Scientific Research Publishing

Journal of Computer and Communications, 2024, 12, 67-79
https://www.scirp.org/journal/jcc
ISSN Online: 2327-5227
ISSN Print: 2327-5219

Rapid Prototype Development Approach for Genetic Programming

Pei He[1], Lei Zhang[2]

[1]School of Computer Science and Cyber Engineering, Guangzhou University, Guangzhou, China
[2]School of Cyberspace Security, Guangzhou University, Guangzhou, China
Email: bk_he@126.com

How to cite this paper: He, P. and Zhang, L. (2024) Rapid Prototype Development Approach for Genetic Programming. *Journal of Computer and Communications*, **12**, 67-79.
https://doi.org/10.4236/jcc.2024.122005

Received: January 7, 2024
Accepted: February 18, 2024
Published: February 21, 2024

Abstract

Genetic Programming (GP) is an important approach to deal with complex problem analysis and modeling, and has been applied in a wide range of areas. The development of GP involves various aspects, including design of genetic operators, evolutionary controls and implementations of heuristic strategy, evaluations and other mechanisms. When designing genetic operators, it is necessary to consider the possible limitations of encoding methods of individuals. And when selecting evolutionary control strategies, it is also necessary to balance search efficiency and diversity based on representation characteristics as well as the problem itself. More importantly, all of these matters, among others, have to be implemented through tedious coding work. Therefore, GP development is both complex and time-consuming. To overcome some of these difficulties that hinder the enhancement of GP development efficiency, we explore the feasibility of mutual assistance among GP variants, and then propose a rapid GP prototyping development method based on πGrammatical Evolution (πGE). It is demonstrated through regression analysis experiments that not only is this method beneficial for the GP developers to get rid of some tedious implementations, but also enables them to concentrate on the essence of the referred problem, such as individual representation, decoding means and evaluation. Additionally, it provides new insights into the roles of individual delineations in phenotypes and semantic research of individuals.

Keywords

Genetic Programming, Grammatical Evolution, Gene Expression Programming, Regression Analysis, Mathematical Modeling, Rapid Prototype Development

1. Introduction

Genetic Programming (GP) [1] is an automatic programming approach applied

in a wide range of application areas, such as circuit design, mathematical modeling, data mining, image analysis, regression analysis, natural disaster prediction, etc. [2] [3] [4] [5] [6]. Its fundamental design idea comes from a genetic algorithm [7] [8], which is derived from such a rule as "survival of the fittest", that is, evolving constantly populations of individuals, followed by the evaluations, and finally obtaining the desired solution with the best fitness value to some given problem. With the rapid development of computing technology, this search-based method has become one of the most important tools to deal with many optimization problems.

GP algorithm consists of 5 steps: a) constructing an initial population of individuals associated with approximate solutions to some given problem; b) evaluating individuals in the population for their fitness values and algorithmic termination condition. If the condition holds, go to step e); otherwise, execute the next step; c) generating a new generation of individuals on the basis of genetic operators and strategies; d) go to step b); e) regarding the individual with the best fitness value as the desired result. This is the common structure of various GP systems.

Up to now, there appear a great many GP variants, which include Grammatical Evolution (GE) [9] [10] [11], Gene Expression Programming (GEP) [12] [13] [14], Multi-Expression Programming (MEP) [3] [14], Cartesian Genetic Programming (CGP) [3] [15] [16], etc. Generally speaking, both their designs and implementations will be concerned with individual representations, genetic operators, evolutionary controls, evaluations, and the like. Consequently, novel GPs designed for some specific problems have the characteristics of structural consistency, but it will be very time-consuming and laborious to design and develop them from scratch.

In this paper, we intend to make a preliminary discussion on the rapid prototyping development of Genetic programming. To our knowledge, similar work related to it is relatively rare. Since many GP variants share the same structure, we manage to build a suitable bridge between them, so that the rapid prototyping development conception and target GP variants can be explored and tested on certain public GP platforms.

2. Basic Principle

Representation plays an important role in GP system. As Rothlauf put it in reference [17] without representations, no use of genetic and evolutionary algorithms is possible. Several other concepts related to genotype representations range from decoding, solution space or phenotypic space to individual evaluation. Search, control and other related operations are primarily applied in genotype space. After individuals have been obtained from some constrained rules, they are generally interpreted as an approximate solution in phenotype space by decoding method, and whether or not it is a good result depends on the fitness evaluation.

Consider GP_1 and GP_2 as two GP systems with the same solution space, regressively analyzed sample dataset S, and evaluation criteria E: *SolutionSpace* × $\{S\}$ → *Real*, I_1 and I_2 as their individual spaces, D_1: I_1 → *SolutionSpace* and D_2: I_2 → *SolutionSpace* as their decoding methods respectively, then what is the relationship, mathematically speaking, between their search processes? Can they help each other to some extent?

