On The Role Of Execution Order In Hybrid Evolutionary Algorithms

1st Antonio LaTorre

Center for Computational Simulation (CCS)
Universidad Politécnica de Madrid
Madrid, Spain
a.latorre@upm.es

2nd Daniel Molina
Andalusian Institute of Data Science (DASCI)
and Computational Intelligence Dept.
University of Granada
Granada, Spain
dmolina@decsai.ugr.es

***Abstract*—Many real-world problems can be formulated as the optimization of a continuous function. Furthermore, these problems are becoming increasingly more complex every year, reaching, or even exceeding, the thousand of variables. Evolutionary Algorithms have been traditionally successful at solving this kind of problems, due to their good balance in terms of solution quality and computation time. However, the aforementioned growth in the size of the problems requires of novel approaches to deal with the increased complexity of larger solutions spaces. Hybrid evolutionary algorithms are a powerful alternative in these scenarios as they are able to combine the strengths of multiple search methods to solve more complex problems. These hybrid approaches normally do not pay attention to the execution order of their components, being the most frequent strategy to always run them in a predefined sequence. In this contribution we study the role of execution order in hybrid evolutionary algorithms within the context of the multiple offspring sampling framework, one of the best algorithms in large-scale global optimization. As shown in the experimentation, a proper execution order policy can boost the performance of MOS to improve the results of other state-of-the-art algorithms.**

***Index Terms*—Large-scale global optimization, LSGO, Hybrid evolutionary algorithms, Memetic algorithm, Execution order.**

I. Introduction

The field of continuous optimization is a very active research area as many real-world problems of different domains can be formulated as the optimization of a continuous function. Among the different approaches that can be used to deal with this kind of problems, Evolutionary Algorithms (EAs) [1] are among the most successful ones, because they are able to reach accurate solutions in complex scenarios without any knowledge of the problem that is being solved, apart from the fitness function. This is a very important characteristic in real-world scenarios, as shown in [2]. The list of EAs that are currently applied to these problems is extensive, and includes Genetic Algorithms [3] or Differential Evolution [4], to name a few.

However, the performance of EAs depends, to a large extent, on the size (number of variables) of the problem under consideration, as the domain search increases exponentially with the number of dimensions. In this context, there has been an increased interest in Large-Scale Global Optimization (LSGO) during the recent years, which is focused on problem sizes that reach, or even exceed, the thousand of variables, both in academic [5] and engineering problems [6].

The multiple offspring sampling (MOS) framework has shown to be an excellent tool to create hybrid EAs combining multiple individual algorithms to solve LSGO problems [7]–[9], becoming the state-of-the-art method in LSGO for several years. In MOS, the multiple individual algorithms are used in sequence, taking the population created by one algorithm as the initial population for the following one. Previous work [7], [9] evaluate several of the controlling parameters of MOS in different experimental scenarios. However, the execution order of the algorithms remained an open issue and was always manually fixed. As far as the authors are concerned, this is the common approach followed in other hybrid EAs, which makes it a good research subject as shown later in this paper. In particular, several ordering strategies are proposed. Some of them are general and could be used by any other hybrid EA, whereas others are dependent on the dynamic nature of MOS, as it adjusts the participation of each algorithm according to the quality of the solutions that they produce. It seemed thus interesting to consider these two measures (quality and participation ratio) as other criteria to determine the execution order of the composing algorithms. The experimentation carried out confirms the hypothesis that execution order plays an important role on the performance of the hybrid EA, especially as the complexity of the problems increases. With the correct execution order policy, MOS is able to improve the results of the winner of the 2019 IEEE CEC LSGO competition.

Finally, the remainder of this paper is organized as follows. Section II quickly reviews the MOS framework. In Section III, we discuss on several alternatives to determine the execution order of the methods participating in a hybrid algorithm, whereas the experimental details are presented in Section IV. Section V provides the results obtained and a thorough discussion on the results and their consequences, whereas Section VI concludes this study with the main conclusions of the work.

II. A brief description of the MOS framework

Multiple Offspring Sampling (MOS) is a framework for developing hybrid evolutionary algorithms with dynamic ad-

978-1-7281-6929-3/20/$31.00 ©2020 IEEE

justment of the contribution of each of its components. In MOS, the overall search process is divided into a number of phases (called steps) in which the hybridized methods run in sequence until the maximum number of fitness evaluations (FEs) allocated to that step is exhausted. At this level, the framework considers each algorithm to be combined as a black box to which it assigns a budget of FEs (a percentage of the budget assigned to the whole step) and an initial set of candidate solutions. Then, each algorithm evolves this population independently and, once the budget of FEs has been consumed, it returns the final population to the framework that, subsequently, passes it to the next algorithm.

