Recovering from population extinction in the Animal Life Cycle Algorithm (ALCA)

J. C. Felix-Saul, Mario Garcia Valdez

Abstract In previous work, we introduced an algorithm inspired by the biological Life Cycle of animal species, consisting of the stages: birth, growth, reproduction, and death. As in nature, population individuals grow older and age. In this algorithm's reproduction process, couples must match by mutual attraction, where sometimes individuals won't procreate offspring because of the mate's low appeal. As time passes, the environment kills its individuals either because of low fitness or age, causing occasional population extinction. When this condition presents, it is required to create a new population if evaluations are available and termination conditions are favorable to continue. Some alternatives we explored to restart this new population are the random generation of a new set of candidate solutions and the creation of a new set of candidate solutions based on either: a mix of elements from the best historically found candidate solutions set (Elite); the mutation with uniform modification from the Elite or the historically best-found solution (Champion); crossing the Champion with the Elite; the projection of the Elite towards the Champion; based on random use of the previous alternatives. In this paper, we present one of the most promising alternatives to solve population extinction caused by nature's pressure in the Life Cycle algorithm: the projection of the Elite towards the Champion, where we compare the results obtained with the latter and the random use of all the alternatives, using classic benchmark functions for optimization for comparison.

J. C. Felix-Saul
TecNM, Tijuana Institute of Technology, Tijuana, Mexico, e-mail: jose.felix201@tectijuana.edu.mx

Mario García Valdez
TecNM, Tijuana Institute of Technology, Tijuana, Mexico, e-mail: mario@tectijuana.edu.mx

1 Introduction

Biologically inspired algorithms have proven to be very effective when solving complex optimization problems [1, 2, 3], but the more challenging the problem, the more computing power it will require to solve them [4]. We consider the main reason it requires more computing power is related to the evaluation of the fitness function, which means calculating the solution to the real problem our algorithm is searching for. One technique to manage this processing power demand is to use distributed computing [5] or the cloud resources [6, 7]. This strategy allows adaptation of the required computing need according to the problem's complexity.

Traditionally, bio-inspired algorithms are developed with a sequential (synchronous) perspective [8, 9], where a process must pause for the previous task to finish before continuing. Some architectures address this issue [10, 11, 12] by working on the cloud and finding solutions on distributed technologies. In this research, we present a distributed algorithm totally built as a native cloud solution, where its processes execute asynchronously and in parallel, managing the processing workload among several computers. This strategy allows to elastically increase (or reduce) the computing power according to the nature of the challenge [13].

In previous work, we introduced an algorithm inspired by the biological life-cycle of animal species [14]. As in nature, population individuals grow older and age. In this algorithm's reproduction process, couples must match by mutual attraction, where sometimes individuals won't procreate offspring because of the mate's low appeal. As time passes, the environment kills its individuals either because of low fitness or age, causing occasional population extinction.

We define extinction as a situation in which something no longer exists; in our case scenario, when we refer to extinction in our algorithm, it means the population of individuals no longer exists. When this condition presents itself, it is required to create a new population to continue evolution (restart).

Population restart is the process in which we create a new set of individuals to replace the others that have already died (or were discarded) due to their aptitude. In a previous work [14] we used to generate a new random set of solutions, with the cost of losing all evolution knowledge. In the state of the art, we have found other methodologies that have successfully sought to solve the problem of population extinction with restarts [15, 16, 17, 18, 19].

One contribution of this paper is to prove that it is possible to evolve a population of individuals, similar to a Genetic Algorithm (GA), using a distributed, parallel, and asynchronous methodology by algorithm implementation and testing.

The main contribution of this research is to demonstrate how to improve the algorithm's behavior, by solving population extinction caused by nature's pressure in the Animal Life Cycle Algorithm with one of the most promising alternatives: the projection of the Elite towards the Champion. To validate our findings, we compare the results obtained with the latter and the random use of the more traditional alternatives (mutation and crossover), using classic benchmark functions for optimization for comparison.

This paper is organized as follows. First, we illustrate our algorithm model, the encountered problem with the occasional population extinction, and our proposed solution in section 2, followed by our experimental configuration and results in section 3, where we continue to analyze and describe some of our research findings in section 4. We finalize by presenting some inferences based on the results of our experiments in section 5.

2 Proposal

In previous work, we presented an algorithm inspired by nature [14], where we model our algorithm based on the generalization of the life cycle of animal species. Our algorithm, as in nature, consists of four stages [20]: birth, growth, reproduction, and death. One key concept of our idea is that combining those processes evolves the population. We propose to execute all these stages in parallel and asynchronously on a continuously evolving population.

