Event-driven multi-population optimization: mixing Swarm and Evolutionary strategies

First Author[1[0000−1111−2222−3333]], Second Author[2,3[1111−2222−3333−4444]], and Third Author[3[2222−−3333−4444−5555]]

[1] Princeton University, Princeton NJ 08544, USA
[2] Springer Heidelberg, Tiergartenstr. 17, 69121 Heidelberg, Germany
lncs@springer.com
http://www.springer.com/gp/computer-science/lncs
[3] ABC Institute, Rupert-Karls-University Heidelberg, Heidelberg, Germany
{abc,lncs}@uni-heidelberg.de

Abstract. Recently, in the field of nature-inspired optimization, researchers have proposed multi-population asynchronous algorithms that distribute the evolutionary process among heterogeneous search paradigms.These algorithms execute the search strategy by reading streams of populations from message queues, and replacing them with evolved versions of them. Moreover, current studies suggest that when we have a high number of populations interacting in parallel, the effect of the individual parameters of each population is compensated by those selected in other populations, improving the overall performance of the algorithm. In this work, we propose a simple reactive migration method for the asynchronous execution of multi-population, multi-strategy algorithms that improves over homogeneous configurations. We evaluate this method by comparing between homogeneous and an ensemble of multi-populations, using Genetic Algorithms (GAs) and Particle Swarm Optimization (PSO) in the noiseless BBOB testbed for the optimization of continuous functions. Results show, that this method offers better performance, even when compared with other asynchronous population based algorithms.

Keywords: heterogeneous multi-population algorithms · genetic algorithms · cloud-native systems.

Nature-inspired optimization algorithms have been used successfully in the last decades to tackle complex problems [48]. These algorithms include evolutionary algorithms (EAs) [2] and swarm intelligence (SI) [29]. Genetic Algorithms (GAs) [26,8], Differential Evolution (DE) [28], and Genetic Programming (GP) [2] are popular EAs, while examples of SI algorithms [29] are particle swarm optimization (PSO) [4], artificial bee colony (ABC) [27], Grey Wolf Optimization (GWO) [35], and ant colony algorithms (ACO) [7]. A common characteristic of these kind of methods is the use of an initial set of random candidate solutions that are manipulated by the algorithm to generate a new set of candidates, and because of this, we commonly referred to them as population-based algorithms. There are other stochastic optimization methods, that even if they are

not nature-inspired, they employ the concept of an evolving population. An example of this kind of methods is population-based incremental learning (PBIL) [3], which is an optimization and an estimation of distribution algorithm. Since instead of piecewise constructing a population or changing it incrementally, all members of the population have to be evaluated to obtain a fitness or score that will be used to select them (or not), a drawback of this kind of algorithms is that they can be computationally expensive since it is difficult to overcome that n factor, where n is the population size, in the complexity of every iteration of the algorithm. That is why researchers have been proposing some form of parallelization since early on [36] with the goal of decreasing their execution time. One of the first methods of parallelization was the island model, which lead to an increased performance [17,19]. The concept was to divide the population into smaller populations that communicated with each other. Other population-based algorithms have adopted the technique, and since then, researchers have found additional advantages besides a reduced execution speed, these include avoiding a premature convergence and maintaining the diversity of the global population [30], we are going to call these methods multi-population algorithms [34]. The relative isolation in which populations carry out the algorithm, together with the synchronous or asynchronous communication, helps to increase the overall diversity since each population will search in a particular area, at least between communications. Multi-population algorithms use a form of communication to recombine or migrate candidate solutions between populations to avoid premature convergence, since smaller populations are known to perform better for a given problem than bigger populations [32,47]. Even in some cases, a multi-population based algorithm scales better due to these interactions, and the parallelism of the operation [1].

Having many populations offers researchers many configuration options and additional challenges when designing efficient multi-population algorithms. Options include the number and size of populations, the interaction between them, the search area of each population, and the search strategy and its parametrization for each population. In this paper, we are interested in the latter, that is, having multiple populations running with distinct parameters and optimization algorithms. We can find in the literature, several heterogeneous multi-population algorithms that integrate variations of optimization algorithms, and these often perform better than single-population or homogeneous multi-population optimization algorithms[47,38]. Heterogeneous algorithms add to the problem of finding the correct parameter settings for each population; because some parameters affect the accuracy of the solution and the convergence speed of the individual algorithms as they tip the balance between exploration and exploitation of the search space. On the other hand, current studies show that by having a high number of populations communicating in parallel, the effect of the individual parameters of each population is compensated by those selected in other populations [32,46]. Some level of heterogeneity can be implemented by just changing the configuration parameters of each population, but in this case, we are interested in heterogeneous search strategies.

Therefore, in this paper, we implemented an asynchronous multi-population algorithm version, using a message queue for inter-process communication [20], and a reactive migration procedure, in which we compare three heterogeneous configurations using a randomized parameter technique. We experimented with all populations using a GA or PSO search strategies, versus an ensemble multi-population with both GA and PSO algorithms, using as a benchmark, the separable functions of the BBOB noiseless testbed [25]. We compare the options by measuring the average running time (aRT) as the number of functions (#FEs), as the objective is to prove that the advantage of heterogeneous configurations resides not only in the increased scalability but also in the search performance.