In principle, we can establish the relation Equation (1) for these two search-based GP systems, where ε stands for the error term.

$$\begin{aligned} &E\left(D_1\left(\underset{s.t.\ a_1\in I_1}{\arg OPT}\left(E\left(D_1\left(a_1\right),S\right)\right)\right),S\right) \\ &=E\left(D_2\left(\underset{s.t.\ a_2\in I_2}{\arg OPT}\left(E\left(D_2\left(a_2\right),S\right)\right)\right),S\right)+\varepsilon \end{aligned} \tag{1}$$

If having h: I_1 → I_2, then there is Equation (2):

$$\begin{aligned} &E\left(D_1\left(\underset{s.t.\ a_1\in I_1}{\arg OPT}\left(E\left(D_1\left(a_1\right),S\right)\right)\right),S\right) \\ &=E\left(D_2\left(h\left(\underset{s.t.\ a_1\in I_1}{\arg OPT}\left(E\left(D_2\left(h\left(a_1\right)\right),S\right)\right)\right)\right),S\right)+\varepsilon' \end{aligned} \tag{2}$$

$$\begin{aligned} &E\left(D_1\left(\underset{s.t.\ a_1\ \in I_1}{\arg OPT}\left(E\left(D_1\left(a_1\right),S\right)\right)\right),S\right)\Bigg|_{D_1}^{D_2h} \\ &=E\left(D_2\left(h\left(\underset{s.t.\ a_1\in I_1}{\arg OPT}\left(E\left(D_2\left(h\left(a_1\right)\right),S\right)\right)\right)\right),S\right) \end{aligned} \tag{3}$$

Since the non-error term on the right side of Equation (2) can be formally obtained from the left side of Equation (2) by replacing its two occurrences of D_1 with D_2h, we can formally get Equation (3). This means the calculation of the right side of Equation (3) can be approximately computed in GP_1 through substitution of D_2h for D_1 in GP_1. Since GP_1 embraces services many other GPs need, we call it the basic GP development platform.

It is worth noticing that D_2 and h play a critical role in rapid prototype development of GP system. In Section 3, we will solve the selection problem of h, a mapping from individuals of the basic GP platform onto that of target GPs.

3. Proposed Approach

The first step of the proposed method is to choose πGrammatical Evolution (*πGE*) [18] as the basic GP platform. GE is a GP variant with variable length genotypes [10] [19] [20]. One of its advantages is that it can use a context-free grammar to describe phenotypes of individuals, therefore theoretically generating programs in an arbitrary programming language. However, GE is usually implemented in terms of leftmost derivations. For example, let $\delta = \alpha A\beta$ be a sentential form of some grammar $G = (V_N, V_T, S, P)$ [21] [22], where V_N is a finite set of non-terminal symbols, V_T on the contrary a finite set of terminals.

Any derivation of sentential form in G will be derived from the start symbol S based on some production rule in P. So, the leftmost derivation of $\alpha\gamma\beta$ from $\alpha A\beta$ with respect to $A ::= \gamma$ in P is the substitution of γ for the leftmost non-terminal , say A, in δ. In order to realize this leftmost derivation, *i.e.* $\alpha A\beta \Rightarrow \alpha\gamma\beta$, in GE whose genotype consists of codons, the production $A ::= \gamma$ selected is determined by Equation (4), where the number of alternative productions for A is denoted as *NumChoices*(A).

$$\text{The production rule selected} = (\textit{value of the current codon}) \bmod \textit{NumChoices}(A) \tag{4}$$

To enhance the flexibility of derivations, a novel encoding method which make codons to encode both expanded non-terminal and production information to be used, *i.e.* codon = ($v1$, $v2$), achieves the desired efficacy. This improvement leads to the πGE. The algorithm structure is similar to that of GP, and the detail can be found in [18]. The major difference between πGE and canonical GE lies in the fact that both the expanded non-terminal, say X, and candidate production rule of X, selected for the derivations are determined by Equations (5) and (6). *cNumNonterminals* used below is the number of all non-terminals occurring in current sentential form. Table 1 demonstrates an example grammar and a derivation of a Boolean expression in the case of the grammar.

$$\text{The nonterminal } X \text{ expanded} = v1 \bmod \textit{cNumNonterminals} + 1 \tag{5}$$

$$\text{The candidate production of } X = v2 \bmod \textit{NumChoices}(X) \tag{6}$$

What we should deal with in *the second step of the proposed approach* is to effectively delineate individuals of objective GP using a context-free grammar, and treat them as sentences of the corresponding language. Once πGE has been selected as the basic GP platform, as is done in this paper, there come a lot of

Table 1. Example derivation of a Boolean expression.

