Direct Evolution of Hierarchical Solutions with Self-Emergent Substructures

Xin Li[1], Chi Zhou[2], Weimin Xiao[2], Peter C. Nelson[1]
[1]Department of Computer Science, University of Illinois at Chicago, Chicago, IL 60607
[2]Physical Realization Research Center of Motorola Labs, Schaumburg, IL 60196
[1]{xli1, nelson}@cs.uic.edu, [2]{Chi.Zhou, awx003}@motorola.com

Abstract

Linear genotype representation and modularity have continuously received extensive attention from the Genetic Programming (GP) community. The advantages of a linear genotype include a convenient and efficient implementation scheme. However, most existing techniques using a linear genotype follow the imperative programming language paradigm and a direct hierarchical composition for the functionality of the solution is underachieved. Our work is based on Prefix Gene Expression Programming (P-GEP), a new GP method featured by a prefix notation based linear genotype representation. Since P-GEP uses a functional language paradigm, its framework results in natural self-emergence of substructures as functional components during the evolution. We propose to preserve and utilize potentially useful emergent substructures via a dynamic substructure library, empowering the algorithm to focus the search on a higher level of the solution structure. Preliminary experiments on the benchmark regression problems have shown the effectiveness of this approach.

1. Introduction

Linear genotype representation has been a popular alternative to the standard tree representation since the introduction of Genetic Programming (GP), primarily due to its convenient and efficient implementation scheme with the linear strings. Some major techniques in this line include linear genetic programming [13, 12], grammatical evolution [11] and stack-based genetic programming [14, 15]. However, most of these approaches follow the imperative programming language paradigm, mapping linear genotype into instructions to compose a program producing the corresponding solution. Although the stack-based approach directly defines the functionality of the solutions, its genotype-phenotype mapping mechanism results in the real solution being segmented by the non-effective (i.e., discarded) genes in the linear genotype representation, and therefore its linear genotype is not expressive enough for identifying the functional components for the represented solution.

Meanwhile, it have been widely recognized in the Genetic Programming (GP) community that to solve complex problems, an ideal approach is to make the algorithm capable of hierarchically forming the solution from simpler components. The researched techniques regarding modularity and hierarchy fall into the following main categories: automatically defined functions [4, 6, 5], module acquisition [7], adaptive representation [8], hierarchical genetic programming [9] and subtree encapsulation [16]. Most recently, Keijzer, Ryan and Cattolico [17] have proposed a transferable library technique to maintain subtrees improved over multiple runs as beneficial modules for other difficult problems in the same domain. However, most prior methods have either predefined the structures of functional components or relied on a tree representation, which are not very efficient and somehow sacrifice the flexibility of the evolutionary process itself. Some linear genotype schemes have adopted an adapted version of the aforementioned modularity approaches in tree representation (e.g. [19]). However, again, few of them directly perform on the functional structure of the solutions.

In this paper, we present a more viable scheme of incrementally identifying good solution structures based on the framework of Prefix Gene Expression Programming (P-GEP), a recently proposed GP method with a linear genotype [18]. Enhanced from Gene Expression Programming (GEP) [1, 2], P-GEP also represents solutions as linear character strings of fixed length (called *chromosomes*) which, in the subsequent fitness evaluation, can be expressed as *expression trees* (ETs) of different sizes and shapes. However, the linear genotype of P-GEP adopts the prefix notation, instead of Karva notation [1]. Consequently, in P-GEP a subtree of the ET representing a complete function corresponds to a coherent substring in the linear chromosome. This is virtually a strict functional programming language paradigm and therefore results in a natural hierarchy in forming the solutions and the identification of solution components becomes feasible within the linear genotype.

The scheme presented in this paper encourages structured solutions by preserving and utilizing substructures that can self-emerge in the evolutionary process of P-GEP. Here the substructures mean functional building blocks as coherent elements in a linear genotype that help compose the solution. By saying self-emergence, we mean the substructure is generated, preserved, and evolved automatically during the P-GEP process. The implementation was realized via a dynamic substructure library equipped with two novel genetic operators for substructures, namely compression and expansion operators. Our hypothesis is that by making the naturally evolved useful substructures directly accessible for producing candidates in the later evolution, the P-GEP search process will have better performance in forming the final solution.

