Franken-Swarm: Grammatical Evolution for the Automatic Generation of Swarm-like Meta-heuristics

Anna Bogdanova
University of Tsukuba
Tsukuba, Japan
ann.bogdanova@gmail.com

Jair Pereira Junior
University of Tsukuba
Tsukuba, Japan
jjunior013@gmail.com

Claus Aranha
Tsukuba University
Tsukuba, Japan
caranha@cs.tsukuba.ac.jp

ABSTRACT

In the last 20 years, literally dozens of optimization algorithms based on swarm intelligence have been proposed. Particle Swarm Optimization, Artificial Bee Colony, Cuckoo Search, Firefly Optimization, and Cat Swarm Optimization are just a small sample of the exuberance of swarm-like algorithms. Although they differ in implementation details, they all share a common structure: an update rule is applied to each solution, followed by a drop rule that decides whether to keep the updated solution or not. In this poster we explore the idea of automatically generating swarm-like optimizers. Our proposal is divided in two stages: First we decompose popular, human-crafted, swarm-like optimizers such as PSO, CS, ABC (as well as DE/GA) into a list of basic rules. Second, we use Grammatical Evolution to procedurally generate variations on this base structure by recombining these operators. We generate three instances of algorithms, and observe that they have comparable performance to DE and PSO. Our framework will be useful to gain insight on the design space of meta-heuristics and the nature of swarm-like algorithms.

CCS CONCEPTS

• **Theory of computation → Genetic programming**; • **Software and its engineering → Search-based soft. engineering**;

KEYWORDS

Grammatical Evolution, Swarm Intelligence, Automated Design of Algorithms, Procedural Generation

ACM Reference Format:
Anna Bogdanova, Jair Pereira Junior, and Claus Aranha. 2019. Franken-Swarm: Grammatical Evolution for the Automatic Generation of Swarm-like Meta-heuristics. In *Genetic and Evolutionary Computation Conference Companion (GECCO '19 Companion), July 13–17, 2019, Prague, Czech Republic.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3319619.3321902

1 MOTIVATION

There is a vast number of variations on the design of population-based search meta-heuristics. A common design pattern for these algorithms is to apply update operators on a set of candidate solutions, followed by selection rules to decide whether to keep the updated solution or not. The choice and design of such operators can sometimes feel more like an art than a science, as dozens of algorithms claim to use rules inspired by natural processes [2].

We are interested in exploring the design space of this type of meta-heuristics. We want to find out whether an automated composition framework, given a set of components extracted from common meta-heuristics, could re-create the original methods, or generate completely new designs. The answers to this question would be able to reinforce known rules-of-thumb about meta-heuristic design, and show areas that have been so far unexplored. To answer these questions, we must first find out a good base template, and a representative set of operators to create the algorithm design space. This poster is our first stab at this work.

1.1 Related Work

There is a growing interest in the automated design of metaheuristics. For example, Bezerra et al. [1] describe a framework that automatically chooses components to assemble a MOGA. This work treats the selection of components as a problem of meta-parameter selection, and uses iRACE to select the MOGA components. Miranda et al. [5] describes PSO using a formal grammar. Using grammatical evolution, they constructed PSO variations, optimizing velocity equations but keeping firmly within the PSO framework.

2 PROPOSED MODEL

We formulate an abstract framework that describes an optimization meta-heuristic in general terms, (Algorithm 1) and then use Grammatical Evolution to realize instances of this framework as procedurally generated algorithms. The Grammatical Evolution part is implemented using the PonyGE library [3].

Our abstract meta-heuristic begins by generating a set of random solutions to a real-valued optimization problem. At every iteration, a set of *update procedures* are applied to each solution in turn.

Each update procedure is defined by a *mutation operator*, followed by a *crossover operator*, and followed by a *replacement operator*. In this context, the mutation and crossover operators modify the original solution, while the replacement operator determines whether the modified solution will replace the original solution or not. Table 1 lists the components in the current prototype.

Finally, at the end of each iteration, a *drop procedure* is applied on the entire solution set. The drop procedure determines whether each solution in the set should be "dropped" and replaced with an entirely random new solution.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
GECCO '19 Companion, July 13–17, 2019, Prague, Czech Republic
© 2019 Copyright held by the owner/author(s). Publication rights licensed to the Association for Computing Machinery.
ACM ISBN 978-1-4503-6748-6/19/07…$15.00
https://doi.org/10.1145/3319619.3321902

Algorithm 1 Abstract Meta-heuristic Framework

```
1: Select Operators using Grammatical Evolution
2: Initialize population as set of random solutions S
3: while Not reached maximum number of iterations do
4:     for Each Solution s ∈ S do
5:         for Each Update Procedure p ∈ P do
6:             Use Operator Mut_p to generate s' from s
7:             Use Operator Cross_p to generate s'' from s'
8:             Use Operator Replace_p to choose between s'' and s
9:     Drop Op. replaces some solutions in S with random ones.
```

Component Type	Component List
Mutation Operator	Levy Mutation
	PSO Velocity
	DE Mutation
Crossover Operator	Exponential Crossover
	Blend Crossover
	None
Replacement Operator	Replace Always
	Replace if Better
	Replace if Better (CS version)
Drop Operator	*None*
	Fixed Probability
	Drop Worst

Table 1: Set of operators used to instantiate the abstract meta-heuristic.

