An Event-based Architecture for Multi-population Optimization Algorithms

Erick Vargas Minguela[1][0000−1111−2222−3333], Mario Garcia Valdez[1][1111−2222−3333−4444], and Third Author[2][2222−−3333−4444−5555]

National Technological Institute of Mexico erick.vargas.minguela@gmail.com
{abc,lncs}

Abstract. In this work, we present a software implementation that follows an event-driven architecture, designed to asynchronously distribute the processing of population-based algorithms. The search algorithm uses a multipopulation approach, creating multiple populations with different parameters of execution, allowing the hybridation of multiple algorithms, in this case Genetic Algorithms (GAs) and Particle Swarm Optimization (PSO). Using a message queue, each population is manipulated by asynchronous serverless functions, taking advantage of functional programming. The cloud-ready javascript implementation, includes a web-based application for the configuration and interaction with algorithms. We executed several experiments in order to validate the system, using benchmark functions and several configurations. Results show ...

Keywords: Multi-population · Asynchronous · Sub-population · Serverless · Distributed.

1 Introduction

A universe of solutions can exist for a single optimization problem and sometimes is too big and complex to solve it traditionally. That is why heuristic, population-based algorithms, and other types of methods can be applied. Population-based algorithms are useful to solve combinatory problems; however, the performance of an algorithm depends on the problem and initial conditions. Usually, an algorithm is suitable for only a specific type of problem, and it is not always easy to know which algorithm is more fitting. Even assuming that we selected an appropriate algorithm, we must also find the appropriate parameters. To deal with these problems, parallel and hybrid architectures have been proposed. The clear advantage of these architectures is the faster execution of the algorithms. However, there is also the ability to change the search dynamically, having multiple algorithms with different parameters interacting with each other at the same time.

In this work, we propose the use of several serverless functions to process small fragments from a population in a distributed way, turning a population into a multi-population. Independent functions, running asynchronously, process each sub-population. Also, sub-populations use the migration of data between

them, to help each other even if the algorithms or parameters of each sub-population are different. The objective is to prevent the algorithm from falling into optimal local values.

Distributed and cloud-based architectures are used extensively in the software industry because of their high performance and lower overall cost. Many systems are being created and migrating step by step to microservices and new serverless architectures, which proposes the use of "Function as a Service" (FaaS). Researchers are starting to exploit this type of architectures [References] in heuristic optimization development, and here we take advantage of them to increase the performance of some population-based algorithms.

1.1 Serverless

Recently, cloud providers such as Amazon Web Services (AWS), IBM Cloud, and Google Cloud, offer a new alternative to programming through interfaces called Serverless Computing [References], this platform consists in an effortless mechanism where developers upload the code into the platform and execute it as many times as it is required, scaling and allowing to do this in parallel. This way, the developers do not worry about servers, connections, and other configurations. In serverless, users pay only for what it is used. There is also the option to install some of these platforms locally; for instance, AWS (Amazon Web Services) [Ref] or Apache Open Whisk [Ref]. stall them into your own server to do your own local architecture serverless.

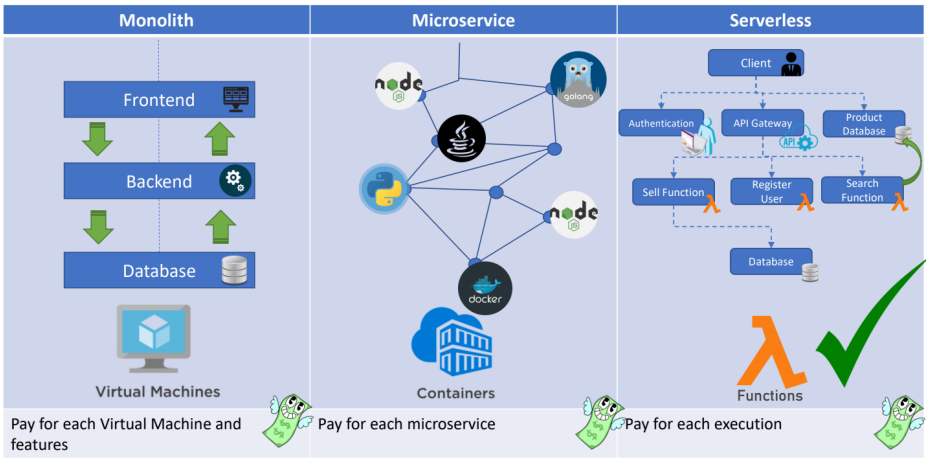

Fig. 1. Software architecture generations.

Serverless Function In math, a function is a relation between a set of inputs and an allowed set of outputs, with the idea that each input goes to a single

output. However, in computer science a function is defined as a unit of code that has a role into a greater code structure, works on varius inputs that usually are variables and throws a concrete output that are the result of process those variables. One of the main features that belongs to functions is that are stateless, they are focused on inputs and result with a simple process that does not require an state, in the moment the inputs get into the function are already generating an output. In serverless, events such as messages or HTTP requests can trigger these functions. Also, each function scales independently and is stateless with a short duration.