2.1 Algorithm Model

This algorithm was inspired by the traditional Genetic Algorithm, meaning that all individuals have a genotype (chromosome) that is a list of values. We calculate the individual's fitness with the evaluation function and do crossover and mutation to the population. What makes our strategy different is that we don't use the concept of evolutionary generations. We manage our set of solutions as a whole that continuously evolve over time. Allowing individuals of different ages to breed and generate offspring, as it happens in nature. The crossover and mutation execute as independent processes that randomly affect the individuals.

Our algorithm's goal is to mimic the animal life cycle, where at any given moment, new individuals are born to be part of the population and participate in the collective evolution. As time passes, they grow older and mature, suffering changes throughout their lives that we chose to represent as mutations. In our analysis, we considered a couple's attraction a fundamental factor in reproduction. We thought of death's work to maintain balance in the population by enforcing the survival of the fittest. As in life, death can happen to everyone: from a newborn to the elderly, where fitness will determine the individual's longevity. We display the general model concept in Figure 1.

2.1.1 Birth

Birth is the first step of the population's evolution. For the algorithm representation, we begin with the generation of a randomly set of individuals.

2.1.2 Growth

In our algorithm representation, there is a direct correlation between elapsed time and the population's evolution. For this to be possible in our proposal, all individuals must grow older and change as time progresses. With each increment of age, an individual may improve or deteriorate. We chose to manifest this idea by performing a small mutation in each change.

2.1.3 Reproduction

In this step, we select a random pair of individuals and evaluate their couple's attraction, where fitness will be the decisive factor that impacts its value. The higher the fitness of the individual, the more attractive it will appear to fellow individuals. As a consequence of the previous concept, we can anticipate that not all couples will produce offspring, meaning reproduction will not always be successful.

2.1.4 Death

Death is the last step of the animal life cycle, as in our algorithm. We chose death as a representation of all the environmental challenges for survival, where its execution enforces the survival of the fittest. As time progresses, so does increase nature's level of danger. The better the individual fitness will increase its chances of survival.

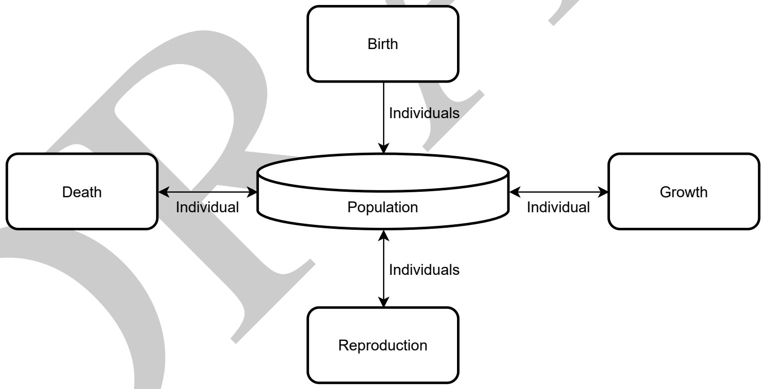

Fig. 1 Animal Life Cycle Algorithm model.

2.2 Problem

In this algorithm's reproduction process, we find mating couples by mutual attraction, where sometimes individuals won't procreate offspring because of the mate's low appeal. As time passes, the environment could kill its individuals either because they have low fitness or age, leading to extinction, meaning there are no more individuals in the population left to reproduce. When this condition presents, if evaluations are available and termination conditions enable the algorithm to continue the search, it is required to create a new set of individuals. We illustrate the general flowchart for the Life-Cycle algorithm in Figure 2.

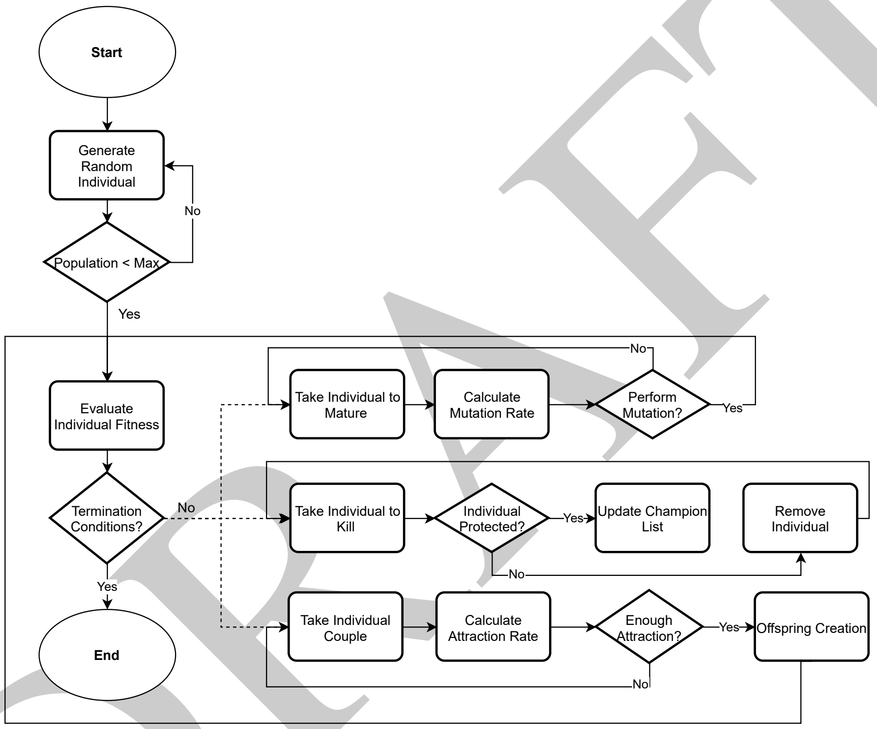

Fig. 2 Animal Life-Cycle Algorithm general flowchart.

