A Training Difficulty Schedule for Effective Search of Meta-Heuristic Design

Jair Pereira Junior
Department of Computer Science
University of Tsukuba
Tsukuba, Japan
pereira-junior.ua.ws@alumni.tsukuba.ac.jp

Claus Aranha
Department of Computer Science
University of Tsukuba
Tsukuba, Japan
caranha@cs.tsukuba.ac.jp

Tetsuya Sakurai
Department of Computer Science
University of Tsukuba
Tsukuba, Japan
sakurai@cs.tsukuba.ac.jp

***Abstract*—In the context of optimization problems, the performance of an algorithm depends on the problem. It is difficult to know *a priori* what algorithm (and what parameters) will perform best on a new problem. For this reason, we previously proposed a framework that uses grammatical evolution to automatically generate swarm intelligence algorithms given a training problem. However, we observed two issues that affected the results of the framework. The first issue was that sometimes the training problems are too easy, and any candidate algorithm could solve it or too difficult that no candidate algorithm could solve them. The second issue was the presence of parameters in the grammar, which causes a significant increase in the search space. In this work, we addressed those issues by investigating three training schedules in which the problems start easy and get harder over time. We also investigated whether numerical parameters should be part of the grammar. We compared these training schedules to the previous one and compared the performance of the generated algorithms against the traditional algorithms, which are DE, PSO, and CS. We found that gradually increasing the difficulty of the training problem produced algorithms that could solve more testing instances than training only in 10-D. The results suggest that a step-by-step increase in difficulty is a better approach overall. We also found that including parameters in the grammar resulted in algorithms on par with the traditional meta-heuristics. Besides, as expected, our results show that removing parameters from the grammar exhibit the worst overall performance. However, interestingly it could solve most of the testing instances within the given testing budget.**

***Index Terms*—automatic algorithm generation, training scheduling, meta-heuristic, hyper-heuristic, grammatical evolution**

I. Introduction

Many problems in engineering, design, and sciences can be described as optimization problems. A well-known way to solve these problems is to use meta-heuristic search algorithms, such as Genetic Algorithm (GA) [1], Differential Evolution (DE) [2], Particle Swarm Optimization (PSO) [3], and Cuckoo Search (CS) [4]. However, the performance of an algorithm depends on the problem, and it is difficult to know *a priori* what algorithm (and what parameters) will perform best on a new problem.

In this context, we previously proposed a framework based on grammatical evolution (GE) to automatically generate swarm intelligence (SI) algorithms given a problem [5]. In that work, the algorithms automatically generated by GE showed comparable performance to the traditional hand-crafted algorithms. However, we identified two issues that affected the results of the framework.

The first problem was observed in the training. Sometimes the problem is too easy, and any candidate algorithm gets the best results, sometimes it is too hard, and no candidate algorithm gets any result. In both cases, the final result is an inability of GE to learn.

The second problem was regarding the setting of control parameters, such as population size and crossover probability. If the parameters are part of the grammar, the search space becomes exceptionally high, and many similar candidate algorithms are generated. On the other hand, if they are not part of the grammar, the value of these parameters will have a significant impact on the evaluation of new individuals.

In this work, we addressed those issues by investigating three training schedules in which the problems start easy and get harder over time. We also investigated whether numerical parameters should be part of the grammar or not. We compared these training schedules to the previous one and compared the performance of the generated algorithms against the traditional algorithms (DE, PSO, and CS) on the COCO [6] benchmark.

We observed that (1) it is effective to increase the difficulty of the problem during the training and that (2) as expected, removing the parameters from the grammar has the worst overall performance. However, interestingly it could solve most of the problems within the given testing budget.

This paper is organized as follows: Section II provides an overview of the field. Section III provides a description of our framework to automatically generate SI-algorithms. Section IV presents the proposed training schedules. Section V describes the experimental design. Section VI presents the results and discussion of our investigation. Section VII presents the conclusion, limitations and future research.

II. Related Work

It has been observed that there is no single algorithm that can perform best in all classes of optimization problems. This phenomenon, known as performance complementarity [7], has motivated many studies on ways to select or generate a proper algorithm for a given class of problems. Algorithms that do this previously mentioned task are called hyper-heuristics.

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

While meta-heuristics searches for solutions in the problem space, in hyper-heuristics, the algorithms operate on a higher level searching for programs in the algorithm design space.