At the end of each step, i.e., when all the algorithms have run to their maximum number of FEs, the framework evaluates the contribution of each method according to the quality of the solutions created by each of them. This measure is used by a participation function to redistribute the overall budget according to this quality. This way, the hybrid algorithm is able to adapt to different stages in the search process and use more efficiently the resources, allowing better performing algorithms to increase their participation and vice versa.

The MOS framework has been successfully used in the past to create hybrid algorithms with an outstanding performance on Large-scale Global Optimization (LSGO) problems. In particular, it obtained the best results in the *Soft Computing Special Issue on scalability of LSGO algorithms* [7] and the Special Sessions and Competitions on LSGO of IEEE CEC 2012 [8] and 2013 [9].

For a detailed explanation of MOS and its governing elements, please refer to [7].

III. Deciding the execution order of the algorithms

As described in Section II, MOS is an adaptive framework for the combination of different algorithms that are applied in sequence with a participation criterion for each of them based on the contribution of each method to the overall search process during their last activation.

As the participation criterion is evaluated after all the algorithms have been run, it could seem that all of them have the same opportunities and that the order in which they are run is not important. However, as the initial population used by one algorithm is the final population created by the previous one, the execution order could have a strong influence on the performance of the algorithm.

MOS has, traditionally, run the algorithms in the same order they were specified in the corresponding configuration file. While this gives the researcher the possibility of manually fixing the desired execution order, it is also true that, in most of the cases, there is no clear sequencing that could be identified as optimal beforehand. It is for this reason that, in this study, we have decided to analyze the role of execution order, proposing different criteria to decide the execution order of the algorithms that are going to be compared:

- **Manual order:** This is the criterion currently used by MOS. The execution order of algorithms is the one in which algorithms are provided in the configuration file. We named it "manual order" because the user can change it manually for each problem. Unfortunately, This is an *a-priori* decision made without any information of the behavior of each algorithm, and it could introduce an unnoticed bias.
- **Random order:** This option randomly selects the execution order for each algorithm at the beginning of each step. This is a simple criterion that tries to run each algorithm in a different order at different steps. The idea is to avoid any bias that could be introduced by a deterministic order as the previous one.
- **Quality order:** In this criterion the algorithms are run according to the value reported by the quality function at each step. This is an order that allows MOS to sort them based on how promising is each algorithm.
  A "descending order" implies that the most promising algorithms are run in the first place. The expected consequence of this approach could be a faster improvement of the results. However, it could be quite conservative (since, presumably, the algorithms executed later will receive better solutions and, therefore, more difficult to improve) and thus the expected gain could be drastically reduced by producing a less flexible order than expected.
  An "ascending order" means that the least performing algorithms are run first. The expected consequence would be, according to the aforementioned influence of execution order on the quality of the population received, that the chances of improving such population increase. On the opposite hand, the execution of the most promising algorithms is postponed and thus convergence could be slower.
- **Participation order:** The last option is to run the algorithms according to their participation ratio (budget of FEs assigned by the framework). In this case, the execution order indirectly depends on the quality measure used by MOS to assign the FEs to each method. This can be seen as a longer term function compared with the quality function, as participation is progressively adjusted according to the quality of the algorithms, which means that, depending on the magnitude of the differences in quality of several methods, the rankings obtained from quality and participation could not be necessarily the same. For example, it could happen that one algorithm is far superior to the others at the beginning of the search, and thus will have a higher quality and participation ratio. As the search advances, it could be surpassed by other algorithms in terms of quality but, if differences are not very large, it could take some time for them to surpass the initially dominating algorithm in terms of participation. This is also influenced by the minimum participation ratio enforced by the framework, that prevents any algorithm from getting a budget smaller than the predefined value.

In Sections IV and V, the different ordering criteria are experimentally compared, in both ascending and descending

order (except for "random order" for obvious reasons).

IV. Experimental framework

The different order criteria are evaluated using the benchmark proposed in the IEEE Congress on Evolutionary Computation, CEC'2013 [5], following its experimental conditions. This benchmark consists of 15 optimization functions for 1000 dimensions, with several degrees of separability, from completely separable functions to fully-non-separable ones:

- Fully separable functions: $f_1 - f_3$.
- Partially separable functions: with a separable subcomponent ($f_4 - f_7$) and without separable subcomponents ($f_8 - f_{11}$).
- Overlapping functions: $f_{12} - f_{14}$.
- Non-separable functions: f_{15}.

