A novel distributed nature-inspired algorithm for solving optimization problems*

J. C. Felix-Saul[1], Mario García Valdez[1], and Juan J. Merelo Guervós[2]

[1] Department of Graduate Studies, Tijuana Institute of Technology, Tijuana, Mexico
[2] Department of Computer Architecture and Technology, Universidad de Granada, Granada, Spain

Abstract. Several bio-inspired algorithms use population evolution as analogies of nature. In this paper, we present an algorithm inspired by the biological life-cycle of animal species, which consists of several stages: birth, growth, reproduction, and death. As in nature, we intend to execute all these stages in parallel and asynchronously on a population that evolves constantly. From the ground up, we designed the algorithm as a cloud-native solution using the cloud available resources to divide the processing workload, among several computers or running the algorithm as a cloud service. The algorithm works concurrently and asynchronously on a constantly evolving population, using different computers (or containers) independently, eliminating waiting times between processes. This algorithm seeks to imitate the natural life cycle, where new individuals are born at any moment and mature over time, where they age and suffer mutations throughout their lives. In reproduction, couples match by mutual attraction, where they may have offspring. Death can happen to everyone: from a newborn to an aged adult, where the individual's fitness will impact their longevity. As a proof-of-concept, we implemented the algorithm with Docker containers by solving the OneMax problem comparing it with a basic (sequential) GA algorithm, where it showed favorable and promising results.

Keywords: Distributed Bioinspired Algorithms · Genetic Algorithms · Cloud Computing.

1 Introduction

Bio-inspired algorithms have been very successful when used to solve complex optimization problems, but as their complexity increases so does the computing power required. One strategy to address this processing need is to use distributed computing, or the resources available in the cloud to help find the solution. This strategy gives us elasticity to increase (or reduce) the computing power to achieve a balance according to the nature of the problem.

One of the problems that we have identified in this research is that most bio-inspired algorithms are designed with a traditionally sequential perspective,

* Supported by organization x.

where each process must wait for the previous task to finish before continuing. Some architectures address this issue, what distinguishes us, is that our proposal presents an algorithm designed in its entirety as a native solution in the cloud, fully distributed, where its processes are executed in parallel and asynchronously. This technique makes it easy to scale the computing power according to the complexity required by the problem.

We designed our algorithm using observation, analysis, and abstraction in nature to identify what most species in the animal kingdom have in common, where individuals of a population that best adapt to the environment have greater chances of survival, reproduction, and improvement of the species. Imagining as if the task of writing these rules of evolution was in our hands, questioning how we could solve it?

Similar to nature, we identified what most species have in common in the life cycle. Our algorithm works on a constantly evolving population, experiencing the stages that all living beings undergo: being born, growing up, reproducing, and dying, where each of these stages will be processes (of our algorithm) that randomly affect the population individuals, emulating the organic way.

This algorithm seeks to imitate the natural life cycle, where new individuals are born at any moment and mature over time, where they age and suffer mutations throughout their lives. In reproduction, couples match by mutual attraction, where they may have offspring. Death can happen to everyone: from a newborn to an aged adult, where the individual's fitness will impact their longevity.

In this research, we seek to demonstrate that it is possible to evolve a population of individuals, similar to a Genetic Algorithm (GA), using a distributed, parallel, and asynchronous methodology.

2 Description of the sections of the Paper

3 State of the art

Multiple citations are grouped [6, 2, 7, 4, 3, 1, 5].

3.1 Background

One of the publications with the most influence on our research proposes a cloud-native optimization framework, that can include multiple (population-based) algorithms [6]. In their research, they work with ideas such as those mentioned in Fig. 1.

4 Proposal

From the ground up, we designed the algorithm as a cloud-native solution using the cloud available resources to divide the processing workload, among several

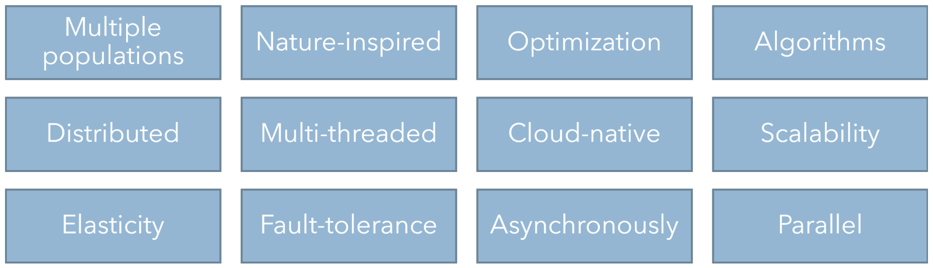

Fig. 1. These ideas planted the seed of thought in our current presentation.

computers or running the algorithm as a cloud service. [The algorithm works concurrently and asynchronously on a constantly evolving population, using different computers (or containers) independently, eliminating waiting times between processes.]