The organization of the paper is as follows: First, Section 1 presents state of the art relevant to our work. In Section 2, we present the proposed method. Section 3 describes the design of the empirical evaluation we designed to assess the effectiveness of the method, and in Section 4, we report and discuss the results. Finally, we offer the conclusions of this paper and suggestions on future work in Section 5.

1 State of the Art

In general, population based algorithms have to keep a balance between exploration and exploitation via the clever use of selection and variation operators [5]. In pursuit of that objective, it is important to keep diversity high [49], but different algorithms have different mechanisms for increasing diversity (exploration) or decreasing it (exploitation). PSO has two constants that rule in which direction the *particle* is going to move: either randomly (exploration) or in the direction of the best particle (exploitation); evolutionary algorithms use mutation and, to a certain point, crossover for exploration and crossover and selection procedures for exploitation. Since these mechanisms are fundamentally different, using several different algorithms at the same time might be considered a win-win situation by way of performing exploitation in several different directions at the same time, which will be able to get closer to the solution as well as generate new possibilities.

This is why hybrid population-based algorithms have been repeatedly suggested in the literature: Pandi et al. [39] suggest a combination of swarm intelligence and another algorithm denominated harmony search; Lien et al. [33] combine particle swarm optimization with bee colony algorithms, while Zhao et al. [50] combine the latter with evolutionary algorithms.

EAs and PSOs share another characteristic, besides the fact that they act on populations: they can use the same data structures to represent every member of the population. Please check Table 1 for equivalence of terms between PSO and EAs; additionally to the data structures used, there is a certain equivalence in the operators, but they are different enough to perform search in different ways. The fact that they use the same data structures means that you can apply them in turns, or simply start calling a “chromosome” a “particle” or the other way round, without any conversion, and keep working with it.

Table 1. Equivalence of terms between particle swarm optimization and evolutionary algorithms.

PSO	EA
Position vector	*genes* in a chromosome
Move towards better	Selection
Random motion	Mutation
Move towards centroid	Crossover

This is what the first papers to propose such a hybrid algorithm, by Robinson et al. [43], explicitly played on this fact and also on what we might call diversity of local minima by running an algorithm on a population until it became stagnated, and then switching to the other one. This was done apparently only once, but eventually the mixture of the two algorithms was able to obtain better results in the design of a kind of antenna called "horn" than each one of them separately, although they report that the best value was obtained by the algorithm that started as a PSO and terminated as an evolutionary one.

Another kind of hybrid was proposed independently by Shi et al. [44], citing the "local optimum" problem, that is, the same as the previous one; it mixes both algorithms testing in different configurations: "parallel" and "serial" testing which configuration works better and is able to avoid that. Their results on benchmark functions are mixed; In general, however, a well-designed evolutionary or PSO algorithm will not fall in that local optimum; it is certainly true that, within a certain evaluation budget, neither might be able to find that global optimum. However, it is probably that they mean that, since exploitation of better-than-average solutions is done by the two algorithms in different directions, a hybrid algorithm might help the single-algorithm version to escape that.

This better exploitation capability was used by Grimaldi et al. [18], with the objective of solving electromagnetic problems. What they do is to split the population into two different populations which will be processed by an EA and a PSO algorithm; the new individuals generated are merged in a single population, which is then split all over again in the next iteration. This guarantees that none of the two algorithms is getting stuck, since the individuals will be randomly subjected to one or the other on every generation.

Later on, Esmin et al. [11] leveraged to design their algorithms; this idea was later extended by Li et al. [31]; in this work, chromosomes/particles represent Support Vector Machines, which are optimized to find the most representative features of gene expression data sets. PSO and EAs are applied serially: 10 iterations of each, after which the finalization criterion is examined; first PSO, then EA. Separate EA and PSO are tested against this hybrid algorithm, finding improvements of a few percent points in the accuracy over classification of the whole dataset. This marginal, but significant, improvement is essentially the kind of results that should be expected. While improvements in diversity always boost the results by allowing the algorithms to find better solutions, an

