Review Article

Jinghui Zhong
School of Computer Science and Engineering, South China University of Technology, Guangzhou, CHINA

Liang Feng
College of Computer Science, Chongqing University, Chongqing, CHINA

Yew-Soon Ong
School of Computer Science and Engineering, Nanyang Technological University, SINGAPORE

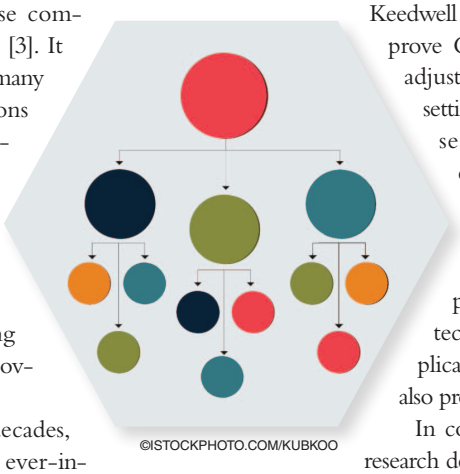
Gene Expression Programming: A Survey

Abstract

Gene Expression Programming (GEP) is a popular and established evolutionary algorithm for automatic generation of computer programs. In recent decades, GEP has undergone rapid advancements and developments. A number of enhanced GEPs have been proposed to date and the real world applications that use them are also multiplying fast. In view of the steadfast growth of GEP and its importance to both the academia and industry, here a review on GEP is considered. In particular, this paper presents a comprehensive review on the recent progress of GEP. The state-of-the-art approaches of GEP, with enhanced designs from six aspects, i.e., encoding design, evolutionary mechanism design, adaptation design, cooperative coevolutionary design, constant creation design, and parallel design, are presented. The theoretical studies and intriguing representative applications of GEP are given. Finally, a discussion of potential future research directions of GEP is also provided.

I. Introduction

Gene Expression Programming (GEP) is an established Evolutionary Algorithm (EA), which solves user-defined tasks by the evolution of computer programs [1], [2]. In GEP, computer programs are typically encoded by fixed-length gene expression strings that are evolved through nature-inspired operators such as mutation and crossover. Today, GEP has been verified as being effective in searching for accurate and concise computer programs [2], [3]. It has been applied to many real world applications with much success reported, including time series predictions [4], classification problems [2], regression problems [2], data mining and knowledge discovery [5]–[7], etc.



©ISTOCKPHOTO.COM/KUBKOO

Over the past decades, GEP has attracted ever-increasing attention from the research communities, leading to a number of enhanced GEPs proposed in the literature. For instance, Li et al. [8] proposed the use of prefix notation to represent computer programs, and this has successfully improved the search efficiency of GEP. Zhong et al. [7] extended the GEP chromosome representation by using Automatically Defined Functions (ADFs). In particular, new schemes such as the uniform design method [9], the Clonal Selection Algorithm (CSA) [10]–[12] and the weight tournament selection [13] have been used to improve the global search ability of GEP. In addition, some researchers have tried to integrate operators of other EAs into GEP to improve its search efficiency [7], [14], [15]. Mwaura and Keedwell [16] sought to improve GEP by adaptively adjusting the parameter settings along with the search, while Shao et al. [17]–[21] attempted to reduce the computational time of GEP via parallel computing technologies. The applications of GEP have also proliferated rapidly.

In contrast to the rapid research development of GEP, so far, few efforts have been made to review on the recent research progress of GEP in the literature. Among those available, Peng et al. [22] has conducted a preliminary survey of GEP. However, the review has focused only on the theoretical research works of GEP. There remains to be lacking a comprehensive survey of GEP that keeps track of the state-of-the-art GEP algorithms and one that discusses on the exciting research areas of GEP. This survey thus represents an attempt to fill in this gap.

To date, work on GEP can be divided into several unique areas. For the sake of brevity, in the present survey, we have classified them into eight groups.

Digital Object Identifier 10.1109/MCI.2017.2708618
Date of publication: 19 July 2017

Corresponding Author: Liang Feng (Email: liangf@cqu.edu.cn).

1556-603X/17©2017IEEE

These include encoding design, evolutionary mechanism design, adaptation design, cooperative coevolutionary design, constant creation design, parallel design, theoretical study, and last but not least, the applications of GEP.

Encoding design, which defines how a solution (i.e., a computer program) is encoded in the chromosome, is the first step to design a GEP. The encoding design has significant influence on the performance of GEP, since it determines the search space as well as the mapping between phenotypes and genotypes. Traditional GEP adopts the Karva expression or K-expression [1] to represent a computer program. This representation is of a fixed length, which is useful for generating concise computer programs [3]. However, the K-expression representation has several drawbacks. For example, good building blocks could be easily destroyed by the genetic operators if the solution is encoded by the K-expression [8]. In recent decades, various efforts have been undertaken in order to overcome the drawbacks of K-expression representation [7], [8], [23].

With respect to the evolutionary mechanisms, traditional GEP adopts several Genetic Algorithm (GA)-based operators, such as the random mutation and the one-point crossover, to evolve the chromosomes. These simple operators become inefficient when attempting to solve complex and large-scale optimization problems. Thus, the literature includes many efforts to enhance the evolutionary mechanisms of GEP. Some of them use mathematical methods such as the uniform design [9], [24], while others use new operators inspired by other EAs [7], [14], [15].

Adaptation design refers to the design of mechanisms to adaptively control the parameters of GEP. It is worth noting that GEP contains a number of control parameters, including the size of population, the length of chromosome, and the mutation rate, etc. The setting of these parameters has significant influence on the performance of the algorithm, thus the finding of proper parameter settings for problems with different features is not a trivial task. To address this issue, several parameter adaptation techniques have been proposed to date. These techniques attempt to adjust the parameter settings of GEP adaptively while the evolution progresses online [16], [25].

Cooperative Coevolutionary (CC) design is commonly used to improve EAs for solving large scale optimization problems. By decomposing the encountered problem into small scale subproblems and using separate EAs to solve the sub-problems cooperatively, CC design is capable of improving the search efficiency of various EAs on a number of problems [26]–[28]. In the literature, several efforts have also been found on applying CC design to improve GEP [29]–[31].

Furthermore, constant creation is an optional operator in GEP that aims to find numerical constants (e.g., the coefficients of formulae) for constructing accurate GEP solutions. Since numerical constant is an integral part of most mathematical formulas, the constant creation operator is useful for finding highly accurate solutions. However, the evolutionary mechanisms of GEP are unsuitable for evolving numerical constants. To tackle this problem, a number of attempts have been made to improve GEP by proposing new schemes which are capable of evolving both structures and numerical constants, simultaneously [32]–[37].

On the other hand, further reduction on the computational time of GEP via the integration of GEP with parallel and distributed computing platforms has become popular in recent years. Since GEP is a population-based search algorithm, it relies on the search approaches that work on the basis of manipulating representative samples of the search subregions within the solution landscape. Such inherently parallel or multi-track searches make GEP suitable for parallelism. So far, a number of parallel GEPs have been developed. Existing parallel GEPs mainly adopt the Master-Slave model [17], [19], [20] and the Island model [18], [21], [38].

The theoretical study on GEP has focused on the theoretical aspects of GEP, such as the proof of convergence and the estimation of convergence speed. This paper will discuss recent research on the theoretical studies of GEP. In addition, this paper also reviews several areas of application where GEP has been successfully applied. Since the applications of GEP have proliferated rapidly, our focus here is placed on five key representative application areas of GEP. Finally, based on the survey, this paper then discusses a number of interesting potential future research areas pertaining to GEP.

The purpose of the current survey is to present a comprehensive multi-facetted exposition of recent research in GEP. For the sake of completeness, the survey begins in Section 2 with a brief review on the background knowledge of GEP. Then, in Section 3, we review the state-of-the-art GEP variants from six perspectives as mentioned above. Section 4 examines the theoretical studies of GEP, followed by a review of the representative application areas of GEP in Section 5. In Section 6, we discuss the future research directions of GEP, and finally draw our conclusions in Section 7.

II. Background

This section provides a brief introduction of related background knowledge to aid readers in understanding the search mechanism of GEP. In particular, the traditional chromosome representation of GEP is described. The algorithm framework of GEP is also given in this section.

A. Chromosome Representation of GEP

Computer programs such as mathematical formulae and logical rules are commonly represented by expression trees (ETs) with two types of nodes: function and terminal. A function (e.g., +, sin) is a node that contains a single child or more which represent the input arguments of this function. For example, a node sin is a function node since it has one child representing its input argument. A terminal is a leaf representing a variable or a constant. In GEP, a computer program is encoded by one or multiple genes. Each gene is a fixed-length string that comprises two parts. The first part (called the *head*) is formed

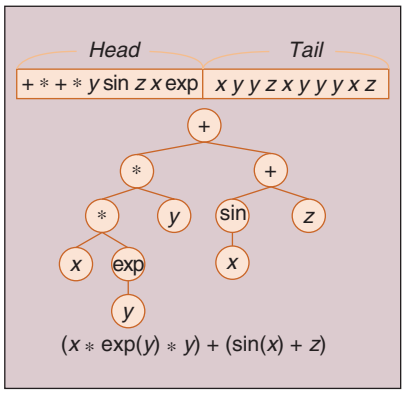


FIGURE 1 An example chromosome of GEP and the decoded expression tree.

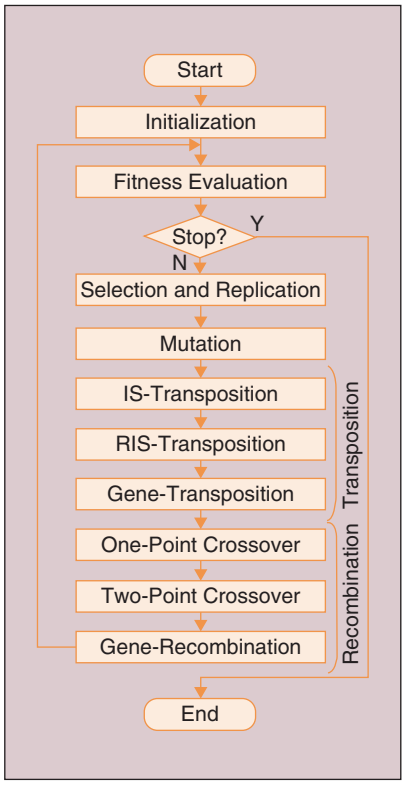


FIGURE 2 The procedure of traditional GEP.

by functions and terminals, while the second part (called the *tail*) is formed by terminals only. Each gene can be converted to an equivalent sub-function by using a width-first search scheme. The sub-functions encoded by genes are linked to form a final solution by a linking function (e.g., +), which is defined in advance or determined by a certain adaptation strategy. To ensure that any chromosome can be translated to a valid expression tree, the lengths of the *head* (h) and *tail* (l) are imposed with the following constraint:

$$l = h \cdot (u - 1) + 1, \tag{1}$$

where u is the maximum number of children of the functions.

A typical example of chromosome with a single gene is given by:

$$[+, *, +, *, y, \sin, z, x, \exp, x, y, \\ y, z, x, y, y, y, x, z]. \tag{2}$$

This chromosome can be converted to an expression tree as shown in Fig. 1, and the solution represented by the chromosome can then be decoded as:

$$(x * \exp(y) * y) + (\sin(x) + z). \tag{3}$$

In the GEP chromosome, the number of nodes in the final ET must not exceed the predefined length of the chromosome. Thus, this representation has a tendency of producing short programs and avoiding the bloating limitation of GP.

B. Algorithm Framework of GEP

Fig. 2 illustrates the procedure of traditional GEP, which consists of the following operations.

Initialization: The first step is to generate a set of random chromosomes to form the initial population. For every chromosome in the initial population, each element of the fixed-length strings is randomly assigned based on the type of the element. Elements belonging to the *head* part are assigned functions and terminals, while elements belonging to the *tail* part are assigned terminals.

Fitness Evaluation: In this step, the fitness values of all chromosomes in the population are evaluated. The fitness evaluation function is problem specific and has a great impact on the performance of the algorithm.

Selection and Replication: This step selects better chromosomes in the population to form a new population for the next generation. There are various selection strategies that can be used, such as the roulette-wheel selection strategy and tournament selection strategy [39]. It has been shown that the roulette-wheel selection with elitism offers better performances in solving complex problems.

Mutation: In the mutation operation, every element at any position of each chromosome is subjected to a random change with a predefined mutation rate (p_m). If an element is to be mutated, it can only be assigned a feasible random symbol according to its type. For example, a *tail* element can only be assigned a terminal.

Transposition: The transposition operation aims to replace some consecutive elements of the chromosome with a segment of consecutive elements in the same chromosome. It contains three sub-steps, which are performed with probabilities of p_{is}, p_{ris}, and p_g, respectively.

- *IS-transposition*: An Insertion Sequence (IS) is a segment of consecutive elements in the chromosome. In this sub-step, an IS is randomly selected. Then a copy of the IS is made and inserted at a random position in the head of a gene. The inserted position should not be the start position of a gene because a gene with a terminal at the root is of little use. After that, the sequence downstream from the insertion point would be replaced by the IS. An example of the IS-transposition is illustrated in Fig. 3, where the chromosome under consideration contains two genes. In the IS-transposition, an IS (i.e., "baa") is randomly chosen from the chromosome. The insertion point (e.g., the 5th position of the 1st gene) is then randomly selected from the head parts of all the genes. Lastly, the elements of the head part downstream from the insertion point are replaced by the selected IS.
- *RIS-transposition*: A Root Insertion Sequence (RIS) is a segment of consecutive elements that starts with a function. Hence, RISs are chosen from the *heads* of genes. In this sub-step, the chromosome and the gene to be modified, the start position of the RIS and its length, are randomly chosen. Once an RIS is selected, a copy of the RIS is made and inserted into the root of the selected gene, as done in the IS-transposition step. Fig. 3 illustrates the RIS-transposition, where the RIS (i.e., "*a+") is selected from the *head* of the 1st gene. Then, the RIS is inserted into the 1st position of the 1st

gene. The original *head* is shifted to accommodate the RIS, and thus the last symbols of the *head* are deleted.

- *Gene-transposition*: In gene transposition, the chromosome to be modified is randomly chosen. Then, a random gene (but not the first gene) of the chosen chromosome is selected and transposed to the beginning of the chromosome. Fig. 3 presents a particular example of the gene-transposition, where the chromosome under consideration contains only two genes. In this example, the 2nd gene is selected and transposed to the beginning of the chromosome.