Value of Codon		Context-free Grammar B ::= V (0) B ::= (BOB) (1) B ::= ~B (2)		O ::= and (0) O ::= or (1)	V ::= t (0) V ::= f (1)	
		Sentential Form	Number of Nonterminals	Nonterminal Expanded	Rule Selected	Number of Choices
10	31	B	1	B	B ::= (BOB)	3
5	11	(BOB)	3	2nd B	B ::= ~B	3
31	3	(BO ~B)	3	O	O ::= or	2
14	9	(B or ~B)	2	1st B	B ::= V	3
22	3	(V or ~B)	2	V	V ::= f	2
45	6	(f or ~B)	1	B	B ::= V	3
88	8	(f or ~V)	1	V	V ::= t	2
73	56	(f or ~t)	0			

Notes: Nonterminal and production rule are determined by Equations (5) and (6).

mappings from πGE onto individual space of the target GP system. In the present paper, Formulas (5) and (6) are also used to define the mapping relation h discussed in Section 2.

Finally, as far as *the third step* is concerned, we will focus on the programming of individual decoder and evaluation component which play an essential part in all GP designs. After completing the previous work, the rapid GP prototyping development system without extra coding requirements will be successfully achieved. In addition, the decoding process may also encounter such problem as incomplete mapping of individuals, that is, the derived sentential form still contains some non-terminal symbols, therefore, it is not a valid expression. The measure taken here is to design and implement default mapping rules (see *complete mapping* used in [19] [20]) compatible with description grammar of individuals for dealing with it. The following provides an overview of the rapid prototype development procedure:

a) Implementing a basic GP prototyping development platform like πGE;

b) Specifying target GP individuals using a context-free grammar;

c) Designing targeted individual decoders, evaluation components and default mapping rules in terms of sample dataset and results of steps a and b;

d) Running programs and analyzing the obtained results.

4. Experiment and Analysis

This section intends to conduct regression analysis on problems from the literature [10] [23] in the context of the above method. The observation dataset is sampled in the same way as that of [10] at 20 observation points of variable y like {−1, −0.9, −0.8, −0.76, −0.72, −0.68, −0.64, −0.4, −0.2, 0, 0.2, 0.4, 0.63, 0.72, 0.81, 0.90, 0.93, 0.96, 0.99, 1} in the range of [−1, 1].

$$f(y) = y^4 + y^3 + y^2 + y \tag{7}$$

$$g(y) = 0.3y\sin(2\pi y) \tag{8}$$

$$h(y) = 1 + 3y + 3y^2 + y^3 \tag{9}$$

To demonstrate that developing GP on rapid prototyping development system is of easiness and efficiency, we have designed and implemented two GP systems, which can be functionally categorized into the standard tree-based GP and the improved GEP (ImpGEP) [24] according to their decoding means. Having implemented πGE as the basic development platform, experiments with them on Formula (7) through Formula (9) can be carried out. The individual descriptions, default mapping rules and decoding algorithms used by these GPs are shown in **Figures 1-4**, respectively. Except that their decoding methods are different from each other and representations should be specified by some grammars, other components such as evolutionary controls, genetic operators implemented in πGE, the basic GP development platform, are shared among various objective GPs. Major parameters employed here are given in **Table 2**.

Experiment with GP on the regression problems

Table 2. Parameters used in these experiments.

	Runs	GenSize	PopSize	MxLength	MnLength	FixLength	ProCro	ProMut
GP	100	100	50	50	20	no	0.9	0.15
ImpGEP	100	100	50	50	20	yes	0.9	0.15

Notes: MxLength = Max Length of individual; FixLength = Fixed Length individual; ProCro = Probability of Crossover.

Grammar for the tree-based GP individuals:

<TreeGP>::= <Leaf> | (<TreeGP> <Op> <TreeGP>) | <Fun>(<TreeGP>)
<Op>::= + | - | * | /
<Fun>::= exp | sin | cos | log
<Leaf>::= y | 1.0

Default mapping:

<TreeGP>::= y
<Op>::= +
<Fun>::= sin
<Leaf>::= 1.0

Figure 1. Grammar and default mapping for individuals of tree-based GP.

