A Survey of Genetic Programming and Its Applications

Milad Taleby Ahvanooey[1], Qianmu Li[1,2], Ming Wu[1], Shuo Wang[1]
[1] School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing, P.O. Box 210094 P.R. China.
[2]Intelligent Manufacturing Department, Wuyi University, Jiangmen, P.O. Box 529020 P.R, China.
[E-mail: Taleby@njust.edu.cn, Qianmu@njust.edu.cn, Wuming@njust.edu.cn, Sharon_Wang@njust.edu.cn]
*Corresponding Authors : Milad Taleby Ahvanooey & Qianmu Li

Received December 28, 2017; revised April 27, 2018; revised August 21, 2018; accepted November 13, 2018; published April 30, 2019

Abstract

Genetic Programming (GP) is an intelligence technique whereby computer programs are encoded as a set of genes which are evolved utilizing a Genetic Algorithm (GA). In other words, the GP employs novel optimization techniques to modify computer programs; imitating the way humans develop programs by progressively re-writing them for solving problems automatically. Trial programs are frequently altered in the search for obtaining superior solutions due to the base is GA. These are evolutionary search techniques inspired by biological evolution such as mutation, reproduction, natural selection, recombination, and survival of the fittest. The power of GAs is being represented by an advancing range of applications; vector processing, quantum computing, VLSI circuit layout, and so on. But one of the most significant uses of GAs is the automatic generation of programs. Technically, the GP solves problems automatically without having to tell the computer specifically how to process it. To meet this requirement, the GP utilizes GAs to a “population” of trial programs, traditionally encoded in memory as tree-structures. Trial programs are estimated using a “fitness function” and the suited solutions picked for re-evaluation and modification such that this sequence is replicated until a “correct” program is generated. GP has represented its power by modifying a simple program for categorizing news stories, executing optical character recognition, medical signal filters, and for target identification, etc. This paper reviews existing literature regarding the GPs and their applications in different scientific fields and aims to provide an easy understanding of various types of GPs for beginners.

Keywords: Automatic Programming, Genetic Programming, Genetic Algorithm, Genetic Operators;

This paper supported by the Project of ZTE Cooperation Research (2016ZTE04-11), Jiangsu province key research and development program: Social development project (BE2017739), Jiangsu province key research and development program:Industry outlook and common key technology projects (BE2017100), 2018 Jiangsu Province Major Technical Research Project "Information Security Simulation System";

http://doi.org/10.3837/tiis.2019.04.002 **ISSN : 1976-7277**

1. Introduction

Genetic programming is a type of Evolutionary Algorithms (EAs), a subset of machine learning, i.e., a search algorithm inspired by the Darwinian's theory of biological evolution. For the first time, the GP was introduced by Mr. John Koza which enables computers to solve problems without being clearly programmed [1]. The GP functions based on John Holland's GAs to generate programs for solving various complex optimization and search problems automatically. In the 1970s, Holland designed the GA as a way of exploiting the potential of the natural evolution to employ on computers. Natural evolution has observed the growth of complex organisms such as animals and plants from simpler single-celled life forms. Holland's GAs are models of the vitals of natural evolution and inheritance [2-3]. During the past forty years, the GPs have been applied to solve a wide range of complex optimization problems, patentable new inventions, producing a number of human-competitive results, etc. in the emerging scientific fields. Like many other fields of computer science, GP still is developing briskly, with new ideas and applications being continuously advanced [4]. While it demonstrates how remarkably abundant GP is and, moreover, makes it hard for new researchers to get familiarized with the main ideas of the GP. Even for students who slightly anxious in this field for a while, it is challenging to keep up by the pace of new advancements.