2 Proposed architecture

The propose is an architecture that allows to process a simple population that is divided into sub-populations distributing them to process in different ways and then communicating each other with purpose of increase the possibility to find the best fitness of a function. This architecture can accept the use of an indeterminate number of algorithms, allowing an easy hybridation and continuous adaptability for different problems.

This architecture consist in 3 nodes, they are explained on the next points:

- Manager: Here a population conformed by sub-populations is created and the architecture initialize parameters to run the algorithm. Every time a sub-population is created triggers an event that sends the subpopulations to be stored in a file JSON of MongoDB (preventing to saturate the memory) and to a message queue that is directed to its subsequent processing in the "Receiver" section that is our cluster of functions (Faas), because each subpopulation requires the execution of a different algorithm, there is a different channel in the web sockets for each type of algorithm that triggers its respective serverless function. Once a subpopulation is processed, it is returned and a selection is made for the sub-population migration. The migration selection is made by taking the population attribute of the subpopulation that was returned and the subpopulation that it have been selected among the best 2, it should be noted that the decision to identify the best from the 2 is made randomly. Once the selection is made, a Splitting Point Uniform crossing will be made. The 2 new subpopulations replace their self and are resent it back to their respective serverless function and this process is repeited until completing the number of assigned migrations for the multi-population. Of course this whole process is performed asynchronously avoiding wait for all responses from serverless functions to perform a crossover or a update of the multi-population status [1, 2]. In the following figure you can see from illustrative way how the multi-population is composed.
- Message Provider: Its purpose is the creation of a sub-population messages queue which is the comunication channel between the sub-population Manager and the Receiver (FaaS), each message is a trigger for the execution of a GA or PSO function. Thanks to the message queue, it is possible to perform

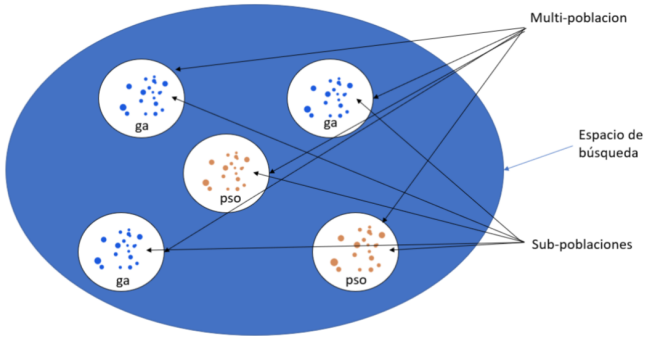

Fig. 2. Multipopulation representation.

the serverless functions asynchronously, avoiding that the algorithm wait for responses independently of their duration and the simultaneous evaluation of different sub-populations independent of its algorithm or characteristics.
– Receiver: The following section contains the Serverless functions of the algorithms to be executed, reduced the best possible using the functional programming paradigm so that they can be converted into FaaS without problems, in addition to achieving a completely clean and fast execution [3] . Each message received on this node is executed in the form of a multi-threaded process in parallel, this allows to having more than one population-based search algorithm running at the same time, and making a copy of itself each algorithm function as required.

To develop this architecture the applied technologies are based in JavaScript using Node JS as it can be see it in the General Architecture Flowchart.

2.1 Sub-population definition

Individuals are created composed of 2 types of information, the one that is active or useful for crossing and the one that is not. The population contains the series of possible solutions, while the rest contains the information on how the processing for the search of the optimal solutions will be executed, linked directly with their respective algorithms.

2.2 Splitting Point Uniform Migration

A uniform mask is created to apply the migration between individuals from 2 sub-populations. The selected data are combined using the middle point between the selected points by the mask. This process iterates the 2 sub-populations to randomly swap values and replace some of them with middle points.

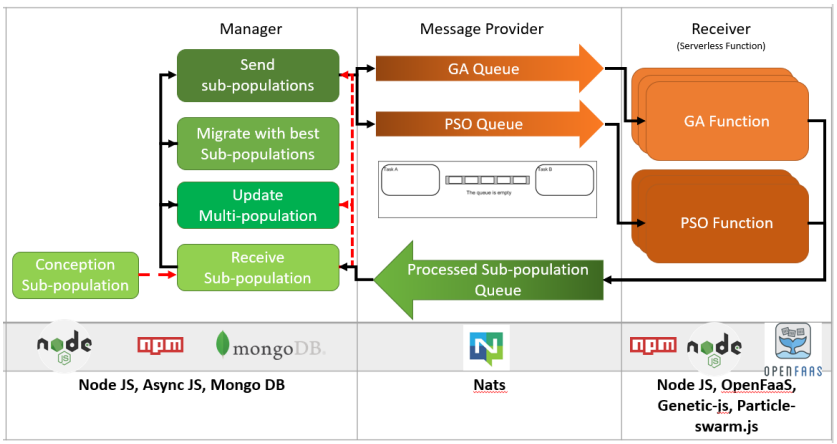

Fig. 3. General Architecture Flowchart.

Fig. 4. Sub-population composition.