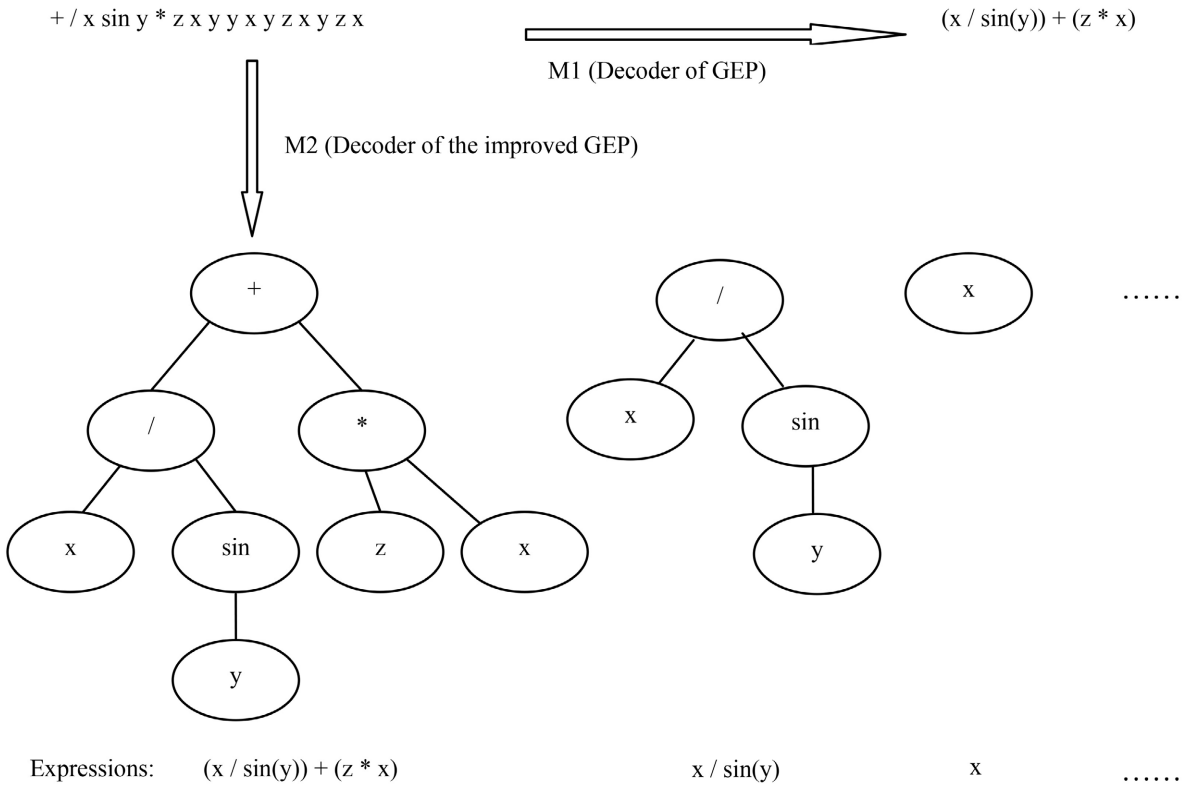


Figure 2. Decoder of the improved GEP with its decoded expressions.

1) Brief comment on GP

Classical GP [1] is proposed by Koza for automatically programming computer programs to solve given problems. The major idea embedded in GP is to decode the evolved tree-based individuals into expressions on the basis of its five-step algorithm framework (see Section 1). On the one hand, individuals can

```
Grammar for the improved GEP individuals (head length=10, tail length=11)
    <GenotypeGEP>::= <Head> <Tail>
    <Head>::= <op> <h> <h> <h> <h> <h> <h> <h> <h> <h>
    <Tail>::= <vc> <vc> <vc> <vc> <vc> <vc> <vc> <vc> <vc> <vc> <vc>
    <op> ::= + | - | * | / | sin | cos | log | exp
    <h>::= <op>|<vc>
    <vc>::= y|1.0

Default mapping:
    <GenotypeGEP>::= * y - y 1.0 y sin y exp y 1.0 1.0 y y y 1.0 y y 1.0 y y
    <Head>::= * y + y 1.0 y log y exp y
    <Tail>::= y 1.0 y 1.0 y 1.0 y y 1.0 y y
    <op>::= +
    <h>::= y
    <vc>::= y
```

Figure 3. Grammar and default mapping for individuals of the improved GEP.

```
Decoding algorithm M2 for individuals of the improved GEP:
function M2(Individual: string; var pos: integer): string;
  var
    cPos: integer;       sGene, subExp1, subExp2: string;
begin
  cPos:= pos;     subExp1:= DecodeOfDepthFirstGEP(Individual, cPos);
  sGene:= GetFuncOrOpnd(Individual, pos);       pos:= pos+length(sGene)+1;
  if pos <= (length(Individual) div 2)
    then
      begin
        subExp2:= M2(Individual, pos);
        if LeastSquareError(TargetFunction, subExp1,'y', Points)>
          LeastSquareError(TargetFunction, subExp2,'y', Points)
          then subExp1:= subExp2
      end;
  M2:= subExp1
end;
```

Figure 4. Decoder for individuals of the improved GEP.

be constantly generated from the algorithm, decoding mechanism within it is responsible for the transformation of them into expressions on the other. Since a context-free grammar can represent tree-based embedded relations between sub-expressions, we introduce it to specify both genotypes and phenotypes. In this case, the phenotypic space is the same as the language obtained. Naturally, the *identity* mapping is suitable for the decoder.