2.3 Proposed Solution

In this article, we test various strategies to generate a new population to replace individuals that have already died (or were discarded) due to their aptitude, as happens in the life cycle. We previously used to generate a new random set of solutions, with the cost of losing all evolution knowledge.

This article proposes some strategies to carry out this operation, where the presented distribution strategy obtained the best results. We propose one promising alternative to restart the population, to solve extinction caused by nature's pressure in the Life Cycle algorithm: the projection of the historically best-found candidate solutions (Elite) towards the best-found candidate solution (Champion).

The strategy we describe in this paper use an elite set, that includes the historically best-found candidate solutions to take advantage of the historical evolution. Our goal is for the offspring to be somewhat close to the best individuals of the Elite. We use the golden ratio to compute the distance between new individuals in the population, making an improved distribution. We can consider our strategy an improvement over other alternatives because of the findings of our latter experiments.

Based on the historically best candidate solution and the elite set, we use the Golden Ratio (shown in Equation 1) to compute the distance between new individuals in the population, making an improved distribution. It randomly selects a solution from the elite set and uses the Golden Ratio [21, 22, 23, 24, 25] to compute multiple solutions against the best-found candidate.

$$\alpha = \frac{1+\sqrt{5}}{2} \tag{1}$$

This strategy is how we can calculate the projection of the historically Elite towards the Champion solution. To illustrate this concept, we include Figure 3, placing the Elite solution on the far right and the Champion solution on the left corner. We can observe the higher amount of new solutions progressively closer to the Champion.

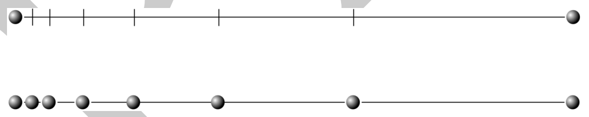

Fig. 3 Projection of an Elite solution (found on the right) towards the Champion (at the left).

This process generates new solutions that will be the new population set. It uses the Golden Ratio to progressively approximate from an Elite solution to the Champion. This strategy facilitates to find apparently hidden solutions, having an exploitation emphasis. Figure 4 displays the projection of five Elite solutions (on the corners) towards the Champion (in the center).

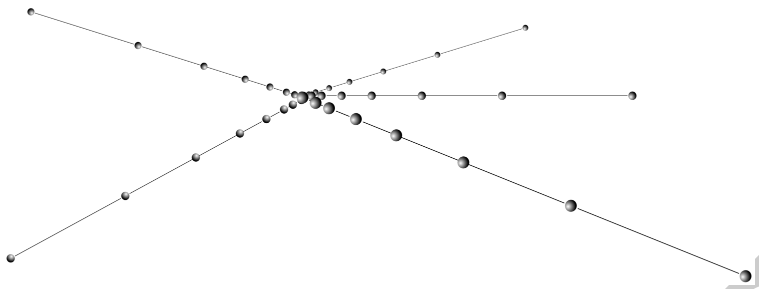

Fig. 4 Projection of five Elite solutions towards the Champion, results in a mini-swarm.

3 Experiments

As a proof-of-concept, we implemented the algorithm with Docker containers, where we compare the results obtained with the projection of the Elite towards the Champion strategy and the random use of the more traditional alternatives (such as mutation and crossover), using classic benchmark functions for optimization as implemented by the DEAP (Distributed Evolutionary Algorithms in Python) library [26].

3.1 Experimental Setup

The following tables show the configuration used to execute the experiments for the analysis and comparison in this paper. In Table 1, we can find the General Configuration for the Animal Life Cycle Algorithm. All values from this configuration remained constant during all phases of the experiment runs. The following table, Table 2, shows the values used for the Classic Benchmark Functions for Optimization, where these are adjusted accordingly for each function.

Table 1 Animal Life Cycle Algorithm (ALCA) General Configuration.

ALCA General Configuration	
Function name	ALL
Population	500
Target error	1.00E-08
Crossover rate	100
Mutation rate	7
Max age	40
Tournament rep.	100
Sample size	20
Base approval	80
Goal approval	200

Table 2 Classic Benchmark Functions for Optimization Configuration.