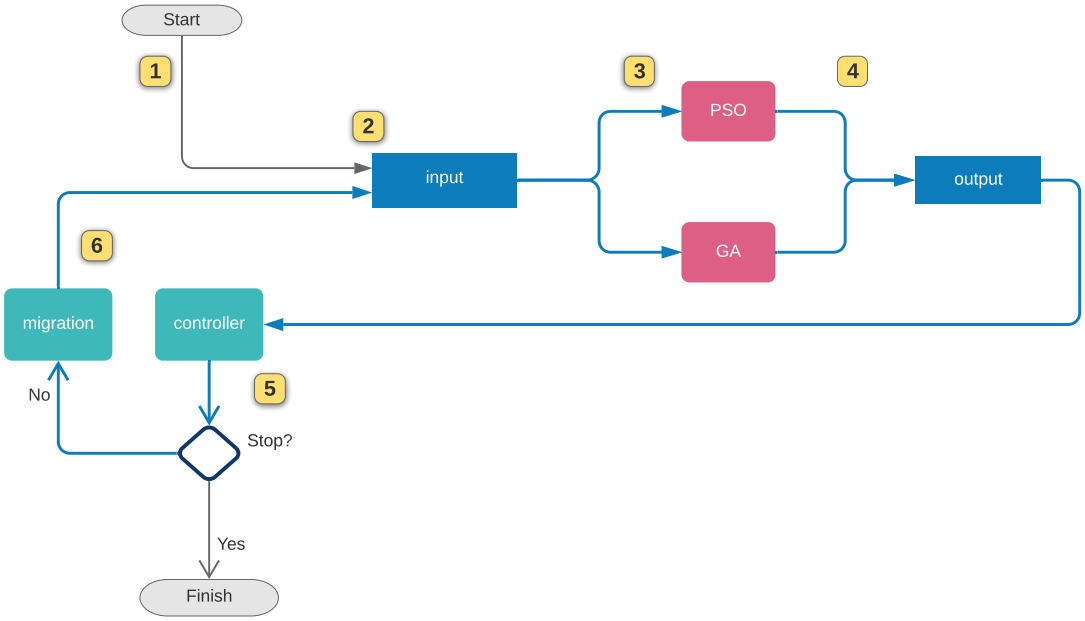

Fig. 1. General scheme of the architecture of this method. Numbers will be used to refer to its different elements in text.

order-of-magnitude difference is not usually achieved, since both algorithm, by themselves, have good mechanisms for global exploration.

Several other hybrid algorithms have been proposed more recently: Gulia et al. [22] mix ant colony optimization, which is a swarm intelligence algorithm, and EAs to select software testing cases. ACO and EA are used serially, with GA acting to refine the test suite initially selected by the ant algorithm.

The state of the art is, then, use swarm intelligence and evolutionary algorithms coupled to a population to which they are applied either one after the other, or splitting the population so that every one is applied to a part. In our case, however, population and algorithms are decoupled, and this decoupled architecture allows to mix algorithms in several different ways, which will be tested in this paper using the method explained next.

2 Proposed Method

In this section, we present a model for the execution of multi-population based asynchronous algorithms. As a first step, we describe the general architecture, for handling a stream of continuously updated populations, and then our approach for decoupling the processing in stateless functions, to finally give details of the migration mechanism.

2.1 Architecture

In this section we present our proposed model for the execution of multi-population based asynchronous algorithms. As we have mentioned before, when designing efficient multi-population algorithms, we need to consider additional issues [34], including the number and size of populations, how they interact or communicate between them, and the search strategy and parametrization of populations. We have considered these requirements and, in this paper, propose a cloud-native multi-population solution. In this model, populations are the primary data structure, and we package them as messages that are part of a continuous stream flowing from one computing node, performing an algorithm, to the next. To achieve this "continuous stream," everything must happen asynchronously without computing nodes needing to wait for others, and becoming available as soon as it is needed.

We have implemented this streaming functionality by using a message queue system, whose general architecture is shown in Figure 1. We can explain the model by using an analogy of producers and consumers of messages. There are two queues, one labeled `input`, and the other one `output`. In a queue, *push* operations are represented by solid arrows connecting to the left side and pull or pop operations as solid arrows leaving from the right side. The `controller` process, is responsible for the migration of individuals from one population to the other, and keeping track of the iterations of the algorithm. There is at least one `stateless function` worker process responsible for running the isolated algorithms. The two processes `PSO` and `GA`, are shown.

We can follow the path of messages as follows:

1. In this step, the specified number of populations are created according to the configuration. Populations at this moment are just static data messages, including each individual inside. Each population includes a metadata section where its algorithm and execution parameters are specified. For instance, for a GA, the mutation rate, type of crossover operator, and other values are indicated.
2. The setup process pushes each population on to the `input` queue so that they can be consumed by stateless functions responsible for the execution of the search.
3. One or more `stateless functions` are constantly pulling population messages from the `input` queue. These functions have the task of executing the optimization algorithm. They take the current state of the population and start to run the specified algorithm for a certain number of iterations.
4. Once they finish the execution, the resulting populations are pushed to the `output` queue; when the computing node is done with a population, it draws another one from the queue.
5. The `controller` pulls populations from the `output` queue, inspects the metadata describing the current state of the populations, and if the algorithm has found an optimal solution or the maximum number of iteration has been reached it stops the execution.
6. Otherwise, it passes the stream of messages to a migration function, where populations are mixed. Migration generates new populations, and they are again pushed to the `input` queue to continue in a loop. The `controller` is also responsible for logging the metadata.

This reactive architecture has the following advantages:

- It decouples the population and the population-based algorithm. In this design, we can have one or many processing nodes, each running a different search strategy.
- Also, the reactive controller gives designers more control over the multi-population algorithm. In this process, designers can dynamically change the number of populations, population parameters, and migration details on-the-fly.
- Another advantage is that algorithm designers have many options for implementing this simple architecture. The same basic components can be implemented as a single multi-threaded program, or as a highly scalable serverless cloud application.

2.2 Stateless functions

This message queue pattern is an essential component of a highly scalable reactive architecture; one of the reasons for this scalability is the use of stateless functions that have no secondary effects, or the need to read data from an external entity. Stateless functions do not need to read, keep, or modify data outside of the scope of the method, it will have no side effects reading inputs

and producing an output, mapping input to output as it were. If we implement population-based optimization algorithms using stateless functions, then to the system, there is no difference between having one or many copies of the same function pulling work from the queue all at the same time, because there are no unwanted side effects, including problems of resource locking resulting from this concurrency. This architecture has been implemented successfully for developing cloud-native multi-population algorithms, using serverless functions [15], and concurrent programming [21].