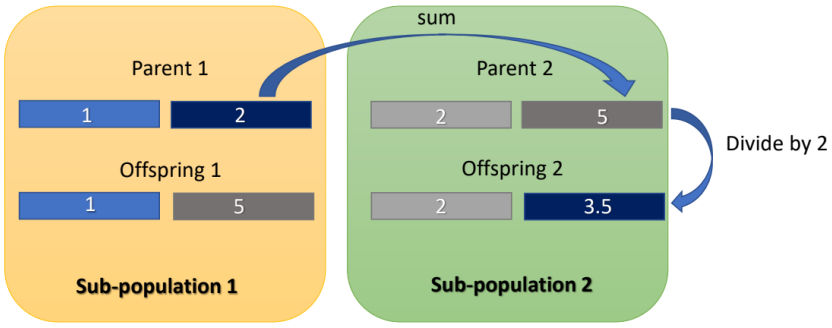

Fig. 5. Splitting Point Uniform process.

2.3 Migration Selection

Using migration selection by tournament keeping up the information of the best sub-population of the multi-population.

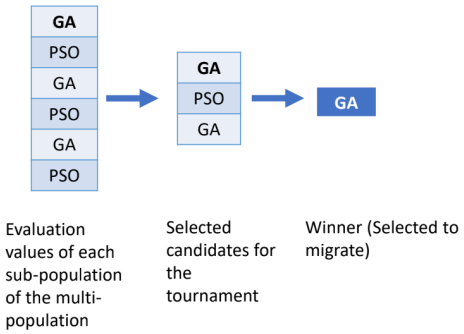

Fig. 6. Selection by tournament.

3 Experiments and Results

3.1 Experiments

Now that an interaction between sub-populations with different algorithms it is working and hybridation have been a success, using until now the added algorithms (GA and PSO) algorithms, all thanks to the developed architecture, lets procede to the experiments. This section is going to be the execution of several experiments from 2 to 40 dimensions, with a stop criterial of an error below 0.5E-8, without a parameter optimization method, waiting that the architecture by its self would be enough to increase the possibility to find a better optimal result than the traditional methods. All this hoping that the results will probe the needness of this kind of architecture on increasing dimensions. To test if the architecture was useful, several experiments were made to solve benchmark functions, for this case the functions are Sphere, Rastrigin and Rosenbrock. Using 10 sub-population for each experiment and maximum 4 migrations per sub-population with different algorithms and parameters for each sub-population.

3.2 Parameters Configuration

This architecture modifies the traditional way to work with population based algorithms, then the experiments could not be parameterized as usually are.

Then the experiments are scaled by their number of evaluations and the parameters must be configured to be adjusted to the next criterial, using the next expression:

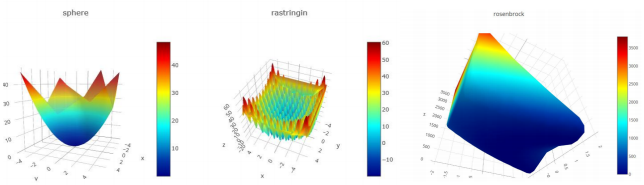

Fig. 7. Benchmark functions for experimentation.

$$Evaluations = 10^5 Dimensions \tag{1}$$

For example, if the experiment has 2 dimensions, the maximum number of evaluations will be 200,0000, for 10 dimensions will be 1,000,000 of evaluacions and the same with the others dimensions.

Table 1. 2 dimension parameters

Parameter	Value
GA Optimization	Minimize
GA Generations	50
GA Dimensions	2
GA Population size	100
GA Mutation	Random(Tournament2,Tournament3,Random ,RandomLinearRank,Sequential,Fittest)
GA Crossover	Tournament3
GA Crossover percentage	Random[10%, 80%]
GA Mutation percentage	Random[10%,50%]
GA Crossover function	Uniforme de punto medio
GA Mutation Function	gaussian
PSO Optimization	Minimiza
PSO Iterations	50
PSO Dimensions	2
PSO Vector size	100
PSO Social factor	Random[0.5,4.0]
PSO Individual factor	Random[0.5,4.0]
PSO Inercia factor	Random[0.5,4.0]

4 Conclusion

This architecture is completely scalable and useful for hybridation of multiple algorithms, until now is only GA and PSO but according with the results with this kind of architecture there is no limit and it works better than multi-populations