In this paper, we present an algorithm inspired by the biological life-cycle of animal species, which consists of several stages: birth, growth, reproduction, and death. As in nature, we intend to execute all these stages in parallel and asynchronously on a population that evolves constantly. We show the LifeCycle Algorithm Proposal Model in Fig. 2.

This algorithm seeks to imitate the natural life cycle, where new individuals are born at any moment and mature over time, where they age and suffer mutations throughout their lives. In reproduction, couples match by mutual attraction, where they may have offspring. Death can happen to everyone: from a newborn to an aged adult, where the individual's fitness will impact their longevity.

4.1 Birth

This algorithm starts with a randomly generated population, where the processes will interact with the population independently. This means that at any given time, any individual can experience any of these processes. Birth is the first process of our algorithm, and it is responsible for the initial generation of individuals.

4.2 Growth

As in nature, all individuals constantly grow, mature, or age. With increasing age, individuals may lose strength but also gain more knowledge to solve problems. We represent this with a possible mutation in each increment of age. The growth process will take an individual to assess whether it's ready to mature and undergo changes.

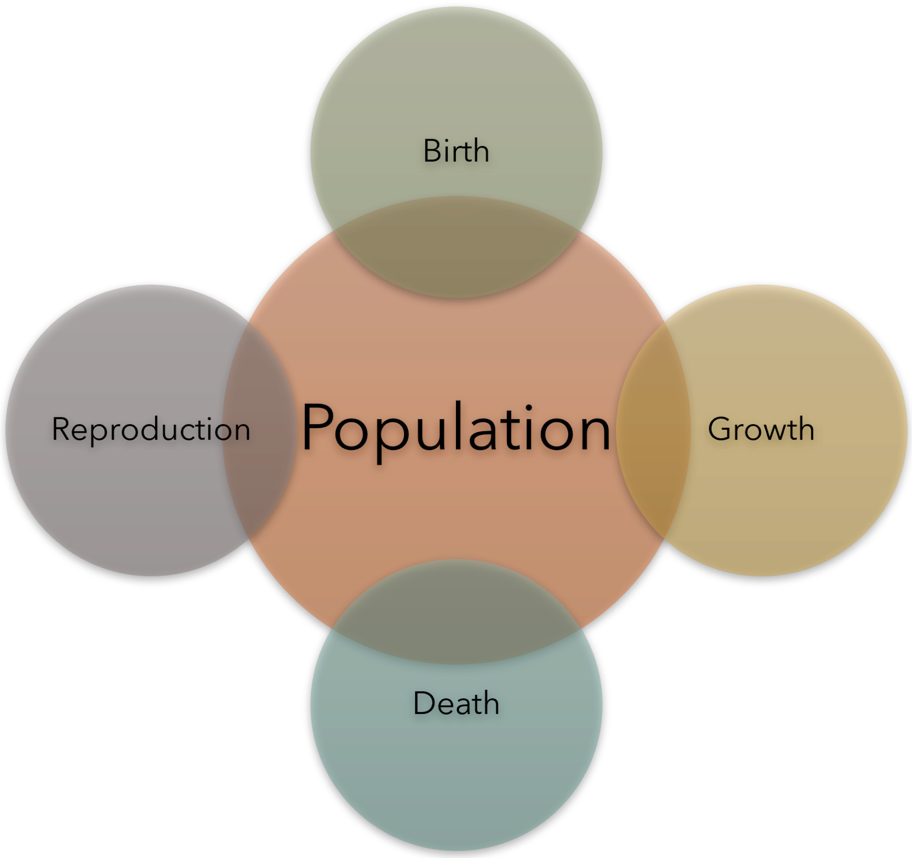

Fig. 2. LifeCycle Algorithm Proposal Model.

4.3 Reproduction

The attraction of a couple will depend on fitness: the better individual's fitness, the more attractive it will be, making it easier to find mating matches. Not all couples will be compatible, so reproduction will not always be possible, but the problem arises: how to quantify the attraction between two individuals?. This process will be taking random pairs of individuals to evaluate their attraction as a couple, to try to breed; when the gestation is successful, a new pair of individuals will be born (as the offspring).

4.4 Death

The death stage represents the challenges and adversities that life presents (to overcome). This process evaluates the individual resistance to survive in the environment. The better fitness the individual has will increase its chances of survival. As time progresses, the demands of nature will also increase, pushing for only the best individuals to survive.

5 Experiments

As a proof-of-concept, we implemented the algorithm with Docker containers by solving the OneMax problem to compare it with a traditional (sequential) GA algorithm using the DEAP library. The OneMax problem uses a genetic algorithm to evolve a population of randomly generated individuals with zeros and ones, and it stops until a solution of only ones is found.

In this experiment, we needed to match or balance the experimentation parameters according to the specifications of the algorithm used by the DEAP library. This is to be able to verify if our algorithm would also converge on the solutions in a similar number of evaluations and execution time.

5.1 Experimental Setup

Table 1 shows the configuration for the OneMax DEAP version, while Table 2 shows the LifeCycle Algorithm setup.