During the last four decades, many books and survey papers have been written and published on various aspects of GP [1-30]. Some provided a profound introduction to the GPs and GPAs as a whole, and others presented a detailed introduction to them by focusing on specific application domains. Hence, there has been written no comprehensive survey on GP during the last decade, and most of the newcomers aim to learn about various types of GP and its applications due to having a variety of applications in different scientific fields such as computer science, biomedical, chemistry, etc. It caused to draw towards writing an easy overview to help their understanding about the GPs. This survey aims to fill the gap, by providing an easy understanding of various types of GP for both newcomers and researchers. The main contributions of this survey are briefly expressed as follows.

- We overview some existing literature on the GPs such as definitions, workflow, and operations, etc.
- We investigate various types of GPs and their applications.
- We suggest some guidelines and directions to provide an easy understanding of GPs for newcomers.

The rest of this paper is organized as follows. Section (2) affords an overview of the existing literature concerning GPs. Section (3) describes various types of GP with some highlight capabilities and limitations and introduces some applications of various types of GP among related source codes for each kind of GP separately. Section (4) suggests some guidelines and research directions. Finally, section (4) draws some conclusions.

2. Literature Review

2.1. Definition of GAs and GPs

This section provides a basic introduction to GAs and GPs. It summarizes some terms and

explains how a simple GA works, but it is not a complete tutorial, i.e., for more detail background information, we suggest readers look the books in [4], [20]. The base of GAs is some characteristics which are inspired by plants and animals. The growth of species (e.g., plants from seeds, animals from eggs, etc.), is controlled by the genes which are inherited from their parents. The genes are stashed on one or more threads of DNA. The DNA is a copy of the parent's DNA in case of asexual reproduction, likely with some random mutations. In the same trends, DNA from two parents is inherited through the new child in sexual reproduction. Often about half of each parent's DNA may transfer to a child where it combines with DNA copied from another parent. The child's DNA is mainly changed from that in either parent.

Holland [2] introduced GAs in the early 1970s as computer programs that imitate the process of evolutionary development in nature. GAs evolve a population of possible solutions to solve complex optimization and search problems. Particularly, the GAs work on encoded models (symbols) of the solutions (i.e., similar to the genetic material of individuals in nature), and they do not operate exactly on the solutions themselves. Moreover, Holland's GA works based on the encoding of solutions as binary strings from a binary alphabet as in nature; the selection makes the essential driving mechanism to achieve better solutions to remain. Every solution is assigned with a fitness value which indicates how to fit it is at solving the problem by comparing with other solutions in the population, i.e., the superior fitness value of a gene, the superior changes of survival, reproduction, and the larger its symbol in the succeeding generation. The process of recombination for genetic material in GAs is formulated by a crossover procedure which swaps parts between binary strings. Another operation, named mutation, produces a sporadic and random change through the bits of binary strings. The mutation also has a straight analogy from nature that can play the role of reproducing lost genetic material [3-10].

As depicted in **Fig. 1**, we designed a workflow to illustrate the primary steps of developing a GA. In the first step, GA begins from problem analysis to estimate the solution domain and determines fitness function to assess the solution domain. In the second step, a specific binary string (or real code) is assigned to denote each solution. In the third step, an initial population is randomly produced. Afterward, genetic operators consisting of "selection," "crossover," and "mutation" are presented for reproducing new solutions. Finally, by repetitive application of genetic operators and fitness evaluations, an optimal solution will be achieved till GA faces the termination and solution criteria [10-30].

The main five key steps (operations) of GAs are summarized as the following points.

- Initialization (Problem Analysis): this strategy involves population parameter setting up, consisting of the greatest evolutional generation, value size, a probability of crossover and mutation rate. Nevertheless, the setting desirable values for these parameters are challenging in designing a practical GA, and there is no specific standard [1], [4], [20], [25], [31].
- Fitness: this is a numeric value allocated to every member of a population to afford a measure of the proportion of a solution to the problem. The fitness measure may combine

any countable, observable, measurable characteristic, behavior or combination of behaviors or features. The fitness measure is indicated in terms of "what requires to be performed" not "how to process it" [11]. A fitness function is a procedure which denotes the fitness of a gene as a solution to the problem in which the aim is to discover a gene with a minimum (or maximum) fitness [1].

