

1

Pyrimidine: An algebra-inspired Programming

2

framework for evolutionary algorithms

3




Congwei Song  <sup>1</sup>

4

<sup>1</sup> Beijing Institute of Mathematical Sciences and Applications, Beijing, China

DOI: [10.xxxxxx/draft](https://doi.org/10.xxxxxx/draft)

Software

- [Review](#) 
- [Repository](#) 
- [Archive](#) 

5

Pyrimidine: An algebra-inspired Programming framework for

6

evolutionary algorithms

7

Summary

Editor: [Sébastien Boisgérault](#) 

Reviewers:

- [@mmore500](#)
- [@sjvrijn](#)

Submitted: 11 December 2023

Published: unpublished

License

Authors of papers retain copyright  
and release the work under a  
Creative Commons Attribution 4.0  
International License ([CC BY 4.0](#)).

8

[Pyrimidine](#) stands as a versatile framework designed for genetic algorithms (GAs), offering

9

exceptional extensibility for a wide array of evolutionary algorithms.

10

Leveraging the principles of object-oriented programming (OOP) and the meta-programming,

11

we introduce a distinctive design paradigm coined as “algebra-inspired Programming”, signifying

12

the fusion of algebraic methodologies with the software architecture.

13

Statement of need

14

GAs ([Holland, 1975](#); [Katoch et al., 2021](#)) have found extensive application across various

15

domains and have undergone modifications and integrations with new algorithms ([Alam et al.,](#)

16

[2020](#); [Cheng & Alkhalifah, 2023](#); [Katoch et al., 2021](#)). For details about the principles of GA,

17

refer to the references ([Holland, 1975](#); [Simon, 2013](#)).

18

In a typical Python implementation, populations are defined as lists of individuals, with each

19

individual represented by a chromosome composed of a list of genes. Creating an individual

20

can be achieved utilizing either the standard library’s array or the widely-used third-party

21

library [numpy](#) ([Harris et al., 2020](#)). The evolutionary operators are defined on these structures.

22

A concise comparison between pyrimidine and other frameworks is provided in [Table 1](#).

Table 1: Comparison of the popular genetic algorithm frameworks.

Library	Design Style	Versatility	Extensibility	Visualization
pyrimidine	OOP, Meta-programming, Algebra-insprited	Universal	Extensible	export the data in DataFrame
<a href="#">DEAP</a>	OOP, Functional, Meta-programming	Universal	Limited by its philosophy	export the data in the class LogBook
<a href="#">gaft</a>	OOP, decoration pattern	Universal	Extensible	Easy to Implement

Library	Design Style	Versatility	Extensibility	Visualization
<a href="#">geppy</a>	based on DEAP	Symbolic Regression	Limited	-
<a href="#">tpot /gama</a>	<a href="#">scikit-learn</a> Style	Hyperparameter Optimization	Limited	-
<a href="#">gplearn/pysr</a>	scikit-learn Style	Symbolic Regression	Limited	-
<a href="#">scikit-opt</a>	scikit-learn Style	Numerical Optimization	Unextendible	Encapsulated as a data frame
<a href="#">scikit-optimize</a>	scikit-learn Style	Numerical Optimization	Very Limited	provide some plotting function
<a href="#">NEAT</a>	OOP	Neuroevolution	Limited	use the visualization tools

23 Tpot/gama (Gijssbers & Vanschoren, 2021; Olson et al., 2016), gplearn/pysr, and scikit-opt  
 24 follow the scikit-learn style (Buitinck et al., 2013), providing fixed APIs with limited extensibility.  
 25 They are merely serving their respective fields effectively (including NEAT (McIntyre et al.,  
 26 2019)).

27 DEAP (Fortin et al., 2012) is feature-rich and mature. However, it adopts a tedious meta-  
 28 programming style and some parts of the code lack decoupling, limiting its extensibility. Gaft  
 29 is highly object-oriented and scalable, but inactive now.

30 Pyrimidine fully utilizes the OOP and meta-programming capabilities of Python, making the  
 31 design of the APIs and the extension of the program more natural. So far, we have implemented  
 32 a variety of optimization algorithms by pyrimidine, including adaptive GA (Hinterding et  
 33 al., 1997), quantum GA (Supasil et al., 2021), differential evolution (Radtke et al., 2020),  
 34 evolutionary programming (Fogel & Fogel, 1986), particle swarm optimization (Wang et al.,  
 35 2018), as well as some local search algorithms, such as simulated annealing (Kirkpatrick et al.,  
 36 1983).

37 To meet diverse demands, it provides enough encoding schemes for solutions to optimization  
 38 problems, including Boolean, integer, real number types and their hybrid forms.

## 39 Algebra-inspired programming

40 The innovative approach is termed “algebra-inspired Programming”. It should not be confused  
 41 with so-called algebraic programming (Kapitonova & Letichevskii, 1993), but it draws inspiration  
 42 from its underlying principles.