To illustrate the applicability of proposed approach, an artificial structured regression problem was first fully examined. The experimental results demonstrated that this work assists in finding meaningful building blocks to incrementally form the final solution with a faster fitness convergence. Additional experiments were run using six benchmark regression problems from the Weka datasets, comparing P-GEP, P-GEP with substructure library, and standard model tree as implemented in WEKA knowledge analysis tool [10]. These results have shown that P-GEP with substructure library can find better regression models, and emergent substructures play a significant role in composing solutions.

2. A Brief Overview of P-GEP

To solve a specific problem, P-GEP requires a function set, terminal set, fitness function, control parameters, and a stop condition.

Derived from GEP, each chromosome in P-GEP is a fixed length character string, which can be composed of any elements (also called *genes*) from the function or the terminal sets. Using the elements from the function set {+, -, *, /, sqrt} and the terminal set {a, b, c, d, 1}, the following is an example P-GEP chromosome of length fifteen, where "." is used to delimit individual genes; sqrt denotes the square-root function; 1 is a numeric constant; and a, b, c, d are variable (or attribute) names.

sqrt.*.+.*.a.*.sqrt.a.b.c./.1.-.c.d (2.1)

P-GEP distinguishes itself from GEP with a new genotype-phenotype mapping mechanism using a prefix notation. A chromosome can be transformed into an ET with a depth-first, left-to-right procedure. A chromosome segment mapped into an ET is also called a prefix expression. A branch of the ET stops growing when the last node in this branch is a terminal. For instance, the ET shown in Figure 1 corresponds to the sample chromosome (2.1), and can be interpreted into a mathematical formula as (2.2). The conversion of an ET into a prefix expression is also very straightforward as traversing the nodes in the tree in preorder to form the string. As GEP, in P-GEP all of the chromosomes are of a fixed length, but the size of ETs can vary, and the possible non-effective genes all come at the end of the chromosome. The significance with this genotype representation is that subtrees in an ET with independent functionality now appear in the linear chromosome as naturally coherent string segments, which can be easily identified. Therefore, the linear genotype also inherently encodes the complexity hierarchies of solutions as revealed by the tree structure [18].

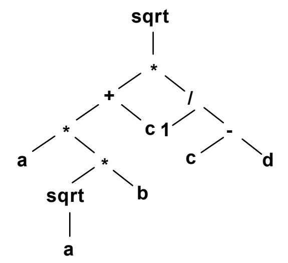


Figure 1. Example of P-GEP Expression Tree

$$\sqrt{(a\sqrt{ab}+c)(\frac{1}{c-d})} \quad (2.2)$$

The P-GEP algorithm begins with the random generation of linear fixed-length chromosomes for individuals of the initial population. The fitness of each individual is evaluated based on a pre-defined fitness function after being mapped into an expression tree. The individuals are selected with respect to fitness to reproduce with modification. This process is repeated for a pre-specified number of generations, or until a satisfying solution has been found. In P-GEP, the selection procedure is determined by roulette-wheel sampling with elitism [3] based on individuals' fitness, and the genetic operators are in accordance with those described in [2].

3. Emergent Substructures in P-GEP

Under the P-GEP's genotype representation, a solution is formed hierarchically upon substructures (defined as functional building blocks coherent in a linear genotype that help compose the solution). For example, in the example chromosome (2.1), the following substrings are all valid substructures:

{sqrt.*.+.*.a.*.sqrt.a.b.c./.1.-.c.d, /.1.-.c.d, -.c.d, *.+.*.a.*.sqrt.a.b.c./.1.-.c.d, +.*.a.*.sqrt.a.b.c, *.a.*.sqrt.a.b, *.sqrt.a.b, sqrt.a}.

When the linear chromosome subjects to the genetic operations, substructures that will be generated are totally determined by the evolutionary process. Due to their high value in composing good solutions, some substructures evolve and persist as single components (referred as *self-*

emergence of substructures in P-GEP). These are functional components favored by the search procedure and can potentially be explicitly utilized to construct the final solution. Since the direct measure for the fitness of a substructure is not easy, we assume that fitter individuals in the population also likely have good solution structure components. If a substructure appears frequently in these individuals, it is prone to be useful for composing the final solution. We currently follow this intuition to specify the emergent substructures in P-GEP as substructures having a high appearance frequency in the fittest individuals (called the *elite group*) among the whole population.