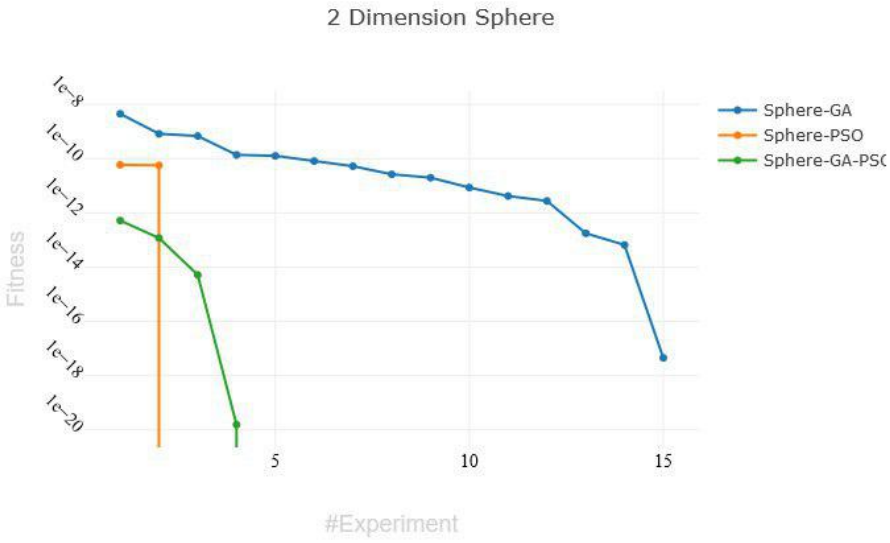

Fig. 8. 2 dimension experiments Sphere.

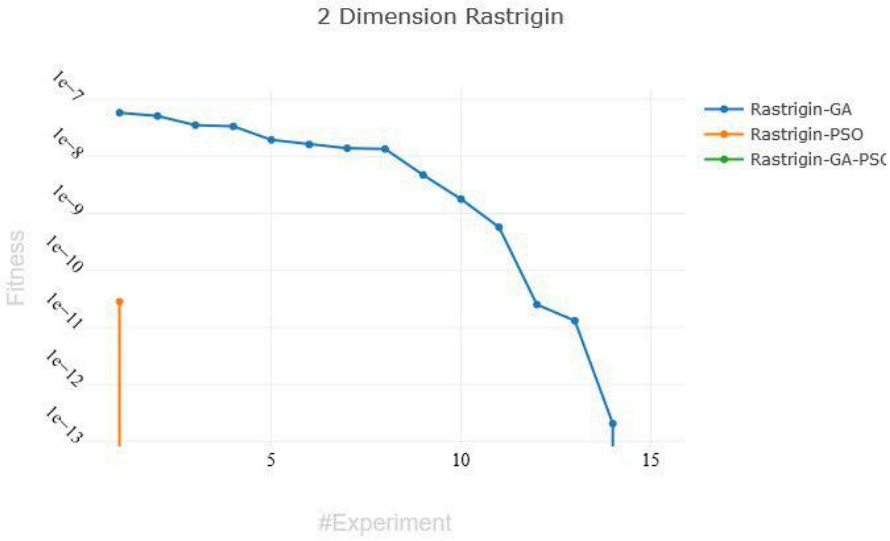

Fig. 9. 2 dimension experiments Rastrigin.

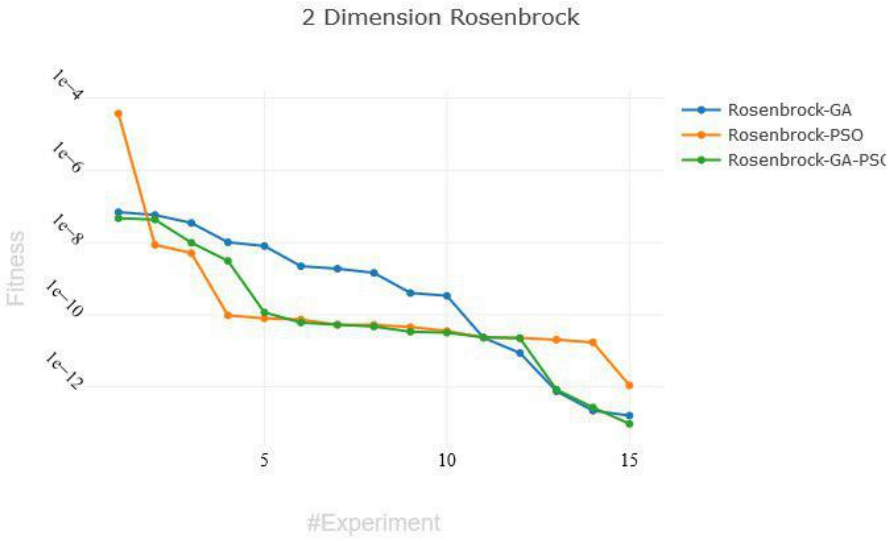

Fig. 10. 2 dimension experiments Rosenbrock.

Table 2. 2 dimensional experiment results

Fn	Best	AVG	Experiment Number
Rastrigin GA	0	1.65377E-08	15
Rastrigin PSO	0	1.8872E-12	15
Rastrigin GA-PSO	0	0	15
Sphere GA	4.53222E-18	4.36977E-10	15
Sphere PSO	0	7.8012E-12	15
Sphere GA-PSO	0	4.33161E-14	15
Rosenbrock GA	1.62335E-13	1.24176E-08	15
Rosenbrock PSO	1.11674E-12	2.47795E-06	15
Rosenbrock GA-PSO	9.5809E-14	6.90695E-09	15