- Selection: this operator is a mechanism for picking genes from the current population to reproduce a new generation. There have been proposed a lot of selection methods so far (e.g., stochastic, linear, roulette wheel, tournament, truncation, and so on) [1], [4], [20], [32].
- Crossover: it is a combination operator which produces a child by recombining selected parts from its parent during the evolutionary process [1], [4], [20-25].
- Mutation: it randomly manipulates a small part of the genetic material (genotype) of one selected parent [1], [4], [22-25].
- Termination: it is a significant part of GP which evaluates Pareto scoring criteria (e.g., functionality and efficiency) for achieving a proper argument in time to discontinue the search. There exist three different termination strategies including termination after a fixed generation, termination until the solution reaches the pre-set optimal requirement, or termination after the Pareto-optimal solution with no better results can be generated [1], [12], [22].
-

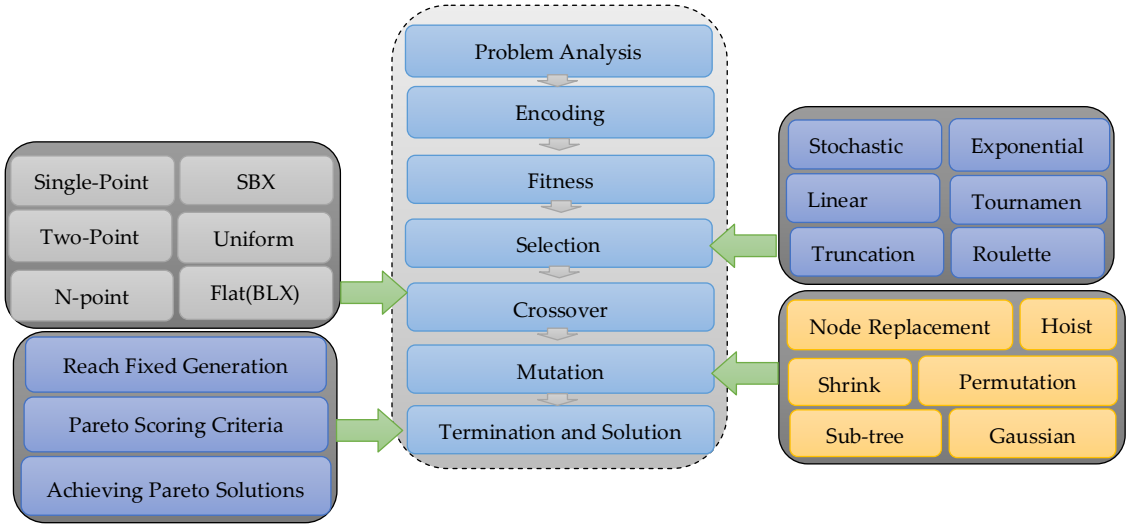


Fig. 1. A workflow of GAs, the primary process is the dashed box; other optional items are methods for each function operator.

Those five key steps mentioned above considerably affect the efficiency of GAs. For instance, a higher crossover probability may cause premature convergence, and theretofore a higher mutation rate may terminate in the loss of proper solutions. **Fig. 2** shows a standard flowchart of the key steps of GAs.

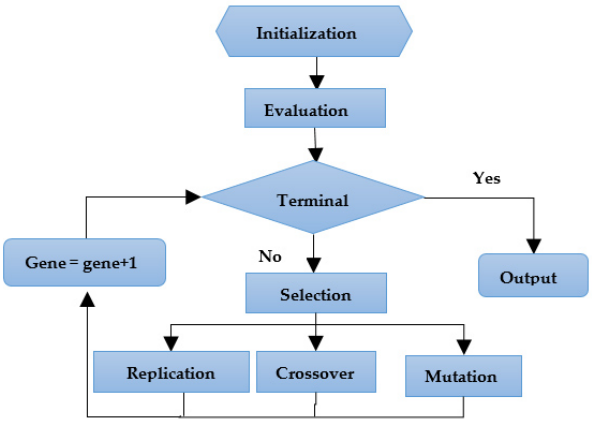


Fig. 2. the standard flowchart of GP [25], [26]

To design a GP, we require to define specific components to mimic the evolutionary process. These components include decision rules (or variables), arithmetic operations (or functions), and genetic operators such as crossover and mutation, to figurative expressions.