To preserve and reuse the emergent substructures during the P-GEP process, generally two steps are involved, including determining the minimum required appearance frequency in the elite group and extracting an emergent substructure into a single representative symbol (called *derived gene*). The first step is trivial and the appearance frequency requirement is empirically set close to the number of the elites. The second step requires more crafts. First, since it is common that one substructure is built upon the others, a brutal extraction can demolish the hierarchical relationship between the substructures. Moreover, an undistinguished hierarchy between substructures will produce inaccurate statistics on their appearance frequency, since an occurrence of one substructure is also an occurrence of all of its component substructures. A better choice is to incrementally extract the substructures at different hierarchies (called *complexity level*). The valid substructures of chromosome (2.1) can be represented as derived genes shown in Table 1. The resulted implementation of emergent substructures can then be simplified to only concern about a uniform operator-parameter construction.

Table 1. Example of derived genes

Derived gene	Substructures	Complexity level
DG0	sqrt.a	0
DG1	-.c.d	0
DG2	*.DG0.b	1
DG3	/.1.DG1	1
DG4	*.a.DG2	2
DG5	+.DG4.c	3
DG6	*.DG5.DG3	4
DG7	sqrt.DG6	5

4. Compression and Expansion Operators

Inspired by [7], we have designed two additional genetic operators for dynamically implementing emergent substructures in P-GEP as proposed in section 3.

Compression operator. The compression operator compresses an emergent substructure into a single derived gene. For the example chromosome (2.1), suppose the substructure "sqrt.a" is first found as a qualified emergent substructure and compressed into a derived gene DG0. DG0 then replaces the original substring in the chromosome to yield "sqrt.*.+.*.a.*.DG0.b.c./.1.-.c.d". Next, the compression operator can further operate on substructures "-.c.d" (denoted by DG1) and "*.DG0.b" (denoted by DG2) to produce an even more compact solution "sqrt.*.+.*.a.DG2.c./.1.DG1". Figure 2 illustrates the above procedure with the downward solid arrows.

The significance of the compression operator is that it assures that the compressed substructure is coherent under the ordinary genetic operations. Moreover, the derived genes are equivalent to position independent subroutines. The evolutionary process is more flexible with the ability of choosing constructive components of the solution from both primitive genes and potentially useful subroutines.

Expansion operator. The expansion operator works as opposite to the compression, by restoring a derived gene symbol back to the gene segment it denotes, which is also illustrated in Figure 2 with the upward dashed arrows.

The expansion operator furthers the goal of preserving only useful substructures to ease the search process. Since the elite group evolves along with the general P-GEP process, the emergent substructures also update over time. A derived gene produced earlier may receive much less usage in later generations as the evolutionary process digs into the search space. The presence of useless derived genes distracts the search process and increases computational cost, and therefore expansion needs to be applied. In addition, expansion enables the ever frozen substructure piece to subject to the genetic operators again, which may possibly tear the piece apart and dramatically reframe the corresponding solution structure. This fosters a diverse search space.

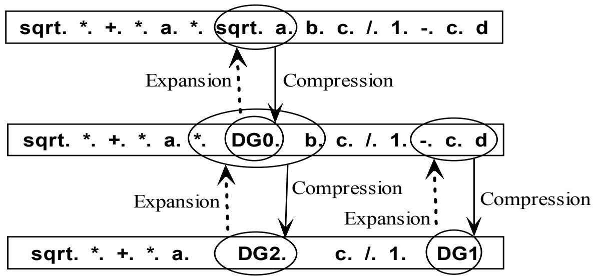
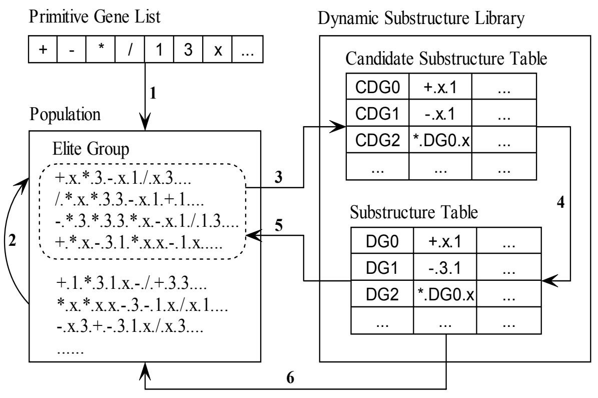

sqrt. *. +. *. a. *. sqrt. a. b. c. /. 1. -. c. d
Expansion
Compression
sqrt. *. +. *. a. *. DG0. b. c. /. 1. -. c. d
Expansion
Compression
Expansion
Compression
sqrt. *. +. *. a. DG2. c. /. 1. DG1