Recombination: The aim of the recombination operation is to exchange the gene information of two parent chromosomes to generate two offspring. This operation also contains three sub-steps. These are one-point recombination, two-point recombination, and gene recombination. In gene recombination, a gene of one parent is selected randomly. Then, the selected gene is exchanged with the corresponding gene of the other parent to generate two offspring. In the recombination, the three sub-steps are performed with probabilities of p_{r1}, p_{r2}, and p_{rg}, respectively.

After the recombination operation, a new population, which has the same size of the parent population, is generated. Then the algorithm turns to the fitness evaluation step, and the evolutionary process continues until the termination conditions (e.g., reaching maximum generation or achieving a satisfactory solution) are met.

C. GEP vs GPs and other EAs

GEP can be considered as a special variant of GP. The major difference between GEP and other GP variants lies in the chromosome representation. Traditionally, in GPs, computer programs are represented using expression trees implemented by tree representation. To ensure that any chromosome is valid and promising, specific operators are required for initialization and reproduction (e.g., the *full* and *grow* methods [40]). This makes the implementation of GP more difficult than other EAs. Besides, as the evolution goes on, the size of the expression trees will grow quickly, which results in the "Bloating Problem". To avoid the above problems, GEP uses a linear representation to encode computer programs. The chromosomes of GEP are easier to genetically manipulate. Further, as the size of the chromosome is fixed, the size of the decoded expression tree will never exceed a predefined upper-bound. Therefore, GEP is capable of providing relatively concise solution to the given problem. However, to facilitate the search for high quality solutions, the chromosome size should be configured appropriately for the specific problem, which is often a non-trivial task.

There are also other GP variants that adopt a linear representation, such as Grammatical Evolution (GE) [41] and Linear Genetic Programming (LGP) [42]. Oltean and Grosan [43] have compared the differences between several GPs with linear representation. In general, each GP variant possesses its strengths and weaknesses. GE uses the BNF grammars for solution encoding and decoding, which makes it suitable for evolving programs that can be expressed as BNF rules. However, the decoding process of GE differs and tends to be more complex than that of the GEP. Besides, as some chromosomes of GE translate to invalid expressions, special mechanism to cope with such expressions is needed. In LGP, the computer program evolved consists of a series of low-level instructions and a number of registers. Thus, LGP is more suitable for evolving low-level programs that are run directly on the computer processor, while GEP is more suitable for evolving high-level programs that are human interpretable.

Compared with other EAs such as GA, Particle Swarm Optimization (PSO), and Differential Evolution (DE), the major difference of GEP lies in the form of solution provided. Generally, the solutions provided by GEP are computer programs (e.g., mathematical formula, classification rule, and heuristic rule) which are generated based on function set and terminal set. Meanwhile, in other EAs, the solutions usually are a vector of values, and they do not need function and terminal sets to construct solutions.

III. Algorithm Design Taxonomy

Over the past decades, a number of enhanced GEPs have been proposed. As shown in Fig. 4 and Table 1, these enhanced GEPs can, generally speaking, be classified into six classes. These are encoding design, evolutionary mechanism design, adaptation design, cooperative coevolutionary design, constant creation design, and parallel design.

A. Encoding Design in GEP

Encoding design determines how a solution is encoded in the chromosome, which has important influence on the performance of GEP. In traditional GEP, a computer program is encoded by using a K-expression [1]. A K-expression string can be converted to an ET by using a width-first travelling procedure. Since all chromosomes are of a fixed length, the GEP is capable of generating concise solutions. In addition, the fixed-length representation facilitates it for genetic operators to manipulate the chromosomes.

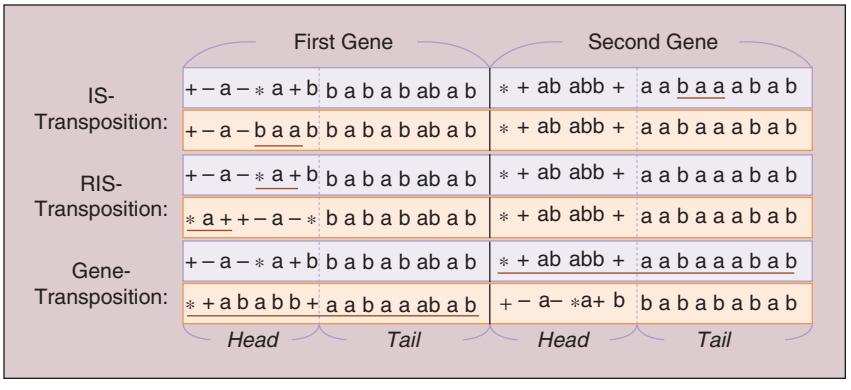


FIGURE 3 Illustration of the transposition operations of GEP.

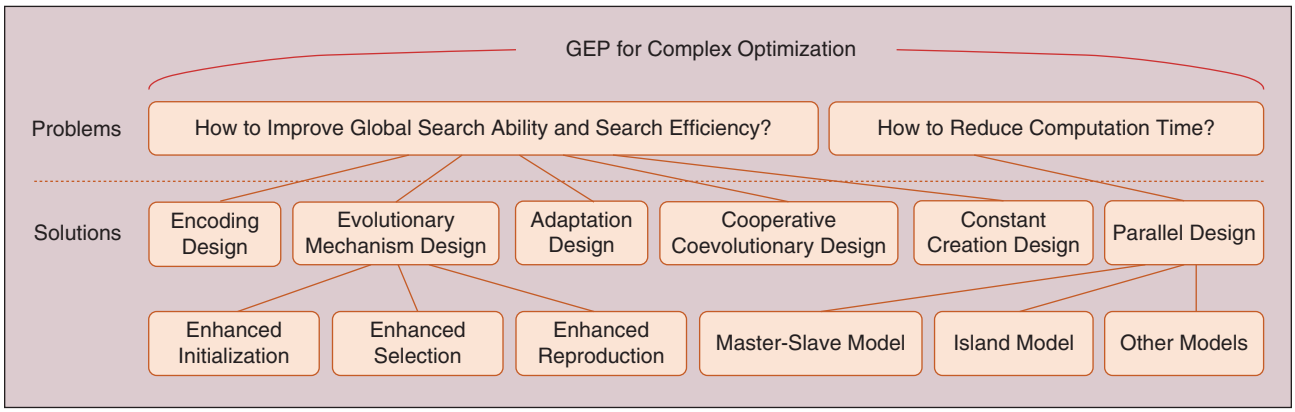


FIGURE 4 Outline of the GEP design taxonomy.

However, the K-expression is very fragile, because good building blocks found can be easily destroyed by the genetic operators such as crossover. To solve this problem, Li et al. [8] proposed a Prefix Gene Expression Programming (P-GEP). The phenotype of P-GEP (i.e., the ET) is the same as that of GEP and the genotype of P-GEP is still a linear string of fixed-length. However, in P-GEP, the phenotype is obtained by travelling the genotype in the depth-first scheme instead of the width-first scheme, as in GEP. For example, suppose a chromosome of P-GEP has the same gene as expressed in (2). Then, the corresponding ET of P-GEP can be obtained as illustrated in Fig. 5, and the final solution can be decoded as:

$$(y * \sin(z) + x) * \exp(x) + y. \qquad (4)$$

It can be observed from (3) and (4) that the same chromosome can lead to quite different solutions by using the decoding methods of GEP and P-GEP, respectively. The authors compared P-GEP with GEP on a symbolic regression problem and several classification problems. The experimental results demonstrated that the prefix notation in P-GEP is more protective for substructures and is capable of improving the search efficiency.

An effective approach to improve GEP is to incorporate high-order knowledge that can accelerate the search. One of the commonly used types of high-order knowledge is the ADF proposed by Koza [52]. ADFs are sub-functions embedded in chromosomes that can be evolved automatically along the search and be used as building blocks to construct the final solution. In the literature, ADFs have been shown to be effective for improving the search efficiency of GPs [52]–[55]. As GEP is a variant of GP, efforts have also been made to integrate ADF with GEP to enhance the search process. For example, in [23], Ferreira proposes to utilize ADFs in GEP by introducing a representation called GEP-ADF. In GEP-ADF, each chromosome consists of a number of conventional genes (i.e., ADFs) and a homeotic gene (i.e., main program), which are illustrated in Fig. 6. All of the conventional genes and the homeotic gene are represented using K-expression. The main program combines different ADFs through linking functions (e.g., +, $*$) for generating the final output. The function set and terminal set of ADFs are the same as that in the traditional K-expression representation. On the other hand, the function set and terminal set of the homeotic gene are the linking function and ADFs, respectively. A typical example chromosome of GEP-ADF can be expressed as: $[*, *, -, y, x, x, z, +, +, -, x, y, z, x, *, *, \sin, 1, +, 2, 2, 1, 2, 1, 2]$. This chromosome encodes two ADFs and one homeotic gene, and the decoded ETs of the chromosome are illustrated in Fig. 6. It can be observed that both ADF_1 and ADF_2 are used twice in the main program. However, it is worth noting that ADFs in GEP-ADF can only be used as terminals of the homeotic gene and the ADFs contain no input argument. These features make GEP-ADF inefficient or non-scalable for complex problems.

To further improve the performance of GEP-ADF, Zhong et al. [7] proposed a new representation called C-ADF, which also tried to use ADFs in GEP. In C-ADF, each chromosome consists of a main program and several ADFs. The main program and ADFs are encoded using the K-expression. However, the building blocks (i.e., function set and terminal set) of the main program and ADFs are different. In particular, for the main program, the function set is composed of functions and ADFs, and the terminal set contains variables and constants. Meanwhile, the function and terminal sets of ADFs consist of functions and input arguments, respectively. Fig. 7 illustrates a chromosome with C-ADF representation. In this example, the ADFs are used three times in the main program. It can further be observed from Fig. 7 that, in C-ADF, the ADFs have input arguments that can come in the form of variables (e.g., x and z), constants (e.g., π), ADFs, or any sub-tree of the main program. In this way, complex solutions can be represented in a concise and readable manner. Experimental results on benchmark symbolic regression problems and even parity problems showed that the C-ADF is effective to improve the search efficiency and to reduce the complexity of solutions.

Quan and Yang [44] proposed a Directed Acyclic Graph (DAG) representation method to improve GEP. The DAG chromosome consists of two parts, namely, the main chromosome and the topological chromosomes. The main chromosome consists of a list of functions

TABLE 1 Algorithm Design Taxonomy.

ENCODING DESIGN	K-EXPRESSION [1]
	PREFIX GENE EXPRESSION [8]
	GENE EXPRESSION WITH ADF [23]
	C-ADF [7]
	DIRECTED ACYCLIC GRAPH REPRESENTATION [44]
EVOLUTIONARY MECHANISM DESIGN	GEP WITH INITIALIZATION USING UNIFORM DESIGN [24]
	GEP WITH INITIALIZATION USING GENE SPACE BALANCE STRATEGY [45].
	GEP WITH WEIGHT TOURNAMENT SELECTION [13]
	GEP WITH COMPONENT THERMODYNAMICAL SELECTION [46]
	GEP WITH CLONAL SELECTION ALGORITHM [11], [12], [47], [48]
	GEP WITH ROTATION OPERATION [3]
	GEP WITH DE OPERATORS [7]
	GEP WITH SA OPERATORS [14]
	GEP WITH AIS OPERATORS [15]
	GEP WITH A DISTANCE-BASED CROSSOVER [49]
	GEP WITH AN ADAPTIVE CROSSOVER [24]
	GEP WITH CROSSOVER USING ORTHOGONAL DESIGN [50]
ADAPTATION DESIGN	ADAPTIVE GEP WITH A GENEMAP [24]
	ADAPTIVE GEP WITH FEEDBACK HEURISTIC [16]
COOPERATIVE COEVOLUTIONARY DESIGN	CCGEP WITH ADFS FOR CLASSIFICATION [29]
	CCGEP FOR THE 3D PROCESS PLANT LAYOUTS PROBLEM [31]
	CCGEP FOR THE DISTRIBUTORS PALLET PACKING PROBLEM [30]
CONSTANT CREATION DESIGN	CONSTANT CREATION USING A DC DOMAIN [32]
	CONSTANT CREATION BASED ERC METHOD [7]
	CONSTANT CREATION BY USING LOCAL SEARCH METHOD [34]–[36]
	CONSTANT CREATION BY USING EXTENDED ENCODING AND DE OPERATORS [37]
	CONSTANT CREATION WITH CREEP MUTATION AND RANDOM MUTATION [33]
PARALLEL DESIGN	PARALLEL DESIGN USING MASTER-SLAVE MODEL [17], [19], [20]
	PARALLEL DESIGN USING ISLAND MODEL [18], [21], [38]
	PARALLEL DESIGN USING AGENT-BASED MODEL [51]

and terminals, while the topological chromosome describes the topology of DAG attached to the main chromosome. A typical DAG representation is illustrated in Fig. 8. In this example, the first symbol in the main chromosome is "∗", and the corresponding values in the topological chromosomes (i.e., the two integers under the first symbol "∗") are 8 and 6. This means that the input arguments of function "∗" are the 8th and 6th symbol in the main chromosome (i.e., "+" and "a"). In this way, we can build the sub-tree for each symbol in the main chromosome. If the symbol in the main chromosome is a terminal, the built sub-tree contains only one node. Each sub-tree represents a candidate solution of the encountered problem of interest, and the best sub-tree is selected as the final solution that is encoded by the entire chromosome. It is worth noting that the DAG representation can store multiple solutions in a single chromosome, which is the same as the MEP proposed in [56]. However, since all sub-trees encoded in the main chromosome need to be evaluated, expensive computational cost is required in DAG.

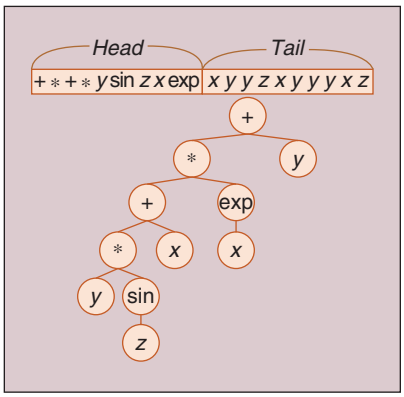


FIGURE 5 An example chromosome of P-GEP.

In summary, the traditional K-expression and the prefix gene expression are simple and easy to implement. However, they may lead to complicated solutions since they cannot reuse high-order building blocks for solution construction. By considering ADFs as terminals, the GEP-ADF is capable of reusing high-order building blocks (i.e., the ADFs) to construct solutions, even though the ADFs cannot be utilized to construct more effective building blocks. Further, the C-ADF representation is more flexible and effective than GEP-ADF since it considers ADFs as functions rather than terminals. The DAG representation can encode multiple solutions in a single solution, which is useful for exploiting the search space. However, it requires extra computational cost for solution evaluations.

