Evolving Mean-Update Selection Methods for CMA-ES

Samuel N. Richter
Missouri University of Science and Technology
Natural Computation Lab
Rolla, Missouri, U. S. A.
snr359@mst.edu

Michael G. Schoen
Missouri University of Science and Technology
Natural Computation Lab
Rolla, Missouri, U. S. A.
ms778@mst.edu

Daniel R. Tauritz
Missouri University of Science and Technology
Natural Computation Lab
Rolla, Missouri, U. S. A.
dtauritz@acm.org

ABSTRACT

This paper details an investigation of the extent to which performance can be improved for the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) by tuning the selection of individuals used for the mean-update algorithm. A hyper-heuristic is employed to explore the space of algorithms which select individuals from the population. We show the increase in performance obtained with a tuned selection algorithm, versus the unmodified CMA-ES mean-update algorithm. Specifically, we measure performance on instances from several real-valued benchmark function classes to demonstrate generalization of the improved performance.

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:
Samuel N. Richter, Michael G. Schoen, and Daniel R. Tauritz. 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, 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Evolutionary Algorithms (EA)s 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. 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 effectively unlimited, 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 to explore the space of new selection functions [30].

Our approach employs a hyper-heuristic to explore the space of selection algorithms, with each search algorithm represented by two components. The first component is a Koza-style Genetic Programming (GP) tree [17], encoding a mathematical function that calculates how desirable an individual is at the current stage of evolution. The second component is a method of selecting individuals, based on how desirable they are calculated to be. We use this hyper-heuristic to evolve a new scheme for selecting individuals in the mean update function of the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [11]. The canonical CMA-ES has shown great success in solving a wide variety of problems, and given the improvement to a canonical EA via evolution of a new selection process shown in [25], we sought to apply the same process to CMA-ES, anticipating a similar improvement.

2 RELATED WORK

The field of hyper-heuristics encompasses many different approaches for the automated design of new algorithms. Methods may utilize offline learning, in which computation is done *a priori* to develop a heuristic, or online learning, in which a heuristic is developed dynamically alongside a running problem. The hyper-heuristic presented in this paper is an offline-learning heuristic that builds a new selection function tuned to an EA solving problem instances from a particular problem class.

A major application of hyper-heuristics is the automated design of algorithmic components. Hyper-heuristics have been used to evolve new algorithms from components of existing algorithms for Ant Colony optimization algorithms [19], Boolean Satisfiability solvers [16], local search heuristics [4], and iterative parse trees representing Black Box Search Algorithms [21].

Previous work has also focused on improvement of targeted components of EAs, including the evolution of new mutation operators [13, 31], mating preferences [9], genetic representation of individuals [26], and crossover operators [8]. Methods for generating selection algorithms, in particular, have been investigated. A random walk through the space of register machines that compute and return a probability of selection for each individual showed that such custom-tuned selection algorithms can outperform typical selection algorithms [30]. A more informed search through the space of selection algorithms may yield an even greater benefit than a random search. In the previous work involving the evolution of Black Box Search Algorithms, the parse trees include evolved selection functions, although the selection functions are limited to two conventional selection functions (k-tournament and truncation) with

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

evolved parameters [21]. An evolutionary search through selection functions developed with Grammatical Evolution showed that better selection functions can be developed using a hyper-heuristic, and that the performance of these selection functions can generalize to new instances within the same function class [20]. A similar hyper-heuristic search employed GP to discover new selection algorithms, again showing a generalized increase in performance for the same EA running on the same problem set [25]. The work described in this paper expands on these ideas by applying the hyper-heuristic search employing GP to the mean-update function of CMA-ES, changing the method used to select the population members used to update the mean in order to tune the performance of CMA-ES for a particular function class.

3 METHODOLOGY

Here we discuss the methodology of our hyper-heuristic, and the meta-EA powering it.

3.1 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 [17]. Rather than encoding entire programs within the GP-Tree, which could result in an infeasibly wide search space of selection algorithms [29], the GP-Tree instead 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.

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. Some selection methods will select with replacement, allowing a single individual to be selected more than once per generation.

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.