Burke et al. [8] proposed a classification of hyper-heuristics methods based on their feedback source during learning (online, offline, and none), and the nature of their search space (selection or generation, and construction or perturbation). For a heuristic generation, GP is the most commonly used method [8]. GP conducts the search using the traditional operators of the genetic algorithm on a set of predefined building blocks. However, GP can have a huge search space, depending on the problem. Another method for heuristic generation is Grammatical Evolution (GE). GE is a variation of GP, in which a context-free grammar restricts the search space.

Studies on using GP and GE for algorithm selection and generation have investigated a self-configuring crossover operator [9], generation and tuning of mutation operators [10], design tuning of genetic algorithm for black-box search [11], design tuning of PSO [12]–[14], tuning of the update-mean rule of CMA-ES [15], designing optimizers largely from scratch [16] and many others.

Our work follows the same direction as the previous research by trying to automate the design of meta-heuristic search algorithms. We, however, investigate different ways to do the training. Our goal is to not only analyze the design space of meta-heuristic algorithms by using a GE to guide the recombination of components of existing algorithms. But, as well as to determine ways to improve the training phase of the hyper-heuristic. Thus, this research aims to enhance our understanding of the design space of meta-heuristics, enabling us to determine components that are useful for which classes of optimization problems, and how to guide the search for those components better.

III. Framework: Franken-swarm

Our framework to automatically generate SI-algorithm explores the design space of meta-heuristics algorithms using Grammatical Evolution (GE). The framework has two main parts: the components library and the recombination framework.

The component library is composed of a set of building blocks extracted from meta-heuristic algorithms, and the recombination framework is composed of a formal grammar that describes how the components are put together. The details of these two parts are described in the two following sections, respectively.

Our framework is implemented in Python[1]. The meta-heuristics components are an in-house implementation, while the GE engine uses the PonyGE2 Grammatical Evolution library [17].

A. Component library

This part is composed of several meta-heuristic components. These components were extracted from the three most traditional swarm intelligence (SI) meta-heuristics in the literature. These algorithms are Differential Evolution (DE), Particle Swarm Optimization (PSO), and Cuckoo Search (CS). In addition to these algorithms, some Genetic Algorithm (GA) operators were also used. Next, all operators were categorized as Initialization, Selection, Variation, Replacement, Dropout, or Repair rules. Table I shows the categorization of all operators present in the proposed framework. A brief description of these categories follows. Initialization operators define how each candidate solution will be initialized at the beginning of the algorithm and how it will be re-initialized after a drop operator. Selection operators select candidate solutions to input to variation operators. Variation operators modify the selected candidate solutions. Replacement operators choose which between the pre-updated solution and the updated solution will be in the next population. Dropout operators choose candidate solutions to be re-initialized. Repair operators define what will be done to the values out of bounds of a candidate solution.

B. Recombination Framework

This part uses GE to search the design space of meta-heuristic algorithms by recombining the building blocks from the component library based on their category. Algorithm 1 shows a simplified version of the formal grammar, focusing on the common structures of SI-algorithms. The terminals of the operators can be found in Table I. This grammar describes algorithms with one, two, or three variation operators. These operators are associated with up to three different selection rules depending on the function's arity, and one replacement rule. A drop rule may be applied at the end of every iteration.

Algorithm 1 Simplified Grammar for Algorithm Generation

```
#components
<S>          ::= <define_repair><main>
<main>       ::= <iteration> | <iteration><drop>
<iteration>  ::= <step>
               | <step><step>
               | <step><step><step>
<step>       ::= S ← <selection>(X)
                 U ← <variation>(S)
                 X ← <replacement>(X, U)
#parameters
<population> ::=  50 | 100 | 200  | 400 | 800 | 1200
<float>      ::= 0.00 | 0.25 | 0.50 | 0.75 | 1.00 | 2.00
<percentage> ::= 0.10 | 0.25 | 0.50 | 0.75 | 1.00
```

C. Algorithm Synthesis

To generate an algorithm for a given problem, our framework is submitted to a training phase. The goal of the training phase is to generate an algorithm that performs well in the training problems and should show similar performance when tested on similar classes of problems.

In the training phase, several algorithms are generated randomly based on the defined grammar. These algorithms are then tested on the training problem. Next, the classical genetic operators are used to generate the next candidate algorithms. The previous step is repeated until the maximum number

[1]All the code, and the data for this paper, including experimental scripts, are available at https://github.com/jair-pereira/FRK/tree/CEC'20.