B. Evolutionary Mechanism Design in GEP

1) *Initialization*: Initialization is the first step in the evolution process. In this step, a population of random individuals is generated. To gain sufficient population diversity for a global search, the initial population should be well scattered in the whole search space. However, the traditional method generates the initial

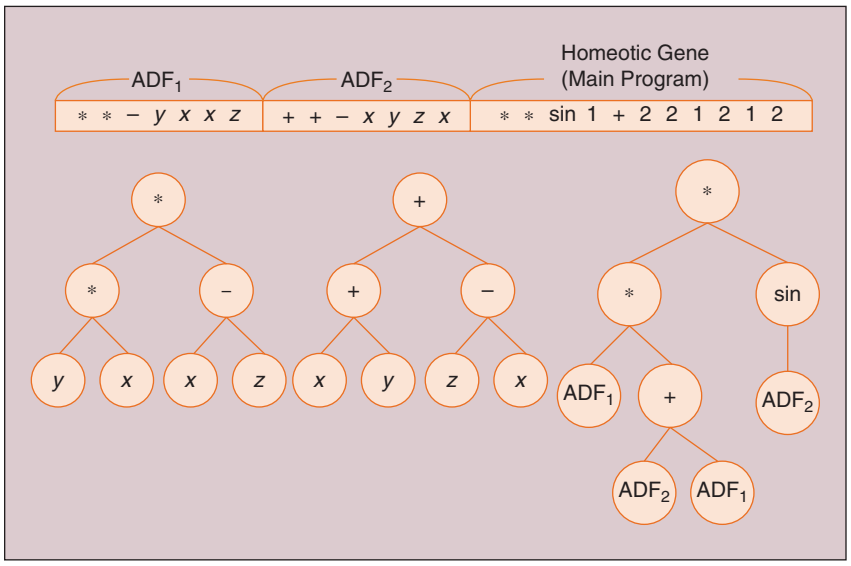


FIGURE 6 An example chromosome of GEP-ADF.

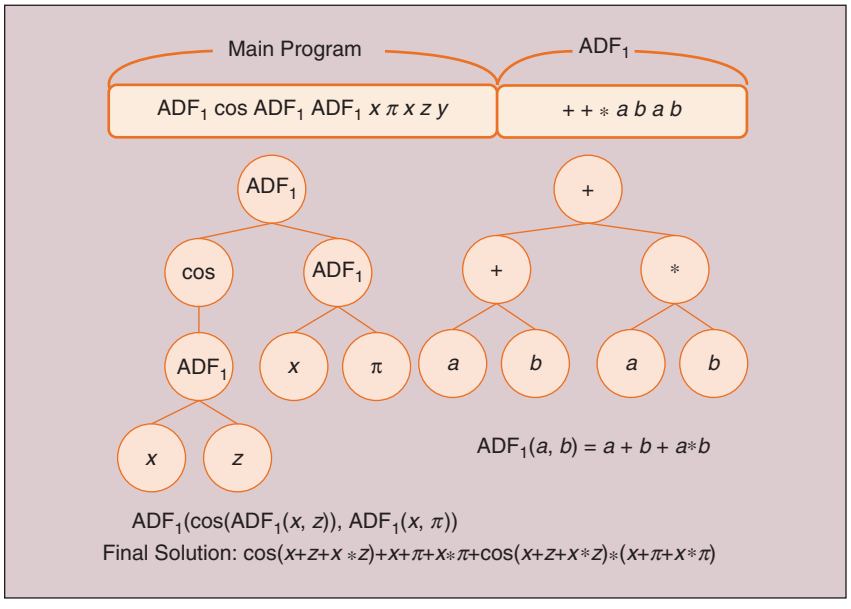


FIGURE 7 An example chromosome with C-ADF representation.

population randomly. Such method is not effective enough to create well-scattered initial population. To address this issue, Chen et al. [24] proposed to use the uniform design method [9]. The uniform design seeks design points that are uniformly scattered throughout the search space. To use the uniform design for initialization, a uniform table is created in advance. The uniform table can be expressed by a matrix $U_M(Q^S)$, where S is the factors, Q is the levels, and M is the number of sample combinations selected from the whole space Q^S. Equation (5) gives a typical example of a uniform matrix with 5 factors and 7 levels.

$$U_7(7^5) = \begin{bmatrix} 2 & 4 & 3 & 7 & 5 \\ 3 & 7 & 5 & 6 & 2 \\ 4 & 3 & 7 & 5 & 6 \\ 5 & 6 & 2 & 4 & 3 \\ 6 & 2 & 4 & 3 & 7 \\ 7 & 5 & 6 & 2 & 4 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}. \quad (5)$$

It can be observed from (5) that the number of levels is the same in each column (i.e., each column has 7 levels in (5)). The uniform table can then be filled with functions and terminals to generate an initial population. Each row of the uniform table represents a chromosome, and each column represents a dimension in the chromosome. Suppose 1, 2, 3, 4, 5, 6, and 7 represent $+, -, *, /, x, y$, and z, respectively, then the two chromosomes generated by the first two rows of (5) are $-, /, *, z, x$ and $*, z, x, y, -$, respectively. Since the chromosome of GEP is divided into *head* and *tail* parts, the authors of [24] constructed a mix-level uniform table to initialize the population, where the *head* can choose more levels than the *tail*.

Similarly, to improve the diversity of the initial population, Hu et al. [45] proposed a Gene Space Balance Strategy (GSBS). The key idea of GSBS is to perform an additional checking procedure after the traditional population initialization process. In this procedure, the dimensions of the chromosomes are checked one by one. For each dimension, if an element has a probability which is higher than the average value, it will be replaced by the element that appears to have the lowest probability. In this way, the elements can be scattered as uniformly as possible in the initial population.

2) *Selection*: Selection operator implements the "survival of the fittest" principle in nature. This operator is used to guide the population to evolve towards the desired direction. The roulette-wheel selection and the tournament selection are two most commonly used selection schemes in the EA community. The traditional GEP adopts the roulette-wheel selection with elitism. In general, various selection schemes provide different levels of selection pressure and can lead to quite different convergence behaviors of the population. Several efforts have been made to design more effective selection schemes for GEP.

Wang et al. [13] proposed a weight tournament selection to improve the population diversity. In the weight tournament selection, a probability vector was introduced to represent the selection probabilities of individuals in the population. To calculate the probability vector,

the authors defined a measure to evaluate the similarity between two individuals s_1 and s_2 by

$$Similarity(s_1, s_2) = \frac{2 * MaximumOverlap}{l(s_1) + l(s_2)}, \tag{6}$$

where $MaximumOverlap$ represents the maximum overlap between s_1 and s_2, s_i represents the length of coding regions in s_i (i.e., the redundant bits in s_i will not be considered in calculating $l(s_i)$). Then, the selection probability of the ith individual is calculated by

$$P_i = \frac{1 - Similarity(s_i, s_{best})}{\sum_{j=1}^{PopulationSize} (1 - Similarity(s_j, s_{best}))}, \tag{7}$$

where s_{best} is the historical best individual. In this way, the more similar an individual is to the historical best individual, the lower selection probability of the individual has. For example, suppose the historical best individual is $\{*, *, x, -, x, y, y, x, y\}$, and an individual in the population is $\{*, x, -, y, x, x, x, y, x\}$. Then, the valid segments of these two individuals are $\{*, *, x, -, x, y\}$ and $\{*, x, -, y, x\}$, respectively. We can calculate the maximum overlap between these two individuals as 3 by comparing each symbol of $\{*, x, -, x, y\}$ and $\{*, x, -, y, x\}$. Thus, the similarity between these two chromosome can be calculated as $(2 * 3)/(6 + 5) = 6/11$. In this way, we can calculate the similarity between each individual and the historical best individual. The similarity values are used as weights to calculate the selection probability, which is then used in the tournament selection. The individual with larger similarity value has a higher selection probability. Once the required number of individuals has been selected for tournament competition, the individual with the best fitness value wins the competition and survives to the next generation. In general, the weight tournament selection only reduces the probability of reproducing an individual with a structure that is similar to the historical best individual so as to increase the population diversity and to avoid being trapped into the local optimum.

Guo et al. [46] proposed a new Component Thermodynamical Selection (CTS) to balance the selection pressure and the population diversity. The CTS is inspired by the principle of minimal free energy in thermodynamics. In the selection process, the newly created offspring are combined with the parent population at first. The free energy component of each individual is then computed by using the mathematical formula defined by the authors. Finally, individuals with larger free energy components are removed, and the remaining individuals form the population for the next generation.

The Clonal Selection Algorithm (CSA) proposed by De Castro and Von Zuben is a new EA inspired by basic immunological principles [10]. The CSA has been validated as effective for solving complex optimization problems. Over the past few years, efforts have been made to integrate CSA and GEP so as to accelerate the search efficiency. For example, Karakasis and Stafylopatis [47] proposed an ECA algorithm based on GEP and CSA [10]. The ECA algorithm adopts the representation of GEP to encode chromosomes, while the search mechanism of the CSA is utilized to evolve the chromosomes. Similarly, Guan et al. [11] and Litvinenko et al. [12] proposed enhanced GEPs by replacing the traditional operators of GEP with those of the CSA so as to maintain population diversity. Liu et al. [48] proposed a hybrid selection strategy. In the hybrid selection, the clonal selection is used to select parent individuals for reproduction, and the roulette-wheel selection method is used to select individuals to form the new population.

3) *Reproduction*: The traditional GEP contains seven operators to generate offspring, including mutation, IS-transition, RIS-transition, gene-transition, one-point crossover, two-point crossover and gene-recombination, which make it hard to configure a GEP for solving a problem of interest efficiently and effectively. Taking this cue, Zhou et al. [3] simplified GEP by using three reproduction operators, namely, crossover, mutation and

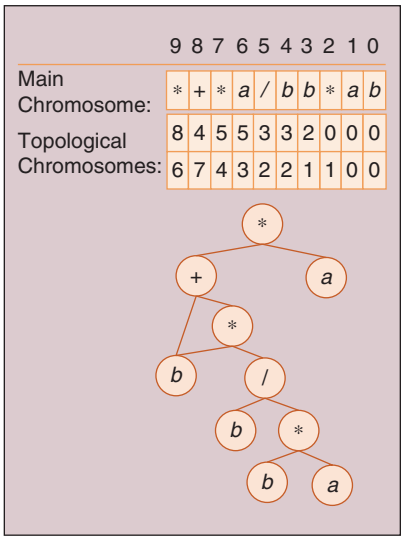


FIGURE 8 An example chromosome with DAG representation.

rotation. As shown in Fig. 9, the one-point crossover and two-point crossover are adopted in this simplified GEP. Both crossover operators are working in the same way as in the traditional GA. In the mutation process, a symbol in the chromosome is mutated to a random symbol in probability. In the rotation operation, the chromosome is divided into two parts with a randomly selected point. Next, the two parts are exchanged to form a new offspring. Please note that the operators of the simplified GEP would generate invalid chromosomes which cannot be translated to an expression tree. To tackle this problem, Zhou et al. [3] proposed to apply a validity test operation to check if a chromosome is valid or not. In particular, if a chromosome produced by the genetic operator cannot pass the test, the operator will be executed repeatedly until the generated chromosome is valid.

Recently, researchers attempted to improve the reproduction process of GEP by using operators of other EAs. For example, Zhong et al. [7] proposed a Self-Learning GEP (SL-GEP) that extends the operations of Differential Evolution (DE) for the evolution of chromosomes. DE is a well-known EA, which has been shown to have strong global search capability in continuous optimization problems. In SL-GEP, the crossover and mutation operators of DE are extended to evolve computer

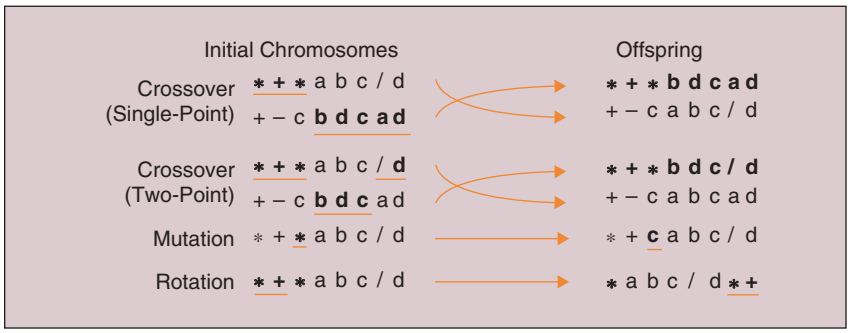


FIGURE 9 Genetic operators of the simplified GEP in [3].

programs, which are encoded in discrete search space. By using the newly defined operators, the number of control parameters in SL-GEP has been reduced and the performance of SL-GEP has been enhanced when compared to GEP on a number of symbolic regression problems and even-parity problems. Further, Jiang et al. [14] proposed a new GEP named GEPS that combines GEP and Simulated Annealing (SA). Similarly, Zeng et al. [15] proposed an Immune GEP (IGEP) based on the Artificial Immune System (AIS) for rule mining. The authors adopted the representation of GEP to encode chromosomes and operators of AIS to evolve the chromosomes.

Qu et al. [49] proposed two new crossover operators for GEP. The authors defined a metric to measure the distance between two individuals. One crossover selects the individuals with farthest distance for recombination, and the other selects the individuals with smallest distance for recombination. Experimental results on five symbolic regression problems indicated that the first crossover can bring more population diversity and better performance.

Chen et al. [24] proposed an adaptive crossover for GEP. In this new crossover, offspring are generated by combining gene segments of multiple parents. Suppose m chromosomes are selected to create an offspring. Each selected chromosome is divided into n exclusive genes and a uniform table is designed to sample n genes from the m chromosomes to form an offspring. The value of m changes adaptively based on the parents' current state. If the distance between the parent's fitness values (f_p)and the historical best fitness value ($f_{\max}$)gets larger, m will increase so as to enable communications among more parents. If the distance gets shorter, it may avoid more excessive mutations from excellent gene segments to have a uniform table of a smaller scale. The number of parents in the uniform table is determined by:

$$m_i = \left\| \left[1 - \frac{f_p}{f_{\max}} + \delta \right] \times m_{i-1} \right|, \qquad (8)$$

where $\delta \in (0, 1)$ and i is the current generation.