Benchmark Configuration Setup							
Function name	Ackley	Bohachevsky	Griewank	Rastrigin	Sphere	Rosenbrock	Rosenbrock
Dimensions	5, 10	5, 10	5, 10	5, 10	5, 10	5	10
Max evaluations	200,000	200,000	200,000	200,000	200,000	500,000	1,000,000
Max stagnation	50,000	50,000	50,000	50,000	50,000	125,000	250,000
Target fitness	-20.0	-12.0	-150.0	-80.0	-50.0	-3500.0	-3500.0
Bound matrix	[-32, 32]	[-2, 2]	[-500, 500]	[-5, 5]	[-5, 5]	[-2, 2]	[-2, 2]

3.2 Experiment Results

For each experiment, we ran 30 independent executions per algorithm and dimensions specified. We recorded the following results: best-found error average, standard deviation, total number of evaluations, and total elapsed time (in seconds). The labels used on the results of our summarized experiments tables are the following:

- Error: is the best-found error average.
- St-Dev: is the standard deviation calculated from the error.
- Evals.: is the total number of evaluations the algorithm executed.
- Time: is the total elapsed or wall clock time, in seconds.

The alternatives to restart this new population are the creation of a new set of candidate solutions based on either: 1. Crossing the Champion with the Elite; 2. The mutation with uniform modification from the Elite; 3. The projection of the Elite towards the Champion; 4. Based on random use of the previous alternatives. We compared three strategies by grouping the previous alternatives in the following:

- Xover Mutation: Crossing the Champion with the Elite, and the mutation with uniform modification from the Elite.
- Elite Projection: The projection of the Elite towards the Champion.
- All Random: Based on random use of the previous alternatives.

3.2.1 Ackley Benchmark Function Experiment

The goal of the first experiment was to confirm that this paper's proposed strategy obtained improved results over our basic technique, which created a new population with a random generation of a new set of candidate solutions. We chose Ackley as the first Classic Benchmark Function for Optimization to test our algorithm behavior, where Figure 5 shows its corresponding Box and Whisker chart, and we summarized the results in Tables 3 and 4 (for five and ten dimensions, respectively).

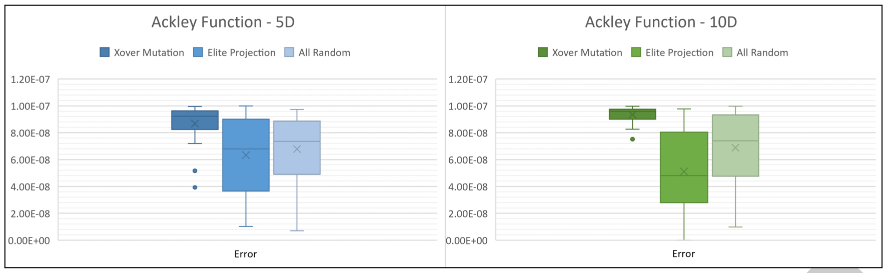

Fig. 5 Box and whisker chart for the Ackley Benchmark Function, 5 vs 10 Dimensions.

Table 3 Ackley Benchmark Function summarized experiments table for 5 Dimensions.

Ackley Function - 5 Dimensions				
	Error	**St-Dev**	**Evals.**	**Time**
Xover Mutation	8.69E-08	1.36E-08	46,331	49.9
Elite Projection	6.33E-08	2.70E-08	13,811	18.1
All Random	6.78E-08	2.55E-08	30,236	32.8

Table 4 Ackley Benchmark Function summarized experiments table for 10 Dimensions.

Ackley Function - 10 Dimensions				
	Error	**St-Dev**	**Evals.**	**Time**
Xover Mutation	9.35E-08	5.93E-09	104,842	111.7
Elite Projection	5.11E-08	3.13E-08	16,019	20.1
All Random	6.89E-08	2.75E-08	43,631	46.3

3.2.2 Bohachevsky Benchmark Function Experiment

As our second experiment to further validate and prove our algorithm behavior, we chose Bohachevsky Classic Benchmark Function for Optimization. Figure 6 shows its corresponding Box and Whisker chart, where we summarized the results in Tables 5 and 6 (for five and ten dimensions, respectively). Due to the observation and analysis of these results, we identify our proposed strategy obtained solutions in the same order as the traditional Xover Mutation, in a significantly reduced number of evaluations.

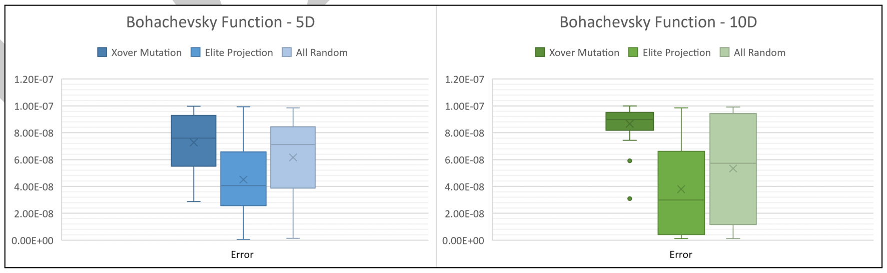

Fig. 6 Box and whisker chart for the Bohachevsky Benchmark Function, 5 vs 10 Dimensions.