In MOS, each individual algorithms keeps its own parameters, which must be tuned to obtain the best performance from the hybrid method. As this is out of scope of this study, we have kept the parameters of the composing algorithms fixed to the recommended values presented in [9]. Additionally, MOS introduces some extra elements:

- A quality measure to determine the quality of new offspring created by each algorithm.
- A participation function to adjust the contribution of each algorithm according to the quality values.
- A step size to determine the budget of FEs to be distributed among the participating algorithms.
- A minimum participation ratio to avoid one algorithm to reach zero, which can be practical when the dominant algorithm is clearly superior in the different stages of the search process.
- The execution order for the algorithms composing the hybrid method.

The first four elements have been thoroughly studied in the past and, for the remainder of this experimentation, we will follow the guidelines established in [9] and summarized in Table I.

TABLE I
MOS parameters

Quality function	Fitness increment average
Participation function	Dynamic participation
Step size	36000 FEs
Minimum participation ratio	20%

The fourth element is studied in this section considering the different criteria described in Section III. We are going to compare different algorithms whose only difference is the execution order criterion:

- **RO**: Random Order, randomly select the execution order for each algorithm at the beginning of each step.
- **AMO**: Ascending Manual Order, run the algorithms in the order specified in the configuration file. It is the original criterion.
- **DMO**: Descending Manual Order, run the algorithms in the opposite order specified in the configuration file.
- **AQO**: Ascending Quality Order, the algorithms are sorted according to their computed quality order (from lower to greater).
- **DQO**: Descending Quality Order, the algorithms are sorted according to their computed quality order (from greater to lower).
- **APO**: Ascending Participation Order, the algorithms are sorted according to their computed participation order (from lower to greater).
- **DPO**: Descending Participation Order, the algorithms are sorted according to their computed participation order (from greater to lower).

Each algorithm is run for each function 25 times, and each run finishes when a maximum number of evaluations, *fitness evaluations, FEs*, is reached ($3 \cdot 10^6$ in this case). Additionally, the best *fitness* is measured at different milestones (in terms of FEs). To simplify the analysis, we are only going to consider the milestones used in previous competitions ($1.2 \cdot 10^5$, $6.0 \cdot 10^5$, $3.0 \cdot 10^6$). The best fitness for each milestone is automatically recorded by the benchmark code.

All the experiments have been run in a computer with the configuration depicted in Table II. Tables and figures have been prepared by using the Tacolab framework[1] [10].

TABLE II
Computer Configuration

PC	Intel Xeon 8 cores 1.86Ghz CPU 22GB RAM
Operating System	Ubuntu Linux 11.10
Prog. Language	C++ & Matlab
C/C++ Compiler	Clang 2.9
Matlab Version	Matlab R2011B

V. Experimental Results and Discussion

In this section we analyze the influence of the execution order in hybrid evolutionary algorithmds by comparing the performance of the alternative criteria proposed in Section III and following the experimental framework defined in Section IV.

First, we compare all the different execution order alternatives among them, to find the one that reports the best results, which will be selected for further comparison. Then, we compare this alternative against the original MOS algorithm and the winner of the last LSGO competition, held at IEEE CEC'2019, CC-RDG3 [11], using the experimental results provided by their authors.

A. Comparison of the different execution order criteria

In this section, we are going to compare all the alternative execution order criteria among them.

First, we report the number of functions for which each algorithm obtains the best results and their average ranking

[1] https://tacolab.org/

TABLE III
NUMBER OF FUNCTIONS IN WHICH EACH ALGORITHM REPORTS THE BEST RESULTS

Algorithm	$1.2 \cdot 10^5$	$6 \cdot 10^5$	$3 \cdot 10^6$
AMO	0	0	0
APO	2	**8**	**6**
AQO	**6**	0	1
DMO	2	2	1
DPO	0	0	0
DQO	3	2	1
RO	2	3	4

TABLE IV
AVERAGE RANKING FOR EACH ALGORITHM AND MILESTONE

Algorithm	$1.2 \cdot 10^5$	$6 \cdot 10^5$	$3 \cdot 10^6$
AMO	4.13	4.20	3.87
APO	3.33	**2.13**	**2.83**
AQO	**3.07**	3.67	3.47
DMO	5.67	5.54	5.13
DPO	3.87	3.94	5.30
DQO	3.47	3.94	3.53
RO	4.47	4.60	3.87

in Tables III and IV, respectively. From Table III, it can be observed that, whereas for FEs=$1.2 \cdot 10^5$ AQO obtains the best results in more functions, from FEs=$6 \cdot 10^5$, APO is clearly the execution order that achieves the best results in more functions. Additionally, Table IV shows that APO has the best average ranking (for the same two last milestones), followed by AQO (that obtains the best results with the lowest FEs). These results seem to be coherent with the behavior of both criteria: while AQO focuses on local performance changes (as it depends on quality, which is an immediate indicator), APO takes participation into consideration, which can have an important inertia and it can take some steps to revert the execution order of algorithms (being thus a bit more conservative). This makes the first option better suited at the beginning of the search process, whereas the second one works better in the long-term.