Table 3. 10 dimensions parameters

Parameter	Value
GA Optimization	Minimiza
GA Generations	70
GA Dimensions	10
GA Population size	200
GA Mutation	Random(Tournament2,Tournament3,Random ,RandomLinearRank,Sequential,Fittest)
GA Crossover	
GA Crossover percentage	Random[10%, 80%]
GA Mutation percentage	Random[10%,50%]
GA Crossover function	Uniforme de punto medio
GA Mutation Function	gaussian
PSO Optimization	Minimiza
PSO Iterations	70
PSO Dimensions	10
PSO Vector size	200
PSO Social factor	Random[0.5,4.0]
PSO Individual factor	Random[0.5,4.0]
PSO Inercia factor	Random[0.5,4.0]

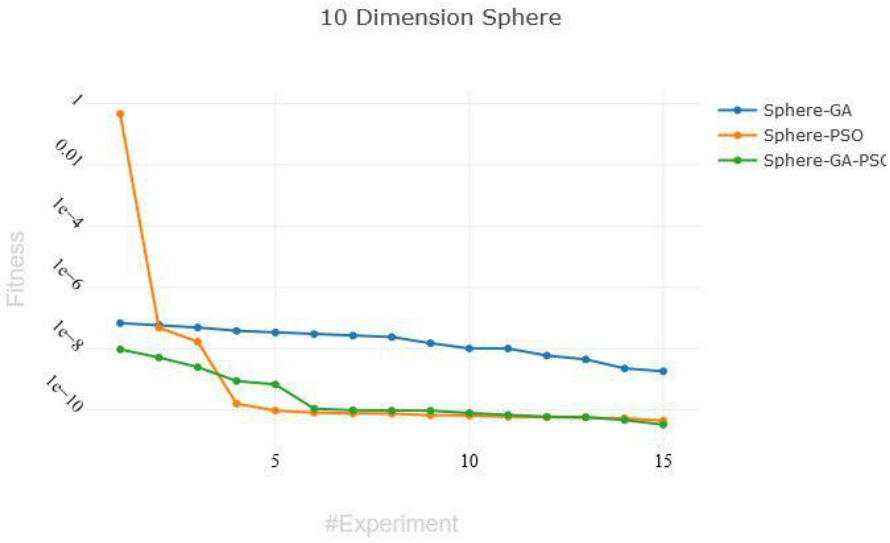

Fig. 11. 10 dimensions experiments Sphere.

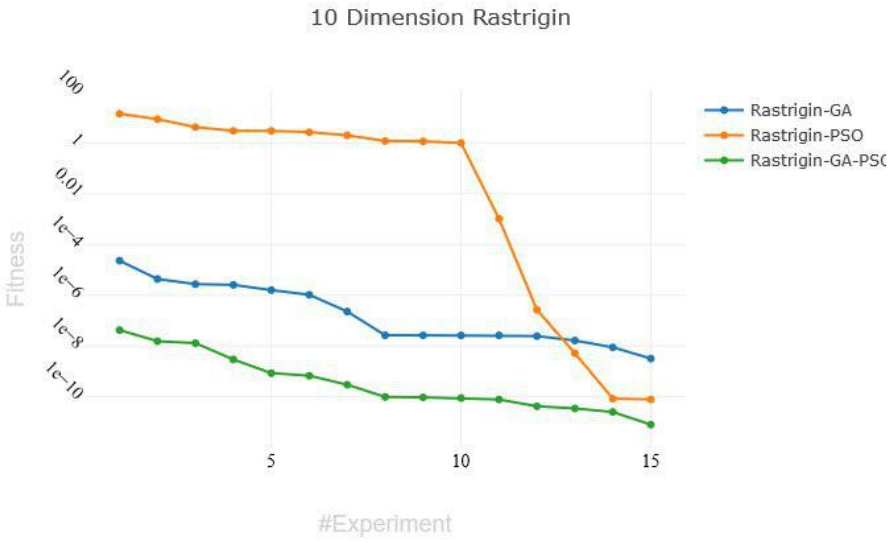

Fig. 12. 10 dimensions experiments Rastrigin.

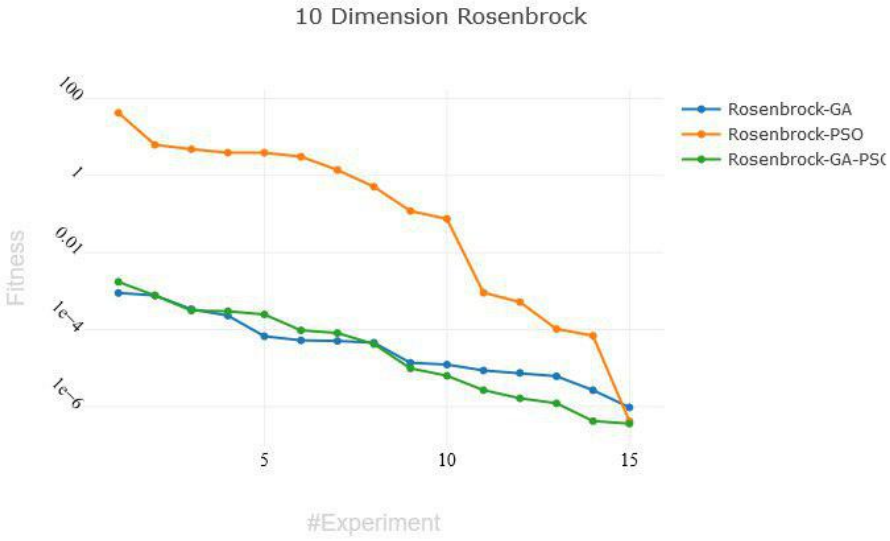

Fig. 13. 10 dimensions experiments Rosenbrock.