Table 5 Bohachevsky Benchmark Function summarized experiments table for 5 Dimensions.

Bohachevsky Function - 5 Dimensions				
	Error	**St-Dev**	**Evals.**	**Time**
Xover Mutation	7.28E-08	2.20E-08	15,010	24.2
Elite Projection	4.51E-08	2.94E-08	8,266	12.6
All Random	6.16E-08	2.85E-08	11,998	21.7

Table 6 Bohachevsky Benchmark Function summarized experiments table for 10 Dimensions.

Bohachevsky Function - 10 Dimensions				
	Error	**St-Dev**	**Evals.**	**Time**
Xover Mutation	8.66E-08	1.38E-08	40,166	44.4
Elite Projection	3.80E-08	3.34E-08	8,506	12.9
All Random	5.34E-08	3.76E-08	19,537	26.8

3.2.3 Griewank Benchmark Function Experiment

The third experiment objective was to confirm Animal Life Cycle Algorithm continued finding solutions in a reduced number of evaluations. For this test, we selected the Griewank Benchmark Function for Optimization. Figure 7 shows its corresponding Box and Whisker chart, where we summarized the results in Tables 7 and 8 (for five and ten dimensions, respectively). With this analysis, we confirmed our strategy continued to obtain solutions in the same order as expected, in a reduced number of evaluations and less time.

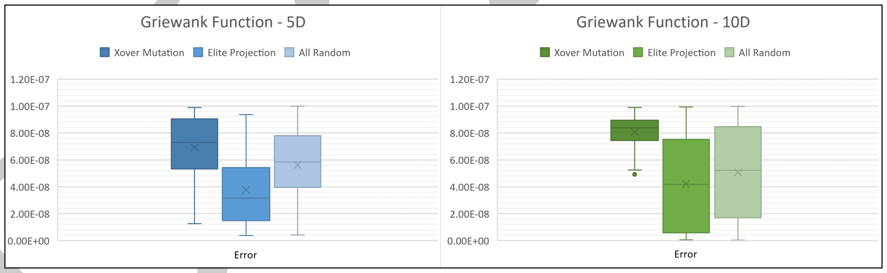

Fig. 7 Box and whisker chart for the Griewank Benchmark Function, 5 vs 10 Dimensions.

Table 7 Griewank Benchmark Function summarized experiments table for 5 Dimensions.

Griewank Function - 5 Dimensions				
	Error	**St-Dev**	**Evals.**	**Time**
Xover Mutation	6.92E-08	2.40E-08	44,566	48.3
Elite Projection	3.77E-08	2.74E-08	16,412	20.1
All Random	5.63E-08	2.99E-08	35,761	38.5

Table 8 Griewank Benchmark Function summarized experiments table for 10 Dimensions.

Griewank Function - 10 Dimensions				
	Error	**St-Dev**	**Evals.**	**Time**
Xover Mutation	8.07E-08	1.39E-08	92,885	100.0
Elite Projection	4.21E-08	3.40E-08	13,534	17.3
All Random	5.08E-08	3.54E-08	40,544	52.5

3.2.4 Rastrigin Benchmark Function Experiment

This experiment's goal was to follow and study the behavior of our proposed strategy, to repeat the results in the same order as expected, in a reduced number of both time and evaluations. We selected Rastrigin Classic Benchmark Function for Optimization for this experiment. Figure 8 shows its corresponding Box and Whisker chart, where we summarized the results in Tables 9 and 10 (for five and ten dimensions, respectively).

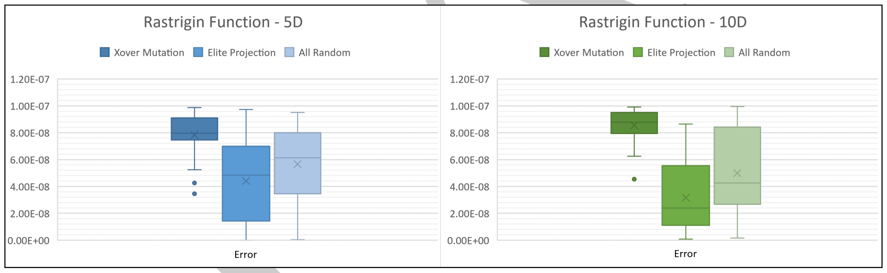

Fig. 8 Box and whisker chart for the Rastrigin Benchmark Function, 5 vs 10 Dimensions.

Table 9 Rastrigin Benchmark Function summarized experiments table for 5 Dimensions.