TABLE I
META-HEURISTIC OPERATORS: BRIEF DESCRIPTION AND CATEGORIZATION

Initialization	
Random Uniform	generates a candidate solution where each element is sampled from a uniform distribution

Selection	
Current	selects the candidate solution currently being updated
Random	selects a candidate solution from the population with uniform probability
K-Tournament	selects the best candidate solution out of k solutions selected from the population with uniform probability

Variation	
PSO Update Rule	velocity$^{i+1} = w.v^i + c1.r1.(pbest^i - x^i) + c2.r2.(gbest^i - x^i)$ x$^{i+1} = x^i + v^{i+1}$
Cuckoo Update Rule	$x^{i+1} = x^i + 0.2 * Levy * (x^i - pbest)$, where *Levy* is a Lévy Flight
Uniform Mutation	each element of the x^i has a fixed probability to be replaced with a value sampled from a uniform distribution
Blend Crossover	x$^{i+1} = (1-\gamma).x_1^i + \gamma.x_2^i$
Exponential Crossover	each element of x_1^i has probability^n to be replaced with elements from x_2^i, where n is the number of previous replacement
Uniform Crossover	each element of x_1^i has a fixed probability to be replaced with elements from x_2^i
DE Mutation	x$^{i+1} = x_1^i + \beta.(x_2^i - x_3^i)$

Replacement	
Current	replaces the current candidate solution by the updated one
Always the Best	replaces the candidate solution by the updated one if there is fitness improvement
Cuckoo Rule	replaces the candidate solution by the updated one if the fitness is better compared to a random updated candidate solution
Later	waits for another update rule before trying replacement again

Drop	
Random	uniformly attempts to drop candidate solutions based on a given probability
Worst	drops k fitness-wise-worst candidate solutions based on a given probability

of iterations is reached. The final output is a meta-heuristic search algorithm that had the best performance in the training problem.

IV. PROPOSED TRAINING SCHEDULING

In this section, we describe the training scheduling and grammar variants.

We proposed different training scheduling to investigate whether training our framework directly in a hard task is better than training in a task that starts easier and gets harder over time. The difference between all the scheduling is that the function's dimension, in which the framework is trained, increases over time. In the baseline, our framework is trained directly in 10-D. In scheduling 2-to-10D, it is trained initially in 2-D and later, it changes to 10-D. Similarly, in the scheduling 5-to-10D, it starts in 5-D and later changes to 10-D. In the scheduling 2,3,5-to-10-D, it starts in 2-D, changes to 3-D, then 5-D, and finally 10-D.

We proposed two variations of the grammar that recombines the meta-heuristics operators to investigate whether numerical parameters should be included in the grammar or not. This question arose because the presence of parameters in the grammar causes a significant increase in the search space. We proposed one grammar that searches for operators only, and we use as a baseline the grammar of our previous work, which searches for operators and its parameters within a certain granularity as specified in 1. For the new grammar, random parameters are used for every run of a candidate algorithm.

V. EXPERIMENTAL DESIGN

We carried out a training-testing experiment to investigate (1) whether training in lower dimension leads to equivalent promising regions in the search space of the same problem in a higher dimension, and (2) whether it is beneficial or not to search for operators altogether with its parameters.

In the training phase, an algorithm is submitted to training problems in order to tune its parameters and/or to select its components. In the testing phase, the algorithm is submitted to different problems in order to assess its performance on problems with similar characteristics.

In order to assess the performance of our framework, we compared it to three traditional methods in the literature, PSO, DE, and CS. It is important to note that all the analyses were done separately for each problem in the training set.

We chose the problem instances from the COCO [6] benchmark, which is a continuous numerical black-box optimization benchmark. COCO implements 24 noiseless functions divided into 5 groups based on their properties and, consequently, difficulty. In this benchmark, a problem instance is a function and two parameters: its scalable dimension (2, 3, 5, 10, 20, and 40) and its instance (1 to 15) - a random, artificial shift on the function space. We selected the functions in the Table II due to the insights obtained about the algorithms behaviour [18]. It is important to emphasize that the selected Rastrigin (f15) is a transformed version that alleviates its symmetry and regularity.

TABLE II
SELECTED FUNCTIONS FROM COCO BENCHMARK.

Function	Modality	Characteristic	Information Obtained
Sphere	Unimodal	Highly Symmetric	Optimal convergence rate
Attractive Sector	Unimodal	Highly Asymmetric	Behaviour in an asymmetric landscape
Rastrigin	Multimodal (highly)	Adequate global structure Alleviated symmetry (transformed) Alleviated regularity (transformed)	Behaviour in an highly multimodal landscape
Gallagher's	Multimodal	Weak global structure (unrelated and randomly chosen 101 optima)	Behaviour in a landscape without global structure

A. *Training and Testing*

We trained the baseline (PSO, DE, and CS) and our proposed framework in instance 1 of 10-D Sphere, Attractive Sector, Rastrigin, and Gallagher's 101-me. The maximum number of functions evaluated (NFE) was 3e+7.