Yang and Ma [50] proposed an enhanced crossover with orthogonal design. In this new crossover operator, offspring are generated by combining segments from multiple parent chromosomes. Instead of evaluating all possible segments combinations from parents, the authors used an orthogonal table to generate a small number of representative combinations. Then, the representative combinations together with the parent chromosomes are ranked and the top m fitter chromosomes are selected as the outputs of the crossover.

In summary, the evolutionary mechanism of GEP consists of three major operators (i.e., initialization, selection, and reproduction). These operators play the key role in determining the search performance for solving the problem of interests. All of the existing works on enhanced evolutionary mechanism design try to better balance the exploration and exploitation abilities of GEP. The major difference among them lies in the operator to be extended (e.g., initialization, selection, and reproduction), the goal to be achieved (e.g., exploration, and exploitation), and the technique to be adopted (e.g., uniform design and orthogonal design).

C. Adaptation Design in GEP

In this section, we examine the adaptation design in GEP. In GEP, a number of control parameters, such as the size of population, the length of chromosomes, mutation rate, one-point crossover rate, etc., have to be defined for problem solving. However, it is well established that the performance of evolutionary search correlates positively to the incorporation of domain specific configuration [57]. That is, different problems require unique parameter configurations for evolutionary search, and a proper configuration could significantly enhance the problem-solving process. Thus, the researchers have successfully proposed adaptive methods to control the configurations of evolutionary search while search progresses [58]–[61].

In contrast to other evolutionary search methods such as GA, adaptation design in GEP received far less attention. Only a few studies have been proposed in the literature with regard to adaptive GEP design. Bautu et al. [24] presented an adaptive GEP called AdaGEP, which adaptively adjusts the number of genes used in the chromosome online. In AdaGEP, each gene is associated with a flag in a binary vector called genemap. The size of genemap is equal to the number of genes in a chromosome. The genes will be ignored in the decoding process if the corresponding flags in the genemap are assigned as zero. This genemap is evolved by the classical GA operators along with the evolutionary search of AdaGEP. Further, Mwaura and Keedwell [16] proposed an adaptive GEP by using a simple feedback heuristic. In [16], the execution probability (e.g., crossover rate and IS-transition rate) of operations that participate in generating offspring with improved fitness will be increased, while those of the operations that generate deteriorated offspring will be reduced.

As we can see, both of the above two methods attempt to adjust the configurations of GEP adaptively while the evolution progresses. However, they focused on different configurations. In

particular, the former tries to adjust the valid bits used in the chromosomes, while the later attempts to adjust the execution probabilities of genetic operators.

D. Cooperative Coevolutionary Design in GEP

Cooperative Coevolutionary (CC) is an evolutionary framework designed for solving large scale optimization problems. The key idea is to decompose the original problem into a number of simpler and easier sub-problems. Then, the sub-problems are solved independently by separate EAs with sub-populations. The fundamental issues in designing CC framework include i) how to decompose the problem and ii) how to evaluate the individuals in the sub-populations. Early works in CC design may include that Potter and De Jong [26] proposed a CC framework to improve GA, where an individual in one sub-population is evaluated by concatenating this individual with the elite individuals from the other sub-populations. To date, CC framework has been successfully applied to improve the search capability of several population based search methods such as PSO [27] and DE [28].

In the literature, the CC framework has also been applied to improve GEP. In particular, Sosa-Ascencio et al. [29] proposed a CC-based GEP framework, where two sub-populations are evolved simultaneously. The first sub-population focuses on evolving the main program, while the second sub-population responses to the evolution of ADFs. The fitness of an individual is given by the best evaluation obtained from all the evaluations of this individual concatenating with the solutions in the other sub-population. Further, Furuholmen et al. tackled the Distributor's Pallet Packing problem with CC-based GEP by dividing it into two sub-problems, i.e., pre-scheduling and packing, in [30], and solved the Three-dimensional Process Plant Layouts problem with CC-based GEP which contains one population of layout heuristics and one population of scheduling heuristics [31]. In both [30] and [31], the fitness of an individual in one sub-population is evaluated by concatenating the individual with the best individual in the other sub-populations.

E. Constant Creation Design in GEP

Numerical constants are an integral part of most mathematical formulas. Thus, constant creation is an important operation for GPs to find highly accurate mathematical formulas. However, it is a challenging task for GPs to find highly accurate constant values, for the numerical constants are continuous values while the chromosome representations of GPs are suitable for combinatorial optimization. In the GP community, several methods have been proposed to create constants. The most commonly used method is the "Ephemeral Random Constant (ERC)" method introduced by Koza [62]. Koza's method introduces a special terminal symbol to represent a constant. In the initialization step, the value of each ERC symbol in the initial individual is assigned a random value within a specific range. After that, these random ERCs are fixed and can be moved from one expression tree to another by using the crossover operator. Other methods such as the local search method [63], nonlinear least squares minimization [64], and EAs [65] have also been used to search for constant values for GP. Most of the methods proposed for GP can also apply to GEP.

Ferreira proposed a new method to create constants in GEP [32]. In Ferreira's method, an extra terminal "?" is used to represent a constant in the formula, and a domain for encoding random constants (Dc) is inserted after the *tail* to indicate constants, which is illustrated in Fig. 10. The length of the Dc domain is equal to that of the *tail*. The symbols in Dc are indices of constants in a constant vector. The constant vector is randomly generated at the beginning. Then, the "?"s in the chromosome are replaced from left to right, and from top to bottom by the constants in the constant vector based on the Dc domain. In this way, numerical constants can be implemented in GEP. However, further research conducted by Ferreira [32] indicated that the GEP itself has the capability of generating simple constants. For some problems, the GEP without using constant creation can perform even better than those with explicit use of numerical constants in terms of accuracy. This is because that using numerical constants will significantly increase the search space, which requires a lot more computational cost to find a satisfactory solution.

Zhong et al. [7] proposed a method based on ERC to create constants. In [7], a set of fixed random constants within a specific range (e.g., [-1, 1]) is generated in the initialization step of the algorithm. ERCs are represented by using a new terminal symbol. In the initialization step and mutation operation, each element in the chromosomes has a probability of c_{rate} of being assigned a random constant which is selected from the constant set. The constant assigned to the element shall remain fixed along

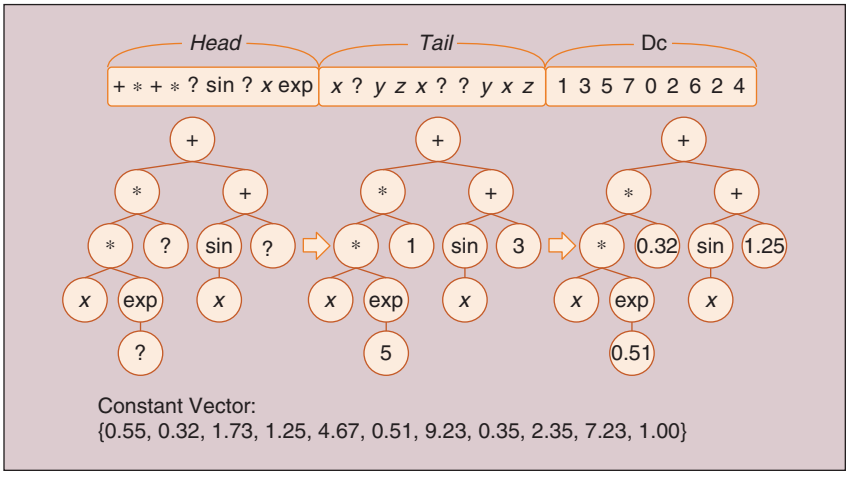


FIGURE 10 The structure of GEP chromosome with Dc domain.

the evolution process on unless a mutation is performed on it.

Li et al. [33] investigated different execution strategies of two constant creation methods, namely creep mutation and random mutation. Their experimental results suggested that constant creation methods applied to the whole population for selected generations performed better than those applied only to the best individual.

Zhang et al. [34], [35] proposed to use DE as a local search method to tune constants of formula because DE is suitable for solving continuous optimization problems. Li et al. [36] proposed a hybrid framework which uses the PSO as the local search method to create constants for GEP.

Cerny et al. [37] proposed a framework named DE-PGEP, in which the traditional discrete genotype is replaced by the continuous genotype. In particular, to create constants for formulae, a predefined number of real constants are added at the end of the chromosome. Each constant is represented by a new terminal, which will be used to construct the final solution. During the evolution process, DE operators are adopted to evolve the chromosomes, including the constant part.

In summary, most existing methods such as [32], [7] and [33] create constants by introducing special terminals and using random mutation. These methods are simple to implement, but lack of effectiveness in finding highly accurate constants. On the other hand, the EA-based methods have the potential to find highly accurate constants. However, these methods are computationally expensive since they require running another EA to tune constants of individuals in every generation.

F. Parallel Design in GEP

GEP contains an iterative evolution process. As a result, it is extremely slow in the context of solving complex and large-scale optimization problems. To address this issue, more and more researchers try to make use of the computing power of parallel and distributed platforms to reduce the computational time of GEP. A number of parallel GEPs have been proposed over the years and they mainly use the Master-Slave Model and the Island Model.

Master-Slave Model: The master-slave model contains a master thread and a number of slave threads, as illustrated in Fig. 11. The master thread controls the execution of the main program while the slave threads solve subtasks of the main program, such as evaluating the fitness of individuals. By solving the subtasks simultaneously, the computational time for solving the main task can be significantly reduced. A typical example is the pGEP proposed by Shao et al. [17]. In pGEP, the main GEP framework is controlled by a master thread. During the evolution process, the fitness values of individuals in the population are calculated in parallel using GPU. Deng et al. [19] proposed a distributed GEP, called DFMGEP-FR. The DFMGEP-FR can be deemed as a master-slave model, where the master (server) sends regression tasks to the slaves (clients), and the slaves concurrently solve the assigned tasks and send the results to the master. Finally, the master integrates all solutions received from the slaves and generates the final solution. Park et al. [20] proposed a parallel GEP which can solve the problem at hand and tune the parameters of GEP simultaneously. They adopted a Master-Slave model that contains two kinds of slaves, namely, the O-client and the G-client. The O-client focuses on turning parameters of GEP, while the G-client focuses on solving the problem at hand.

Island Model: In the Island model, the entire population is divided into a number of sub-populations. Each sub-population is evolved by one processor, as shown in Fig. 12. During the evolutionary process, individuals in one sub-population can be transferred to another to share the search information. For example, Jiang et al. [18] proposed a parallel GEP called GEPSA, where the global population is divided into several sub-populations, which have different parameter settings that serve to balance exploration and exploitation abilities. To share search information and to improve the search efficiency, the best-so-far individual will adaptively replace the worst individual of a sub-population if it does not contain the best-so-far individual. Lin et al. [38] proposed a parallel niching GEP called PNGEP. In PNGEP, the whole population is again divided. Each sub-population is mapped into a processor and then be evolved using the niching technique to maintain the population diversity. The best individual of each sub-population will be exchanged through the individual pool during the evolution. Further, Du et al. [21] proposed a parallel EA architecture. They use GEP as an example to implement their parallel method. In their method, the population is also divided into a number of sub-populations. Each sub-population uses the MapReduce mechanism to further parallelize the fitness evaluation process. During the evolution, individuals can also be transfer across sub-populations.

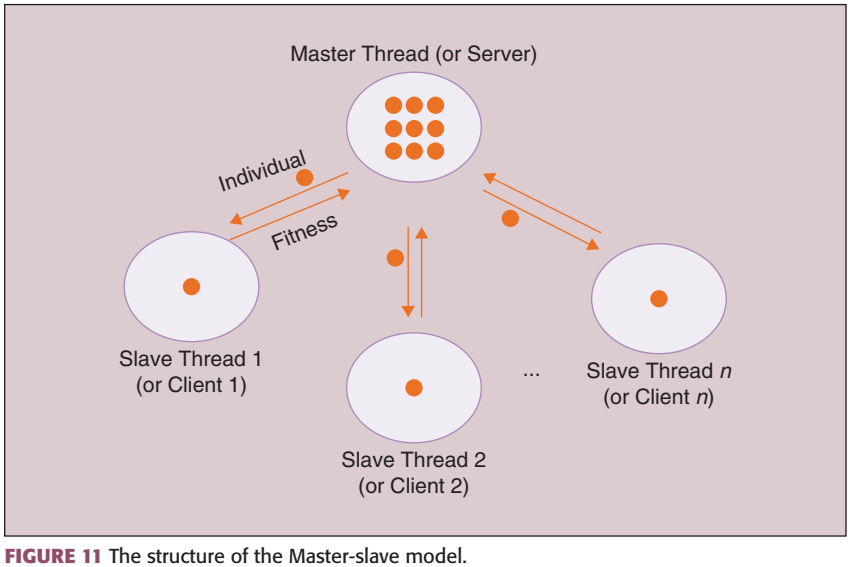


FIGURE 11 The structure of the Master-slave model.

Other Models: Besides the Master-Slave Model and the Island Model, some other parallel models have been used to implement parallel GEP. For example, Jdrzejowicz and Ratajczak-Ropel [51] implemented a parallel GEP using the multi-agent model.

In summary, most existing parallel designs in GEP attempt to reduce the computational cost of GEP by conducting fitness evaluation in parallel. The major difference of existing methods lies in the selection of parallel models (e.g., Master-slave model, Island model, or Agent-based Model) and the parallel computing platforms (e.g., MPI, MapReduce, or GPU).

IV. Theoretical Study of GEP

Despite the broad research deployed and the great success achieved in advanced GEP designs so far, there is a lack of rigorous theoretical GEP studies in the literature, since it is hard for theory to keep track with the state-of-the-art algorithms which have become increasingly elaborate for handling today's complex and big problems.

In [66], Yuan et al. proved that the evolution of GEP for symbolic regression will converge on the global best chromosome in probability and the GEP may also converge on the local best chromosome. Furthermore, Lu et al. [67], [68] used Markov chain and spectrum analysis to analyze the convergence rate of GEP with maintaining elitist (ME-GEP). They proved that the convergence speed of ME-GEP depends on the properties of transition matrices and the upper bound of convergence speed relies on the parameter settings of ME-GEP, such as the population size, mutation probability and selection probability.