Rastrigin Function - 5 Dimensions				
	Error	**St-Dev**	**Evals.**	**Time**
Xover Mutation	7.85E-08	1.59E-08	22,322	34.1
Elite Projection	4.42E-08	3.26E-08	10,860	14.8
All Random	5.66E-08	2.81E-08	21,122	35.0

Table 10 Rastrigin Benchmark Function summarized experiments table for 10 Dimensions.

Rastrigin Function - 10 Dimensions				
	Error	**St-Dev**	**Evals.**	**Time**
Xover Mutation	8.56E-08	1.23E-08	53,579	58.7
Elite Projection	3.17E-08	2.57E-08	8,704	13.0
All Random	5.00E-08	3.16E-08	24,376	27.4

3.2.5 Sphere Benchmark Function Experiment

For our fifth experiment we chose the Sphere Classic Benchmark Function for Optimization. Even though the sphere function is considered one of the simplest to solve, the same behavior was consistent for the algorithm, where it showed a reduced number of evaluations and execution time. We can confirm this affirmation with the results shown in this experiment. Figure 9 displays its corresponding Box and Whisker chart, where we summarized the results in Tables 11 and 12 (for five and ten dimensions, respectively).

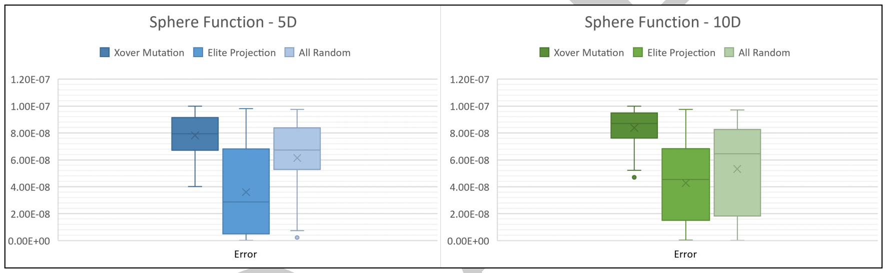

Fig. 9 Box and whisker chart for the Sphere Benchmark Function, 5 vs 10 Dimensions.

Table 11 Sphere Benchmark Function summarized experiments table for 5 Dimensions.

Sphere Function - 5 Dimensions				
	Error	**St-Dev**	**Evals.**	**Time**
Xover Mutation	7.82E-08	1.65E-08	11,312	14.5
Elite Projection	3.60E-08	3.27E-08	7,056	10.9
All Random	6.14E-08	2.79E-08	9,158	12.2

Table 12 Sphere Benchmark Function summarized experiments table for 10 Dimensions.

Sphere Function - 10 Dimensions				
	Error	**St-Dev**	**Evals.**	**Time**
Xover Mutation	8.39E-08	1.45E-08	30,769	34.4
Elite Projection	4.28E-08	3.06E-08	9,236	13.3
All Random	5.32E-08	3.38E-08	21,465	26.3

3.2.6 Rosenbrock Benchmark Function Experiment

The goal of our sixth and last experiment was to study the behavior of the Animal Life-Cycle Algorithm when tested with the Rosenbrock Benchmark Function for Optimization. From all of our previous evaluation functions, Rosenbrock presented the most difficult challenge for our algorithm. This test facilitated the differentiation between the three compared strategies because the solutions found were in different numeric order. After all tests, the Elite Projection proved to be the best alternative. We can corroborate this by analyzing Figures 10 and 11, which shows its corresponding Box and Whisker chart. We present the summarized results in Tables 13 and 14 (for five and ten dimensions, respectively).

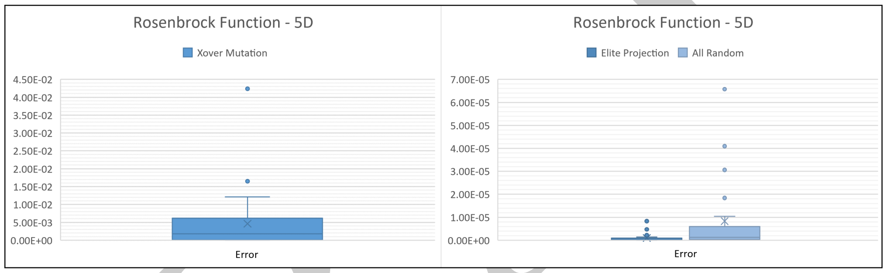

Fig. 10 Box and whisker chart for the Rosenbrock Benchmark Function, 5 Dimensions.

Table 13 Rosenbrock Benchmark Function summarized experiments table for 5 Dimensions.