To have a stateless version of a population-based algorithm, we only need to skip the step of creating a random population. Instead, the function receives the population as a parameter along with the required parameters. After several iterations (specified in the parameters), the function returns the current state of the population, with additional data describing the local execution. This data is necessary to log the #FE and current fitness values.

2.3 Migration

As the controller is pulling population messages from the output queue, it waits until the message queue contains three valid populations to trigger an event handled by the `population_mixer` function, which uses as an argument a list containing the three populations. This design has the advantage of not needing to keep a buffer in memory or external storage, and it follows the reactive paradigm. A possible disadvantage can be that it only mixes populations that have arrived sequentially to the message queue, but we could mitigate this with a larger buffer and in fact populations do not need to arrive to the buffer in the same order they were created or processed, so this shouldn't, in principle, contribute to any loss of diversity.

When `population_mixer` unit receives a list of three populations, let us say [A, B, C], it calls the migration method shown in Algorithm 1 for [A, B], [B, C] and [A, C]. The migration algorithm sorts the individuals of each population and generates a new one by merging the best half from each. Finally, the `population_mixer` method pushes the three generated populations back to the `input` message queue.

Algorithm 1 Migration

procedure CXBESTFROMEACH(pop_1, pop_2)
  $pop_1.sort()$
  $pop_2.sort()$
  $size \leftarrow min(len(pop_1), len(pop_2))$
  $cxpoint \leftarrow (size - 1)/2$
  $pop_1[cxpoint:] \leftarrow pop_2[:cxpoint + 2]$
  return pop_1
end procedure

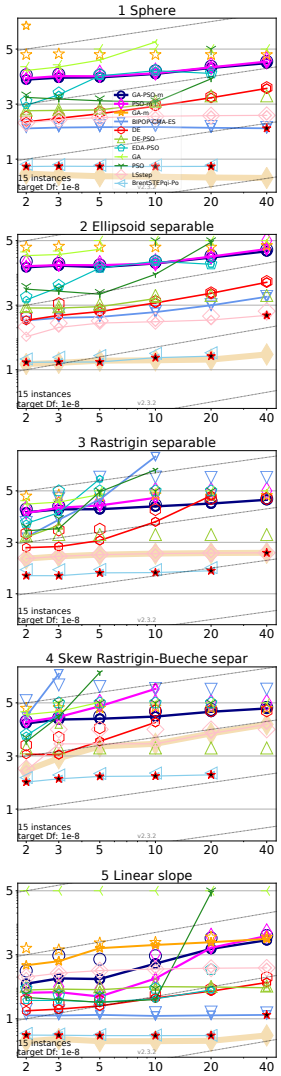

Fig. 2. Average running time (in #FEs as log_{10} value), divided by dimension for target function value 10^{-8} vs dimension. Black stars indicate a statistically better result compared to all other algorithms ($p < 0.01$) and Bonferroni correction number of dimensions. Legend: ○:GA-PSO-m, ◇:PSO-m, ⋆:GA-m, ▽:BIPOP-CMA-ES, ⬡:DE, △:DE-PSO, ⬠:EDA-PSO, ⋋:GA, ⋌:PSO, ◇:LSstep, ◁:BrentSTEP

3 Experimental setup

In this section, we present the experimental setup needed to verify if having a multi-population algorithm, with heterogeneous populations and the support for mixing search strategies, increases the performance of the search by needing fewer function evaluations than a homogeneous setting.

benchmark separable functions ($f_1 to f_5$) from the Continuous Noiseless BBOB testbed, which is part of the Comparing Continuous Optimizers (COCO) framework [24].

Although there are not many real-world problems that are entirely separable, we wanted to focus on separable functions as a primary benchmark for testing the performance of a hybrid multi-population algorithm. And there is interest in solving such problems using more general methods [6,45].

These functions are real-parameter, single-objective, and are usually employed as benchmarks. We have tested with fifteen instances of each function, each one having a different optimal value. The standard benchmark of the testbed 40 dimensions. With the maximum number of function evaluations (#FEs) increasing with the dimension (D), using the expression $10^5 \cdot D$ (i.e. for $D = 2$, #FEs is $200,000$).

The COCO framework offers several tools to compare the performance of algorithms, generating data sets, tables, and reports for an experiment, we have compared using the average required time (aRT). There is a repository[4] of more than 200 results for the noiseless BBOB testbed, collected from BBOB workshops and special sessions between the years 2009 and 2019. To test the heterogeneous multi-population capabilities, we have compared the aRT between a homogeneous and an ensemble of multi-populations, using Genetic Algorithms (GAs) and Particle Swarm Optimization (PSO).

We have used the DEAP framework for the GA stateless function [13] and the EvoloPy library [12] for the PSO algorithm function. We show the parameters for each algorithm in Tables 2 and Table 3 for the GA and PSO, respectively. We have obtained these parameters by following a method used by García et al. in [16]. First, we tested the parameters for the Rastrigin separable function with five dimensions. After about fifteen experiments, the most challenging targets were achieved for this particular function. We tested again with functions one to three, and after obtaining favorable results, we set the PSO and GA algorithm parameters. We have set the mutation and crossover probabilities of the GA randomly to have heterogeneous populations; the entry in the Table shows the range of these parameters. We have not changed these settings during the experiments, and we have only provided the population size and number of generations as parameters.