2) Experimental process

Having implemented the required πGE, our major work towards solving the above problems by GP includes 2 steps:

a) Designing a grammar as well as its corresponding default mapping rule as shown in Figure 1, so as to specify the individuals of concern;

b) Programming both the decoder and evaluation component so that phenotypes and fitness values can be decoded from individuals and well evaluated, respectively.

In this experiment, the decoder and evaluation mechanism are the *identity mapping* and the *least square error* principle, respectively.

Experiment with the Improved GEP on the regression problems

1) Brief comment on GEP

GEP [12] is a linearly represented GP variant with fixed-length individuals of terminal symbols and functions. Its decoding method as partly given in **Figure 2** is impressive. It transforms individuals into expression trees (ET trees) at first, and followed by traversal operations on them to get the expected expressions [12]. However, for the sake of simplicity, the decoder M1 used here is designed according to the depth-first decoding mode.

Another feature is its structural constraint of individuals, that is to say, each individual is composed of a head and a tail that satisfy such length restriction as $t = h * (n - 1) + 1$, where t, h represent length of tail and head, and n is the maximum arity of functions involved. Considering that the decoder of canonical GEP can only find one possible solution in each run, we have made an improvement for figuring out many expressions from genotype reusing technique [24] on it so that the wining chance can be enhanced. For instance, the improved GEP tends to reuse genotype information through continuously applying original GEP decoding approach (say M1 in **Figure 2**) to every element (or word) of an individual as shown in **Figure 2**, thus obtaining many alternative solutions. The improved decoder M2 is also declared in **Figure 4**.

2) Experimental process

Having finished the implementation of πGE as basic development platform, we have the following problem solving process:

a) Designing both the grammar and the corresponding default mapping as given in **Figure 3** for specifying the individuals of concern;

b) Coding both the decoder and evaluation component so as to decode and evaluate individuals and their fitness values, respectively.

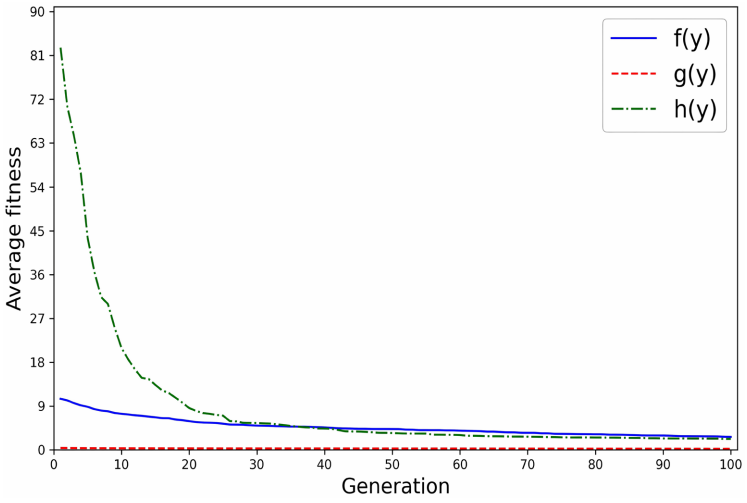
In this experiment, evaluation criterion is the *least square error*. Given the following three functions, the decoder of the improved GEP, denoted M2 in **Figure 2**, can be realized by applying the function DecodeOfDepthFirstGEP firstly to different elements of an individual to get many candidate expressions, then evaluating and choosing the fittest expression as the desired solution. The implementation of M2 is shown in **Figure 4**.

DecodeOfDepthFirstGEP(Individual, cPos): A function to construct an expression from the element indexed by cPos in the individual.