For the baseline, we tuned their parameters using iRace [19] with maxExperiments=3e+3 and NFE of a run=1e+4. For the testing phase, we used only the best set of parameters found. Table III shows the tuned parameters for each problem instance.

For our framework, we trained both grammars using all four training schedules previously introduced. The GE parameters were iteration=30, population=20, and meta-heuristic NFE=1e+4 with 5 repetitions. Only the best algorithm with the lowest mean - out of the 5 repetitions, was used for the testing phase.

We balanced the NFE between the hyper-heuristic and meta-heuristic to have the number of iteration and population of the hyper-heuristic as high as possible while keeping it feasible runtime-wise.

To write a balanced grammar, we followed the guidelines in [20]. We chose the subtree representation to reduce invalids as recommended in [17]. Consequently, mutation and crossover did also use subtree representation. The tournament selection had k=4 for low pressure (20% of the population).

We tested all the configured algorithms on the same functions as in the training but on the others 2 to 15 instances. For each problem instance, the algorithms had NFE=1e+5 with up to 10 restarts. By testing on the same functions, but in different instances, we are looking at whether our framework could capture the function underlying structure during the training phase.

VI. RESULTS

In this section, we present the comparison of the two grammars variants in each training schedule, and the performance during the testing.

A. *Training Schedules*

Here, we assessed whether we are generating better candidate algorithms over time and how good those algorithms are. We compared the two grammars (including parameters vs not including parameters) in each proposed training scheduling by plotting the precision of the best candidate algorithm during the training. The chosen precisions ranges from 1e+02 to 1e-08. Note that a change in one decimal place of precision means ten times increase being in a logarithmic scale.

The training and testing results in Rastrigin were omitted due to limit of space and similarly poor performance of all methods. Including the testing results of the traditional algorithms. We discuss this situation in the Section VII.

Figs. 1-3 compare the grammar that includes parameters against the grammar that does not include it, for all training schedules. They show the precision of the best candidate algorithm at each iteration of the grammatical evolution. Each figure corresponds respectively to the functions Sphere, Attractive Sector, and Gallagher's 101-me. We compare pairs grammar-training schedule by looking at the area under the curve, in which higher precision at early iterations is better.

The gray area on the left side indicates when the candidate algorithms were evaluated in low dimension(s). Similarly, the white area on the right side signals the training in the target dimension (10-D).

As can be seen in fig. 1 (Sphere), the grammar that includes

TABLE III
PSO, DE, AND CS: PARAMETER TUNING (IRACE)

Particle Swarm Optimization				
Problem	**Population**	**Velocity modifier**	**Pbest modifier**	**Gbest modifier**
Sphere	100	0.39	0.00	1.84
Attractive Sector	200	0.56	0.00	1.67
Rastrigin	800	0.12	0.01	1.82
Gallagher's	400	0.49	0.12	1.79

Differential Evolution			
Problem	**Population**	**DE-Mutation modifier**	**Crossover probability**
Sphere	50	0.08	0.18
Attractive Sector	50	0.47	0.92
Rastrigin	50	0.17	0.85
Gallagher's	50	0.5	0.93

Cuckoo Search			
Problem	**Population**	**Drop probability**	**Solutions to be dropped**
Sphere	50	0.57	25
Attractive Sector	50	0.97	25
Rastrigin	50	0.89	25
Gallagher's	50	0.78	25

parameter with 5-to-10D scheduling is the only case that does not lose precision when the problem gets harder. Overall, the training schedule that uses more dimensions seems to be more robust. Similarly, the grammar that includes parameters is also more effective because they lose less precision.

In fig. 2 (Attractive Sector), it is not clear which method is better. Overall, the grammar that includes parameters using 2-10D is as good as the grammar that does not include parameters using 5-10D.

Lastly, fig. 3 (Gallagher's) is consistent with the fig. 1, in which the grammar that includes parameter with 5-to-10D scheduling is the only case that does not lose precision when the problem gets harder and including parameters is overall better.

Looking at all functions for the scheduling 2,3,5-to-10D, both grammars do not lose precision when the problems changes from 2D to 3D. When it changes from 3D to 5D, they quickly recover the precision (except in Rastrigin). This quick recover may indicate that a step-by-step increase in dimension is beneficial for the training. Also, the grammar that does not include parameters is always slightly better, possibly indicating that the components capture better the underlying structure of the functions compared to parameters.