The figurative expressions referred to as solutions (or genes) which are produced for shaping the initial population. A population in the GA is a set of possible solutions during an iteration process of the algorithm. In general, the initial expressions are generated by tree structure based encoding. **Fig. 3** depicts a few instances of such trees. These expressions are shaped by components from two different parameter groups: (II) arithmetic operations or functional primitives (e.g., cos, +, *, sin, ln, etc.), and (III) system decision-variables (terminal set), e.g., r, π, b, etc. The arguments for operations are entered from the terminal set that includes the constants, decision parameters or other variables as listed in **Table 1**. The initial solutions are typically bounded based on the length of expression or tree depth for assigning the first population in the GA by the possible building blocks to be expanded at new steps of the evolutionary process [33-40]. For example: let us suppose that, we want to design a GP to calculate $y = \pi r^2$. To solve this problem, the population of programs might be included a program that computes $y = (\pi * (r * r))$. Therefore, fitness could be obtained by performing each program with each of 'x' values and checking each answer with the corresponding 'y' value. As depicted in **Fig. 3**, the bold circles indicate crossover points on the parents. It forms each child by swapping such nodes from the parents. When a picked child (shown bold) is shifted from the Dad program and added in the Mum (shifting the existing child or offspring, also is highlighted), a new child is generated that may possess even high fitness. In the result, (Child1) actually calculates $y = \pi * r^2$ and, therefore, it is the output of our GP.

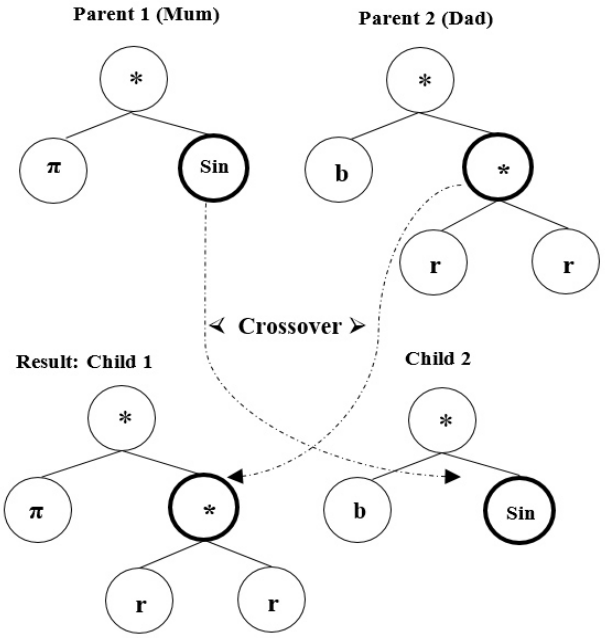


Fig. 3. Tree-based expression of the stated GP and an instance of crossover operation: Child1: $\pi * r^2$, Child2: $b * sin$, Mum: $\pi * sin$, Dad: b*r^2 [16, 34]

The created solutions are finally arithmetic equations that implicitly give the correlation between the decision rules of the system and a related efficiency metric. Thus, each new child (or generated solution) sub-tree is assigned by a fitness value, that refers to how exact the expression describes the training data. The fitness of a child verifies the expression's dependent ability to survive and produce the next generation during the evolutionary process. As shown in **Fig. 3**, to reproduce the subsequent population of new solutions, the child solution could further undergo small alterations using mutation to enable local search. This process can repeat at each generation till a new population is shaped. Evolution is concluded while a stopping criterion, such as a pre-set superiority or computational cost, is satisfied [16], [34].