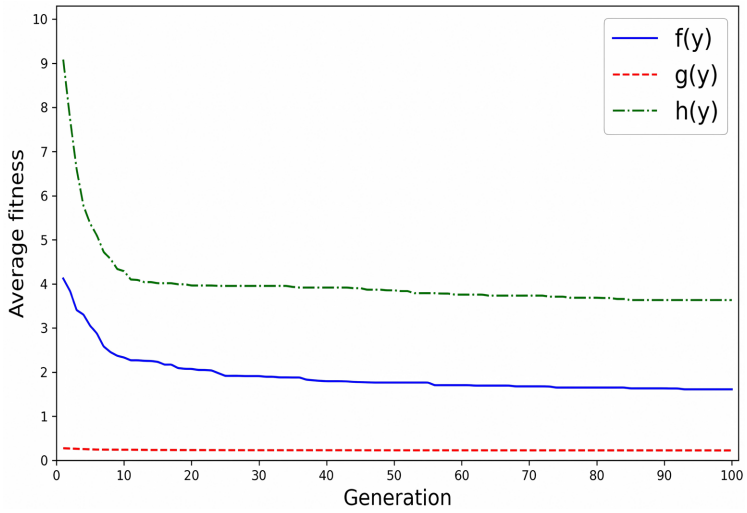
GetFuncOrOpnd(Individual, pos): A function to get a element/word indexed by *pos* in the individual.

LeastSquareError(p, q, 'y', Points): A function to compute the least square error between functions p and q. 'y' stands for the variable of them. The variable Points is a list of the observation points of y.

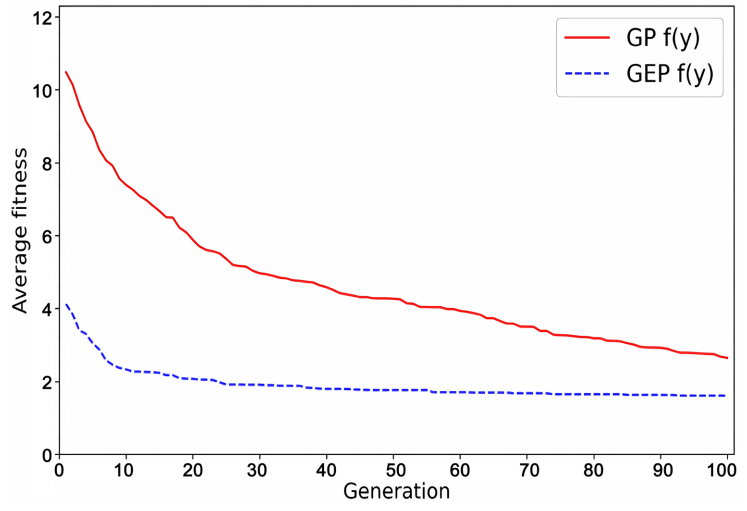
Finally, running the decoders and evaluation components obtained above in the context of πGE, sample dataset and grammars given above will result in **Figure 5** and **Figure 6**. The evaluation procedure used here is realized on the *least square error* principle. It follows from these results that the solutions obtained from each rapid developed method can gradually approach to the optimization



(a)

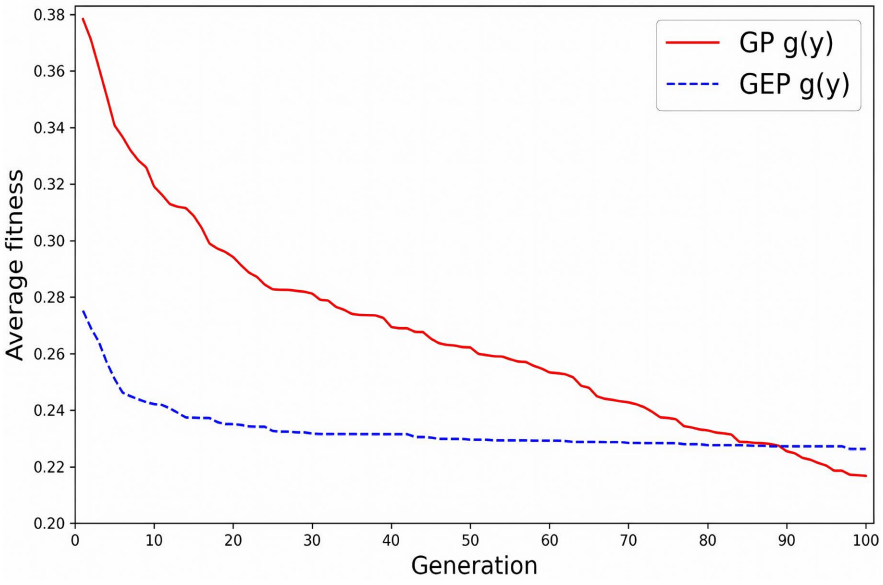


(b)