Table 4. 10 dimensional experiment results

Fn	Best	Average	Experiment Number
Rastrigin GA	3.21768E-09	2.38015E-06	15
Rastrigin PSO	7.8586E-11	2.715716161	15
Rastrigin GA-PSO	8.01492E-12	5.08668E-09	15
Sphere GA	1.84051E-09	2.5389E-08	15
Sphere PSO	4.50351E-11	4.72855E-09	15
Sphere GA-PSO	3.33851E-11	1.30062E-09	15
Rosenbrock GA	9.58323E-07	1.24176E-08	15
Rosenbrock PSO	4.16711E-07	4.431565444	15
Rosenbrock GA-PSO	3.62472E-07	0.000240251	15

Table 5. Parametros experimentos 20 dimensiones

Parameter	Value
GA Optimization	Minimiza
GA Generations	70
GA Dimensions	20
GA Population size	200
GA Mutation	Random(Tournament2,Tournament3,Random ,RandomLinearRank,Sequential,Fittest)
GA Crossover	Tournament3
GA Crossover percentage	Random[10%, 80%]
GA Mutation percentage	Random[10%,50%]
GA Crossover function	Uniforme de punto medio
GA Mutation Function	gaussian
PSO Optimization	Minimiza
PSO Iterations	70
PSO Dimensions	20
PSO Vector size	200
PSO Social factor	Random[0.5,4.0]
PSO Individual factor	Random[0.5,4.0]
PSO Inercia factor	Random[0.5,4.0]

Table 6. Resultados 20 dimensiones

Fn	Best	Average	Experiment Number
Rastrigin GA	0.000808633	0.220596203	15
Rastrigin PSO	3.988070734	25.51777514	15
Rastrigin GA-PSO	9.13E-09	7.38E-02	15
Sphere GA	1.84051E-09	9.22715E-06	15
Sphere PSO	7.04E-11	3.50E-07	15
Sphere GA-PSO	9.11E-11	2.13E-08	15
Rosenbrock GA	0.000348015	0.010958941	15
Rosenbrock PSO	9.119539342	13.37613983	15
Rosenbrock GA-PSO	2.31663E-05	0.005608855	15

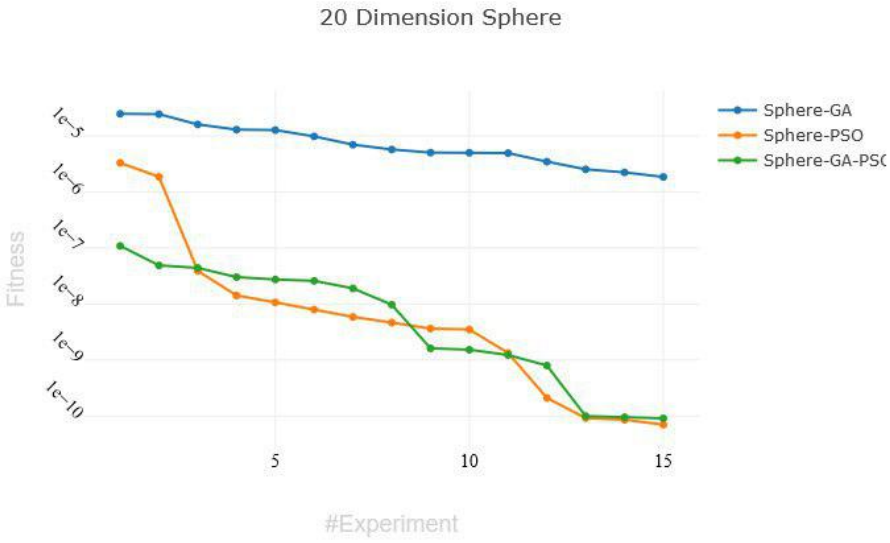

Fig. 14. 20 dimensions experiments Sphere.