Figure 2. Illustration of compression and expansion operators

5. Dynamic Substructure Library

The dynamic substructure library maintains the emergent substructures as derived genes dynamically produced by the compression and expansion operators. Figure 3 illustrates the basic architecture of the library and its relationship with the rest of the P-GEP evolutionary process.

There are several important parameters defining the dynamic substructure library: (1) the library length, defining the maximum number of derived genes that can be hold at any time; (2) the required appearance frequency in the elite group, defining the criteria for a derived gene; (3) the maximum complexity level, defining the permitted maximum embedding hierarchy of the derived genes; (4) a candidate substructure table (CST), to store the potential emergent substructures from the current elite group; (5) a substructure table (ST), to store the qualified derived genes; and (6) a Boolean variable *bAppend*, which indicates whether or not to append derived genes to the primitive gene list. If it is a true value, derived genes have promoted usage in composing candidate solutions and more impact on evolution. These adjustable parameters together produce a viable scheme of incorporating useful emergent structure information into the P-GEP process.



Primitive Gene List
+
-
*
/
1
3
x
...
Dynamic Substructure Library
Candidate Substructure Table
CDG0 +.x.1 ...
CDG1 -.x.1 ...
CDG2 *.DG0.x ...
... ...
1
Population
Elite Group
+.x.*.3.-.x.1./.x.3....
/.*.x.*.3.3.-.x.1.+.1....
-.*.3.*.3.3.*.x.-.x.1./.1.3....
+.*.x.-.3.1.*.x.x.-.1.x.....
+.1.*.3.1.x.-./.+.3.3....
.x..x.x.-.3.-.1.x./.x.1....
-.x.3.+.-.3.1.x./.x.3....
......
2
3
4
5
6
Substructure Table
DG0 +.x.1 ...
DG1 -.3.1 ...
DG2 *.DG0.x ...
... ...

The process denoted by each numbered arrow:
1. Generate the initial population and continuously supply genetic material;
2. The ordinary P-GEP evolutionary process;
3. Generate candidate substructures;
4. Update the substructure table with the new eligible derived genes;
5. Update the elite group with the new substructure table;
6. Supply derived genes for the subsequent evolution.

Figure 3. An overview of the dynamic substructure library

In addition to the ordinary P-GEP process described in Section 2.1, there is an additional substructure library updating procedure if the best individual of the current generation is improved beyond the previous generations. This procedure consists of three phases: (1) compressing and producing the candidate emergent substructures from the elite group; (2) updating the substructure library with the new set of eligible derived gene. This is done through sorting the derived genes in both ST and CST by their total appearance frequency in the current elite group, and picking up the top list to refresh the substructure library; and (3) updating the representation of the elite chromosomes by replacing the occurrences of library substructures with the corresponding derived gene names as well as expanding the obsolete derived genes.

6. Experiments and Discussions

The technique of P-GEP with substructure library is generally aimed at tackling the data mining problems with complex underlying solution structures. However, for the purpose of testing, regression problems are especially suitable since they have a single target solution being evolved, which can better reveal how the emergent substructures relate to the solution structures.

6.1. A Simple Structured Regression Problem

We first experimented with an artificially structured simple regression problem to verify our hypothesis about the existence of emergent substructures, and the advantage of incorporating them into the evolution. The testing problem is shown in (6.1). A set of twenty-one fitness cases equally spaced along the x axis from -10 to 10 inclusively were chosen for the experiments.

$$y = 3 * (x+1)^3 + 2 * (x+1)^2 + (x+1) \qquad (6.1)$$

(1) Experiment settings

The techniques under investigation included P-GEP, P-GEP with the dynamic substructure library and with (denoted as P-GEP_Add) or without (denoted as P-GEP_noAdd) adding derived genes into the primitive gene list. The general experiment setup is summarized as follows: the chromosome size is 128; the population size is 1000; the maximum number of generations is 500; the crossover probability is 0.7, and the mutation and rotation probability is 0.02; function and terminal sets are selected as {+, -, *, /} and {1, 2, 3, 5, 7} respectively; the number of elites is 4. Additionally, the parameters of the substructure library for P-GEP_noAdd and P-GEP_Add are: library length is 5 and maximum complexity level is 1; the required appearance frequency is 3 for P-GEP_noAdd and 4 for P-GEP_Add.