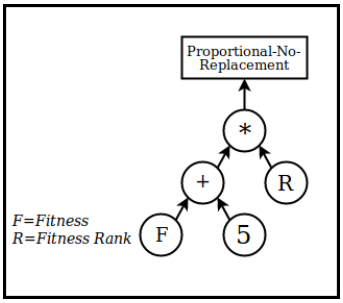

Figure 1: Example of a generated selection function.

An example of this representation is shown in Figure 1. It shows an example of a GP-Tree that represents the function evaluated for each individual, as well as a final selection method used. With this selection function, the desirability of any individual is calculated as the individual's fitness rating plus 5, multiplied by the individual's ranking in the population ordered by fitness. The selection method used is Proportional-No-Replacement, so the probability of any individual being selected is directly proportional to its desirability score, and an individual cannot be selected more than once. See [24] for more information on this particular representation of selection functions.

3.2 Search Methodology

We use a meta-EA to develop the selection functions described in Section 3.1, treating each complete selection function as a member of a higher-order population. The quality of each selection function is determined by running CMA-ES on a suite of static training instances from a benchmark function class. The underlying EA utilizes the selection function in question, and keeps all other parameters constant. The performance of the underlying EA is used to determine the quality of a selection function; selection functions that enable the EA to perform better, with all other parameters constant, are considered to be "higher-quality" selection functions. The size of the GP-Trees is constrained using parsimony pressure.

When the meta-EA concludes, the CMA-ES utilizing the best selection function from the meta-EA is run on a set of separate testing instances from the same function class to test the generalization of the selection function's performance. If modified CMA-ES performs significantly better on the testing instances than the same EA using a standard selection function, then we can say that the evolved selection function successfully generalized to the function class of interest.

The benchmark function 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 [10]. This benchmark set provides a suite of real-valued optimization problems. Each function class

is offered in multiple dimensionalities, and for each dimension, multiple unique instances of the function class are present. By using several different instances of a given problem for a given dimensionality, we can test whether an increase in performance offered by a higher-quality selection function generalizes to other instances within the function class. We do this by using only some of the instances during the meta-EA to evaluate the quality of selection functions, then run CMA-ES utilizing the evolved selection function on the unused, unseen instances.

4 EXPERIMENTAL SETUP

To test our methodology, we target the CMA-ES algorithm for improvement. We use the meta-EA to evolve a new method of selecting which of the sampled points to use for recalculating the search space mean. The individuals of the meta-EA each encode a single selection function in their genome, which modifies the mean-update functionality of CMA-ES. Rather than selecting the μ highest-fitness points, the encoded selection function selects a subset of all the sampled points, which are then ordered by fitness and used to update the mean. Once the mean is updated, all other state variables, such as the covariance matrix, are updated using the same methods as the unmodified CMA-ES.

To select benchmark functions for the CMA-ES, we use the 24 noiseless function classes in the COCO dataset. We use dimensionalities of 2, 3, 5, and 10, excluding 20 and 40 to save computational resources. The meta-EA is run separately for each combination of function class and dimensionality, using 6 of the instances for that function class and dimensionality to train the meta-EA and the rest of the available instances for testing generalization of the evolved selection function. For the parameters of CMA-ES, we use $\lambda = 10 \times D$, $\mu = \lambda/2$, and $\sigma_i = 0.5$. The CMA-ES terminates on population convergence, after $10000 \times D$ evaluations, or after finding the solution to a function instance, as specified by the COCO benchmark platform.

When evaluating the performance of an evolved selection function to assign a fitness value to it, the fitness is taken as the proportion of the runs in which the modified CMA-ES reaches the global optimum, or moves close enough to it to meet the criteria to solve the function.

For each function instance in the testing set, we run the CMA-ES for 200 testing runs, both using the evolved selection function and unmodified. We then measure, for each testing instance, the proportion of runs which solved the function, in each case.

5 RESULTS AND DISCUSSION

For each testing instance, the percentage of runs solved by the modified and unmodified CMA-ES is shown in Table 1.

For the function classes 4, 6, 12, 17, 18, 19, 20, and 21, the CMA-ES modified with the evolved selection function solved the function instances at least 20 percent more often than the unmodified CMA-ES for at least one dimensionality. 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 17 and 18, performance increases are seen for $D = 2$ and $D = 3$, with negligible performance differences on the other dimensionalities. Function class 22 sees about a 12 percent increase in solution rate on $D = 2$, but negligible improvement for other dimensionalities. 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 other function classes, there was no major difference between the success rate of the modified and unmodified CMA-ES. In a few cases, the modified CMA-ES performs marginally worse than the unmodified CMA-ES, but the difference in solution rate usually no more than 5 percent, and only 7.4 percent at most.