To run the experiments, we have deployed the Docker application with 8 worker containers hosting the stateless version of the GA and PSO algorithms described earlier, and a total of 10 populations. Table 4 shows the parameters we have used. The *GA-PSO Ratio* parameter specifies the proportion of populations

[4] `https://coco.gforge.inria.fr/doku.php?id=algorithms-bbob`

Table 2. DEAP GA EvoWorker Parameters

Selection	Tournament size=12
Mutation	Gaussian $\mu = 0.0$, $\sigma = 0.5$, indbp=0.05
Mutation Probability	[0.1,0.3]
Crossover	Two Point
Crossover Probability	[0.2,0.6]

Table 3. EvoloPy PSO Parameters

V_{max}	6
W_{max}	0.9
W_{min}	0.2
C_1	2
C_2	2

that will be using the GA algorithm, with 0 indicating all populations will run a PSO algorithm, and with 0.5 the same proportion of PSO and GA populations, and GA only it is specified with 1. We have specified that the experiment will use only the first five functions. We have used 15 instances, because this is the standard for the BBOB benchmark [24]. The list of dimensions that we have tested is in the *Dimensions* parameter. Then, we have defined for each dimension: the number of populations, and for each population, the number of generations and population size. Finally, we have defined how many complete loops the algorithm will perform. The product of these parameters gives us a maximum #FEs. For example, for $D = 2$, the maximum number evaluations is $200,000$, this is the same as the product of the parameters, that is $40 * 50 * 10 * 10$.

Table 4. Parameters used in the experiments, for ten populations and eight workers.

Dimension	2	3	5	10	20	40
Generations	40	25	28	50	66	80
Population Size	50	60	60	70	100	125
Populations	10	10	10	10	10	10
Iterations	10	20	30	30	30	40

We have deployed the container-based application in a Desktop PC with AMD Ryzen 9 3900x 12-core processor with 24 threads and 16 GB RAM. We used Docker version 19.03.3, build a872fc2f86, and `docker-compose` version 1.21.0, in Ubuntu Linux 18.04, and Python 3.7.5 code. Container images and Docker compose file are available at (https://hub.docker.com/x/x), and (https://github.com/x/). The experiments were performed with COCO [24] version bbob.v15.03 in python, the plots were produced with version 2.3.2.

4 Results

The results obtained by the experiments show (see Figure 3) how the runtime scales with dimension to reach certain target values Δf; The color of the lines indicates the average runtime, for each target reached at least one time. A red circle without a number on top indicates the algorithm has reached the 10^{-8} target on all instances for that dimension. If there is a number on the top, it indicates how many times the algorithm has reached the target. Fixed values of targets $\Delta f = 10^k$ with k colors are given in the legend of f_1, k values are $[-8, -5, -3, -2, -1, 0, 1]$. Results from each configuration are presented in each column, GA, PSO, and GA&PSO populations.

We can see that the GA algorithm (on the left column) has a hard time reaching even the 10^{-5} target in functions ($f_1 - f_4$) for all dimensions, and has the worst performance in 20 and 40 dimensions. Finally, it performs better on f_5 reaching the 10^{-8} on all dimensions but with a higher runtime on lower dimensions.

On the other hand, the multi-population PSO (taking the column on the middle) has better performance; its results take a smaller range, reaching all targets for functions f_1 and f_2 but with a slightly higher runtime than the multi-strategy configuration. It is not able to reach the 10^{-8} target on the higher dimensions of f_3 and f_4. Moreover, the PSO multi-population has the best runtime for the lower dimensions of f_5.

Finally, the proposed PSO&GA mixed algorithm has reached the 10^{-8} target value on all functions ($f_1 - f_5$) and scales well to higher dimensions, even on f_3 and f_4 (charts on the lower rows), which are usually considered difficult functions.

These results are also competitive when compared against other GA and PSO implementations of past BBOB workshops. Figure 2 shows how the runtime scales with dimension to reach the most difficult target for $\Delta f = 10^{-8}$ for each function. The representative algorithms are: The simple binary GA algorithm of Nicolau [37], the PSO algorithm by El-Abd and Kamel [10], an EDA and PSO hybrid [9], Differential Evolution (DE) with adaptive encoding [42] by Pošík and Klemš, the hybrid DE-PSO by García-Nieto et al. [14], and the BIPOP-CMA-ES algorithm [23] by Hansen, this last algorithm has the best overall ($f_1 - f_2 4$) performance on the BOBB-2009 benchmark, as it could solve 23, 22 and 20 functions out of 24 in dimensions 10, 20 and 40, respectively.

We also compare with two methods that are not population-based, but univariate solvers generalized for the optimization of separable functions, the line-search algorithm LSStep by Pošík [40], and the Hybrid Brent-STEP by Pošík and Baudiš [41]. The performance for the Brent-STEP algorithm is the best for this group of functions, finding all targets with le FEs.

We can see that the number of evaluations of GA-PSO-m tends to be higher in most cases, but on the other hand, it could solve all functions while others could not. As expected, the BIPOP-CMA-ES algorithm has the best running time on functions f_1, f_2 and f_5 and reaches three targets at search dimensions 20 and 40. It is worth to notice the performance of the DE algorithm, having the best running time on lower dimensions of f_3 and f_4.