Rosenbrock Function - 5 Dimensions				
	Error	**St-Dev**	**Evals.**	**Time**
Xover Mutation	4.65E-03	8.29E-03	500,000	646.4
Elite Projection	9.62E-07	1.86E-06	307,878	427.0
All Random	8.30E-06	1.58E-05	491,400	697.8

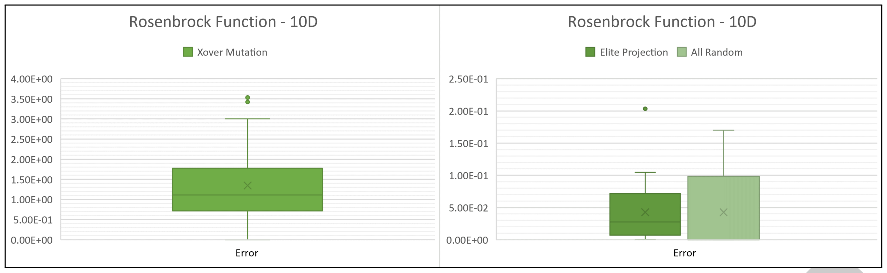

Fig. 11 Box and whisker chart for the Rosenbrock Benchmark Function, 10 Dimensions.

Table 14 Rosenbrock Benchmark Function summarized experiments table for 10 Dimensions.

Rosenbrock Function - 10 Dimensions				
	Error	**St-Dev**	**Evals.**	**Time**
Xover Mutation	1.35E+00	9.83E-01	1,000,000	1,384.3
Elite Projection	4.26E-02	4.34E-02	1,000,000	1,146.0
All Random	4.27E-02	6.02E-02	952,038	1,147.8

4 Discussion

Imagine being an astronaut on a mission repairing a space station in the middle of space. Suddenly, a block of asteroids hits the craft, and you lose your oxygen tank while the explosion pushes you away. From afar, you see the area where your tank might be, covered with something similar to a gas leaked from the station that blocks all visibility. Finding the tank is the only hope to return to your main ship some miles away. The oxygen reserve from your suit is running out, and there are only a couple of minutes left. Now hold your breath in real life and think, what strategy will you choose to search for your tank? We repeated this mental exercise several times until we found the best solution (or blacked out).

If your life depended on it, would you choose a Genetic Algorithm, a variant of a Particle Swarm Optimization Algorithm, or another? We searched for a way to mix both, at least in some way, take the main idea or concepts from them and make them work together. We strongly believe it is possible to combine the features, as previous work has proven to obtain outstanding results [11, 27, 10, 2]. Our goal was to create a swarm out of the best individuals found in evolution. We probably have a long way to go, but at least we are getting promising results. From our experiments, all six Classic Benchmark Functions for Optimization (Ackley, Bohachevsky, Griewank, Rastrigin, Sphere, and Rosenbrock) proved the same behavior for the Animal Life Cycle Algorithm: finding the expected solution in a reduced number of evaluations and less execution time.

5 Conclusions

The Animal Life Cycle Algorithm allows for multiple parameters' fine-tuning, facilitating the freedom to experiment with different alternatives to restart the population and solve extinction caused by nature's pressure, which will impact how quickly, and the quality of its found solution. We compared the results obtained with the use of historical elite projection versus other alternatives, using classic benchmark functions for optimization for comparison, where it showed favorable and promising results.

As the nature of the challenge increases, it is indispensable to have an elastic, scalable, and fault-tolerant model designed with cloud collaboration processes, and asynchronous communication. We have proven that it is possible to evolve a population, using a distributed, parallel, and asynchronous strategy, inspired by the Genetic Algorithm (GA). To further validate this work, we could use some more demanding (or control) problem that requires calculating real numbers [28, 29]. We implemented the algorithm using Docker containers.

Acknowledgements This paper has been supported in part by TecNM Project 15340.22-P.

References

1. Castillo, O., Valdez, F., Soria, J., Amador-Angulo, L., Ochoa, P., Peraza, C.: Comparative study in fuzzy controller optimization using bee colony, differential evolution, and harmony search algorithms. Algorithms 12(1), 9 (2019)
2. Valdez, F.: Swarm intelligence: A review of optimization algorithms based on animal behavior. Recent Advances of Hybrid Intelligent Systems Based on Soft Computing pp. 273–298 (2021)
3. Acherjee, B., Maity, D., Kuar, A.S.: Ultrasonic machining process optimization by cuckoo search and chicken swarm optimization algorithms. International Journal of Applied Metaheuristic Computing (IJAMC) 11(2), 1–26 (2020)
4. Ontiveros, E., Melin, P., Castillo, O.: High order α-planes integration: a new approach to computational cost reduction of general type-2 fuzzy systems. Engineering Applications of Artificial Intelligence 74, 186–197 (2018)
5. Thain, D., Tannenbaum, T., Livny, M.: Distributed computing in practice: the condor experience. Concurrency and computation: practice and experience 17(2-4), 323–356 (2005)
6. García-Valdez, M., Mancilla, A., Trujillo, L., Merelo, J.J., Fernández-de Vega, F.: Is there a free lunch for cloud-based evolutionary algorithms? In: 2013 IEEE Congress on Evolutionary Computation. pp. 1255–1262. IEEE (2013)
7. Eshratifar, A.E., Esmaili, A., Pedram, M.: Bottlenet: A deep learning architecture for intelligent mobile cloud computing services. In: 2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED). pp. 1–6. IEEE (2019)
8. Porto, V.W.: Evolutionary programming. In: Evolutionary Computation 1, pp. 127–140. CRC Press (2018)
9. Back, T.: Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms. Oxford university press (1996)
10. Valdez, M.G., Guervós, J.J.M.: A container-based cloud-native architecture for the reproducible execution of multi-population optimization algorithms. Future Generation Computer Systems 116, 234–252 (2021)