(2) Experimental results

Finding 1: P-GEP algorithms with the substructure library have better fitness convergence curves. Since the ultimate goal is to find the optimal solution effectively, we first examine the general fitness convergence curves revealed by the evolutionary process of every technique. In these experiments the fitness of a solution is defined as the residual, which is better when smaller. For each run of the experiment, the best fitness value of every generation is recorded. Furthermore, the inspection of the log files of the program finds out that the most significant residual reduction at the early generations are the pure result of the general P-GEP evolutionary process. In other words, for P-GEP_noAdd and P-GEP_Add, this kind of residual reduction usually happens before any derived gene is pulled into the substructure library. This is reasonable since the initial population is highly diverse due to the random generation procedure and the elites seldom share any substructure. Therefore, we have plotted the average of this best fitness value over ten runs from generation 100 to 500 for the P-GEP algorithms in Figure 4.

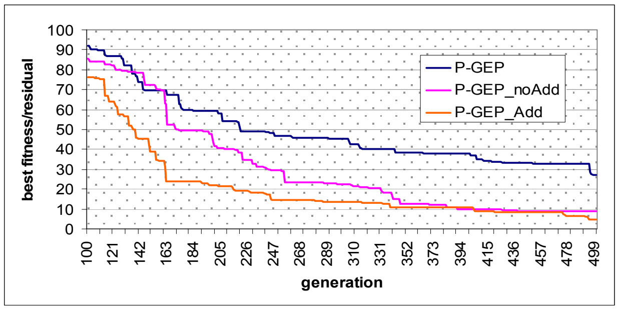


Figure 4. The fitness convergence curves of P-GEP algorithms

The curves exhibited in Figure 4 show the better performance of the P-GEP algorithms with a substructure library. Therefore, preserving and utilizing the emergent substructures is a promising approach to further exploit the advantages of P-GEP's linear genotype. And in terms of the finally converged average best fitness value, the two versions of the P-GEP algorithms with a substructure library seem to be competitive with each other. However, further examination of the individual trials shows that P-GEP_Add actually has found an ideal solution (which exactly fits the given problem with a zero residual) for the problem three times out of ten, while none of the other techniques achieved even a single ideal solution.

Finding 2: dynamic substructure library helps find useful derived genes. We further examined the three ideal solutions evolved by the P-GEP_Add to see whether derived genes helped compose these solutions. The final solutions and their corresponding derived genes present in the substructure library as of the last generation are reported in Table 2, where it is clear that every ideal solution benefited from the derived genes in its formation during the evolution. Some very meaningful substructures were discovered, such as "*.x.x" and "*.x.3", which are apparently useful functional components of the solution.

Table 2. Example of the evolved optimal solutions composed of derived genes

Evolved ideal solution #1	
Solution	((3*(x*((DA4*x)+5)))-DA3)+(7-(1+x))
Derived genes	**DA0**: *.x.1; **DA1**: /.3.5; **DA2**: +.x.5; **DA3**: *.x.x; **DA4**: -.DA2.1
Evolved ideal solution #2	
Solution	(3*((DA1/(1/DA2))-(3-(5+(x*1)+x))))-DA2
Derived genes	**DA0**: *.x.x; **DA1**: +.x.3; **DA2**: +.DA0.x; **DA3**: /.x.5; **DA4**: -.DA0.1
Evolved ideal solution #3	
Solution	((DA3*DA2)+1)-(3-((DA2+1)+DA1))
Derived genes	**DA0**: *.x.3; **DA1**: +.x.2; **DA2**: +.DA0.5; **DA3**: *.DA1.x

Another fact revealed by Table 2 is that useful emergent substructures can take every possible form as long as they can work together with other genes to produce the final solutions. For some other runs of the experiments the most suitable and natural substructure "+.x.1" was observed to emerge as a derived gene in the middle of the evolution. However, it sometimes simply faded from the substructure library due to its unpopularity in the elite group at later generations, or was not fully utilized by the evolution in composing a candidate solution. Future research should investigate these cases.