Moving to the Fig. 4, we picked the training scheduling that hit better precision overall (5-to-10-D) and compared it to the training done directly in 10-D. In this graph, we show only the precision of the evaluations in 10-D. First, looking at the framework that searches for parameters, both scheduling hit the same precision, except in Gallagher's where starting the training at 5-D is much better. It seems faster that training first in 5-D when including parameters in the grammar. We observed a similar trend when looking at the framework that uses random parameters, but here, it is faster to train straight in 10-D. For these schedules, using lower dimension in the training seems more beneficial only if the grammar includes parameters. Overall, Attractive Sector, seems more sensitive to the choice of operators over parameters, while Gallagher's seems sensitive to parameter tuning.

B. Testing

In this subsection, we compared the performance of all generated algorithms by plotting the empirical cumulative distribution function (ECDF). We first compared all the cases for the grammar that includes parameters, and then we compared the other grammar. Finally, we compared the best schedule of each grammar against the traditional algorithms.

In the figs. 5-7, the x-axis represents the budget in logarithmic scale, being the maximum budget NFE=1e+6, represented as 5 in the graph. The y-axis represents the number of problems in the testing set. In this representation, algorithms with bigger area under the curve are considered better.

Fig. 5 shows the testing results for the grammar that includes parameters. As can be seen, the framework was unable to solve Attractive Sector and Gallagher's 101-me when trained straight in 10D. This confirms our previous findings [5].

The training schedule that had the most consistent performance was 2,3,5-to-10D, supporting that a step-by-step increase in the dimension can improve the training.

Surprisingly, 2-to-10D had a very similar performance performance to 5-to-10D, since the leap from 2 to 10D is much higher than 5 to 10D. They performed similarly in Sphere and Gallagher's, but 2-to-10D could solve 100% the problems in Attractive sector while 5-to-10D could solve only about 60%.

Fig. 6 shows the testing results for the grammar without parameters. Due to the random parameters, it is important to recall that these algorithms had testing budget of NFE=1e+5 (each restart) with up to 10 restarts. It is again surprisingly that 2-to-10D performed better than straight in 10-D and 5-to-10D, but it supports the same results of the algorithms generated by grammar that includes parameters. Interestingly, the algorithms using random parameters could solve a fair amount of problems by the exchange of more budget, and in this case the 2-to-10D schedule was better probably by selecting components less sensitive to parameters.

Fig. 7 shows the testing results. As shown, DE is overall the best algorithm, being the the generated algorithm by the grammar with parameters on par with it.

Turning now to the statistical test, we performed the Wilcoxon Signed Rank Test of each training schedule against the baseline 10-D. There was a significant difference only for 2-to-10D ($p=0.03$). While not significant for 2,3,5-to-10D ($p=0.11$) and for 5-to-10D ($p=0.79$), the significance values may indicate that 2D was a good starting point for the training.

Lastly, we performed the Wilcoxon Signed Rank Test comparing the grammar with parameters against the grammar without parameters for each function presented in this paper. None of these differences were statistically significant. This suggests that part of the training budget has to be used to tune parameters, and how large this budget should be depend on the problem. Sphere and Gallagher's 101-me (both $p=0.12$) may suggest a lower budget for parameter tuning compared to Attractive Sector ($p=0.81$).

VII. Conclusion

In this study, we investigated three training schedules in which the problems start easy and get harder over time. Also, we determined whether the numerical parameters should be part of the grammar or not. The results indicate that it is faster to train straight in 10-D in the absence of parameters in the grammar than other training approaches. However, this method may hit lower precision if the problem is sensitive to parameter tuning (e.g., Gallagher's).

The training results also indicate that including parameters in the grammar leads to more robust algorithms as a result of losing less precision when the problem gets harder. In addition, the results using the 2,3,5-to-10D schedule suggest that increasing step-by-step the dimension leads to more robustness in terms of precision.

The testing results of the 2,3,5-to-10D schedule indicate that this approach generates algorithms that can consistently solve

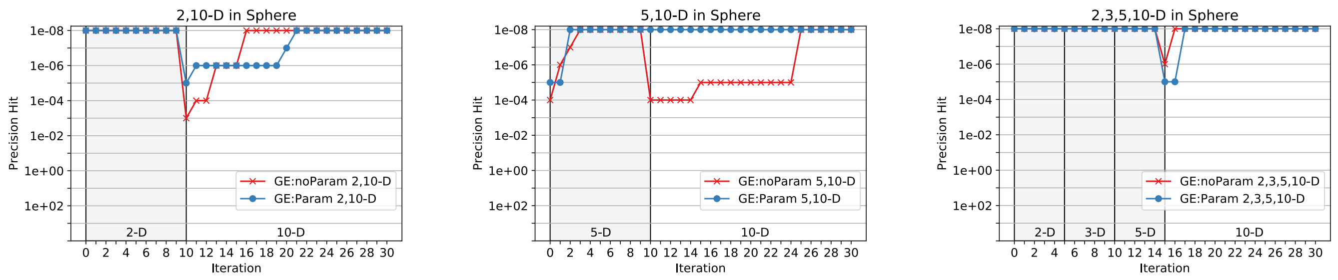

Fig. 1. Comparison between pairs of grammar-training schedules. The best pairs are the ones with higher precision at earlier iterations in 10-D.