Then, we have compared all the MOS versions using the methodology proposed for the WCCI'2020 competition:

1) For each function, algorithms are sorted according to their average mean error.
2) Each algorithm is given a certain number of points according to its ranking, and following the F1 criterion: 25 points to the best algorithm, 18 points to the runner-up, 15 to the third one, etc.
3) The results on each function are aggregated for all the algorithms to obtain their overall score.
4) Algorithms are compared (for example, with bar plots, or stacked bar plots) for each considered milestone.

The overall score of each algorithm at each milestone is depicted in Table V. It can be seen that APO is clearly the execution order that produces the best results (being AQO better in the lowest FEs scenario), confirming the results shown in Table IV.

TABLE V
SCORES OF ALL THE EXECUTION ORDER APPROACHES ACCORDING TO THE F1 CRITERION

Algorithm	$1.2 \cdot 10^5$	$6 \cdot 10^5$	$3 \cdot 10^6$
AMO	184	180	196
APO	223	**298**	**263**
AQO	**255**	205	223
DMO	146	149	156
DPO	195	192	143
DQO	224	199	208
RO	183	187	211

Figures 1, 2, and 3 graphically show the information gathered in Table V. It can be observed that APO (using Ascending Participation Order) not only is the criterion that is clearly better than the other ones, but it also achieves good results in all the groups of functions.

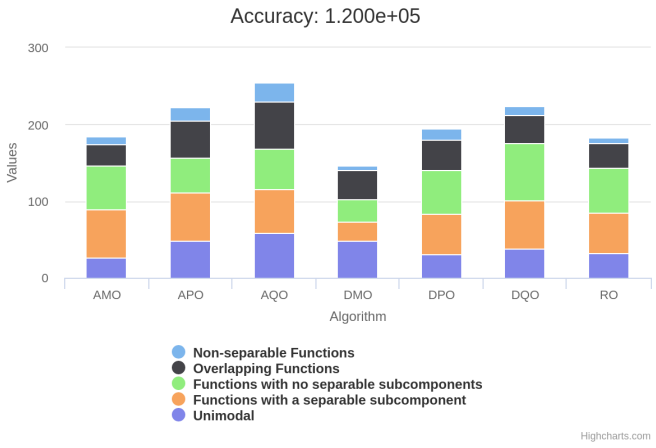


Fig. 1. Results comparison for FES=$1.2 \cdot 10^5$

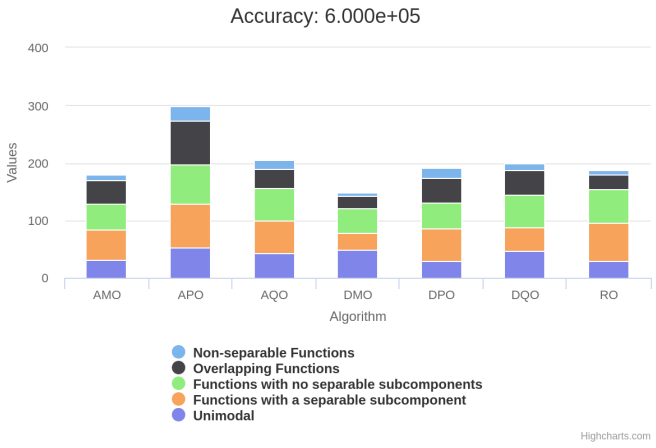


Fig. 2. Results comparison for FES=$6 \cdot 10^5$

Paying more attention to the previous figures it can be seen that the improvement obtained by APO is mainly due to a much better behavior in two groups of functions: overlapping

