# Evolving Mean-Update Selection Methods for CMA-ES


## CCS CONCEPTS

• **Computing methodologies** → **Genetic programming**; • **Theory of computation** → *Design and analysis of algorithms*; • **Software and its engineering** → *Genetic programming*;

## KEYWORDS

Selection, Genetic Programming, Hyper-heuristic, CMA-ES

**ACM Reference Format:**
. 2019. Evolving Mean-Update Selection Methods for CMA-ES. In *Proceedings of the Genetic and Evolutionary Computation Conference 2019 (GECCO '19).* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn


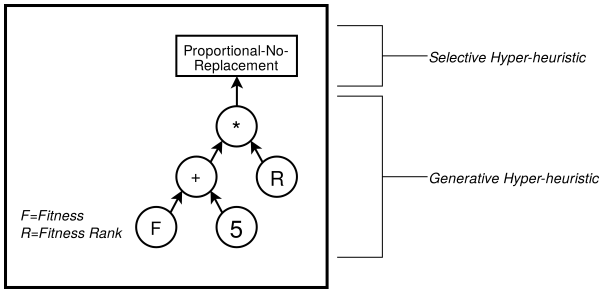


**Figure 1: Example of a generated selection function.**

## 1 INTRODUCTION

EAs employ selection functions to control the method by which an individual's genes are selected, for purposes such as, recombination, survival, or updating internal variables. Each selection algorithm plays a significant role in determining the behavior of the EA and the population, and thus, the average performance of the EA's search through the space of solutions [**?** ].

New selection algorithms can be designed in cases where the performance offered by existing algorithms is insufficient, even with well-tuned parameters. However, the full space of selection algorithms is only limited by the maximum algorithm size, and so it is highly unlikely that any conventionally human-designed algorithm offers the optimal selection behavior, given a specific problem. A performance gain is likely to be attained by exploring the space of selection algorithms to find one that offers better performance than any conventional selection algorithm. Previous work has confirmed this hypothesis, prompting our approach to use a hyper-heuristic, with both generative and perturbative elements, to explore the space of new selection functions [**?** ].

## 2 METHODOLOGY

### 2.1 Encoding Selection Functions

We developed a generalized format to represent a selection function, which can encode both a number of traditional selection functions as well as novel selection functions. The representation consists of two major parts. The first part is a binary Koza-style GP-Tree [**?** ]. Tthe GP-Tree encodes a mathematical function. All of the function inputs (the terminals of the GP-Tree) are real-valued numbers, and all of the operators in the GP-Tree operate on, and return, real-valued numbers. The terminals of the GP-Tree include various factors pertinent to a single individual of the population, including the individual's fitness, the individual's fitness ranking among the population members, the uniqueness of the individual's genome, and the individual's age, in generations. The possible terminal inputs also include information pertinent to the evolution at large, including the total size of the population, the current generation, the maximum and minimum fitness values in the population, and the sum of the individuals' fitness values. Constants are also included, as well as random terminals, which return a random number within a (configurable) closed range. Binary operators in the GP-Tree include various arithmetic and other mathematic functions. When evaluated, the mathematical function encoded by the GP-Tree returns a single real-valued number, corresponding to the relative "desirability" of the individual whose data was input into the function. This GP-Tree is built by the generative part of the hyper-heuristic, and can encompass any valid parse tree built from the available operators and terminals.

The second part of the evolved selection function is a method of selecting individuals based on their desirabilities, as calculated by the mathematical function encoded by the GP-Tree. The possible selection methods are inspired by traditional selection functions, such as truncation, tournament selection, fitness-proportional selection, and stochastic universal sampling. Some selection methods will select with replacement, allowing a single individual to be selected more than once per generation. This component is the part of the selection function built by the perturbative hyper-heuristic, being exactly one choice from a set of pre-determined methods.

To perform selection on a population, the function encoded by the GP-Tree is evaluated once for each member of the population, using the data points for that individual (fitness value, fitness ranking, etc.) as inputs to the function. The number output by the function becomes the desirability score for each individual. Finally, the selection step is used to select individuals based on the individuals' desirability scores. The selected individuals can then be used for recombination, as the survivors for the next generation, or for any other update to the internal variables that depends on a chosen subset of the population, as pertinent to the evolutionary search strategy used.

An example of this representation is shown in Figure 1.



*GECCO '19, July 13–17, 2019, Prague, Czech Republic*



ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

## 2.2 Search Methodology

We use a meta-EA to develop the selection functions, treating each complete selection function as a member of a higher-order population. After generating an initial pool of randomly constructed selection functions, the quality of each complete selection function is determined, and well-performing selection functions are chosen to recombine and mutate into new candidate selection functions. The size of the trees is constrained using parsimony pressure.