11. García-Valdez, M., Trujillo, L., Merelo, J.J., de Vega, F.F., Olague, G.: The evospace model for pool-based evolutionary algorithms. Journal of Grid Computing 13(3), 329–349 (2015)
12. Merelo, J.J., García-Valdez, M., Castillo, P.A., García-Sánchez, P., Cuevas, P., Rico, N.: Nodio, a javascript framework for volunteer-based evolutionary algorithms: first results. arXiv preprint arXiv:1601.01607 (2016)
13. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., et al.: A view of cloud computing. Communications of the ACM 53(4), 50–58 (2010)
14. Felix-Saul, J.C., Valdez, M.G., Guervós, J.J.M.: A Novel Distributed Nature-Inspired Algorithm for Solving Optimization Problems, pp. 107–119. Springer International Publishing, Cham (2022)
15. Hellwig, M., Beyer, H.G.: A modified matrix adaptation evolution strategy with restarts for constrained real-world problems. In: 2020 IEEE Congress on Evolutionary Computation (CEC). pp. 1–8. IEEE (2020)
16. Loshchilov, I.: Cma-es with restarts for solving cec 2013 benchmark problems. In: 2013 IEEE Congress on Evolutionary Computation. pp. 369–376. Ieee (2013)
17. Jansen, T., Zarges, C.: Aging beyond restarts. In: Proceedings of the 12th annual conference on Genetic and evolutionary computation. pp. 705–712 (2010)
18. Tendresse, I.l., Gottlieb, J., Kao, O.: The effects of partial restarts in evolutionary search. In: International Conference on Artificial Evolution (Evolution Artificielle). pp. 117–127. Springer (2001)
19. Mathias, K.E., Schaffer, J.D., Eshelman, L.J., Mani, M.: The effects of control parameters and restarts on search stagnation in evolutionary programming. In: International Conference on Parallel Problem Solving from Nature. pp. 398–407. Springer (1998)
20. Read, K., Ashford, J.: A system of models for the life cycle of a biological organism. Biometrika 55(1), 211–221 (1968)
21. Nematollahi, A.F., Rahiminejad, A., Vahidi, B.: A novel meta-heuristic optimization method based on golden ratio in nature. Soft Computing 24(2), 1117–1151 (2020)
22. Gaikwad, P.S., Kulkarni, V.B.: Face recognition using golden ratio for door access control system. In: Advances in Signal and Data Processing, pp. 209–231. Springer (2021)
23. Khesin, B., Wang, H.: The golden ratio and hydrodynamics. The Mathematical Intelligencer 44(1), 22–27 (2022)
24. Battaloglu, R., Simsek, Y.: On new formulas of fibonacci and lucas numbers involving golden ratio associated with atomic structure in chemistry. Symmetry 13(8) (2021), https://www.mdpi.com/2073-8994/13/8/1334
25. Narushin, V.G., Griffin, A.W., Romanov, M.N., Griffin, D.K.: Measurement of the neutral axis in avian eggshells reveals which species conform to the golden ratio. Annals of the New York Academy of Sciences n/a(n/a) (2022), https://nyaspubs.onlinelibrary.wiley.com/doi/abs/10.1111/nyas.14895
26. Fortin, F.A., De Rainville, F.M., Gardner, M.A.G., Parizeau, M., Gagné, C.: Deap: Evolutionary algorithms made easy. The Journal of Machine Learning Research 13(1), 2171–2175 (2012)
27. García-Valdez, M., Merelo, J.J.: Event-driven multi-algorithm optimization: Mixing swarm and evolutionary strategies. In: International Conference on the Applications of Evolutionary Computation (Part of EvoStar). pp. 747–762. Springer (2021)
28. Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. Evolutionary computation 10(2), 99–127 (2002)
29. Miikkulainen, R., Liang, J., Meyerson, E., Rawal, A., Fink, D., Francon, O., Raju, B., Shahrzad, H., Navruzyan, A., Duffy, N., et al.: Evolving deep neural networks. In: Artificial intelligence in the age of neural networks and brain computing, pp. 293–312. Elsevier (2019)