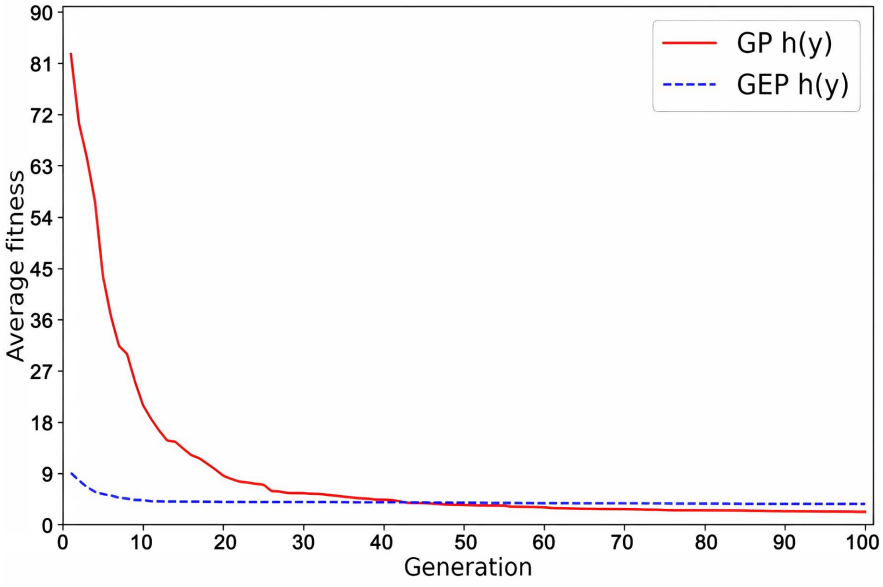


(c)

Figure 5. Solving *f*, *g*, *h* by GP and the improved GEP. (a) Solving *f*, *g*, *h* by GP; (b) Solving *f*, *g*, *h* by GEP; (c) Solving *f* by GP and GEP.



(a)



(b)

Figure 6. Solving g, h by GP and the improved GEP. (a) Solving g by GP and GEP; (b) Solving h by GP and GEP.

goals when applying that method to solve the regression problems of concern. Additionally, it can also be seen from **Figure 6** that in the given environment of these experiments, the tested GP outperforms the GEP variant on g and h with respect to the fitness values, and from **Figure 5(a)** and **Figure 5(b)** that the regression analysis on g seems easier than on f and h. With the help of the proposed method, we can quickly establish a GP prototyping system and simulate it on πGE, the basic GP development platform. Besides, this approach also provides with a common environment for comparisons among many GPs.

Although the number of examples given here is small, we can get some im-

portant cognition and enlightenment from them. So far, there are many GP variants. The essential intention implied in the abovementioned method and effect is to construct and search the target semantic objects in the phenotypic space by changing both the search space and corresponding decoders. As such, this method not only helps designers to improve the efficiency of GP research and development, but also helps to use genotype space, an unpredictable kaleidoscope, to examine and understand semantic objects and domain knowledge.

5. Conclusion

We have made a preliminary investigation into rapid GP prototype development and applications in the present paper, and more systematical and in-depth issues like how to integrate formal structures and semantics into it will become our future work. The major advantage of using this approach to design and implement GP is that designers and implementers can ignore many implementation details and concentrate their energy on the essence of the problem, such as representation, decoding method and population evaluation. For representation, what we need to do is to define individuals by designing a context-free grammar. And decoder and evaluation procedure are the components to be programmed only provided mapping relation between the basic development platform and target GPs is determined. These GE-based task segmentation methods are of great theoretical and practical significance to solving complex practical problems and studying high-performance computing of genetic programming.

Acknowledgements

This work was supported partly by the National Natural Science Foundation of China under Grant No. 61977018.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Koza, J.R. (1992) Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge, MA.

[2] Zhong, J.H., Feng, L. and Ong, Y.-S. (2017) Gene Expression Programming: A Survey. *IEEE Computational Intelligence Magazine*, **12**, 54-72. https://doi.org/10.1109/MCI.2017.2708618

[3] Oltean, M., Groşan, C., Dioşan, L. and Mihăilă, C. (2009) Genetic Programming with Linear Representation: A Survey. *International Journal on Artificial Intelligence Tools*, **18**, 197-238. https://doi.org/10.1142/S0218213009000111

[4] Angelis, D., Sofos, F. and Karakasidis, T.E. (2023) Artificial Intelligence in Physical Sciences: Symbolic Regression Trends and Perspectives. *Archives of Computational Methods in Engineering*, **30**, 3845-3865. https://doi.org/10.1007/s11831-023-09922-z

[5] Bi, Y., Xue, B., Mesejo, P., Cagnoni, S. and Zhang, M. (2023) A Survey on Evolutionary Computation for Computer Vision and Image Analysis: Past, Present, and Future Trends. *IEEE Transactions on Evolutionary Computation*, **27**, 5-25. https://doi.org/10.1109/TEVC.2022.3220747

[6] Hosseini, M. and Lim, S. (2022) Gene Expression Programming and Data Mining Methods for Bushfire Susceptibility Mapping in New South Wales, Australia. *Natural Hazards*, **113**, 1349-1365. https://doi.org/10.1007/s11069-022-05350-7

[7] Holland, J.H. (1975) Adaptation in Natural and Artificial Systems. The University of Michigan Press, Ann Arbor, MI.