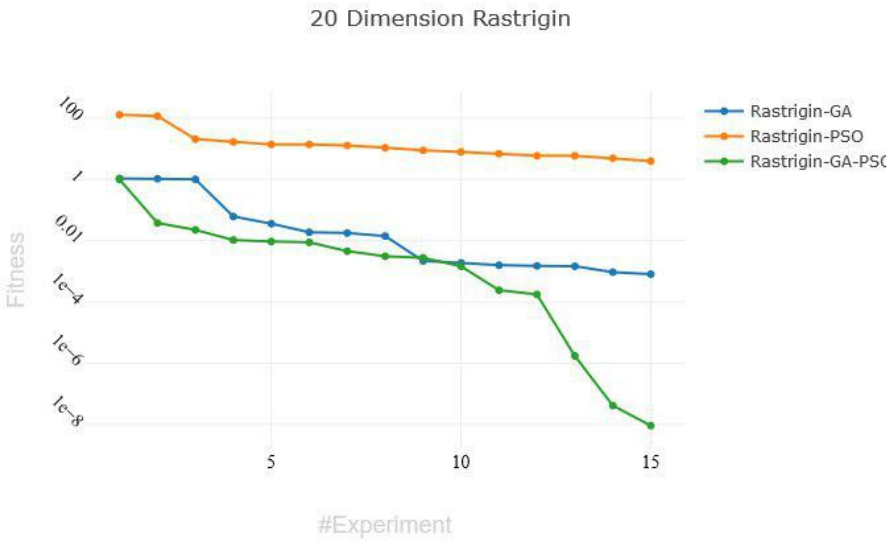

Fig. 15. 20 dimensions experiments Rastrigin.

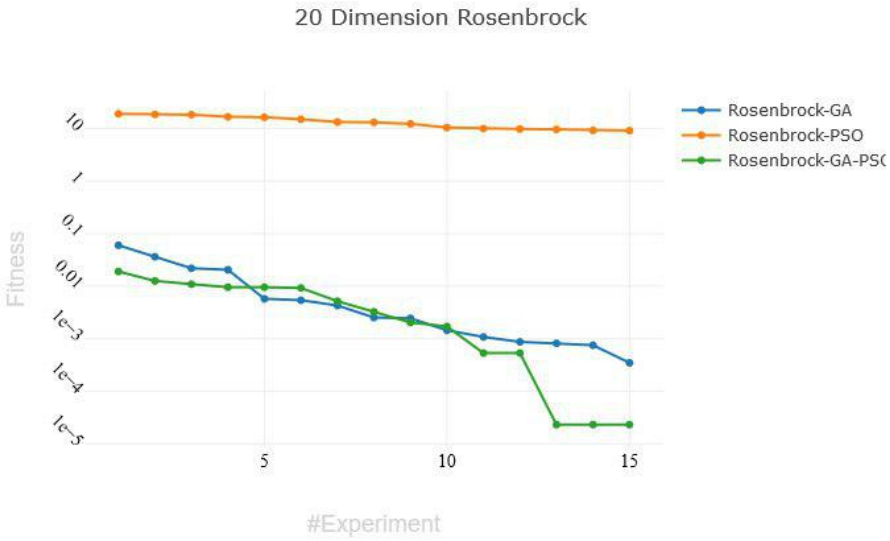

Fig. 16. 20 dimensions experiments Rosenbrock.

Table 7. Parametros experimentos 40 dimensiones

Parameter	Value
GA Optimization	Minimiza
GA Generations	70
GA Dimensions	40
GA Population size	200
GA Mutation	Random(Tournament2,Tournament3,Random ,RandomLinearRank,Sequential,Fittest)
GA Crossover	
GA Crossover percentage	Random[10%, 80%]
GA Mutation percentage	Random[10%,50%]
GA Crossover function	Uniforme de punto medio
GA Mutation Function	gaussian
PSO Optimization	Minimiza
PSO Iterations	70
PSO Dimensions	40
PSO Vector size	200
PSO Social factor	Random[0.5,4.0]
PSO Individual factor	Random[0.5,4.0]
PSO Inercia factor	Random[0.5,4.0]

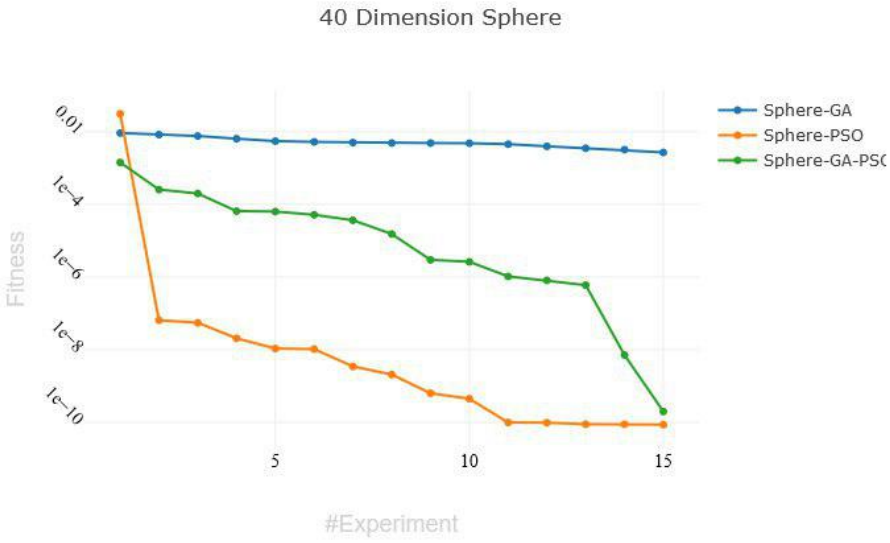

Fig. 17. 40 dimensions experiments Sphere.