Figure 4, highlights the performance of our proposal in search space dimensions 10, 20, and 40. The figure shows the fraction of functions-target pairs by the number of FEs. In this case, an algorithm that requires less FEs to reach all targets will have more area under the curve. For higher dimensions, the GA-PSO multi-population gives competitive results by reaching all targets within budget. The Brent-STEP algorithm did not provide results for 40D.

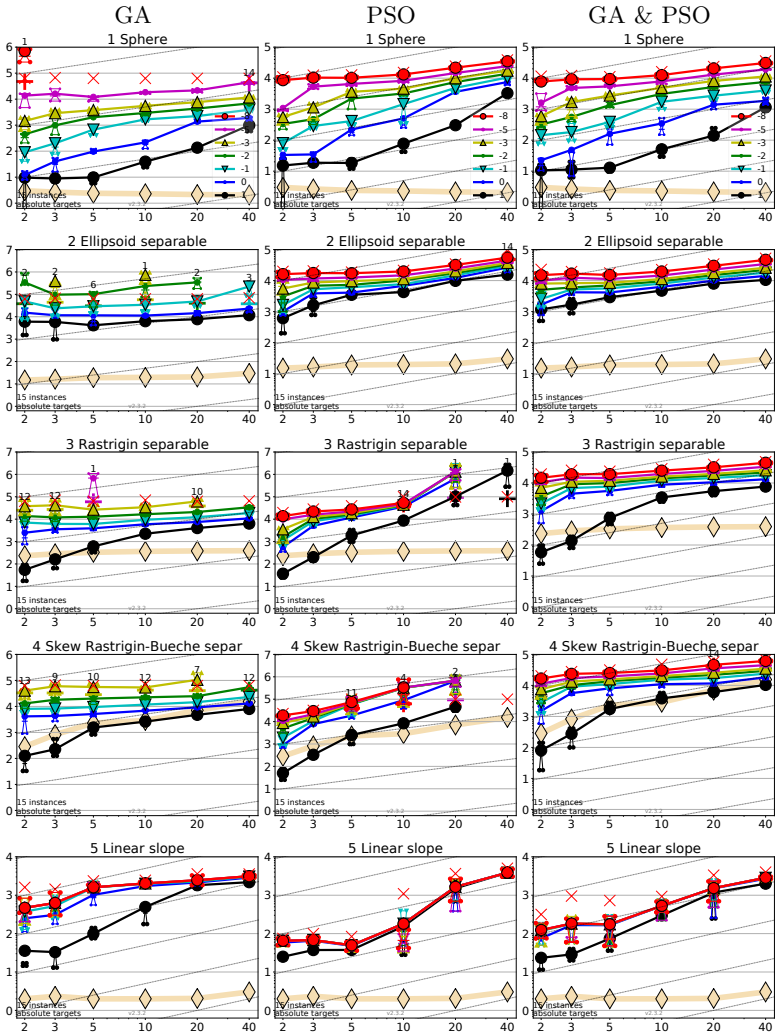

Fig. 3. Scaling of running time with problem dimension to reach certain target values Δf. Lines: average runtime (aRT); Cross (+): median runtime of successful runs to reach the most difficult target that was reached at least once (but not always); Cross (×): maximum number of f-evaluations in any trial. Notched boxes: interquartile range with median of simulated runs; all values are divided by dimension and plotted as log_{10} values versus dimension.

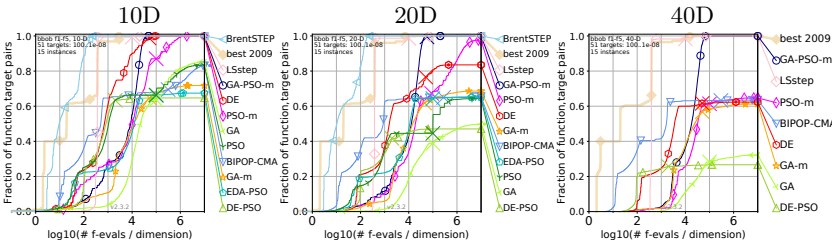

Fig. 4. Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/DIM) for separable functions.

5 Conclusions and future lines of work

In this paper we have presented a new multi-population, hybrid, stateless bioinspired algorithm that uses different processing units to evolve sets of populations, which behave as a stream of continuously updated messages. Population and evolution (or any other kind of change) are totally decoupled, and its processing is stateless, which makes possible to create a mixture of processing algorithms that work asynchronously and in a way that's totally independent of each other.

We have tested this combination of algorithms on the BBOB benchmark functions, obtaining results that are better that those obtained by any of them separated. This confirms results obtained by previous authors in hybrid algorithms, except that, in this case, the architecture is different and there are improvements all across the tested functions; besides, results are better than results that have been published for other algorithms, including a hybrid one, making the hybrid PSO/GA a firm contender in the function optimization arena. It is interesting to note that this is done despite mixing both algorithms in a totally random way, so that it's really impossible to know whether an individual population will be processed by a GA or PSO, or how many cycles of each has undergone. The migration process, however, guarantees a mixing of results and is probably the key to the ultimate success of the algorithm proposed here.