More recently, Du et al. [69] used Markov chain theory to analyze the time complexity of ME-GEP. They obtained the upper and lower limits of the average time complexity of ME-GEP, and discovered the relationship between the upper limit and the parameter settings of ME-GEP. In particular, suppose the population size is n, the generation number is t, the best individual in the population is $x_0(t)$, the ith individual in the population is $x_i(t), 1 \le i \le n$, and $\xi(t) = (x_0(t), x_1(t), x_2(t), ..., x_n(t))$. According to [69], $\{\xi(t); t \ge 0\}$ is a Markov Chain. Define the optimal state set of ME-GEP as:

$$E^* = \{(x_0, x_1, ..., x_n) \in E : \exists x_j (0 \le j \le n) \in B\}, \quad (9)$$

where E represents the state space, and B represents the best solution set in the search space. Then, according to [69], the ME-GEP converges to E^*, i.e.,

$$\lim_{t \to \infty} P\{\xi(t) \in E^*\} = 1. \quad (10)$$

In addition, Huang [70] studied the schema theory of GEP. The genetic modifications provided by each operator of GEP were analysed. In [70], a set of theorems for predicting the propagation of the schema from one generation to another and a set of experiments designed to validate the developed schema theory are also presented.

V. Applications of GEP

Over the past decades, GEP has been applied to a range of applications owing to its high effectiveness and efficiency. It is worth noting that the applications of GEP are very large and are increasing rapidly. Thus, it is impossible to list all of them in this article. For the sake of space economy, in this section, we focus on describing five representative application fields of GEP. Most of the applications of GEP can be converted to the problems that belong to these five representative fields. Our objective is to provide a brief description of these representative fields so as to facilitate readers applying GEP in practice. Table 2 summaries the example applications of GEP in the five representative areas. In general, most applications adopt the classical single objective GEP proposed by Ferreira [1], [2], while some applications adopt GEP with problem-specific modifications.

A. Symbolic Regression Problem (SRP)

SRP is one of the most common application fields of GEP. It requires finding a mathematic formula to fit the given dataset. The mathematical formula, which is formed by combining building blocks such as numerical functions (e.g., + and sin) and input variables (e.g., x, y), is expected to describe the insight relationships between the inputs and outputs of the system and be capable of predicting outputs of new inputs. Each sample of the dataset consists of input variables and outputs, and can be expressed as:

$$\{x_{i,1}, x_{i,2}, ..., x_{i,n}, o_{i,1}, ..., o_{i,m}\}, \quad (11)$$

where n is the number of input variables and m is the number of outputs, $x_{i,j}$ and $o_{i,j}$ are the jth input and the jth output of the ith sample, respectively. The quality of the formula (Γ) is evaluated by the accuracy of its fitting, which is commonly achieved by using the root-mean-square-error (*RMSE*). Generally, given a

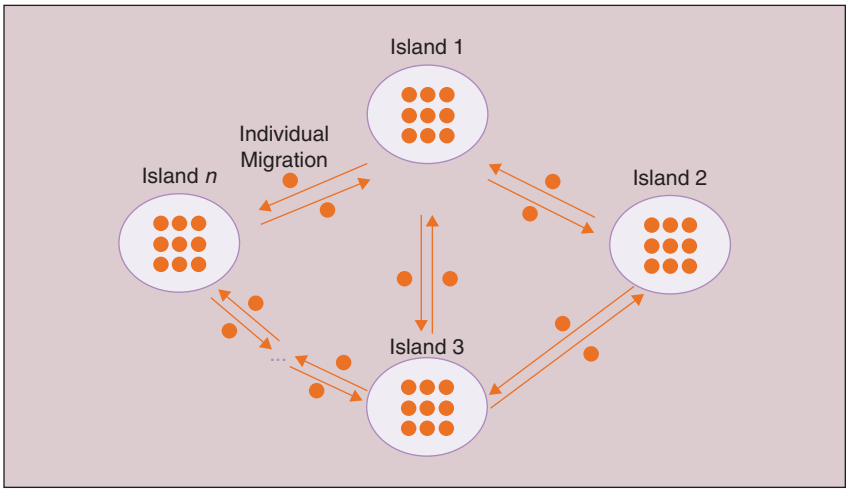


FIGURE 12 The structure of the Island model.

TABLE 2 Example applications of GEP.

PROBLEM	TYPE OF GEP	SUMMARY
SRP	SOGEP	PREDICT REFERENCE EVAPOTRANSPIRATION IN ARID CLIMATE [71]
	SOGEP	PREDICT VELOCITY FIELD [72]
	SOGEP	PREDICT THE DISCHARGE COEFFICIENT IN RECTANGULAR SIDE WEIRS [73]
	SOGEP	PREDICT CONSTRUCTION AND DEMOLITION WASTE [74]
	SOGEP	DETERMINATION OF THE ULTIMATE LIMIT STATES OF SHALLOW FOUNDATIONS [75]
	SOGEP	PREDICT BUS DWELL TIME [76]
	SOGEP	STOCK MARKET PREDICTION [77]
	SOGEP	PREDICT CASPIAN SEA LEVEL CHANGES [78]
	SOGEP	PREDICT FRICTION FACTOR FOR SOUTHERN ITALIAN RIVERS [79]
	SOGEP	ESTIMATION OF CRITICAL VELOCITY FOR SLURRY TRANSPORT THROUGH PIPELINE [80]
	SOGEP	ESTIMATION PERFORMANCE OF LIBRAH2O ABSORPTION COOLING SYSTEM [81]
	SOGEP	ESTIMATION OF FLASH POINT TEMPERATURE OF NON-ELECTROLYTE ORGANIC COMPOUNDS [82]
	SOGEP	SHORT-TERM LOAD FORECASTING IN THE ELECTRIC POWER INDUSTRY [83]
	SOGEP	STAGE-DISCHARGE RELATIONSHIP PREDICTION [84]
	SOGEP	TIME SERIES PREDICTION [4]
	IMPROVED SOGEP WITH BLOCK STRATEGY	PREDICT SOFTWARE RELIABILITY [85]
	SOGEP	ESTIMATE FLOW TIME OF JOBS IN A MULTI-STAGE JOB SHOP [86]
	SOGEP	WEATHER PREDICTION [87]
	SOGEP	PREDICT NORMALIZED SHEAR MODULUS AND DAMPING RATIO OF SANDS [88]
	SOGEP	PREDICT VOLTAGE OF DIFFERENT PROTON EXCHANGE MEMBRANE FUEL CELLS [89]
	SOGEP	PREDICT EVAPOTRANSPIRATION [90]
	SOGEP	FLOW DURATION CURVE REGIONALIZATION [91]
	SOGEP	PREDICT ELECTRICITY DEMAND [92]
CP	SOGEP	DISCOVER CLASSIFICATION RULES [93]
	REVISED SOGEP WITH BACKWARD CHANGING EA	DISCOVER RULES FOR MUSIC EMOTION CLASSIFICATION [94]
	SOGEP	MULTI-LABEL CLASSIFICATION [95]
	SOGEP	EVEN SELECTION IN HIGH ENERGY PHYSICS [96]
	SIMPLIED SOGEP	BENCHMARK CLASSIFICATION PROBLEMS [3]
AMDP	SOGEP, SL-GEP	CROWD MODEL DESIGN [5], [97]
	GEP WITH SOME MODIFICATION	ELECTRONIC CIRCUITS DESIGN [98]
	MODIFIED SOGEP WITH MULTIPLE GENE DOMAINS	ARTIFICIAL NEURAL NETWORKS DESIGN [99]
	SOGEP	CIRCUITS DESIGN [100], [101]
	SOGEP	PRODUCTION LINE DESIGN [102]
	SOGEP WITH EXTENDED ENCODING DESIGN	ROBOT BEHAVIOR MODEL DESIGN [103]
COP	SOGEP	GENERATE HYPER-HEURISTIC FOR COMBINATORIAL OPTIMIZATION PROBLEMS [6], [104]
	SOGEP	DYNAMIC MACHINE SCHEDULING PROBLEMS [105], [106]
	SOGEP WITH EXTENDED OPERATORS	TSP AND TASK ASSIGNMENT PROBLEM [2]
ROP	SOGEP WITH ENCODING EXTENSION	THE HZERO AND GEP-PO ALGORITHMS [2], THE UC-GEP [107]

SOGEP represents the traditional single objective GEP proposed by Ferreira [1], [2].

set of building blocks (e.g., numerical functions and input variables), the symbolic regression requires finding the optimal Γ^* that minimizes the *RMSE* for the given dataset:

$$\Gamma^* = \arg\min_{\Gamma} f(\Gamma), \tag{12}$$

where $f(\Gamma)$ returns the fitting error of Γ.

Many practical problems such as the time series prediction problems can be converted to an SR problem. In the literature, various GEP based methods have been proposed to solve problems that can be converted to an SR problem. For example, Yassin et al. [71] converted the problem of estimating reference evapotranspiration in arid climate as an SR problem and applied GEP to solve it. Gholami et al. [72] used GEP to predict the velocity field in a 90° channel bend. Ebtehaj et al. [73] treated the prediction of discharge coefficient in rectangular side weirs as an SR problem and used GEP to solve the problem. Similar works in this area can be found in [4], [74]–[92], [108].

B. Classification Problem (CP)

Classification is a fundamental and active research topic in data mining and knowledge discovery. Formally, given a set of predetermined target classes $C = \{C_1, C_2, ..., C_n\}$, a set of input features $\{A_1, A_2, ..., A_m\}$, and a set of training data $S = \{S_1, S_2, ..., S_N\}$ where each sample S_i has m features and is associated with one target class. The task is to construct a set of rules, which can be used to predict the target classes of samples, given the input feature values of the samples. Machine learning (ML) techniques such as Artificial Neural Networks (ANNs) and Support Vector Machine (SVM) are commonly used to solve classification problems [109]. Recently, GP and its variants have also been applied to classification. Espejo et al. [110] has done a comprehensive survey on using GP for classification. As a variant of GP, GEP has also been applied to classification over the past decades. For example, Zhou et al. [3] applied GEP to several classification problems and showed that GEP is capable of evolving more concise solutions than the other methods. Karakasis and Stafylopatis [111] proposed an enhanced GEP with Clonal Selection to evolve accurate classification rules. Ávila et al. [95] applied GEP to the solution of multi-label classifications. Wagner et al. [112] proposed an enhanced GEP-based method to discover classification rule for data mining.

Over the past decades, GEP has been applied to a range of applications, such as symbolic regression, classification, automatic model design, combinatorial optimization, and real parameter optimization.

C. Automatic Model Design Problem (AMDP)

Recently, GEP has been used for AMDP. As an example, Zhong et al. [5] used GEP to design crowd simulation model. In this application, the GEP is used to combine the predefined building blocks to form model components (i.e., behavior rules) so as to fit the given real dataset. To evaluate the fitness of a candidate behavior rule, a crowd simulation model with the behavior rule is performed first. The final simulation results are then compared with the objective data to calculate the fitness value of the behavior rule. A similar technique was applied to another crowd modelling application [97].

GEP has also been used for automatic circuit design. For example, Janeiro et al. [100], [101] proposed a GEP-based method to design components of sensor circuits. In their method, two functions (+ and =) are used to represent series and parallel structures of the circuit, respectively. Three terminals are used to represent resistances, inductors and capacitors. In this way, an electrical circuit can be represented by a linear string of functions and terminals. Similar work in this area can be found in [98], [99], [102], [103].

D. Combinatorial Optimization Problem (COP)

Combinatorial optimization aims to find an optimal subset in a given family of subsets of a finite set so as to maximize (or minimize) a given cost function [113]. Many combinatorial optimization problems such as the Travelling Salesman Problem (TSP) and the Job Shop Scheduling problem (JSP) are NP-hard problems. Thus, they are infeasible for exhaustive search methods to solve. Recently, using GEP to solve combinatorial optimization problems has attracted increasing attention. For example, Ferreira [2] applied GEP to two scheduling problems, i.e., the TSP and the Task Assignment Problem. Sabar et al. [6], [104] applied GEP to designing high-level heuristics for combinatorial optimization problems. Nie et al. [105], [106] applied GEP to evolving generic scheduling rules for dynamic machine scheduling problems. A major feature of using GEP for combinatorial optimization is that the solutions provided by GEP are often general enough to solve multiple problems. For example, the high-level heuristics generated by GEP in [104] are capable of solving multiple problems across different domains.

E. Real Parameter Optimization Problem

Typically, the Real parameter Optimization Problem (ROP) can be stated as follows:

$$\begin{aligned} &\text{maximize } F(x) \\ &\text{subject to } x \in \mathbb{R}^n, \end{aligned} \tag{13}$$

where n is the dimension of the problem, $\mathbb{R}^n$ denotes the decision space, and $F : \mathbb{R}^n \to \mathbb{R}$ represents the objective function. In the literature, ROPs are commonly solved by EAs such as GA, DE, and PSO. As GEPs are straightforward to be employed for evolving programs with discrete tree structure, the research of GEP for ROPs received far less attention with only few works reported to date. In particular, Ferreira [2] proposed two GEP variants for ROPs, namely, the HZero

algorithm and the GEP-PO algorithm, while Xu et al. [107] presented an Uniform-Constants based GEP (UC-GEP) for ROPs. The key idea of these methods is to use multiple genes to represent one solution of the ROP, with each gene representing one variable value. They mainly differ in how to encode a real value by a gene. For example, in the HZero algorithm, each gene contains a *tail* domain and a Dc domain. Only one terminal "?" is considered to represent a real constant. Thus each gene is essentially a formula that consists of different constant values and functions. The final result of the formula is assigned to the corresponding ROP variable. Further, it is worth noting that a large chromosome length is required if the high dimensionality ROPs are encountered, which leads to a huge search space that cannot be efficiently solved by existing GEPs. It is an unexplored direction whether the GEP based methods can outperform other EAs such as DE and PSO on large scale ROPs.

VI. Open Research Issues

In this section, several open research issues of GEP are identified and discussed for further exploration.

A. Advanced Mechanism Design in GEP

GEP contains several key components, such as transposition, mutation, constant creation, and a number of control parameters. The design of these components as well as the control of the search process have a great impact on the performance of GEP. Although many works have been proposed for dedicated configuration mechanisms in GEP, further exploration of advanced designs in GEP is expected.

1) Evolutionary Operator and Parameter Control Design: In the literature, various advanced evolutionary operators and parameter control approaches have been proposed and verified to be effective in other EAs, such as the orthogonal design method [114], [115], the aging concepts [116], the compact design mechanism [117], and the opposite-based evolutionary mechanism [118]. In contrast, dedicated designs of evolutionary operators and parameter settings for GEP are limited, such as the adaptive and self-adaptive GEP designs discussed in Section III.C. More efficient and effective designs of the search operators as well as control approaches of parameter configuration are necessary to achieve advanced GEP for problem solving.