To test our methodology, we target the CMA-ES algorithm for improvement, evolving a new selection function for the mean-update algorithm.

The quality of each selection function is determined by running CMA-ES on a suite of static training instances from a benchmark problem class. The benchmark problem classes used for the underlying EA are selected from the Comparing Continuous Optimizers (COCO) platform used for the GECCO Workshops on Real-Parameter Black-Box Optimization Benchmarking [**?** ]. The COCO platform provides 24 unique benchmark problems. Each problem is offered in multiple dimensions, and for each dimension, multiple problem instances are present. We select 11 of the 24 problem classes, and use dimensions 2, 3, 5, and 10. For each class and dimension, we use some instances for the meta-EA, and set aside the rest of the instances to test for generalization. The problem classes were selected by running unodified CMA-S on each of the 24 problem classes, selecting those where it failed to solve some dimension $D \leq 10$ on at least half of the trial runs.

# 3 RESULTS AND DISCUSSION

For the function classes 4 and 19, the success rate of CMA-ES increased by 20-30 percent when modified with the evolved selection function at $D = 2$, but performed similarly to the unmodified CMA-ES at other dimensionalities. For function classes 20 and 21, a performance increase is seen on dimensionalities $D = 2, 3$, and 5, but not $D = 10$; curiously, the modified CMA-ES performs worse on function class 21 when $D = 10$. For function classes 6 and 12, performance is similar for $D = 2, 3$, and 5, but for $D = 10$, there is a significant performance increase: on function 6, the success rate increased from 0 percent to around 96 percent, and on function class 12, the success rate increased from 18-67 percent, varying across the function instances, to 100 percent for all function instances.

For the function classes 3, 15, 16, 23, and 24, there was no major difference between the success rate of the modified and unmodified CMA-ES.

## 3.1 Discussion of Results

We observed that evolving a new selection function for CMA-ES increased its solution quality on 6 of the 11 functions tested. In particular, we observed two cases with high dramatic improvements: the tests for COCO function classes 6 and 12 for dimensionality $D = 10$. In the case of function class 6, the unmodified CMA-ES was unable to solve any of the testing instances, but the CMA-ES using the evolved selection solved these instances nearly 100% of the time. In the case of function class 12, the success rate of the unmodified CMA-ES varied strongly between instances, ranging from 18 to 67%, with an average success rate of 44.05% across all instances. The modified CMA-ES, however, solved every test instance, reaching the global best fitness in each of 200 runs, on each of the testing instances, achieving a 100% success rate. By observing the increases in success rate for some of the functions, it is clear that evolving a new selection scheme for CMA-ES provides a substantial benefit in some cases.

The five cases where no improvement was observed involved functions that were highly multimodal. Three of these function were variants of the Rastrigin function, a highly multimodal function and the other two—the Weierstrass Function and the Katsuura Function—are highly rugged. It is likely, in these cases, that CMA-ES requires some other improvement aside from a new selection scheme to better learn and traverse the global structures of these functions. Because we only replaced the selection scheme of CMA-ES, we only changed how it internally updates the mean. Improving the performance of CMA-ES on these functions likely requires more intelligent updating of the other internal variables of CMA-ES, such as the evolution paths, the covariance matrix, and the step size.

$F = 21$, $D = 10$ is the one case tested where CMA-ES performed markedly worse than unmodified CMA-ES. This effect is likely due to overspecialization to the set of training instances.

# 4 CONCLUSIONS

We hypothesized that a hyper-heuristic search through the space of selection functions for EAs could improve the performance of an EA on a particular problem class by discovering a specialized selection function. We developed a representation of selection functions that uses a Koza-style GP-Tree to relate an individual's fitness value and fitness ranking to its relative probability of selection, and used a meta-EA to search through the space of selection functions in this representation.

With this meta-EA, we have shown that it is possible to generate new selection functions, tuned to a particular benchmark problem, that can enable an EA to significantly outperform conventional selection functions on those problems. Thus, we show that, in order to discover the optimum selection method for an EA operating on a particular problem, it may not be sufficient to use any of the static conventional selection functions tested. We have also shown that, in some cases, this performance increase from a custom selection algorithm will generalize to similar problems in the same problem class. Therefore, if one expects to run the same EA on many problems from the same problem class, one might expect to gain a performance increase by doing some *a priori* calculation to develop a specialized selection algorithm trained on instances of that problem class, which would then enable an EA utilizing that selection function to perform better on other instances of that problem class. However, our experiments have also shown that, for certain functions, replacing only the selection function may not yield significant performance improvements, depending on the behavior of the search strategy and the nature of the function being optimized by the EA. Careful consideration must be given to determine what the effect of tuning the selection scheme of a given EA will be on the performance of that EA, and whether such tuning will cause the EA to have an appreciable performance increase on the problem class in question.