As future lines of work, we will try to add different population-based, data-compatible algorithms to the mix, to see what kind of results they obtain and what could possible be the best possible mix. A way of automatically scaling to get the number of nodes needed would also be a very interesting feature.

An extensive study on the real effects of the mix of algorithms should also be performed, with a theoretical basis if possible, using the size of the basins of attractions, for instance, or how different algorithms move along the fitness landscape.

Acknowledgments

The authors would also like to thank the anonymous referees for their valuable comments and helpful suggestions. The work is supported by the so and so.

References

1. Alba, E.: Parallel evolutionary algorithms can achieve super-linear performance. Information Processing Letters **82**(1), 7 – 13 (2002). https://doi.org/https://doi.org/10.1016/S0020-0190(01)00281-2, evolutionary Computation
2. Back, T.: Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms. Oxford university press (1996)
3. Baluja, S.: Population-based incremental learning. a method for integrating genetic search based function optimization and competitive learning. Tech. rep., Carnegie-Mellon Univ Pittsburgh Pa Dept Of Computer Science (1994)
4. Clerc, M.: Particle swarm optimization, vol. 93. John Wiley & Sons (2010)
5. Črepinšek, M., Liu, S.H., Mernik, M.: Exploration and exploitation in evolutionary algorithms: A survey. ACM computing surveys (CSUR) **45**(3), 1–33 (2013)
6. Doerr, B., Sudholt, D., Witt, C.: When do evolutionary algorithms optimize separable functions in parallel? In: Proceedings of the twelfth workshop on Foundations of genetic algorithms XII. pp. 51–64 (2013)
7. Dorigo, M., Di Caro, G.: Ant colony optimization: a new meta-heuristic. In: Proceedings of the 1999 congress on evolutionary computation-CEC99 (Cat. No. 99TH8406). vol. 2, pp. 1470–1477. IEEE (1999)
8. Eiben, A.E., Smith, J.E.: Genetic algorithms. In: Introduction to evolutionary computing, pp. 37–69. Springer (2003)
9. El-Abd, M., Kamel, M.S.: Black-box optimization benchmarking for noiseless function testbed using an eda and pso hybrid. In: Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers. pp. 2263–2268 (2009)
10. El-Abd, M., Kamel, M.S.: Black-box optimization benchmarking for noiseless function testbed using particle swarm optimization. In: Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers. pp. 2269–2274 (2009)
11. Esmin, A.A.A., Lambert-Torres, G., Alvarenga, G.B.: Hybrid evolutionary algorithm based on pso and ga mutation. In: 2006 Sixth International Conference on Hybrid Intelligent Systems (HIS'06). pp. 57–57. IEEE (2006)
12. Faris, H., Aljarah, I., Mirjalili, S., Castillo, P.A., Guervós, J.J.M.: Evolopy: An open-source nature-inspired optimization framework in python. In: IJCCI (ECTA). pp. 171–177 (2016)
13. Fortin, F.A., Rainville, F.M.D., Gardner, M.A., Parizeau, M., Gagné, C.: Deap: Evolutionary algorithms made easy. Journal of Machine Learning Research **13**(Jul), 2171–2175 (2012)
14. García-Nieto, J., Alba, E., Apolloni, J.: Noiseless functions black-box optimization: evaluation of a hybrid particle swarm with differential operators. In: Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers. pp. 2231–2238 (2009)
15. García-Valdez, J.M., Merelo-Guervós, J.J.: A modern, event-based architecture for distributed evolutionary algorithms. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion. pp. 233–234 (2018)
16. García-Valdez, M., Merelo, J.: Benchmarking a pool-based execution with ga and pso workers on the bbob noiseless testbed. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion. pp. 1750–1755 (2017)