2) Constant Creation Design: Constant creation is an important operator in GEP, which is helpful in finding high quality GEP solutions. However, the optimization of constants significantly increases the search space. Existing constant creation techniques are not effective in handling the additional complexity incurred by constant creation. In particular, Ferreira [32] has compared two commonly used constant creation methods in GEP on symbolic regression problems. The experimental results showed that the GEPs with a constant creation operator performed even worse than the GEPs without a constant creation operator for checking the solution accuracy. It is therefore desirable to design advanced constant creation techniques, which can balance the incurred complexity in a search space and the efforts made for accurate solution exploration.

3) GEP Design for Solution Complexity Reduction: The GEP adopts a fixed length string to represent a computer program. This enables the GEP to find concise solutions when the length of chromosome is small. However, when the problems become complicated, especially with a large number of terminals and functions, the dimension of the corresponding chromosome will increase accordingly to ensure the accuracy of the optimized solution. This, however, can lead to complicated GEP solutions that are not general, and difficult for humans to interpret. Thus, it is necessary to design effective mechanisms to reduce the solution complexity of the GEP. One potential approach is to treat the solution complexity as another objective and use multi-objective optimization techniques [119]–[121] to find solutions that have trade-offs between the accuracy and the complexity of solutions.

4) GEP Design for Ill-defined Problems: Traditional GEP and its variants are usually applied to well-defined problems. However, in many real world applications (e.g., classification problems with unbalanced data [122]), the problems encountered may contain specific features, which make GEP inapplicable or inefficient. In these cases, problem-specific GEP has to be designed based on the features of the problem encountered. For example, in the application of determining the water quality and stress on lakes or rivers as a result of pollutants found in the wastewater, evaluation systems were installed to measure the changes of the environment over time in different perspectives[1]. Due to the circumstances relating to measurement, the measurement data may have missing values. This problem can be easily converted to a symbolic regression problem, which is the common application of GEP. However, without an effective method to deal with the missing data, the GEP cannot be applied directly to solve it.

B. GEP Meets Machine Learning

Another open research issue relates to the use of ML techniques to enhance the search performance of GEP. Specifically, ML can be used to learn high-order or domain-specific knowledge to enhance the search efficiency of GEP. In the GP community, some work has already been done on using ML to learn high-order or domain-specific knowledge in order to enhance the problem-solving performance of GP [52], [53], [123]–[126]. As an example, Kameya et al. [125] used ML to capture the building blocks (frequently used sub-functions) from historical search experiences, which are then reused in an effort to improve GP. The GEP is an iterative search algorithm that generates a great amount of history search information during the search. Thus, it is also possible to learn useful domain-specific knowledge from history search information to improve the search efficiency of GEP.

Furthermore, ML could be used to define surrogate models aimed at reducing the computational cost of GEP. For

[1]The readers are referred to the detailed descriptions of the application in http://www.spotseven.de/gecco-challenge/gecco-challenge-2014

many optimization problems such as those that require running simulations, evaluating the fitness of an individual is expensive [5], [97]. In these cases, ML techniques, such as ANNs, can be used to build an approximation model of the real system (a surrogate model) to estimate the fitness of a given individual. Since performing with the surrogate model is much cheaper than doing so with the real system, using surrogate model can significantly reduce the computational costs of the search process [127].

In addition, ML could be used to reduce the dimension of the problem encountered so as to improve the search efficiency of GEP. In practical applications, the problem at hand may contain a large number of redundant dimensions, such as redundant input features. In fact, only a small subset of the features is useful for constructing the final GEP solutions. In these cases, we can use ML techniques, such as feature selection, to choose a small number of important features for solution construction. In this way, the search space would be reduced and the search efficiency of GEP could be improved.

C. Exploration of New GEP Framework

With the rapid development of Internet and cloud computing technology, the problems that humans face today are becoming more and more complicated. Optimization problems with large-scale size, high dimensional decision spaces, many objectives, etc., create common challenges. To solve these challenging problems, new GEP frameworks need to be developed.

1) Big GEP Framework: Today, the data generated by human increases exponentially. This leads to an urgent need for designing effective methods to solve big optimization problems, as traditional GEP may become inapplicable in these cases [128], [129]. Among others, there are at least two main reasons. First, the search space is huge due to the large dimensions of the problems encountered. For example, the DNA micro array data usually contain thousands of features [130]. Thus, traditional GEP may quickly be affected by local stagnations. Second, for big optimization problems, the fitness evaluation involves processing great quantities of data and could be quite expensive. GEP is an iterative search algorithm that requires a large number of fitness evaluations. Thus, the total computational time for GEP to solve big optimization problems could be very long. To address the above issues, one potential approach is to design parallel GEP models based on cloud computing platforms or distributed computing platforms.

There are still many challenges and open research issues for further exploration of GEP, such as the exploration of advanced designs in GEP, the design of new GEP framework for complicated optimization, and rigorous theoretical analysis of GEPs.

2) Multi-Tasking GEP Framework: Multi-task learning has become an active research topic in the machine learning community [131]–[135]. The justification for multi-task learning is that problems seldom exist in isolation, and related problems often contain useful information that can be utilized to improve problem-solving efficiency [53], [126], [136]. Recently, the concept of Multi-Factorial Optimization (MFO) has been introduced in [137] as a new evolutionary paradigm that would promote evolutionary multitasking. In contrast to traditional EAs that solve a single task each time, the MFO intends to conduct evolutionary searches on multiple concurrently existing search spaces, corresponding to different tasks. A Multi-Factorial Evolutionary Algorithm (MFEA) has been designed based on the concept of MFO, and has been shown to be effective on a number of continuous and combinatorial optimization problems in [137]. The MFO has great potential to improve the problem solving efficiency of GEP. However, the original MFEA is designed based on the specific problems and cannot be applied with GEP directly for the evolution of computer programs. Thus, one of the possible directions for multi-tasking GEP is from the perspective of MFO.

3) Many-Objective GEP Framework: Many-Objective Optimization problems (MaOPs), which have more than three objectives to be optimized simultaneously, have widely existed in various applications [138]–[140]. MaOPs are more challenging compared with the bi- and tri-objective optimization problems, since the Pareto fronts of MaOPs are much more complicated. Traditional multi-objective GEPs (see, for example, [141]) can be applied to solving MaOPs, but their performance will worsen as the number of objectives increases. The reason for this is that when the number of objectives is large, most of the individuals in the population will be equally good (i.e., nondominated to each other). This causes the algorithms to fail to converge due to the loss of selection pressure in the direction of the Pareto front. Therefore, designing new many-objective GEP frameworks to handle MaOPs is required.

4) Dynamic GEP Framework: Many practical optimization problems are dynamic, which requires an optimization algorithm finding the global optimal solution under a specific environment as well as tracking the trajectory of the changing optima over dynamic environments. In the literature, many EAs for Dynamic Optimization Problems (DOPs) have been proposed, which can be generally divided into two categories: (1) finding/tracking optima over time (algorithms are mainly for DOPs in a continuous space) [142], [143], and (2) adapting current solutions against changes (algorithms are mainly for DOPs in a combinatorial space) [144], [145]. In contrast, significantly fewer studies on GEPs for solving DOPs, including both continuous and combinatorial DOPs, have been explored, making it a fertile area for further research investigations.

D. Theoretical Studies of GEP

As discussed above, GEP is one of computational intelligence areas in which the

empirical work has outpaced the theoretical work. While there has been some work on the theory of GEP [66]–[70], more rigorous theoretical analysis is expected to provide more insights on the GEP search process. Possible areas for theoretical study on GEP may include the proof of convergence, time complexity analysis, the convergence speed estimation, and the analysis of evolution efficiencies of operators. Besides the classical GEP framework, theoretical studies on the state-of-the-art GEP frameworks are also expected.

VII. Conclusion

GEP is a variant of GP for automatic generation of computer programs. It uses a fixed-length gene expression representation to encode computer programs, and be able to find concise and readable solution efficiently. Over the past decades, GEP has undergone rapid advancements and has been widely used in many applications, including classification problems, time series predictions, and others. In this survey, we reviewed the recent research progress of GEP from various perspectives. In particular, we first discussed the state-of-the-art techniques designed for improving the search performance of GEP. In general, these techniques try to improve GEP from six aspects, including encoding design, evolutionary mechanism design, parameter adaptation design, cooperative coevolutionary design, constant creation design, and parallel design. Next, we presented the theoretical studies of GEP to date. The theoretical study of GEP has not attracted sufficient attentions from researchers yet. Existing works focused mainly on the proof of convergence and the estimation of convergence speed. Further, we summarized the major applications of GEP. Generally, most applications of GEP can be converted into five representative optimization problems, i.e., symbolic regression, classification, automatic model design, combinatorial optimization, and real parameter optimization.

Although GEP has achieved fast advancements and developments, there are still many challenges and open research issues for further exploration. First, to further improve the performance of GEP, exploration of advanced designs in GEP is expected. Potential techniques include the new parameter adaptive controlling design, constant creation design, solution complexity reduction design, and ill-problem handling design. Second, using ML to assist GEP is desirable. ML could be used in at least three aspects for enhancing GEP for problem solving, i.e., learning problem-specific knowledge, constructing surrogate models, and dimension reduction. Third, to solve complicated problems that humans face today. The design of new GEP framework for complicated optimization (e.g., those with large-scale size, high dimensional decision spaces, and many objectives, etc.) is urgent. These include the Big GEP framework for solving problems in big data environment, Multi-Tasking GEP framework for handling multiple tasks simultaneously, Many-objective GEP Framework to deal with problems with more than three objectives, and Dynamic GEP framework for problems with optima changing over dynamic environments. Last but not least, rigorous theoretical analysis of GEP is required for providing deeper insights of the GEP search process, such as the proof of convergence, time complexity analysis, convergence speed estimation, and analysis of evolution efficiencies of operators.

Acknowledgment

This work is partially supported under the National Natural Science Foundation of China (Grant No. 61602181, 61603064), Fundamental Research Funds for the Central Universities (Grant No. 2017ZD053), Frontier Interdisciplinary Research Fund for the Central Universities (Grant No. 106112017CDJQJ188828), and the Data Science and Artificial Intelligence Center (DSAIR) at the Nanyang Technological University. Further, the authors would like to express their sincere thanks to Dr. Athanasios V. Vasilakos for his time in proofreading the first version of the paper.

References

[1] C. Ferreira, "Gene expression programming: A new adaptive algorithm for solving problems," *Complex Syst.*, vol. 13, no. 2, pp. 87–129, 2001.

[2] C. Ferreira, *Gene Expression Programming: Mathematical Modelling by an Artificial Intelligence*. New York: Springer-Verlag, 2006.

[3] C. Zhou, W. Xiao, T. M. Tirpak, and P. C. Nelson, "Evolving accurate and compact classification rules with gene expression programming," *IEEE Trans. Evolut. Comput.*, vol. 7, no. 6, pp. 519–531, Dec. 2003.

[4] J. Zuo, C.-J. Tang, C. Li, C.-A. Yuan, and A.-L. Chen, "Time series prediction based on gene expression programming," in *International Conference on Web-Age Information Management*. Berlin Heidelberg: Springer-Verlag, 2004, pp. 55–64.

[5] J. Zhong, L. Luo, W. Cai, and M. Lees, "Automatic rule identification for agent-based crowd models through gene expression programming," in *Proc. Int. Conf. Autonomous Agents and Multiagent Systems, Int. Foundation for Autonomous Agents and Multiagent Systems. ACM*, May 2014, pp. 1125–1132.

[6] N. R. Sabar, M. Ayob, G. Kendall, and R. Qu, "A dynamic multiarmed bandit-gene expression programming hyper-heuristic for combinatorial optimization problems," *IEEE Trans. Evolut. Comput.*, vol. 45, no. 2, pp. 217–228, Feb. 2015.

[7] J. Zhong, Y.-S. Ong, and W. Cai, "Self-learning gene expression programming," *IEEE Trans. Evolut. Comput.*, vol. 20, no. 1, pp. 65–80, Feb. 2016.

[8] X. Li, C. Zhou, W. Xiao, and P. C. Nelson, "Prefix gene expression programming," in *Proc. Genetic and Evolutionary Computation Conf. ACM*, June 2005, pp. 25–31.

[9] K.-T. Fang, D. K. Lin, P. Winker, and Y. Zhang, "Uniform design: Theory and application," *Technometrics*, vol. 43, no. 3, pp. 237–248, 2000.

[10] L. N. D. Castro and F. J. V. Zuben, "Learning and optimization using the clonal selection principle," *IEEE Trans. Evolut. Comput.*, vol. 6, no. 3, pp. 239–251, June 2002.

[11] G. L. Z. Gan, Z. Yang and M. Jiang, "Automatic modelling of complex functions with clonal selection-based gene expression programming," in *Proc. Int. Conf. Natural Computation*, Aug. 2007, pp. 228–232.

[12] V. Litvinenko, P. Bidyuk, J. Bardachov, V. Sherstjuk, and A. Fefelov, "Combining clonal selection algorithm and gene expression programming for time series prediction," in *Proc. IEEE Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications*, Sept. 2005, pp. 133–138.

[13] L. Wang, B. Yang, S. Wang, and Z. Liang, "Building image feature kinetics for cement hydration using gene expression programming with similarity weight tournament selection," *IEEE Trans. Evolut. Comput.*, vol. 19, no. 5, pp. 679–693, Oct. 2015.

[14] S. Jiang, Z. Cai, D. Zeng, Y. Liu, and Q. Li, "Gene expression programming based on simulated annealing," in *Proc. IEEE Int. Conf. Wireless Communications, Networking and Mobile Computing*, Sept. 2005, pp. 1264–1267.

[15] T. Zeng, C. Tang, Y. Xiang, P. Chen, and Y. Liu, "A model of immune gene expression programming for rule mining," *J Univers. Comput Sci.*, vol. 13, no. 10, pp. 1484–1497, Oct. 2007.

[16] J. Mwaura and E. Keedwell, "Adaptive gene expression programming using a simple feedback heuristic," in *Proc. AISB Conf.*, 2009.

[17] S. Shao, X. Liu, M. Zhou, J. Zhan, X. Liu, Y. Chu, and H. Chen, "A GPU-based implementation of an enhanced GEP algorithm," in *Proc. 14th Annu. Conf. Genetic and Evolutionary Computation*. ACM, July 2012, pp. 999–1006.

[18] S.-W. Jiang, Z.-H. Cai, D. Zeng, Q. Li, and Y.-F. Cheng, "Parallel gene expression programming algorithm based on simulated annealing method," *DianziXuebao (Acta Electronica Sinica)*, vol. 33, no. 11, pp. 2017–2021, Nov. 2005.