5.1 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 $D = 10$. The five cases where no improvement was observed involved functions that were highly multimodal. It is likely, in these cases, that CMA-ES requires some other improvement aside from a new mean-update scheme to better learn and traverse the global structures of these functions.

6 CONCLUSIONS

We hypothesized that a search through the space of selection functions could improve the performance of CMA-ES on a particular problem class by discovering a specialized selection function with which to modify the mean-update function of CMA-ES. We developed a representation of selection functions that utilizes a GP-Tree and selection method 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 function, that can enable CMA-ES to significantly outperform conventional selection functions on those functions. We have also shown that this performance increase from a custom selection algorithm will generalize to similar functions in the same function class. Therefore, if one expects to run CMA-ES, or any EA on many functions from the same function 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 function 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.

6.1 Future Work

The work presented in this paper opens a number of potential avenues for future research. Of primary concern is the fact that the meta-EA presented in this paper requires a large amount of *a priori* computation to generate a high-quality selection function.

Table 1: Percentage of Runs Solved By Modified CMA-ES/Unmodified CMA-ES, averaged over all instances

COCO Function Class	D=2	D=3	D=5	D=10
1	100.0% / 100.0%	100.0% / 100.0%	100.0% / 100.0%	100.0% / 100.0%
2	100.0% / 100.0%	100.0% / 100.0%	100.0% / 100.0%	100.0% / 100.0%
3	35.65% / 34.2%	22.0% / 21.35%	7.6% / 6.6%	1.2% / 1.35%
4	32.15% / 3.3%	0.15% / 0.25%	0.0% / 0.0%	0.0% / 0.0%
5	100.0% / 100.0%	100.0% / 100.0%	100.0% / 100.0%	100.0% / 100.0%
6	100.0% / 100.0%	100.0% / 100.0%	99.1% / 100.0%	96.0% / 0.0%
7	98.05% / 94.35%	100.0% / 97.65%	99.35% / 99.4%	100.0% / 99.95%
8	99.85% / 100.0%	100.0% / 100.0%	99.9% / 99.5%	99.8% / 100.0%
9	100.0% / 100.0%	100.0% / 100.0%	100.0% / 100.0%	100.0% / 100.0%
10	100.0% / 100.0%	100.0% / 100.0%	100.0% / 100.0%	100.0% / 100.0%
11	100.0% / 100.0%	100.0% / 100.0%	100.0% / 100.0%	100.0% / 100.0%
12	99.55% / 100.0%	99.5% / 100.0%	99.85% / 100.0%	100.0% / 44.05%
13	100.0% / 100.0%	100.0% / 100.0%	100.0% / 100.0%	100.0% / 100.0%
14	100.0% / 100.0%	100.0% / 100.0%	100.0% / 100.0%	100.0% / 100.0%
15	41.45% / 41.95%	32.85% / 32.45%	24.55% / 24.0%	11.95% / 16.85%
16	65.7% / 65.65%	59.95% / 64.0%	58.2% / 57.1%	40.05% / 39.7%
17	96.05% / 75.0%	96.15% / 81.9%	96.7% / 94.6%	99.2% / 99.2%
18	96.85% / 64.05%	97.6% / 79.25%	93.5% / 91.75%	93.9% / 94.9%
19	55.05% / 36.1%	20.1% / 20.9%	6.85% / 5.8%	6.6% / 0.45%
20	48.35% / 24.8%	19.35% / 12.15%	2.25% / 1.5%	0.0% / 0.0%
21	56.8% / 27.65%	55.4% / 28.7%	36.05% / 23.3%	28.6% / 36.0%
22	47.15% / 35.25%	25.8% / 27.85%	14.2% / 12.75%	5.95% / 0.4%
23	70.35% / 73.3%	54.7% / 52.3%	32.05% / 31.2%	7.35% / 8.05%
24	1.6% / 1.45%	0.1% / 0.3%	0.0% / 0.0%	0.0% / 0.0%

While this computational cost may be worth it for EAs that will run on functions from the same function class many times, a more efficient method of finding good selection functions has a much greater potential to benefit EAs in general. Exploring a method of online learning could allow for the elimination of the expensive *a priori* computation time, allowing specialized selection functions to be generated during the evolution.