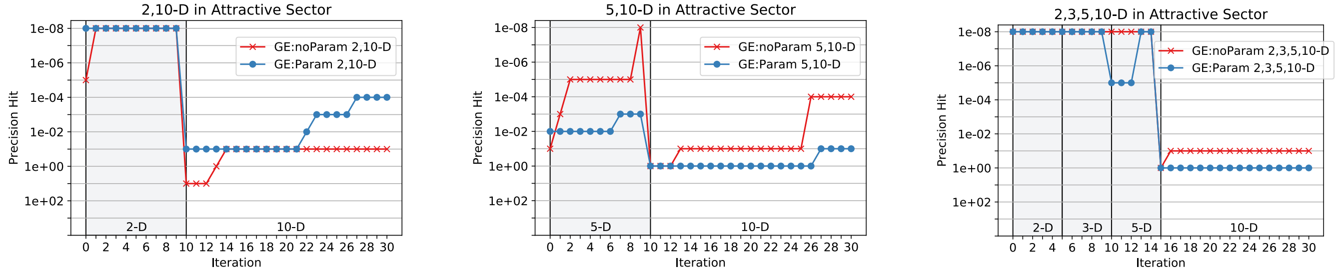

Fig. 2. Comparison between pairs of grammar-training schedules. The best pairs are the ones with higher precision at earlier iterations in 10-D.

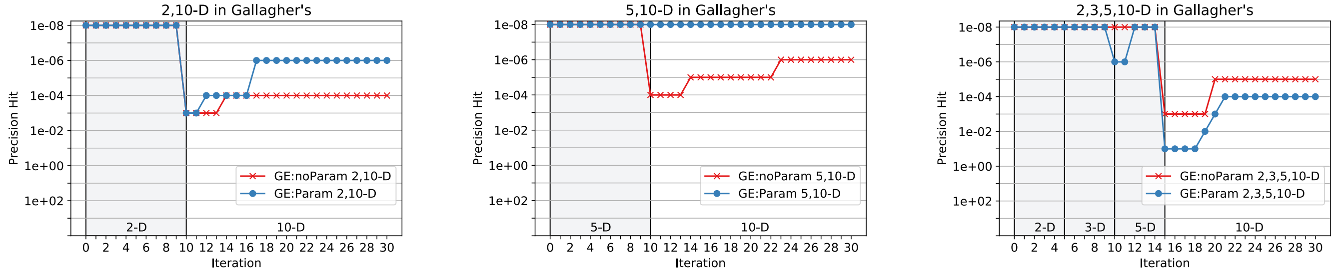

Fig. 3. Comparison between pairs of grammar-training schedules. The best pairs are the ones with higher precision at earlier iterations in 10-D.

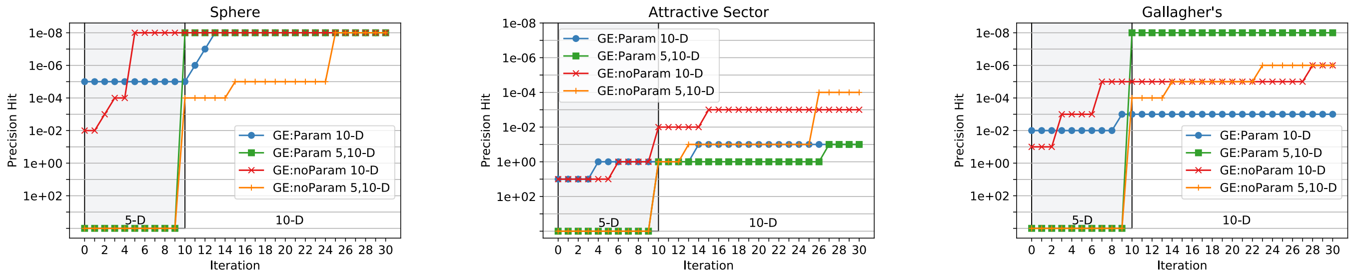

Fig. 4. Comparison between both grammars, each trained using the schedule 5,10-D and straight in 10-D. Only evaluations in 10-D are shown.

more testing instances. A more granular increase in difficulty may lead to even better results.