[19] S. Deng, D. Yue, L.-C. Yang, X. Fu, and Y.-Z. Feng, "Distributed function mining for gene expression programming based on fast reduction," *PloS One*, vol. 11, no. 1, pp. e0146698, Jan. 2016.

[20] H.-H. Park, A. Grings, M. V. dos Santos, and A. S. Soares, "Parallel hybrid evolutionary computation: Automatic tuning of parameters for parallel gene expression programming," *Appl. Math Comput.*, vol. 201, no. 1, pp. 108–120, July 2008.

[21] X. Du, Y. Ni, Z. Yao, R. Xiao, and D. Xie, "High performance parallel evolutionary algorithm model based

on MapReduce framework," *Int. J. Comput. Appl. Technol.*, vol. 46, no. 3, pp. 290–295, Mar. 2013.
[22] Y.-Z. Peng, C.-A. Yuan, X.-F. Mai, and X. Qin, "Survey on theoretical research of gene expression programming," *Appl. Res. Comput.*, vol. 28, no. 2, pp. 413–419, Feb. 2011.
[23] C. Ferreira, *Automatically Defined Functions in Gene Expression Programming*. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 21–56.
[24] Y. Chen, D. Chen, S. U. Khan, J. Huang, and C. Xie, "Solving symbolic regression problems with uniform design-aided gene expression programming," *J Supercomput.*, vol. 66, no. 3, pp. 1553–1575, Dec. 2013.
[25] A. Brbulescu and E. Butu, "Time series modelling using an adaptive gene expression," *Int. J. Math. Models Methods Appl. Sci.*, vol. 3, pp. 85–93, 2009.
[26] M. A. Potter and K. A. De Jong, "A cooperative coevolutionary approach to function optimization," in *International Conference on Parallel Problem Solving from Nature*. New York: Springer-Verlag, 1994, pp. 249–257.
[27] X. Li and X. Yao, "Cooperatively coevolving particle swarms for large scale optimization," *IEEE Trans. Evolut. Comput.*, vol. 16, no. 2, pp. 210–224, Apr. 2012.
[28] M. N. Omidvar, X. Li, Y. Mei, and X. Yao, "Cooperative co-evolution with differential grouping for large scale optimization," *IEEE Trans. Evolut. Comput.*, vol. 18, no. 3, pp. 378–393, June 2014.
[29] A. Sosa-Ascencio, M. Valenzuela-Rendón, and H. Terashima-Marín, "Cooperative coevolution of automatically defined functions with gene expression programming," in *Proc. IEEE 11th Mexican Int. Conf. Artificial Intelligence*, Oct. 2012, pp. 89–94.
[30] M. Furuholmen, K. Glette, M. Hovin, and J. Torresen, "Coevolving heuristics for the distributor's pallet packing problem," in *Proc. IEEE Congress on Evolutionary Computation*, May 2009, pp. 2810–2817.
[31] M. Furuholmen, K. Glette, M. Hovin, and J. Torresen, "A coevolutionary, hyper heuristic approach to the optimization of three-dimensional process plant layoutsa comparative study," in *Proc. IEEE Congress on Evolutionary Computation*, July 2010, pp. 1–8.
[32] C. Ferreira, "Function finding and the creation of numerical constants in gene expression programming," in *Advances in Soft Computing*. New York: Springer-Verlag, 2003, pp. 257–265.
[33] X. Li, C. Zhou, P. C. Nelson, and T. M. Thomas, "Investigation of constant creation techniques in the context of gene expression programming," in *Proc. 6th Annu. ACM Conf. Genetic and Evolutionary Computation*, 2004.
[34] Q. Zhang, C. Zhou, W. Xiao, P. C. Nelson, and X. Li, "Using differential evolution for GEP constant creation," in *Proc. Late breaking paper at Genetic and Evolutionary Computation Conf.*, Seattle, WA, USA, 2006.
[35] Q. Zhang, C. Zhou, W. Xiao, and P. C. Nelson, "Improving gene expression programming performance by using differential evolution," in *6th IEEE Int. Conf. Machine Learning and Applications*, Dec 2007, pp. 31–37.
[36] T. Li, T. Dong, J. Wu, and T. He, "Function mining based on gene expression programming and particle swarm optimization," in *Proc. 2nd IEEE Int. Conf. Computer Science and Information Technology*, Aug 2009, pp. 99–103.
[37] B. M. Cerny, P. C. Nelson, and C. Zhou, "Using differential evolution for symbolic regression and numerical constant creation," in *Proc. 10th Annu. Conf. Genetic and Evolutionary Computation*. ACM, July 2008, pp. 1195–1202.
[38] Y. Lin, H. Peng, and J. Wei, "A niching gene expression programming algorithm based on parallel model," in *International Workshop on Advanced Parallel Processing Technologies*. New York: Springer-Verlag, 2007, pp. 261–270.
[39] J. Zhong, X. Hu, J. Zhang, and M. Gu, "Comparison of performance between different selection strategies on simple genetic algorithms," in *Proc. Int. Conf. Computational Intelligence for Modelling, Control and Automation and Int. Conf. Intelligent Agents, Web Technologies and Internet Commerce*, vol. 2, Nov. 2005, pp. 1115–1121.
[40] R. Poli, W. B. Langdon, N. F. McPhee, and J. R. Koza, *A Field Guide to Genetic Programming*. Lulu. com, 2008.
[41] M. O'Neill and C. Ryan, "Grammatical evolution," *IEEE Trans. Evolut. Comput.*, vol. 5, no. 4, pp. 349–358, Aug. 2001.
[42] M. F. Brameier and W. Banzhaf, *Linear Genetic Programming*. New York: Springer-Verlag, 2007.
[43] M. Oltean and C. Grosan, "A comparison of several linear genetic programming techniques," *Complex Syst.*, vol. 14, no. 4, pp. 285–314, 2003.
[44] H-y. Quan and G. Yang, "Gene expression programming with DAG chromosome," in *Proceedings of the 2Nd International Conference on Advances in Computation and Intelligence*, ser. ISICA'07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 271–275.
[45] J. Hu, C.-J. Tang, L. Duan, J. Zuo, J. Peng, and C.-A. Yuan, "The strategy for diversifying initial population of gene expression programming," *Chin. J. Comput.*, vol. 30, no. 2, pp. 305, Feb. 2007.
[46] Z. Guo, Z. Wu, X. Dong, K. Zhang, S. Wang, and Y. Li, "Component thermodynamical selection based gene expression programming for function finding," *Math. Probl. Eng.*, vol. 2014, 2014.
[47] V. K. Karakasis and A. Stafylopatis, "Efficient evolution of accurate classification rules using a combination of gene expression programming and clonal selection," *IEEE Trans. Evolut. Comput.*, vol. 12, no. 6, pp. 662–678, Dec. 2008.
[48] J. Y.-C. Liu, J.-H. A. Chen, C.-T. Chiu, and J.-C. Hsieh, "An extension of gene expression programming with hybrid selection," in *Proceedings of the 2nd International Conference on Intelligent Technologies and Engineering Systems (ICITES2013)*. New York: Springer-Verlag, 2014, pp. 635–641.
[49] L. Qu, C. Hongbing, and H. X. Lin, "Edit distance based crossover operator in gene expression programming," in *Proc. 8th IEEE Int. Conf. Biomedical Engineering and Informatics,* Oct. 2015, pp. 468–472.
[50] J. Yang and J. Ma, "A hybrid gene expression programming algorithm based on orthogonal design," *Int. J. Comp. Intell. Tech.*, vol. 9, no. 4, pp. 778–787, June 2016.
[51] P. Jdrzejowicz and E. Ratajczak-Ropel, "Agent-based gene expression programming for solving the RCPSP/max problem," in *International Conference on Adaptive and Natural Computing Algorithms*. New York: Springer-Verlag, 2009, pp. 203–212.
[52] J. R. Koza and J. Noyes, *Genetic Programming II Videotape: The Next Generation*. Cambridge, MA: MIT Press, 1994.
[53] R. Meuth, M.-H. Lim, and Y.-S. Ong, D. C. W, II, "A proposition on memes and meta-memes in computing for higher-order learning," *Memetic Comput.*, vol. 1, no. 2, pp. 85–100, June 2009.
[54] T. V. Belle and D. H. Ackley, "Code factoring and the evolution of evolvability," in *Proc. Genetic Evol. Comput. Conf.*, July 2002, pp. 1383–1390.
[55] J. A. Walker and J. F. Miller, "The automatic acquisition, evolution and reuse of modules in cartesian genetic programming," *IEEE Trans. Evolut. Comput.*, vol. 12, no. 4, pp. 397–417, Aug. 2008.
[56] M. Oltean and D. Dumitrescu, "Multi expression programming," Cluj-Napoca, Romania: Babe-Bolyai University, Tech. Rep. UBB-01-2002, Jan. 2002.
[57] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Trans. Evolut. Comput.*, vol. 1, no. 1, pp. 67–82, Apr. 1997.
[58] A. K. Qin and P. N. Suganthan, "Self-adaptive differential evolution algorithm for numerical optimization," in *Proc. IEEE Congress on Evolutionary Computation*. Sept. 2005, vol. 2, pp. 1785–1791.
[59] J. Brest, S. Greiner, B. Boskovic, M. Mernik, and V. Zumer, "Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems," *IEEE Trans. Evolut. Comput.*, vol. 10, no. 6, pp. 646–657, Dec. 2006.
[60] Z.-H. Zhan, J. Zhang, Y. Li, and H. S.-H. Chung, "Adaptive particle swarm optimization," *IEEE Trans. Syst. Man Cybern. B Cybern.*, vol. 39, no. 6, pp. 1362–1381, Dec. 2009.
[61] J. Zhang and A. C. Sanderson, "JADE: Adaptive differential evolution with optional external archive," *IEEE Trans. Evolut. Comput.*, vol. 13, no. 5, pp. 945–958, Oct 2009.
[62] J. R. Koza, *Genetic Programming: vol.1, On the Programming of Computers by Means of Natural Selection*. Cambridge, MA: MIT Press, 1992.
[63] M. Zhang and W. Smart, "Genetic programming with gradient descent search for multiclass object classification," in *2004 European Conference on Genetic Programming (EuroGP)*. New York: Springer-Verlag, Apr. 2004, pp. 399–408.
[64] M. Kommenda, G. Kronberger, S. Winkler, M. Affenzeller, and S. Wagner, "Effects of constant optimization by nonlinear least squares minimization in symbolic regression," in *Proc. 15th Annu. Conf. Companion on Genetic and Evolutionary Computation*. ACM, July 2013, pp. 1121–1128.
[65] S. Mukherjee and M. J. Eppstein, "Differential evolution of constants in genetic programming improves efficacy and bloat," in *Proc. 14th Annu. Conf. Companion on Genetic and Evolutionary Computation. ACM*, July 2012, pp. 625–626.
[66] C. Yuan, C. Tang, Y. Wen, J. Zuo, J. Peng, and J. Hu, "Convergency of genetic regression in data mining based on gene expression programming and optimized solution," *Int. J. Comput. Appl.*, vol. 28, no. 4, pp. 359–366, July 2006.
[67] X. Du, L. X. Ding, C. W. Xie, X. Xu, S. W. Wang, and L. Chen, "Convergence analysis of gene expression programming based on maintaining elitist," in *Proc.the first ACM/SIGEVO Summit on Genetic and Evolutionary Computation*. ACM, June 2009, pp. 823–826.
[68] X. Du and L. Ding, "About the convergence rates of a class of gene expression programming," *Sci. China Inf. Sci.*, vol. 53, no. 4, pp. 715–728, Mar. 2010.
[69] X. Du, Y. Ni, D. Xie, X. Yao, P. Ye, and R. Xiao, "The time complexity analysis of a class of gene expression programming," *Soft Comput.*, vol. 19, no. 6, pp. 1611–1625, Dec. 2015.
[70] Z. Huang, "Schema theory for gene expression programming," Ph.D. dissertation, Brunel Univ. School Eng. Design PhD. Theses, Sept. 2014.
[71] M. A. Yassin, A. Alazba, and M. A. Mattar, "Artificial neural networks versus gene expression programming for estimating reference evapotranspiration in arid climate," *Agric. Water Manag.*, vol. 163, pp. 110–124, Sept. 2016.
[72] A. Gholami, H. Bonakdari, A. H. Zaji, A. A. Akhtari, and S. R. Khodashenas, "Predicting the velocity field in a 90 open channel bend using a gene expression programming model," *Flow Meas. Instrum.*, vol. 46, pp. 189–192, Oct. 2015.
[73] I. Ebtehaj, H. Bonakdari, A. H. Zaji, H. Azimi, and A. Sharifi, "Gene expression programming to predict the discharge coefficient in rectangular side weirs," *Appl. Soft Comput.*, vol. 35, pp. 618–628, July 2015.
[74] Z. Wu, H. Fan, and G. Liu, "Forecasting construction and demolition waste using gene expression programming," *J. Comput. Civil Eng.*, vol. 29, no. 5, pp. 04014059, Oct. 2013.
[75] A. T. poora, A. Bararib, M. Behniac, and T. Najafid, "Determination of the ultimate limit states of shallow foundations using gene expression programming (GEP) approach," *Soils Found.*, vol. 55, no. 3, pp. 650–659, Apr. 2015.
[76] S. Rashidi and P. Ranjitkar, "Bus dwell time modelling using gene expression programming," *Comput. Aided Civil Infrastruct. Eng.*, vol. 30, no. 6, pp. 478–489, Apr. 2015.
[77] A. Karatahansopoulos, G. Sermpinis, J. Laws, and C. Dunis, "Modelling and trading the Greek stock market with gene expression and genetic programing algorithms," *J. Forecast.*, vol. 33, no. 8, pp. 596–610, Sept. 2014.
[78] A. Karatahansopoulos, G. Sermpinis, J. Laws, and C. Dunis, "Forecasting caspian sea level changes using satellite altimetry data (June 1992-December 2013) based on evolutionary support vector regression algorithms and gene expression programming," *Glob. Planet. Chang.*, vol. 121, pp. 53–63, July 2014.
[79] H. M. Azamathulla, "Gene-expression programming to predict friction factor for southern Italian rivers," *Neural Comput. Appl.*, vol. 23, no. 5, pp. 1421–1426, Aug. 2013.
[80] H. M. Azamathulla and Z. Ahmad, "Estimation of critical velocity for slurry transport through pipeline using adaptive neuro-fuzzy interference system and gene-expression programming," *J. Pipeline Syst. Eng. Pract.*, vol. 4, no. 2, pp. 131–137, July 2012.
[81] E. Dikmen, "Gene expression programming strategy for estimation performance of LiBr-H_2O absorption cooling system," *Neural Comput. Appl.*, vol. 26, no. 2, pp. 409–415, Feb. 2015.
[82] F. Gharagheizi, P. Ilani-Kashkouli, N. Farahani, and A. H. Moham-madi, "Gene expression programming strategy for estimation of flash point temperature of non-electrolyte organic compounds," *Fluid Phase Equilib.*, vol. 329, pp. 71–77, May 2012.