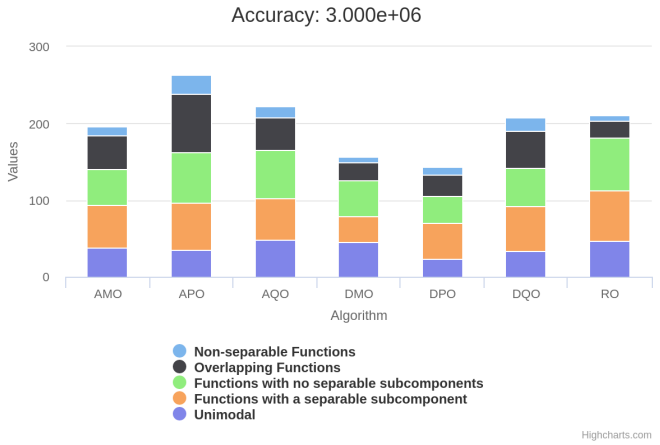


Fig. 3. Results comparison for FES=$3 \cdot 10^6$

and non-separable functions. This is especially interesting because these functions are the most difficult to optimize.

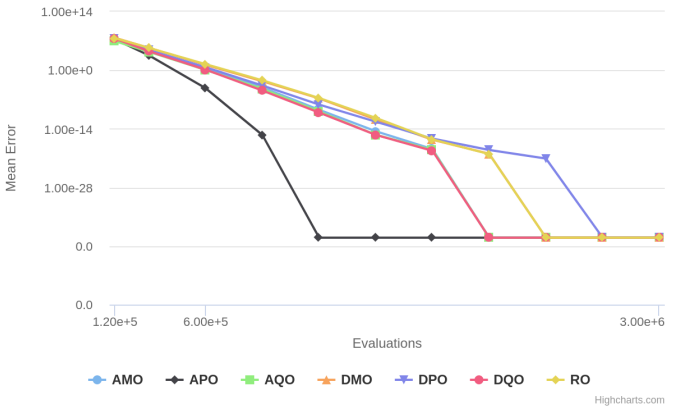
B. Convergence of the different execution order criteria

In LSGO it is very important how quickly an algorithm is able to obtain competitive results (because in real-world problems, it is interesting to use as few as possible FEs with the least possible loss of error). In this section we are going to show and analyze the convergence plots comparing the efficiency of each algorithm.

In order to analyze convergence speed of the different methods, we have measured the error for each of the previous algorithms at different milestones: $\{1.2, 3.0, 6.0, 9.0, 12, 15, 18, 21, 24, 27, 30\} \cdot 10^5$. To avoid overwhelming the reader with all the convergence plots, we are only going to show a subset of representative functions, covering the different observed scenarios.

In the following paragraphs, we are going to present several conclusions considering the convergence plots:

Fig. 4. Convergence plot for function F_1



- In separable functions such as F_1, see Figure 4, although all the algorithms achieve the same results, sometimes APO is able to do it more quickly.

Fig. 5. Convergence plot for function F_5

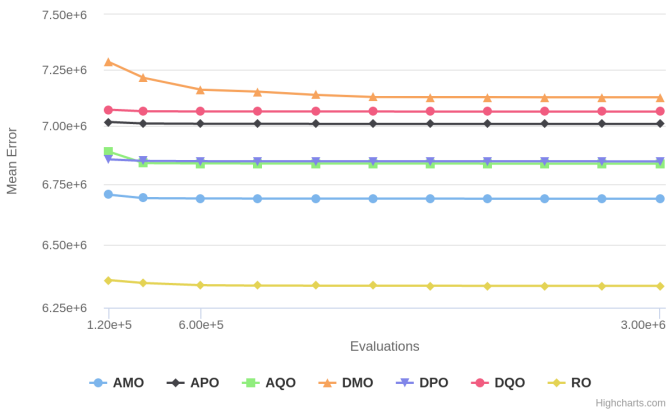
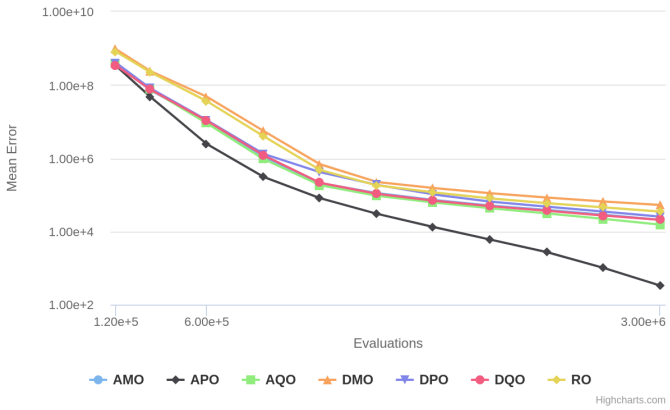