[8] Mitchell, M. (1996) An Introduction to Genetic Algorithm. MIT Press, Cambridge.

[9] Brabazon, A., O'Neill, M. and McGarraghy, S. (2015) Grammatical Evolution. In: *Natural Computing Algorithms*, Springer, Berlin, Heidelberg, 357-373. https://doi.org/10.1007/978-3-662-43631-8_19

[10] O'Neill, M. and Ryan, C. (2001,) Grammatical Evolution. *IEEE Transactions on Evolutionary Computation*, **5**, 349-358. https://doi.org/10.1109/4235.942529

[11] Bartoli, A., Castelli, M. and Medvet, E. (2020) Weighted Hierarchical Grammatical Evolution. *IEEE Transactions on Cybernetics*, **50**, 476-488. https://doi.org/10.1109/TCYB.2018.2876563

[12] Ferreira, C. (2001) Gene Expression Programming: A New Adaptive Algorithm for Solving Problems. *Complex Systems*, **13**, 87-129.

[13] Ferreira, C. (2006) Automatically Defined Functions in Gene Expression Programming. In Nedjah, N., Mourelle, L.d.M. and Abraham, A., Eds., *Genetic Systems Programming*, Vol. 13, Springer, Berlin, Heidelberg, 21-56. https://doi.org/10.1007/3-540-32498-4_2

[14] Oltean, M. and Grosan, C. (2003) A Comparison of Several Linear Genetic Programming Techniques. *Complex Systems*, **14**, 285-313. https://wpmedia.wolfram.com/uploads/sites/13/2018/02/14-4-1.pdf

[15] Miller, J.F. (2020) Cartesian Genetic Programming: Its Status and Future. *Genetic Programming and Evolvable Machines*, **21**, 129-168. https://doi.org/10.1007/s10710-019-09360-6

[16] Miller, J.F. (2011) Cartesian Genetic Programming. In: Miller, J., Ed., *Cartesian Genetic Programming*, Springer, Berlin, Heidelberg, 17-34. https://doi.org/10.1007/978-3-642-17310-3_2

[17] Rothlauf, F. (2006) Representations for Genetic and Evolutionary Algorithms. In: *Representations for Genetic and Evolutionary Algorithms*, Springer, Berlin, Heidelberg. https://link.springer.com/content/pdf/10.1007/3-540-32444-5_2.pdf

[18] Fagan, D., Nicolau, M., O'Neill, M., Galván-López, E., Brabazon, A. and McGarraghy, S. (2010) Investigating Mapping Order in πGE. *IEEE Congress on Evolutionary Computation*, Barcelona, 18-23 July 2010, 1-7.

[19] He, P., Johnson, C.G. and Wang, H.F. (2011) Modeling Grammatical Evolution by Automaton. *Science China Information Sciences*, **54**, 2544-2553. https://doi.org/10.1007/s11432-011-4411-8

[20] He, P., Deng, Z.L., Gao, C.Z., Wang, X.N. and Li, J. (2017) Model Approach to Grammatical Evolution: Deep-Structured Analyzing of Model and Representation. *Soft Computing*, **21**, 5413-5423. https://doi.org/10.1007/s00500-016-2130-1

[21] Hopcroft, J.E., Motwani, R. and Ullman, J.D. (2008) Introduction to Automata Theory, Languages, and Computation. 3rd Edition, Pearson Education, Inc., San Antonio, TX.

[22] Aho, A.V., Lam, M.S., Sethi, R. and Ullman, J.D. (2007) Compilers: Principles, Techniques, and Tools. 2nd Edition, Pearson Education, Inc., San Antonio, TX.

[23] Gupt, K.K., Raja, M.A., Murphy, A., Youssef, A. and Ryan, C. (2022) GELAB—The Cutting Edge of Grammatical Evolution. *IEEE Access*, **10**, 38694-38708. https://doi.org/10.1109/ACCESS.2022.3166115

[24] Deng, W., He, P. and Qian, J.Y. (2013) Multi-Gene Expression Programming with Depth-First Decoding Principle. *Pattern Recognition and Artificial Intelligence*, **26**, 819-828. (In Chinese)