to live in. The algorithm combines the effectiveness of swarm optimization in Prides and individual roaming that provides escaping optimas. It also harnesses Genetic Algorithm through mating in prides. The information exchange between Pride and Nomads is key for stuck prides to escape and for previous nomads to improve solutions.

References

[1] Sachin Desale, Akhtar Rasool, Sushil Andhale, and Priti Rane. Heuristic and meta-heuristic algorithms and their relevance to the real world: A survey. 351:2349–7084, 01 2015.

[2] R. Eberhart and J. Kennedy. A new optimizer using particle swarm theory. *MHS95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, 1995.

[3] John H. Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence.* MIT Press, 2010.

[4] B.r. Rajakumar. The lions algorithm: A new nature-inspired search algorithm. *Procedia Technology*, 6:126–135, 2012.

[5] Maziar Yazdani and Fariborz Jolai. Lion optimization algorithm (loa): A nature-inspired metaheuristic algorithm. *Journal of Computational Design and Engineering*, 3(1):24–36, 2016.