Fig. 6. Convergence plot for function F_7



- In non-separable functions there are two different behaviors. In some of them, such as F_5, the best solution is obtained in few FEs, and that value is not improved for the remaining FEs (see Figure 5). In this function, curiously, the best average results are obtained by the random order criterion. This could be explained by the fact that, whereas the median results are very similar, the other criteria do not report robust results, which means that average values are worse than those obtained by the random criterion. In other functions, such as F_7, APO shows a greater efficiency, with a difference between APO and the other execution order criteria that increases with the FEs (see Figure 6).
- In non-separable functions, the behavior obtained is similar to the one shown in Figure 7. In these functions, APO clearly obtains the best results.
- Finally, in the overlapping function, F_{15}, there are several algorithms with bad behavior, such as RO and DMO, and the other ones are more similar, with a slight advantage of APO.

Fig. 7. Convergence plot for function F_{13}

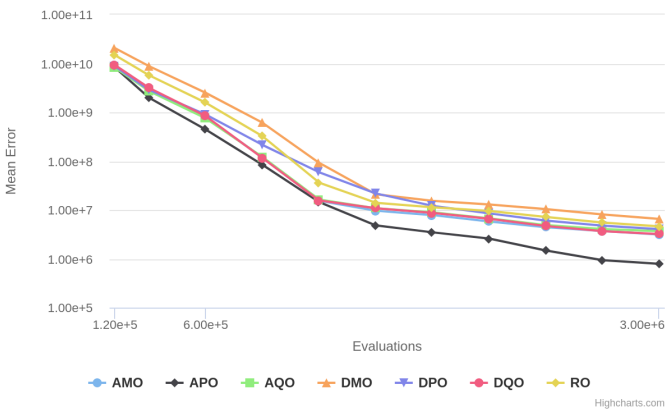
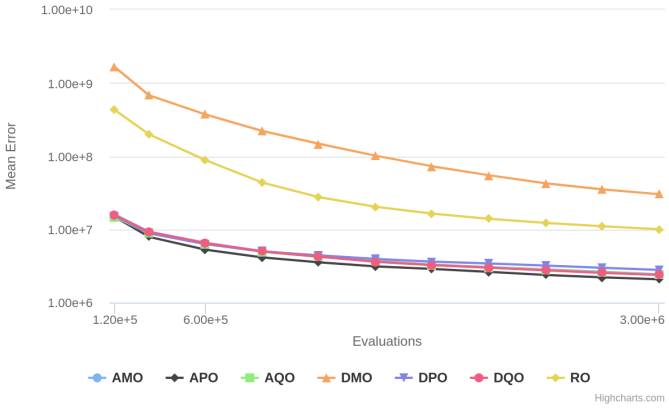


Fig. 8. Convergence plot for function F_{15}



C. Comparison with other algorithms

Finally, we compare our proposal (MOS using the APO criterion) with state-of-the-art algorithms to analyze how competitive it is. In particular, we have selected for comparions the previous winner of the last LSGO competition held in CEC'2019[2], the CC-RDG3 algorithm [11]. Additionally, we are going to compare it against the original MOS algorithm, to observe the improvement obtained by the new execution order.

As in Section V-A, we are going to compare the algorithms following two different procedures: first, with the average ranking, and later using the LSGO competition procedure.

TABLE VI
AVERAGE RANKING OF THE PROPOSAL AND REFERENCE ALGORITHMS

Algorithm	$1.2 \cdot 10^5$	$6 \cdot 10^5$	$3 \cdot 10^6$
CC-RDG3	**1.40**	**1.67**	1.93
MOS	2.13	2.40	2.30
MOS_APO	2.47	1.93	**1.77**

[2]http://www.tflsgo.org/special_sessions/cec2019.html

In Table VI, the average ranking is shown. We can see that MOS_APO achieves better relative results as FEs increase. Finally, for $FE = 3{\cdot}10^6$ the proposal achieves a better average ranking that the current state-of-the-art, CC-RDG3.