The CMA-ES in our experiments was tuned to increase performance on the COCO benchmark function classes. A major next step for this work is to apply the meta-EA to real-world EAs that could benefit from new, specialized selection functions.

Because the objective of this paper is similar to the work done to develop selection algorithms via Grammatical Evolution [20] and register machines [30], it remains to be seen which cases each method is more effective for, and a direct comparison of the methods on the same benchmark functions may yield more insight into which offers better performance benefits under certain conditions.

The framework of CMA-ES used was fairly basic, lacking features such as restarts. In addition, many techniques have been developed to improve CMA-ES, such as in $(1 + 1)$-CMA-ES [14] and the Active CMA-ES [15]. Evolving a specialized selection function for these new forms of CMA-ES may lead to even greater performance gains, or may not even be necessary for particular problem classes. Additionally, we only made efforts to improve the mean-update step of CMA-ES, which allowed to it gain increased performance in some, but not all, of the problem cases tested. Further experiments to tune the methods used for updating the other internal variables, such as evolution path, covariance matrix, and step-size, may lead to greater performance in more problem classes.

The meta-EA parameters were manually tuned to allow for a high degree of exploration. A sensitivity analysis of these parameters, as well an investigation of parameter tuning/control, could lead to increased performance of the meta-EA.

REFERENCES

[1] Edmund K Burke, Michel Gendreau, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and Rong Qu. 2013. Hyper-heuristics: A Survey of the State of the Art. *Journal of the Operational Research Society* 64, 12 (2013), 1695–1724.

[2] Edmund K Burke, Matthew Hyde, Graham Kendall, and John Woodward. 2010. A Genetic Programming Hyper-heuristic Approach for Evolving 2-d Strip Packing Heuristics. *IEEE Transactions on Evolutionary Computation* 14, 6 (2010), 942–958.

[3] Edmund K Burke, Matthew R Hyde, and Graham Kendall. 2006. Evolving Bin Packing Heuristics with Genetic Programming. In *Parallel Problem Solving From Nature-ppsn Ix*. Springer, 860–869.

[4] Edmund K Burke, Matthew R Hyde, and Graham Kendall. 2012. Grammatical Evolution of Local Search Heuristics. *IEEE Transactions on Evolutionary Computation* 16, 3 (2012), 406–417.

[5] Edmund K Burke, Mathew R Hyde, Graham Kendall, Gabriela Ochoa, Ender Ozcan, and John R Woodward. 2009. Exploring Hyper-heuristic Methodologies with Genetic Programming. In *Computational Intelligence*. Springer, 177–201.

[6] Ágoston E Eiben, Robert Hinterding, and Zbigniew Michalewicz. 1999. Parameter Control in Evolutionary Algorithms. *IEEE Transactions on Evolutionary Computation* 3, 2 (1999), 124–141.

[7] Pablo Garrido and María Cristina Riff. 2010. DVRP: a Hard Dynamic Combinatorial Optimisation Problem Tackled By An Evolutionary Hyper-heuristic. *Journal of Heuristics* 16, 6 (2010), 795–834.

[8] Brian W Goldman and Daniel R Tauritz. 2011. Self-configuring Crossover. In *Proceedings of the 13th Annual Conference Companion on Genetic and Evolutionary Computation*. ACM, 575–582.

[9] Lisa M Guntly and Daniel R Tauritz. 2011. Learning Individual Mating Preferences. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*. ACM, 1069–1076.

[10] N. Hansen, A. Auger, O. Mersmann, T. Tušar, and D. Brockhoff. 2016. COCO: A Platform for Comparing Continuous Optimizers in a Black-Box Setting. *ArXiv e-prints* arXiv:1603.08785 (2016).

[11] Nikolaus Hansen and Andreas Ostermeier. 1996. Adapting Arbitrary Normal Mutation Distributions in Evolution Strategies: The Covariance Matrix Adaptation. In *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation*. IEEE, 312–317.

[12] Sean Harris, Travis Bueter, and Daniel R Tauritz. 2015. A Comparison of Genetic Programming Variants for Hyper-heuristics. In *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*. ACM, 1043–1050.