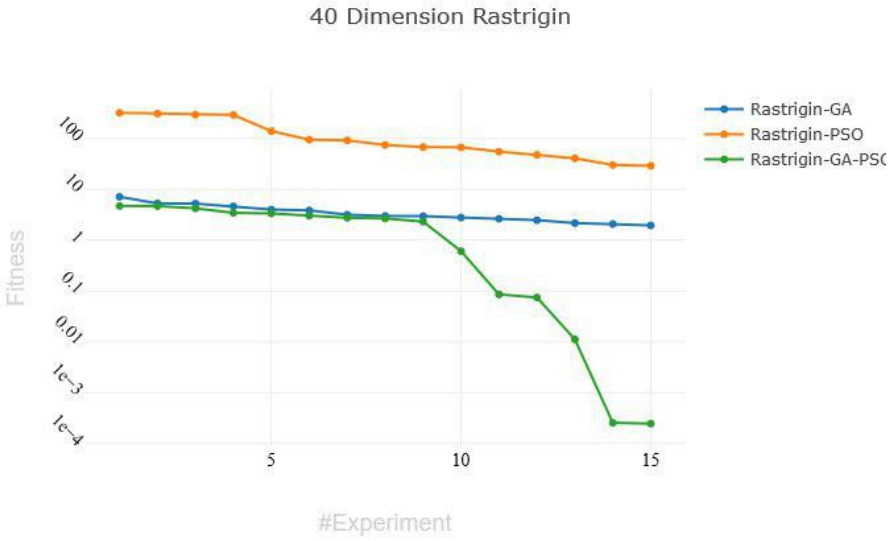

Fig. 18. 40 dimensions experiments Rastrigin.

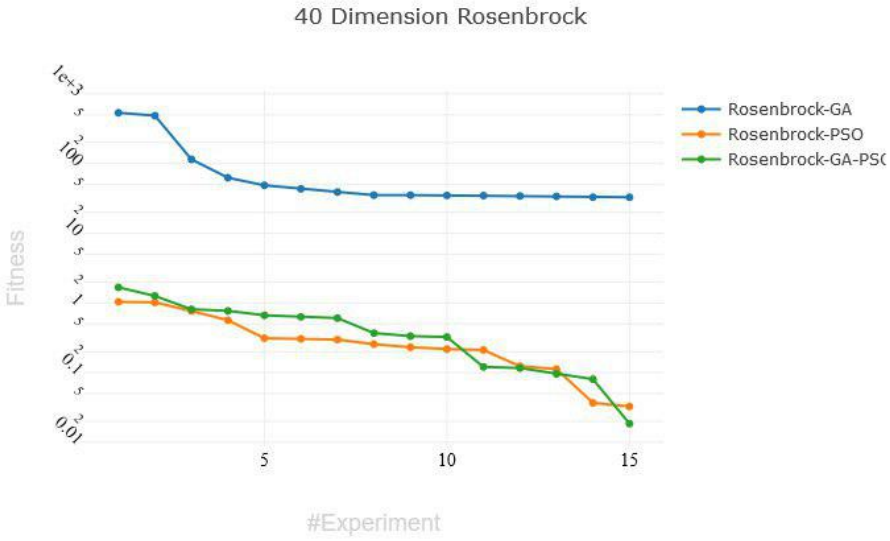

Fig. 19. 40 dimensions experiments Rosenbrock.

Table 8. Resultados 40 dimensiones

Fn	Best	Average	Experiment Number
Rastrigin GA	1.95478879	3.560837088	15
Rastrigin PSO	29.06596132	130.2865863	15
Rastrigin GA-PSO	2.46E-04	2.13E+00	15
Sphere GA	0.002686956	0.005302951	15
Sphere PSO	8.68E-11	2.07E-03	15
Sphere GA-PSO	2.00E-10	1.41E-04	15
Rosenbrock GA	0.000348015	106.9287542	15
Rosenbrock PSO	0.032708559	0.368395353	15
Rosenbrock GA-PSO	0.018538924	0.525086565	15

with only one algorithm, also every experiment executes in short times because serverless functions searching in an asynchronous way getting a fast convergence.

5 Future work

To get a continuous improvement it is believed that it is required a sort of mutation applied to the sub-populations. This mutation would be a swapping type, taking the algoritm parameters from the best and the worst sub-populations, increasing the possibilities to get and optimal result, preventing get stucked into a local optimum. Of course it is expected to use this architecture using more algorithms than GA and PSO.

References

1. M. Løvbjerg and T. K. Rasmussen, "Hybrid Particle Swarm Optimiser with Breeding and Subpopulations," *Proc. 3rd Genetic Evolutionary Computation Conf.*, pp. 469 –476, 2001.
2. H. M. A. Jimeno, M. J. L. Sánchez, and R. H. Rico, "Multipopulation - based multi - level parallel enhanced Jaya algorithms," *The Journal of Supercomputing*, no. 0123456789, 2019.
3. M. Roberts, "Serverless Architectures," 2016.