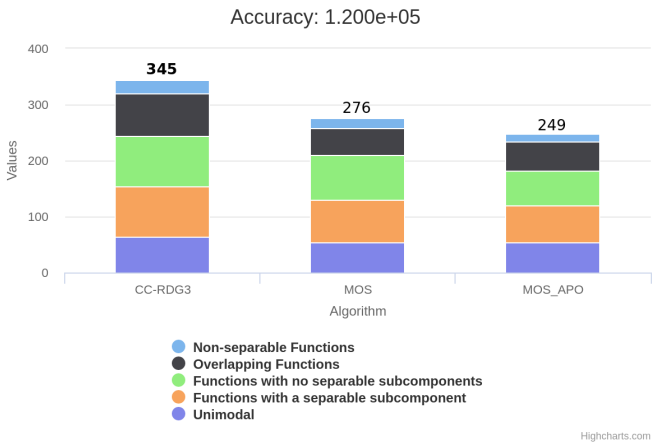


Fig. 9. MOS_APO vs. reference algorithms for FES=$1.2 \cdot 10^5$

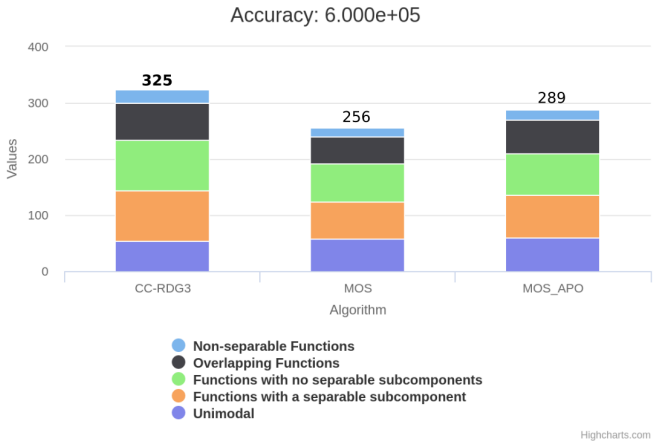


Fig. 10. MOS_APO) vs. reference algorithms for FES=$6 \cdot 10^5$

In Figures 9, 10, and 11 the results following the competition procedure are provided. It is clear that for a lower number of FEs, CC-RDG3 obtains the best results, and that MOS_APO is able to improve as more FEs are used, obtaining the best results with FES=$3 \cdot 10^6$. The good results of MOS_APO demonstrates the convenience of choosing this participation order in MOS to allow the most promising algorithms to be applied first, while being flexible enough to adapt when the situation changes. With these results in mind, it can be concluded that MOS_APO improves the results of the previous LSGO competition winner.

These good results are mainly due to the non-separable and overlapping functions, as can been in Figures 12 and 13, in which MOS_APO obtains the best results compared with both the original MOS and CC-RDG3. It is a remarkable improvement because the original MOS had not a good behavior in these functions.

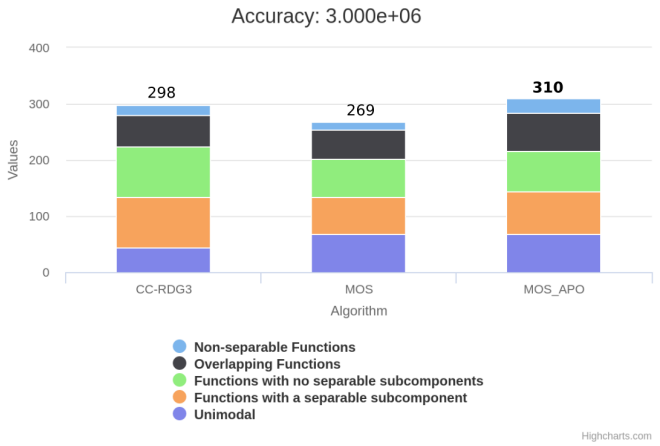


Fig. 11. MOS_APO) vs. reference algorithms for FES=$3 \cdot 10^6$

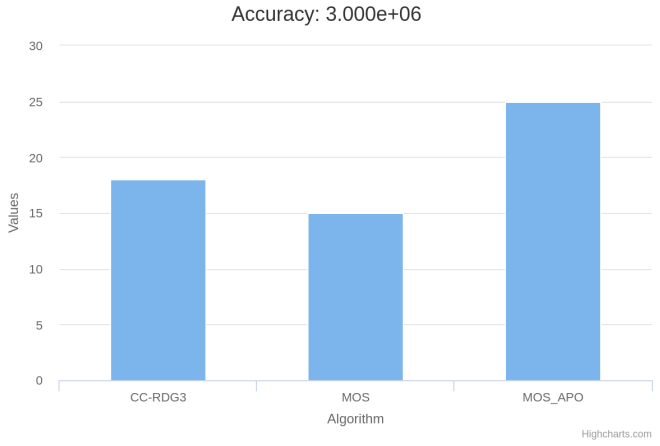


Fig. 12. Performance of MOS_APO vs. reference algorithms in non-separable functions