6.2. Benchmark Testing

We further assessed the overall performance of P-GEP with the dynamic substructure library technique by testing on a set of six benchmark regression problems (shown in Table 3) cited from WEKA knowledge analysis tool [10].

Table 3. Summary of the regression datasets

Dataset	# of attributes	# of cases	Target variance
fruitfly	5	125	15.879
quake	4	2178	0.189
sensory	12	576	0.823
strike	7	625	560.660
housing	14	506	9.197
puma8NH	9	8192	5.622

(1) Experiment settings

This time we evaluated P-GEP_Add in comparison with P-GEP, and the model tree (denoted as MT) implemented in WEKA (a standard regression method for inducing piecewise models). The general experiment setup is the same as in the simple regression problem, except that function and terminal sets are selected as {IF, +, -, *, /, power, sqrt, log, exp, gauss, sigmoid, sin, cos, tan, atan} and {PI, 1, 2, 3, 5, 7} respectively. The parameters of the substructure library for P-GEP_Add are: library length is 10 and maximum complexity level is 1; the required appearance frequency is 3.

(2) Experimental results

Each dataset has been separated into a training set (with 80% of the data) and a testing set (with the remaining data) by applying the StratifiedRemoveFolds utility included in WEKA tool. For P-GEP methods, each experiment has been run ten times with the average and best results reported. Additionally, to make the results comparable across the different datasets, the relative residual (RR) is used for the measurement, as shown in (6.2), where n is the number of cases; v_i is the expected value; v_i' is the predicted value; and v is the average of the expected values. Table 4 gives the testing results.

$$RR = \sqrt{\left(\sum_{i=1}^{n}(v_i - v_i')^2\right) \Big/ \left(\sum_{i=1}^{n}(v_i - v)^2\right)} \qquad (6.2)$$

From Table 4, we can conclude the following for the conducted experiments: (1) on average P-GEP_Add performs consistly better than P-GEP, and the best runs of P-GEP_Add have also achieved smaller residuals than P-GEP for all of the datasets but fruitfly; (2) Although we have run only 500 generations, P-GEP_Add has already exhibited comparable performance to MT in terms of the average results for all of the datasets but puma8NH, and the best runs of P-GEP_Add usually produce a better solution than MT. Again, all of the best solutions found by P-GEP_Add include derived genes from the substructure library, which might indicate some critical factors for the nature of the problems. Moreover, P-GEP_Add induces a single model to fits the whole dataset, as opposed to a number of linear models for the sub-datasets in MT (which can easily go over twenty for a bit complex problems). This is usually desirable when the overall underlying trend of the data needs to be described.

Table 4. Summary of the predictive performance of P-GEP_Add, P-GEP and MT on the benchmark regression problems in terms of relative residual

Dataset	**P-GEP_Add**		**P-GEP**		**MT**
	Avg	**Best**	**Avg**	**Best**	
Fruitfly	1.0043	0.9904	1.0193	0.9884	1.0000
Quake	0.9979	0.9955	0.9990	0.9966	1.0028
Sensory	0.9661	0.9145	1.0402	0.9848	1.1050
Strike	0.9685	0.9245	0.9736	0.9388	0.9437
Housing	0.5274	0.4480	0.5608	0.5265	0.5040
puma8NH	0.7270	0.6682	0.7898	0.7733	0.5713
Average	0.8652	0.8235	0.8971	0.8681	0.8545

7. Conclusions and Future Work

This paper has introduced a method to decompose the evolution of solutions, by preserving and utilizing emergent substructures in P-GEP, which adopts a prefix notation based linear genotype to directly encode the solution structure. The overall implementation scheme is realized as P-GEP with a dynamic substructure library. The preliminary experiments have shown that emergent substructures do exist and can help construct the good solutions without sacrificing the flexibility of the search power inherent in the evolution.