43 The advantages of the model are summarized as follows:

- 44 1. The population system and genetic operations are treated as an algebraic system, and  
 45 genetic algorithms are constructed by imitating algebraic operations.
- 46 2. It is highly extensible. For example it is easy to define multi-populations, even so-called  
 47 hybrid-populations.
- 48 3. The code is more concise.

## 49 Basic concepts

50 We introduce the concept of a **container**, simulating an **(algebraic) system** where specific  
 51 operators are not yet defined.

52 A container  $s$  of type  $S$ , with elements of type  $A$ , is represented by the following expression:

$$s = \{a : A\} : S \quad \text{or} \quad s : S[A], \quad (1)$$

53 where the symbol  $\{\cdot\}$  signifies either a set, or a sequence to emphasize the order of the  
54 elements. The notation  $S[A]$  mimicks Python syntax, borrowed from the module [typing](#).

55 Building upon the concept, we define a population in pyrimidine as a container of individuals.  
56 The introduction of multi-population further extends this notion, representing a container of  
57 populations, referred to as “the high-order container”. Pyrimidine distinguishes itself with its  
58 inherent ability to seamlessly implement multi-population GAs.

59 An individual is conceptualized as a container of chromosomes, without necessarily being an  
60 algebraic system. Similarly, a chromosome acts as a container of genes.

61 In a population system  $s$ , the formal representation of the crossover operation between two  
62 individuals is denoted as  $a \times_s b$ , that can be implemented as the command `s.cross(a, b)`.  
63 Although this system concept aligns with algebraic systems, the current version diverges  
64 from this notion, and the operators are directly defined as methods of the elements, such as  
65 `a.cross(b)`.

66 The lifting of a function/method  $f$  is a common approach to defining the function/method for  
67 the system:

$$f(s) := \{f(a)\},$$

68 unless explicitly redefined. For example, the mutation of a population typically involves the  
69 mutation of all individuals in it. Other types of lifting are allowed.

70 transition is the primary method in the iterative algorithms, denoted as a transform:

$$T(s) : S \rightarrow S.$$

## 71 Metaclasses

72 A metaclass should be defined to simulate abstract algebraic systems, which are instantiated as  
73 a set containing several elements, as well as operators and functions on them. Currently, the  
74 metaclass `MetaContainer` is proposed to create container classes without defining operators  
75 explicitly.

## 76 Mixin classes

77 Mixin classes specify the basic functionality of the algorithm.

78 The `FitnessMixin` class is dedicated to the iteration process focused on maximizing fitness,  
79 and its subclass `PopulationMixin` represents the collective form.

80 When designing a novel algorithm, significantly differing from the GA, it is advisable to start  
81 by inheriting from the mixin classes.

## 82 Base Classes

83 There are three base classes in pyrimidine: `BaseChromosome`, `BaseIndividual`,  
84 `BasePopulation`, to create chromosomes, individuals and populations respectively.

85 Generally, the algorithm design starts as follows.

```
class UserIndividual(MonoIndividual):
    element_class = BinaryChromosome

    def _fitness(self):
        # Compute the fitness

class UserPopulation(StandardPopulation):
    element_class = UserIndividual
    default_size = 10
```

86 Here, `MonoIndividual` represents an individual with a single chromosome. `UserIndividual`  
 87 (or `UserPopulation`) serves as a container of elements in type of `BinaryChromosome` (or  
 88 `UserIndividual`). Instead of setting the fitness attribute, users are recommended to override  
 89 the `_fitness` method, where the concrete fitness computation is defined. Following is the  
 90 equivalent expression, using the notion in Equation 1:

```
UserIndividual = MonoIndividual[BinaryChromosome]
UserPopulation = StandardPopulation[UserIndividual] // 10
```

91 Algebraically, there is no difference between `MonoIndividual` and `Chromosome`. And the  
 92 population also can be treated as a container of chromosomes as follows.

```
class UserChromosome(BaseChromosome):
    def _fitness(self):
        # Compute the fitness

UserPopulation = StandardPopulation[UserChromosome] // 10
```