[83] S. S. S. Hosseini and A. H. Gandomi, "Short-term load forecasting of power systems by gene expression programming," *Neural Comput. Appl.*, vol. 21, no. 2, pp. 377–389, Mar. 2012.
[84] A. Guven and A. Aytek, "New approach for stage-discharge relationship: Gene-expression programming," *J. Hydrol. Eng.*, vol. 14, no. 8, pp. 812–820, Aug. 2009.
[85] Y. Zhang and J. Xiao, "A software reliability modelling method based on gene expression programming," *Appl. Math. Inf. Sci.*, vol. 6, no. 1, pp. 125–132, Jan. 2012.
[86] A. Baykasolu and M. Gken, "Gene expression programming based due date assignment in a simulated job shop," *Expert Syst. Appl.*, vol. 36, no. 10, pp. 12 143–12 150, Mar. 2009.
[87] A. Bakhshaii and R. Stull, "Deterministic ensemble forecasts using gene-expression programming," *Weather Forecast.*, vol. 24, no. 5, pp. 1431–1451, Oct. 2009.
[88] A. Keshavarz and M. Mehramiri, "New gene expression programming models for normalized shear modulus and damping ratio of sands," *Eng. Appl. Artif. Intell.*, vol. 45, pp. 464–472, July 2015.
[89] A. Nazari, "Prediction performance of PEM fuel cells by gene expression programming," *Int. J. Hydrogen Energy*, vol. 37, no. 24, pp. 18 972–18 980, Aug. 2012.
[90] S. Traore and A. Guven, "New algebraic formulations of evapotranspiration extracted from gene-expression programming in the tropical seasonally dry regions of west Africa," *Irrigation Sci.*, vol. 31, no. 1, pp. 1–10, Jan. 2013.
[91] M. Z. Hashmi and A. Y. Shamseldin, "Use of gene expression programming in regionalization of flow duration curve," *Adv. Water Resour.*, vol. 68, pp. 1–12, Feb. 2014.
[92] S. M. Mousavi, E. S. Mostafavi, and F. Hosseinpour, "Gene expression programming as a basis for new generation of electricity demand prediction models," *Comput. Ind. Eng.*, vol. 74, pp. 120–128, May 2014.
[93] Z. Yu, H. Lu, H. Si, S. Liu, X. Li, C. Gao, L. Cui, C. Li, X. Yang, and X. Yao, "A highly efficient gene expression programming (GEP) model for auxiliary diagnosis of small cell lung cancer," *PloS One*, vol. 10, no. 5, pp. e0125517, May 2015.
[94] K. Zhang and S. Sun, "Web music emotion recognition based on higher effective gene expression programming," *Neurocomputing*, vol. 105, pp. 100–106, June 2013.
[95] J. L. Ávila, E. Gibaja, A. Zafra, and S. Ventura, "A gene expression programming algorithm for multi-label classification," *J. Mult. Valued Logic Soft Comput.*, vol. 17, pp. 183–206, 2011.
[96] L. Teodorescu, "Gene expression programming approach to event selection in high energy physics," *IEEE Trans. Nucl. Sci.*, vol. 53, no. 4, pp. 2221–2227, Aug. 2006.
[97] J. Zhong and W. Cai, "A hyper-heuristic framework for agent-based crowd modeling and simulation," in *Proc. Int. Conf. Autonomous Agents & Multiagent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2016, pp. 1331–1332.
[98] X.-S. Yan, W. Wei, R. Liu, S.-Y. Zeng, and L.-S. Kang, "Designing electronic circuits by means of gene expression programming," in *Proc. First NASA/ESA Conference on Adaptive Hardware and Systems*, 2006, pp. 194–199.
[99] Y. Yang and A. Alexeev, "Designing neural networks using gene expression programming," in *Proc. Applied Soft Computing Technologies: The Challenge of Complexity*, 2006, pp. 517–535.
[100] F. M. Janeiro and P. M. Ramos, "Sensor characterization using gene expression programming evolutionary algorithms," in *Proc. Instrumentation and Measurement Technology Conf.*, May 2012, pp. 1–5.
[101] F. M. Janeiro, J. Santos, and P. M. Ramos, "Gene expression programming in sensor characterization: Numerical results and experimental validation," *IEEE Trans Instrum Meas.*, vol. 62, no. 5, pp. 1373–1381, May 2013.
[102] A. Baykasolu, "Gene expression programming based meta-modelling approach to production line design," *Int. J. Comput. Integrat. Manuf.*, vol. 21, no. 6, pp. 657–665, Aug. 2008.
[103] J. Mwaura and E. Keedwell, "Evolving robot subbehaviour modules using gene expression programming," *Genet. Program. Evolvable Mach.*, vol. 16, no. 2, pp. 95–131, June 2015.
[104] N. R. Sabar, M. Ayob, G. Kendall, and R. Qu, "Automatic design of a hyper-heuristic framework with gene expression programming for combinatorial optimization problems," *IEEE Trans. Evolut. Comput.*, vol. 19, no. 3, pp. 309–325, June 2015.
[105] L. Nie, X. Shao, L. Gao, and W. Li, "Evolving scheduling rules with gene expression programming for dynamic single-machine scheduling problems," *Int. J. Adv. Manuf. Technol.*, vol. 50, no. 5–8, pp. 729–747, Sept. 2010.
[106] L. Nie, L. Gao, P. Li, and X. Li, "A GEP-based reactive scheduling policies constructing approach for dynamic flexible job shop scheduling problem with job release dates," *J. Intell. Manuf.*, vol. 24, no. 4, pp. 763–774, Aug. 2013.
[107] K. Xu, Y. Liu, R. Tang, J. Zuo, J. Zhu, and C. Tang, "A novel method for real parameter optimization based on gene expression programming," *Appl. Soft Comput.*, vol. 9, no. 2, pp. 725–737, Sept. 2009.
[108] H. M. Azamathulla, "Gene expression programming for prediction of scour depth downstream of sills," *J. Hydrol.*, vol. 460, pp. 156–159, June 2012.
[109] C. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. New York: Springer-Verlag, 2007.
[110] P. G. Espejo, S. Ventura, and F. Herrera, "A survey on the application of genetic programming to classification," *IEEE Trans. Syst. Man Cybern. Syst. Part C*, vol. 40, no. 2, pp. 121–144, Mar. 2010.
[111] V. K. Karakasis and A. Stafylopatis, "Efficient evolution of accurate classification rules using a combination of gene expression programming and clonal selection," *IEEE Trans. Evolut. Comput.*, vol. 12, no. 6, pp. 662–678, Dec. 2008.
[112] W. R. Weinert and H. S. Lopes, "GEPCLASS: A classification rule discovery tool using gene expression programming," in *2006 International Conference on Advanced Data Mining and Applications*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 871–880.
[113] C. Helmberg, "Semidefinite programming for combinatorial optimization," Tech. Rep. ZIB-Report ZR-00-34, Konrad-Zuse-Zentrum Berlin, Oct. 2000.
[114] Z.-H. Zhan, J. Zhang, Y. Li, and Y.-H. Shi, "Orthogonal learning particle swarm optimization," *IEEE Trans. Evolut. Comput.*, vol. 15, no. 6, pp. 832–847, Dec. 2011.
[115] Y.-W. Leung and Y. Wang, "An orthogonal genetic algorithm with quantization for global numerical optimization," *IEEE Trans. Evolut. Comput.*, vol. 5, no. 1, pp. 41–53, Feb. 2001.
[116] W.-N. Chen, J. Zhang, Y. Lin, N. Chen, Z.-H. Zhan, H. S.-H. Chung, Y. Li, and Y.-H. Shi, "Particle swarm optimization with an aging leader and challengers," *IEEE Trans. Evolut. Comput.*, vol. 17, no. 2, pp. 241–258, Apr. 2013.
[117] F. C. E. Mininno, F. Neri, and D. Naso, "Compact differential evolution," *IEEE Trans. Evolut. Comput.*, vol. 15, no. 1, pp. 32–54, Feb. 2011.
[118] S. Rahnamayan, H. R. Tizhoosh, and M. M. Salama, "Opposition-based differential evolution," *IEEE Trans. Evolut. Comput.*, vol. 12, no. 1, pp. 64–79, Feb. 2008.
[119] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evolut. Comput.*, vol. 6, no. 2, pp. 182–197, Apr. 2002.
[120] E. Zitzler and S. Knzli, "Indicator-based selection in multiobjective search," in *Proc. Parallel Problem Solving from Nature*, 2004, pp. 832–842.
[121] Q. Zhang and H. Li, "MOEA/D: A multi-objective evolutionary algorithm based on decomposition," *IEEE Trans. Evolut. Comput.*, vol. 11, no. 6, pp. 712–731, Dec. 2007.
[122] U. Bhowan, M. Johnston, M. Zhang, and X. Yao, "Evolving diverse ensembles using genetic programming for classification with unbalanced data," *IEEE Trans. Evolut. Comput.*, vol. 17, no. 3, pp. 368–386, June 2013.
[123] X. Chen, Y.-S. Ong, M.-H. Lim, and K. C. Tan, "A multi-facet survey on memetic computation," *IEEE Trans. Evolut. Comput.*, vol. 15, no. 5, pp. 591–607, Oct. 2011.
[124] Y.-S. Ong, M.-H. Lim, and X. Chen, "Memetic computation past, present & future [research frontier]," *IEEE Comput. Intell. Mag.*, vol. 5, no. 2, pp. 24–31, May 2010.
[125] Y. Kameya, J. Kumagai, and Y. Kurata, "Accelerating genetic programming by frequent subtree mining," in *2008 Genetic Evol. Comput. Conf.* New York, NY, USA: ACM, July 2008, pp. 1203–1210.
[126] M. Iqbal, W. Browne, and M. Zhang, "Reusing building blocks of extracted knowledge to solve complex, large-scale boolean problems," *IEEE Trans. Evolut. Comput.*, vol. 18, no. 4, pp. 465–480, Aug. 2014.
[127] Y. Jin, "A comprehensive survey of fitness approximation in evolutionary computation," *Soft Comput.*, vol. 9, no. 1, pp. 3–12, Jan. 2005.
[128] Z. H. Zhou, N. V. Chawla, Y. Jin, and G. J. Williams, "Big data opportunities and challenges: Discussions from data analytics perspectives [discussion forum]," *IEEE Computat. Intell. Mag.*, vol. 9, no. 4, pp. 62–74, Nov. 2014.
[129] Y. Zhai, Y.-S. Ong, and I. W. Tsang, "The emerging "big dimensionality," *IEEE Computat. Intell. Mag.*, vol. 9, no. 3, pp. 14–26, Aug. 2014.
[130] A. Brazma, H. Parkinson, U. Sarkans, M. Shojatalab, J. Vilo, N. Abeygunawardena, E. Holloway, M. Kapushesky, P. Kemmeren, G. G. Lara, A. Oezcimen, P. Rocca-Serra, and S.A. Sansone, "ArrayExpressła public repository for microarray gene expression data at the EBI," *Nucleic Acids Res.*, vol. 31, no. 1, pp. 68–71, Jan. 2003.
[131] R. Caruana, "Multitask learning," *Mach. Learn.*, vol. 28, no. 1, pp. 41–75, 1997.
[132] T. Evgeniou and M. Pontil, "Regularized multitask learning," in *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: ACM, 2004, pp. 109–117.
[133] T. Evgeniou, C. A. Micchelli, and M. Pontil, "Learning multiple tasks with kernel methods," *J. Mach. Learn. Res.*, vol. 6, no. 4, pp. 615–637, Apr. 2005.
[134] O. Chapelle, P. Shivaswamy, S. Vadrevu, K. Weinberger, Y. Zhang, and B. Tseng, "Multi-task learning for boosting with application to web search ranking," in *Proc. 16th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining*, pp. 1189–1198. 2010.
[135] P. Gong, J. Ye, and C. Zhang, "Robust multi-task feature learning," in *Proc. 18th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining*, 2012, pp. 895–903.
[136] M. Iqbal, "Improving the scalability of XCS-based learning classifier systems," Ph.D. dissertation, Victoria University of Wellington. 2014.
[137] A. Gupta, Y.-S. Ong, and L. Feng, "Multifactorial evolution: Toward evolutionary multitasking," *IEEE Trans. Evolut. Comput.*, vol. 20, no. 3, pp. 343–357, June 2016.
[138] A. Mukhopadhyay, U. Maulik, S. Bandyopadhyay, and C. A. C. Coello, "A survey of multiobjective evolutionary algorithms for data mining: Part I," *IEEE Trans. Evolut. Comput.*, vol. 18, no. 1, pp. 4–19, Feb. 2014.
[139] H. K. S. M. Asafuddoula and T. Ray, "Six-sigma robust design optimization using a many-objective decomposition-based evolutionary algorithm," *IEEE Trans. Evolut. Comput.*, vol. 19, no. 4, pp. 490–507, Aug. 2015.
[140] J. G. Herrero, A. Berlanga, and J. M. M. Lpez, "Effective evolutionary algorithms for many-specifications attainment: Application to air traffic control tracking filters," *IEEE Trans. Evolut. Comput.*, vol. 13, no. 1, pp. 151–168, Feb. 2009.
[141] Y. Zheng, L. Jia, and H. Cao, "Multi-objective gene expression programming for clustering," *Inf. Technol. Control*, vol. 41, no. 3, pp. 283–294, 2012.
[142] S. Jiang and S. Yang, "A steady-state and generational evolutionary algorithm for dynamic multiobjective optimization," *IEEE Trans. Evolut. Comput.*, vol. 21, no. 1, pp. 65–82, Feb. 2017.
[143] S. Yang, "Evolutionary computation for dynamic optimization problems," in *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO Companion '15. New York, NY, USA: ACM, 2015, pp. 629–649.
[144] M. Mavrovouniotis, F. M. Mller, and S. Yang, "Ant colony optimization with local search for dynamic traveling salesman problems," *IEEE Trans. Cybern.*, pp. 1–14. in pressing 2016.
[145] M. Liu, H. K. Singh, and T. Ray, "A memetic algorithm with a new split scheme for solving dynamic capacitated arc routing problems," in *Proc. IEEE Congress on Evolutionary Computation*, July 2014, pp. 595–602.