17. Gorges-Schleuter, M.: Explicit parallelism of genetic algorithms through population structures. In: International Conference on Parallel Problem Solving from Nature. pp. 150–159. Springer (1990)
18. Grimaldi, E.A., Grimaccia, F., Mussetta, M., Pirinoli, P., Zich, R.: Genetical swarm optimization: a new hybrid evolutionary algorithm for electromagnetic applications. In: 2005 18th International Conference on Applied Electromagnetics and Communications. pp. 1–4. IEEE (2005)
19. Grosso, P.: Computer simulations of genetic adaptation: Parallel subcomponent interaction in multilocus model. Ph. D. Dissertation, University of Michigan (1985)
20. Guervós, J.J.M., García-Valdez, J.M.: Introducing an event-based architecture for concurrent and distributed evolutionary algorithms. In: International Conference on Parallel Problem Solving from Nature. pp. 399–410. Springer (2018)
21. Guervós, J.J.M., Laredo, J.L.J., Castillo, P.A., Valdez, M.G., Rojas-Galeano, S.: Improving the algorithmic efficiency and performance of channel-based evolutionary algorithms. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion. pp. 320–321 (2019)
22. Gulia, P., et al.: Hybrid swarm and ga based approach for software test case selection. International Journal of Electrical & Computer Engineering (2088-8708) **9** (2019)
23. Hansen, N.: Benchmarking a bi-population cma-es on the bbob-2009 function testbed. In: Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers. pp. 2389–2396 (2009)
24. Hansen, N., Auger, A., Mersmann, O., Tusar, T., Brockhoff, D.: COCO: a platform for comparing continuous optimizers in a black-box setting (2016), arXiv preprint arXiv:1603.08785
25. Hansen, N., Finck, S., Ros, R., Auger, A.: Real-parameter black-box optimization benchmarking 2009: Noiseless functions definitions (2009)
26. Holland, J.H., et al.: Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. MIT press (1992)
27. Karaboga, D.: An idea based on honey bee swarm for numerical optimization. Tech. rep., Technical report-tr06, Erciyes university, engineering faculty, computer ... (2005)
28. Karaboğa, D., Ökdem, S.: A simple and global optimization algorithm for engineering problems: differential evolution algorithm. Turkish Journal of Electrical Engineering & Computer Sciences **12**(1), 53–60 (2004)
29. Kennedy, J.: Swarm intelligence. In: Handbook of nature-inspired and innovative computing, pp. 187–219. Springer (2006)
30. Li, C., Nguyen, T.T., Yang, M., Yang, S., Zeng, S.: Multi-population methods in unconstrained continuous dynamic environments: The challenges. Information Sciences **296**, 95–118 (2015)
31. Li, S., Wu, X., Tan, M.: Gene selection using hybrid particle swarm optimization and genetic algorithm. Soft Computing **12**(11), 1039–1048 (2008)
32. Li, X., Ma, S., Wang, Y.: Multi-population based ensemble mutation method for single objective bilevel optimization problem. IEEE Access **4**, 7262–7274 (2016)
33. Lien, L.C., Cheng, M.Y.: A hybrid swarm intelligence based particle-bee algorithm for construction site layout optimization. Expert Systems with Applications **39**(10), 9642–9650 (2012)

34. Ma, H., Shen, S., Yu, M., Yang, Z., Fei, M., Zhou, H.: Multi-population techniques in nature inspired optimization algorithms : A comprehensive survey. Swarm and Evolutionary Computation **44**(July 2017), 365–387 (2019). https://doi.org/10.1016/j.swevo.2018.04.011
35. Mirjalili, S., Mirjalili, S.M., Lewis, A.: Grey wolf optimizer. Advances in engineering software **69**, 46–61 (2014)
36. Mühlenbein, H., Gorges-Schleuter, M., Krämer, O.: Evolution algorithms in combinatorial optimization. Parallel computing **7**(1), 65–85 (1988)
37. Nicolau, M.: Application of a simple binary genetic algorithm to a noiseless testbed benchmark. In: Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers. pp. 2473–2478 (2009)
38. Nseef, S.K., Abdullah, S., Turky, A., Kendall, G.: An adaptive multi-population artificial bee colony algorithm for dynamic optimisation problems. Knowledge-based systems **104**, 14–23 (2016)
39. Pandi, V.R., Panigrahi, B.K.: Dynamic economic load dispatch using hybrid swarm intelligence based harmony search algorithm. Expert Systems with Applications **38**(7), 8509–8514 (2011)
40. Pošík, P.: Bbob-benchmarking two variants of the line-search algorithm. In: Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers. pp. 2329–2336 (2009)
41. Pošík, P., Baudiš, P.: Dimension selection in axis-parallel brent-step method for black-box optimization of separable continuous functions. In: Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation. pp. 1151–1158 (2015)
42. Pošík, P., Klemš, V.: Benchmarking the differential evolution with adaptive encoding on noiseless functions. In: Proceedings of the 14th annual conference companion on Genetic and evolutionary computation. pp. 189–196 (2012)
43. Robinson, J., Sinton, S., Rahmat-Samii, Y.: Particle swarm, genetic algorithm, and their hybrids: Optimization of a profiled corrugated horn antenna. vol. 1, pp. 314 – 317 vol.1 (02 2002). https://doi.org/10.1109/APS.2002.1016311
44. Shi, X., Lu, Y., Zhou, C., Lee, H., Lin, W., Liang, Y.: Hybrid evolutionary algorithms based on pso and ga. In: The 2003 Congress on Evolutionary Computation, 2003. CEC'03. vol. 4, pp. 2393–2399. IEEE (2003)
45. Swarzberg, S., Seront, G., Bersini, H.: Step: The easiest way to optimize a function. In: Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence. pp. 519–524. IEEE (1994)
46. Tanabe, R., Fukunaga, A.: Evaluation of a randomized parameter setting strategy for island-model evolutionary algorithms. In: Evolutionary Computation (CEC), 2013 IEEE Congress on. pp. 1263–1270. IEEE (2013)
47. Wu, G., Mallipeddi, R., Suganthan, P.N., Wang, R., Chen, H.: Differential evolution with multi-population based ensemble of mutation strategies. Information Sciences **329**, 329–345 (2016)
48. Yang, X.S.: Nature-inspired optimization algorithms. Elsevier (2014)
49. Yuan, B., Gallagher, M.: On the importance of diversity maintenance in estimation of distribution algorithms. In: Proceedings of the 7th annual conference on Genetic and evolutionary computation. pp. 719–726 (2005)
50. Zhao, H., Pei, Z., Jiang, J., Guan, R., Wang, C., Shi, X.: A hybrid swarm intelligent method based on genetic algorithm and artificial bee colony. In: International Conference in Swarm Intelligence. pp. 558–565. Springer (2010)