Further research under this general topic will primarily include the follows: (1) benchmark testing for the proposed method as compared to the traditional ADF approach in GP; (2) the current definition for emergent substructures actually is the specification for derived attributes (which act as terminals after being compressed). More generally we can consider derived functions which extract the functional elements of the substructure into an abstract operator and parameterize the terminals. The evolutionary process can therefore be guided more toward the priority of discovering ideal solution structures.

8. References

[1] C. Ferreira, "Gene Expression Programming: a New Adaptive Algorithm for Solving Problems", *Complex Systems* 13(2), 2001, pp. 87-129.

[2] C. Zhou, W. Xiao, P.C. Nelson, and T.M. Tirpak, "Evolving Accurate and Compact Classification Rules with Gene Expression Programming", *IEEE Transactions on Evolutionary Computation* 7(6), 2003, pp. 519-531.

[3] D.E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley Pub. Co., 1989.

[4] J. Koza, *Genetic Programming II: Automatic Discovery of Reusable Programs*, MIT Press, Cambridge, MA, 1994.

[5] T. Yu and C. Clack, "PolyGP: A Polymorphic Genetic Programming System in Haskell", in *Late Breaking Paper at Genetic Programming Conference(GP-97)*, Stanford, CA, 1997.

[6] O. Brock, "Evolving Reusable Subroutines for Genetic Programming", in *Artificial Life at Stanford*, edited by J. Koza, Stanford, CA, 1994.

[7] P.J. Angeline, "Genetic Programming and Emergent Intelligence", in *Advances in Genetic Programming*, edited by K.E. Kinnear Jr., MIT Press, Cambridge, MA, 1994, pp. 75-98.

[8] J.P. Rosca and D.H. Ballard, "Hierarchical Self-Organization in Genetic Programming", in *Proceedings of the11th International Conference on Machine Learning*, Morgan Kaufmann, 1994.

[9] W. Banzhaf, D. Banscherus, and P. Dittrichc, "Hierarchical Genetic Programming using Local Modules", *Series Computational Intelligence, Internal Report of SFB* 531, No. 56/99, ISSN 1433-3325, University of Dortmund, Germany, 1999.

[10] I.H. Witten and E. Frank, *Data Mining: Practical machine learning tools with Java implementations*. Morgan Kaufmann, San Francisco, 2000.

[11] C. Ryan, J.J. Collins, and M. O'Neill, "Grammatical Evolution: Evolving Programs for an Arbitrary Language", in *EuroGP'1998(LNCS1391)*, Springer-Verlag, 1998, pp. 83-96.

[12] W.B. Langdon and W. Banzhaf, "Repeated Sequences in Linear GP Genomes", in *Late-breaking Papers at GECCO-2004*, Seattle, WA, 2004.

[13] M. Brameier and W. Banzhaf, "A Comparison of Linear Genetic Programming and Neural Networks in Medical Data Mining", *IEEE Transactions on Evolutionary Computation* 5(1), 2001, pp. 17-26.

[14] T. Perkis, "Stack-Based Genetic Programming", in *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*, Orlando, FL, 1994, pp. 148-153.

[15] L. Spector and A. Robinson, "Genetic Programming and Autoconstructive Evolution with the Push Programming Language", *Genetic Programming and Evolvable Machines* 3, Kluwer Acdemic Publishers, Netherlands, 2002, pp. 7-40.

[16] S.C. Roberts, D. Howard, and J.R. Koza, "Evolving modules in genetic programming by subtree encapsulation", in *Proceedings of EuroGP'2001(LNCS2038)*, Springer-Verlag, Lake Como, Italy, 2001, pp. 160-175.

[17] M. Keijzer, C. Ryan, and M. Cattolico, "Run Transferable Libraries – Learning Functional Bias in Problem Domains", in *Proceedings of GECCO-2004*, Seattle, WA, 2004, pp. 531-542.

[18] X. Li, C. Zhou, W. Xiao, and P.C. Nelson, "Prefix Gene Expression Programming", in *Late Breaking Paper at the Genetic and Evolutionary Computation Conference(GECCO-*2005), Washington, D.C., 2005.

[19] M. O'Neill and C. Ryan, "Grammar Based Function Definition in Grammatical Evolution", in *Proceedings of the Genetic and Evolutionary Computation Conference(GECCO-2000)*, 2000, pp. 485-490.