[13] Libin Hong, John Woodward, Jingpeng Li, and Ender Özcan. 2013. Automated Design of Probability Distributions as Mutation Operators for Evolutionary Programming Using Genetic Programming. In *European Conference on Genetic Programming*. Springer, 85–96.

[14] Christian Igel, Thorsten Suttorp, and Nikolaus Hansen. 2006. A Computational Efficient Covariance Matrix Update and a (1+ 1)-CMA For Evolution Strategies. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*. ACM, 453–460.

[15] Grahame A Jastrebski and Dirk V Arnold. 2006. Improving Evolution Strategies Through Active Covariance Matrix Adaptation. In *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*. IEEE, 2814–2821.

[16] Ashiqur R. KhudaBukhsh, Lin Xu, Holger H. Hoos, and Kevin Leyton-Brown. 2009. SATenstein: Automatically Building Local Search SAT Solvers from Components". In *Twenty-first International Joint Conference on Artificial Intelligence (IJCAI-09)*. 517–524.

[17] John R Koza. 1994. Genetic Programming as a Means for Programming Computers By Natural Selection. *Statistics and Computing* 4, 2 (1994), 87–112.

[18] Natalio Krasnogor and Steven Gustafson. 2004. A Study on the Use of "Self-Generation" in Memetic Algorithms. *Natural Computing* 3, 1 (2004), 53–76.

[19] Manuel Lopez-Ibanez and Thomas Stutzle. 2012. The Automatic Design of Multi-objective Ant Colony Optimization Algorithms. *IEEE Transactions on Evolutionary Computation* 16, 6 (2012), 861–875.

[20] Nuno Lourenço, Francisco Pereira, and Ernesto Costa. 2013. Learning Selection Strategies for Evolutionary Algorithms. In *International Conference on Artificial Evolution (evolution Artificielle)*. Springer, 197–208.

[21] Matthew A Martin and Daniel R Tauritz. 2013. Evolving Black-box Search Algorithms Employing Genetic Programming. In *Proceedings of the 15th Annual Conference Companion on Genetic and Evolutionary Computation*. ACM, 1497–1504.

[22] Jorge Maturana, Frédéric Lardeux, and Frédéric Saubion. 2010. Autonomous Operator Management for Evolutionary Algorithms. *Journal of Heuristics* 16, 6 (2010), 881–909.

[23] Stephen Remde, Peter Cowling, Keshav Dahal, Nic Colledge, and Evgeny Selensky. 2012. An Empirical Study of Hyperheuristics for Managing Very Large Sets of Low Level Heuristics. *Journal of the Operational Research Society* 63, 3 (2012), 392–405.

[24] Samuel Richter. 2019. *Evolved Parameterized Selection for Evolutionary Algorithms*. Master's thesis. Missouri University of Science and Technology.

[25] Samuel N Richter and Daniel R Tauritz. 2018. The Automated Design of Probabilistic Selection Methods for Evolutionary Algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. ACM, 1545–1552.

[26] Eric O Scott and Jeffrey K Bassett. 2015. Learning Genetic Representations for Classes of Real-valued Optimization Problems. In *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*. ACM, 1075–1082.

[27] David H Wolpert, William G Macready, et al. 1995. *No Free Lunch Theorems for Search*. Technical Report. Technical Report SFI-TR-95-02-010, Santa Fe Institute.

[28] John R Woodward. 2010. The Necessity of Meta Bias in Search Algorithms. In *Computational Intelligence and Software Engineering (cise), 2010 International Conference On*. IEEE, 1–4.

[29] John R Woodward and Ruibin Bai. 2009. Why Evolution Is Not a Good Paradigm for Program Induction: A Critique of Genetic Programming. In *Proceedings of the First Acm/sigevo Summit on Genetic and Evolutionary Computation*. ACM, 593–600.

[30] John Robert Woodward and Jerry Swan. 2011. Automatically Designing Selection Heuristics. In *Proceedings of the 13th Annual Conference Companion on Genetic and Evolutionary Computation*. ACM, 583–590.

[31] John R Woodward and Jerry Swan. 2012. The Automatic Generation of Mutation Operators for Genetic Algorithms. In *Proceedings of the 14th Annual Conference Companion on Genetic and Evolutionary Computation*. ACM, 67–74.