Some limitations are present in this study. First, the number of variation operators is limited. We did not use operators that behave well in a highly multimodal landscape, as revealed by our results in Rastrigin. Second, we trained and tested only on four benchmark functions. These limitations should be addressed in future works by adding more operators from other meta-heuristics, and by using more functions to assess the generalizability of the investigated methods better.

There are two questions raised by this study that need to be investigated. First, because we observed that it is effective to increase the dimension during the training gradually, the natural progression of this work is to analyze a good trade-off between training in easier problems and the performance of the generated algorithms. Second, because we also observed that the generated algorithms from the grammar without parameters took more budget but could a part of the problems in the testing set, the goal then is how to better divide the training resources between the search for components and the search for parameters.

REFERENCES

[1] M. Mitchell, *An introduction to genetic algorithms.* MIT press, 1998.

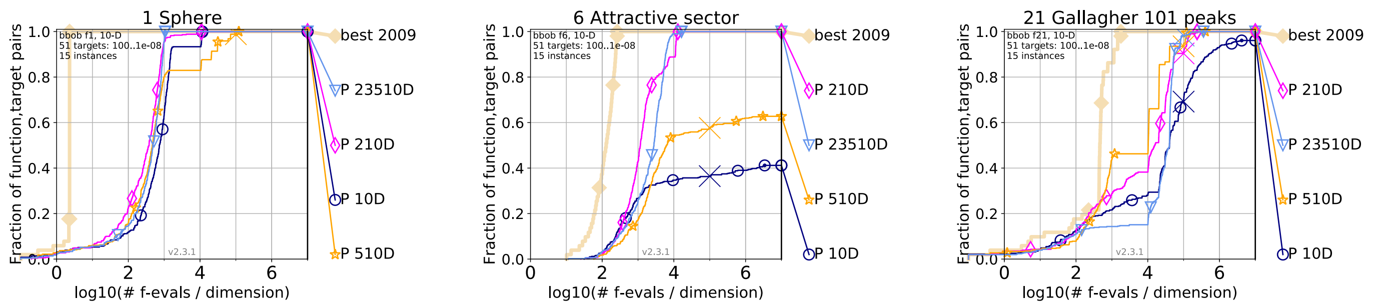

Fig. 5. ECDF of the algorithms generated by grammar that includes parameters.

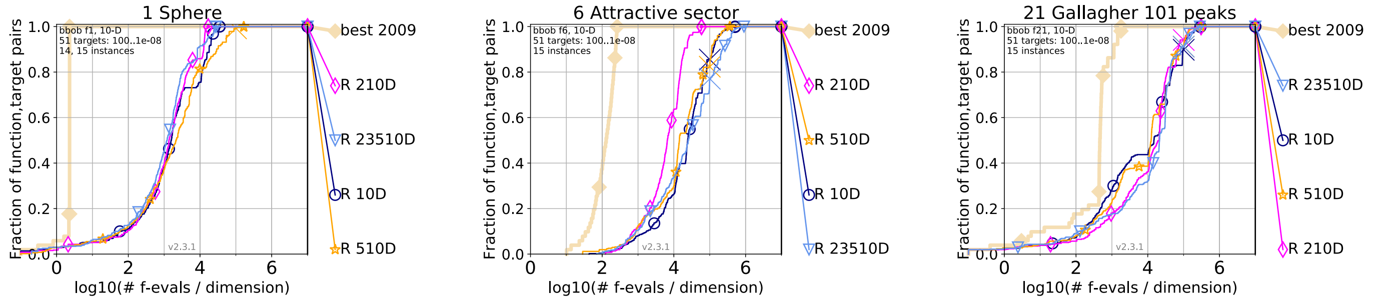

Fig. 6. ECDF of the algorithms generated by grammar that does not include parameters.