5.2 Results

Table 3 shows the OneMax DEAP experiment results, while Table 4 shows the LifeCycle Algorithm experiment results.

6 Discussion

In our experiments, we found that for simple problems, the cost of communication between containers will increase the total time of execution, however, we believe that the opposite must also be true: for complex problems, distributing

Table 1. OneMax DEAP configuration.

OneMax (DEAP)	
Population	60
Max Generation	20
Stagnation	Off
Chromosome Length	20
Target Fitness	20
Crossover Rate	100
Mutation Rate	7
Tourney Reprod. Rate	50
Tourney Sample	3

Table 2. LifeCycle Algorithm configuration.

LifeCycle Algorithm	
Population	60
Max Generation	20
Stagnation	Off
Chromosome Length	20
Target Fitness	20
Crossover Rate	100
Mutation Rate	7
Tourney Reprod. Rate	50
Tourney Sample	4
Max Age	80
Open Reprod. Rate	5
Closed Reprod. Rate	45
Fertility Rate	100
Base Approval	80
Goal Approval	100
Max Wait Time	2

Table 3. OneMax DEAP experiment results.

OneMax DEAP				
Run No.	Generation	Best Individual	Total Evaluations	Total Time (seconds)
1	6	20	360	0.016
2	5	20	300	0.032
3	7	20	420	0.061
4	9	20	540	0.081
5	8	20	480	0.080
6	6	20	360	0.062
7	7	20	420	0.062
8	6	20	360	0.016
9	8	20	480	0.058
10	6	20	360	0.050
11	7	20	420	0.073
12	8	20	480	0.076
13	8	20	480	0.065
14	6	20	360	0.060
15	6	20	360	0.045
16	6	20	360	0.059
17	6	20	360	0.061
18	6	20	360	0.058
19	5	20	300	0.060
20	7	20	420	0.067
21	7	20	420	0.060
22	7	20	420	0.085
23	7	20	420	0.071
24	4	20	300	0.049
25	6	20	360	0.050
26	9	20	540	0.081
27	6	20	420	0.074
28	6	20	360	0.064
29	6	20	360	0.065
30	4	20	240	0.016
Avg.	6.5	20	394	0.059

Table 4. LifeCycle Algorithm experiment results.

LifeCycle Algorithm				
Run No.	Generation	Best Individual	Total Evaluations	Total Time (seconds)
1	8	20	486	6.65
2	6	20	381	5.20
3	6	20	369	5.69
4	7	20	434	6.37
5	12	20	731	8.20
6	5	20	306	4.55
7	13	20	764	8.34
8	4	20	233	4.09
9	12	20	692	7.27
10	6	20	338	4.76
11	9	20	559	7.78
12	12	20	730	8.35
13	4	20	219	3.43
14	8	20	462	5.28
15	4	20	217	3.37
16	4	20	234	4.54
17	4	20	256	4.35
18	12	20	714	8.14
19	12	20	694	7.88
20	15	20	892	9.74
21	11	20	633	7.98
22	3	20	169	4.02
23	10	20	611	6.89
24	10	20	629	7.32
25	7	20	416	6.22
26	7	20	446	5.55
27	6	20	379	5.95
28	8	20	466	6.98
29	3	20	197	3.36
30	6	20	381	6.16
Avg.	7.8	20	468	6.15

the work on multiple resources must reduce time, making the communication cost negligible.

This scheme allows for multiple parameter's fine tunning, granting us the freedom to experiment with different selection and reproduction strategies simultaneously, which will impact how fast it finds the solution and the obtained quality.

7 Conclusions

We implemented the algorithm with Docker containers by solving the OneMax problem comparing it with a traditional (sequential) GA algorithm, where it showed favorable and promising results. To further validate this work, we could use control or some more complex and demanding problem that requires computing real numbers.

As the complexity of problems increases, it is essential to have a scalable, replicable, and fault-tolerant model that uses collaborative techniques to work in the cloud, where multiple resources will be communicating asynchronously. This research has shown that it is possible to evolve a population of individuals, similar to a Genetic Algorithm (GA), using a distributed, parallel, and asynchronous methodology.

References

1. 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)
2. Krejca, M.S., Witt, C.: Lower bounds on the run time of the univariate marginal distribution algorithm on onemax. Theoretical Computer Science **832**, 143–165 (2020)
3. Merelo, J.J., Castillo, P.A., García-Sánchez, P., de las Cuevas, P., Rico, N., García Valdez, M.: Performance for the masses: Experiments with a web based architecture to harness volunteer resources for low cost distributed evolutionary computation. In: Proceedings of the Genetic and Evolutionary Computation Conference 2016. pp. 837–844 (2016)
4. 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)
5. Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. Evolutionary computation **10**(2), 99–127 (2002)
6. 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)
7. Witt, C.: Upper bounds on the running time of the univariate marginal distribution algorithm on onemax. Algorithmica **81**(2), 632–667 (2019)