## 93 References

- 94 Alam, T., Qamar, S., Dixit, A., & Benaida, M. (2020). Genetic algorithm: Reviews,  
 95 implementations, and applications. *CompSciRN: Computer Principles (Topic)*. <https://doi.org/10.22541/au.159164762.28487263>
- 97 Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., Niculae, V.,  
 98 Prettenhofer, P., Gramfort, A., Grobler, J., Layton, R., VanderPlas, J., Joly, A., Holt, B.,  
 99 & Varoquaux, G. (2013). API design for machine learning software: Experiences from the  
 100 scikit-learn project. *ECML PKDD Workshop: Languages for Data Mining and Machine*  
 101 *Learning*, 108–122. <https://doi.org/10.48550/arXiv.1309.0238>
- 102 Cheng, S., & Alkhalifah, T. (2023). Robust data driven discovery of a seismic wave equation.  
 103 *Geophysical Journal International*, 236(1), 537–546. <https://doi.org/10.1093/gji/ggad446>
- 104 Fogel, L. J., & Fogel, D. B. (1986). *Artificial intelligence through evolutionary programming:*  
 105 *Prediction and identification* [Final Report]. U.S. Army Research Institute. <https://doi.org/10.21236/ada171544>
- 107 Fortin, F.-A., Rainville, F.-M. D., Gardner, M.-A., Parizeau, M., & Gagné, C. (2012). DEAP:  
 108 Evolutionary algorithms made easy. *Journal of Machine Learning Research*, 13, 2171–2175.  
 109 <https://github.com/DEAP/deap>
- 110 Gijsbers, P., & Vanschoren, J. (2021). GAMA: A general automated machine learning assistant.  
 111 In Y. Dong, G. Ifrim, D. Mladenić, C. Saunders, & S. Van Hoecke (Eds.), *Machine learning*  
 112 *and knowledge discovery in databases. Applied data science and demo track* (pp. 560–564).  
 113 Springer International Publishing. [https://doi.org/10.1007/978-3-030-67670-4\\_39](https://doi.org/10.1007/978-3-030-67670-4_39)
- 114 Harris, C. R., Millman, K. J., Walt, S. J. van der, Gommers, R., Virtanen, P., Cournapeau, D.,  
 115 Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., Kerkwijk,  
 116 M. H. van, Brett, M., Haldane, A., Río, J. F. del, Wiebe, M., Peterson, P., ... Oliphant,  
 117 T. E. (2020). Array programming with NumPy. *Nature*, 585(7825), 357–362. <https://doi.org/10.1038/s41586-020-2649-2>
- 119 Hinterding, R., Michalewicz, Z., & Eiben, A. E. (1997). Adaptation in evolutionary computation:  
 120 A survey. *Proceedings of 1997 IEEE International Conference on Evolutionary Computation*  
 121 *(ICEC '97)*, 65–69. <https://doi.org/10.1109/ICEC.1997.592270>
- 122 Holland, J. (1975). *Adaptation in natural and artificial systems*. The Univ. of Michigan.  
 123 <https://doi.org/10.7551/mitpress/1090.001.0001>
- 124 Kapitonova, Y. V., & Letichevskii, A. A. (1993). Algebraic programming: Methods and tools.

- 125      *Cybern Syst Anal*, 29, 307–312. <https://doi.org/10.1007/BF01125535>
- 126      Katoch, S., Chauhan, S. S., & Kumar, V. (2021). A review on genetic algorithm: Past,  
127      present, and future. *Multimed Tools Appl*, 80, 8091–8126. <https://doi.org/10.1007/s11042-020-10139-6>  
128
- 129      Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing.  
130      *Science*, 220, 671–680. <https://doi.org/10.1126/science.220.4598.671>
- 131      McIntyre, A., Kallada, M., Miguel, C. G., & Silva, C. F. da. (2019). Neat-python. *CodeRe-*  
132      *claimers/Neat-Python*. <https://github.com/CodeReclaimers/neat-python>
- 133      Olson, R. S., Urbanowicz, R. J., Andrews, P. C., Lavender, N. A., Kidd, L. C., & Moore, J. H.  
134      (2016). Automating biomedical data science through tree-based pipeline optimization. *Jour-*  
135      *nal of Machine Learning Research*, 123–137. [https://doi.org/10.1007/978-3-319-31204-0\\_](https://doi.org/10.1007/978-3-319-31204-0_9)  
136      9
- 137      Radtke, J. J., Bertoldo, G., & Marchi, C. H. (2020). DEPP - differential evolution parallel  
138      program. *Journal of Open Source Software*, 5(47), 1701. [https://doi.org/10.21105/joss.](https://doi.org/10.21105/joss.01701)  
139      01701
- 140      Simon, D. (2013). *Evolutionary optimization algorithms: Biologically inspired and population-*  
141      *based approaches to computer intelligence*. John Wiley & Sons. [https://api.semanticscholar.](https://api.semanticscholar.org/CorpusID:60429433)  
142      [org/CorpusID:60429433](https://api.semanticscholar.org/CorpusID:60429433)
- 143      Supasil, J., Pathumsoot, P., & Suwanna, S. (2021). Simulation of implementable quantum-  
144      assisted genetic algorithm. *Journal of Physics: Conference Series*, 1719(1), 012102.  
145      <https://doi.org/10.1088/1742-6596/1719/1/012102>
- 146      Wang, D., Tan, D., & Liu, L. (2018). Particle swarm optimization algorithm: An overview.  
147      *Soft Computing*, 22(2), 387–408. <https://doi.org/10.1007/s00500-016-2474-6>