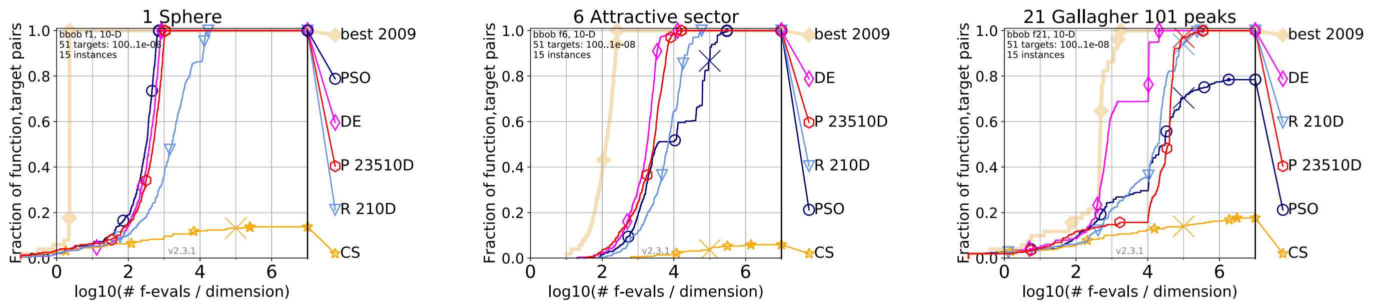

Fig. 7. ECDF of the traditional algorithms and of the generated algorithms for both grammars on the best training scheduling.

[2] R. Storn and K. 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, 1997.

[3] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95-International Conference on Neural Networks*, vol. 4, pp. 1942–1948, IEEE, 1995.

[4] X.-S. Yang and S. Deb, "Cuckoo search via lévy flights," in *2009 World congress on nature & biologically inspired computing (NaBIC)*, pp. 210–214, IEEE, 2009.

[5] A. Bogdanova, J. P. Junior, and C. Aranha, "Franken-swarm: grammatical evolution for the automatic generation of swarm-like metaheuristics," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pp. 411–412, 2019.

[6] N. Hansen, D. Brockhoff, O. Mersmann, T. Tusar, D. Tusar, O. A. ElHara, P. R. Sampaio, A. Atamna, K. Varelas, U. Batu, D. M. Nguyen, F. Matzner, and A. Auger, "COmparing Continuous Optimizers: numbbo/COCO on Github," Mar. 2019.

[7] P. Kerschke, H. H. Hoos, F. Neumann, and H. Trautmann, "Automated algorithm selection: Survey and perspectives," *Evolutionary computation*, vol. 27, no. 1, pp. 3–45, 2019.

[8] E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. R. Woodward, "A classification of hyper-heuristic approaches," in *Handbook of metaheuristics*, pp. 449–468, Springer, 2010.

[9] B. W. Goldman and D. R. Tauritz, "Self-configuring crossover," in *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation*, pp. 575–582, 2011.

[10] J. R. Woodward and J. Swan, "The automatic generation of mutation operators for genetic algorithms," in *Proceedings of the 14th annual conference companion on Genetic and evolutionary computation*, pp. 67–74, 2012.

[11] M. A. Martin and D. R. Tauritz, "Evolving black-box search algorithms employing genetic programming," in *Proceedings of the 15th annual conference companion on Genetic and evolutionary computation*, pp. 1497–1504, 2013.

[12] P. B. Miranda and R. B. Prudêncio, "Gefpso: A framework for pso optimization based on grammatical evolution," in *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pp. 1087–1094, 2015.

[13] R. H. R. de Lima and A. T. R. Pozo, "A study on auto-configuration of multi-objective particle swarm optimization algorithm," in *2017 IEEE Congress on Evolutionary Computation (CEC)*, pp. 718–725, IEEE, 2017.

[14] P. B. Miranda and R. B. Prudêncio, "A novel context-free grammar for the generation of pso algorithms," *Natural Computing*, pp. 1–19, 2018.

[15] S. N. Richter, M. G. Schoen, and D. R. Tauritz, "Evolving mean-update selection methods for cma-es," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pp. 1513–1517, 2019.

[16] M. Lones, "Optimising optimisers with push gp," *arXiv preprint arXiv:1910.00945*, 2019.

[17] M. Fenton, J. McDermott, D. Fagan, S. Forstenlechner, E. Hemberg, and M. O'Neill, "Ponyge2: Grammatical evolution in python," in

Proceedings of the Genetic and Evolutionary Computation Conference Companion, pp. 1194–1201, 2017.
[18] S. Finck, N. Hansen, R. Ros, and A. Auger, "Real-parameter black-box optimization benchmarking 2010: Presentation of the noisy functions," tech. rep., Citeseer, 2010.
[19] M. López-Ibáñez, J. Dubois-Lacoste, L. P. Cáceres, M. Birattari, and T. Stützle, "The irace package: Iterated racing for automatic algorithm configuration," *Operations Research Perspectives*, vol. 3, pp. 43–58, 2016.
[20] M. Nicolau and A. Agapitos, "Understanding grammatical evolution: Grammar design," in *Handbook of Grammatical Evolution*, pp. 23–53, Springer, 2018.