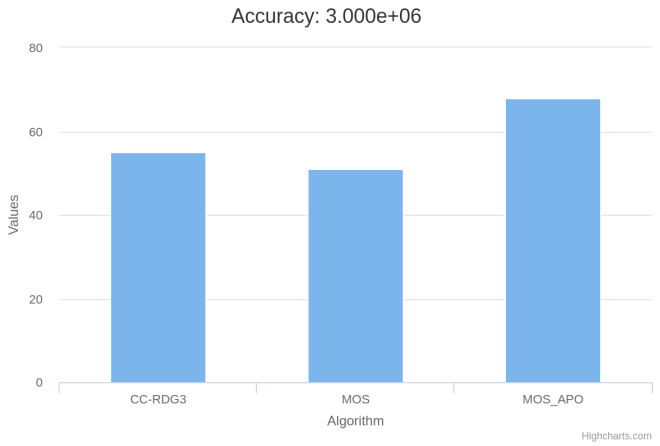


Fig. 13. Performance of MOS_APO vs. reference algorithms in overlapping functions

VI. Conclusions

Multiple Offspring Sampling (MOS) is a framework for developing hybrid evolutionary algorithms, and has been considered the state-of-art in LSGO for several years. MOS is characterized by a dynamic adjustment of the contribution of each of its components as a function of the improvement obtained by each of them.

Although previous works have analyzed several MOS parameters, all the algorithms were traditionally applied in the same order of the configuration file. However, the execution order can have a strong influence on the search, because the initial population used by one algorithm is the final population created by the previous one.

In this work, we propose and compare different execution order criteria, considering, apart from the aforementioned fixed criterion, a random order, and two dynamic criteria that depend, respectively, on the quality and participation computed for each algorithm at each step (both in ascending and descending order). From the comparisons, it is shown that the best results are obtained when MOS uses the execution order that considers participation and arranges algorithms in ascending order.

This proposal, MOS with Ascending Participation Order (MOS_APO), is also compared not only with the original MOS but also with the winner in the previous LSGO competition, CC-RDG3, showing that, for the largest FEs, the proposal obtained the best results. Thus, it can be concluded that MOS_APO should be considered the new state-of-the-art method in LSGO.

Acknowledgment

This work was supported by grants from the Spanish Ministry of Science (TIN2017-83132-C2-2-R, TIN2017-89517-P).

References

[1] T. Back, D. B. Fogel, and Z. Michalewicz, *Handbook of Evolutionary Computation*, 1st ed. GBR: IOP Publishing Ltd., 1997.

[2] N. Xiong, D. Molina, M. L. Ortiz, and F. Herrera, "A Walk into Metaheuristics for Engineering Optimization: Principles, Methods and Recent Trends," *International Journal of Computational Intelligence Systems*, vol. 8, no. 4, pp. 606–636, May 2015.

[3] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, 1st ed. USA: Addison-Wesley Longman Publishing Co., Inc., 1989.

[4] R. Storn and K. V. Price, "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces," *Journal of Global Optimization*, vol. 11, no. 4, pp. 341–359, Jan. 1997.

[5] X. Li, K. Tang, M. N. Omidvar, Z. Yang, and K. Quin, "Benchmark Functions for the CEC'2013 Special Session and Competition on Large Scale Global Optimization," Evolutionary Computation and Machine Learning Group, Tech. Rep., Jan. 2013.

[6] G. Vanderplaats, "Very large-scale optimization," National Aeronautics and Space Administration (NASA), Langley Research Center, Tech. Rep. NASA/CR-2002-211768, 2002.

[7] A. LaTorre, S. Muelas, and J. M. Peña, "A MOS-based dynamic memetic differential evolution algorithm for continuous optimization: a scalability test," *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, vol. 15, no. 11, pp. 2187–2199, 2011.

[8] ——, "Multiple Offspring Sampling in Large Scale Global Optimization," in *2012 IEEE Congress on Evolutionary Computation (CEC 2012)*. IEEE Press, Jan. 2012, pp. 964–971.

[9] ——, "Large Scale Global Optimization: Experimental Results with MOS-based Hybrid Algorithms," in *2013 IEEE Congress on Evolutionary Computation (CEC 2013)*. IEEE Press, Jan. 2013, pp. 2742–2749.
[10] D. Molina and A. LaTorre, "Toolkit for the Automatic Comparison of Optimizers: Comparing Large-Scale Global Optimizers Made Easy," in *2018 IEEE Congress on Evolutionary Computation (CEC 2018)*, Jul. 2018, pp. 1229–1236.
[11] Y. Sun, X. Li, A. Ernst, and M. Omidvar, "Decomposition for large-scale optimization problems with overlapping components," in *2019 IEEE Congress on Evolutionary Computation (CEC)*, Jul. 2019, pp. 326–333.