3 EXPERIMENTAL ANALYSIS

Using the proposed framework, we created 5 instances of meta-heuristics for testing. First we re-created traditional DE and PSO using the proposed Abstract Meta-heuristic model. Then we created three original algorithms using Grammatical Evolution. The three algorithms were created using different ways to evaluate their fitness in the GE: *Franken10* used the Ackley function on D50 for 10 generations, *Franken30* used Ackley on D50 for 30 generations, and *Franken KA* used the Katsuura function (F23) on D40 for 10 generations [1]. The short evaluation time for each algorithm instance allows us to evaluate a larger number of them during GE.

The description of the five instances is on table 2. The source code for the framework, as well as the implementation instances and the experiments are available online [2]. From table 2 we can see that the three instances obtained have strong similarities: All of them prefer the greedy "Replace if Better" selection, and the guided DE and PSO mutation operators. This may reflect the short training period used. Also, the drop operator, present in algorithms such as CS and ABC, was not used often. We compared the performance of these instances on functions 20, 21, 22 and 24 of the BBOB library [4], and the GE instances showed similar performance to the handcrafted instances, which is quite encouraging for an early result.

4 DISCUSSION

Our results show that the training regimen and choice of fitness function should be a priority for the continuation of this work. Our early results are encouraging, and we plan to add further operators and reproduce other classical methods in this framework as well.

[1]Note that "generation" here is the evaluation time of each meta-heuristic generated by the grammar. The Grammatical Evolution itself ran for 30 generations in all cases.
[2]https://github.com/jair-pereira/GECCO19/tree/GECCO_Poster

Name	Operators
DE	*Proc1*: DE Mut, Xover: Exponential, Replace if Better
	Drop: *None*
PSO	*Proc1*: PSO Velocity, Xover: *None*, Replace Always
	Drop: *None*
Franken30	*Proc1*: PSO Velocity, Xover: *None*, Replace If Better
	Drop: Fixed Probability
Franken10	*Proc1*: DE Mut, Xover: Expo., Replace If Better (CS)
	Proc2: PSO Velocity, Xover: *None*, Replace Always
	Drop: *None*
Franken Ka	*Proc1*: DE mut, Xover: *None*, Replace if Better
	Proc2: PSO Vel., Xover: *None*, Replace if Better (CS)
	Drop: *None*

Table 2: Algorithm instances generated in this work. PSO and DE were hand-crafted, the others genrated by GE.

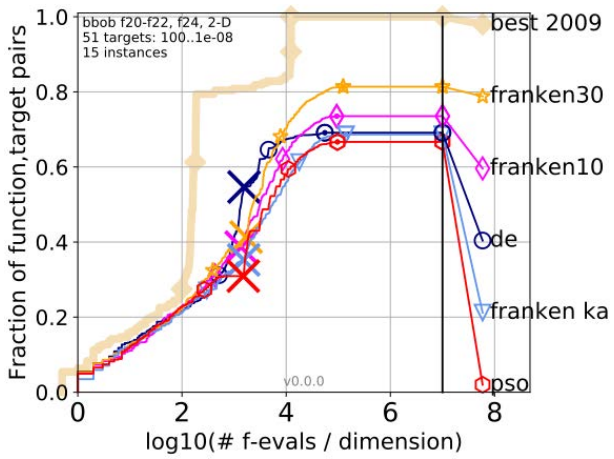

Figure 1: Preliminary results of the procedurally generated meta-heuristics: Franken30, Franken10 and FrankenKa.

REFERENCES

[1] Leonardo Bezerra, Manuel López-Ibañez, and Thomas Stützle. 2016. Automatic Component-Wise Design of Multi-Objective Evolutionary Algorithms. *IEEE Trans. Evolutionary Computation* 20, 3 (2016), 403–417.
[2] Felipe Campelo and Claus Aranha. 2018. EC Bestiary: A bestiary of evolutionary, swarm and other metaphor-based algorithms. (June 2018).
[3] Michael Fenton, James McDermott, David Fagan, Stefan Forstenlechner, Erik Hemberg, and Michael O'Neill. 2017. PonyGE2: Grammatical Evolution in Python. In *Proc. of the Genetic and Evolutionary Computation Conference Companion*. ACM, New York, NY, USA, 1194–1201.
[4] N. Hansen, S. Fink, R. Ros, and A. Auger. 2009. *Real-parameter black-box optimization benchmarking 2009: Noiseless functions definitions.* Technical Report RR-6829. Inria. http://coco.lri.fr/downloads/download15.01/bbobdocfunctions.pdf Updated February 2010.
[5] Péricles B. C. Miranda and Ricardo B. C. Prudêncio. 2018. A novel context-free grammar for the generation of PSO algorithms. *Natural Computing* (mar 2018). https://doi.org/10.1007/s11047-018-9679-9