Table 1. Preliminary Parameters of the GP [16]

Parameters	**Values**
Pop size	1000
Minimum initial tree size	2
Maximum initial tree size	4
Maximum solution length	30
Evaluation limit	10^6
Initialization method	Ramped half and half
Selection strategy	Tournament with size 2
Arithmetic operations	{*, +, -, x^2, sin, cos, ln, etc.}

Algorithm 1 explains the stages of a GP in details according to the predefined parameters in **Table1.** Herein, the "gene" is a term that means individual or chromosome in some existing literature.

Algorithm1: Pseudo-framework of the standard GPs [34], [48]

```
1.  Procedure Genetic_ Algorithm
2.  Input: Setup GP according to the mentioned parametters in Table.1, retrieve training data;
3.  begin
4.  N = population size;
5.  P = create parent population by randomly creating N genes;
6.     While (not done)
7.        C = create empty child population;
8.         While (not enough genes in C)
9.           Parent1 = select parent;
10.          Parent2 = select parent;
11.          Child1, Child2 = Crossover (Parent1, Parent2)
12.          Mutate child1, child2;
13.          Evaluate child1, child2 for fitness;
14.          Insert child1, child2 into C;
15.       End while
16.       P = combine P and C somehow to get N new genes
17.    End while
18.   Return Optimal/best solution so far;
18. End Procedure
```

Many platforms offer specific features in order to implement the GAs. The most popular platforms for performing GAs are the MATLAB, Java, C++, and Python. For example, "Algorithmic Trading program" is an example of GP which is written in python and its source code can be found in [49].

2.2. Selection Strategies

The first two preliminary steps represent the primitive set for GP, and hence, contingently determine the search space GP will seek. This consists of all the programs which could be created by making the primitives in all feasible ways. Nevertheless, the GP does not recognize which regions or elements of this search space are sufficient at this stage (that is, consist of programs which solve or almost solve the problem). Indeed, it is the duty of the fitness measure, that adequately defines the desired purpose of the search process. The fitness measure is only the primary mechanism for providing a high-level statement of the problem's requirements to the GP system. Depending on the optimization problem at hand, fitness could be estimated in terms of the quantity of error among its result and the appropriate output. Also, the amount of time (e.g., money, fuel, and the like) is needed to lead a system to the proper target state, the accuracy of the program in identifying patterns or grouping objects into classes, the final result which a game-playing program builds, the compliance of a structure with user-specified design criteria, and so on [20], [21], [22]. Individuals for creating child or

offspring are selected using a selection strategy after evaluating the fitness value of each gene during the selection process [7]. In other words, the selection strategy determines which one of the genes in the current generation can be applied for reproducing a new child in hopes that the next generation may possess greater fitness. The selection operator is accurately expressed to ensure which fit members of the population (with higher fitness) have a higher probability of being chosen for mutating, but those critical members of the population still have a low possibility of being chosen. Moreover, it essential to guarantee that the search process is universal and does not directly converge to the nearest local optimum genes. Various types of selection mechanisms have different procedures for evaluating the selection probability. The selection approaches evolve genes (solutions) based on the decision rules and, therefore, reproduce new solution (with higher fitness) by passing through the genetic material for generating the next generation in the form of the children. There have been introduced many types of selection strategies so far. Also, we describe four major selection methods including; proportionate reproduction (roulette wheel), tournament, rank based, and truncation, etc. More descriptions of selection strategies can be found in [41-46].

2.2.1 Proportionate Reproduction or Roulette Wheel

Proportionate reproduction was proposed by Holland [2], supposed that the genes are chosen according to their probabilities which are equal to their fitness values. This process is an electing principle which is similar to the roulette wheel. In the roulette wheel, the possibility of choosing a sector is equal to the magnitude of the central angle of the sector. Similarly, in the GA, the total population is divided on the wheel, and each part indicates a child. The proportion of the child's fitness to the whole fitness values of the total population determines the selection probability of that gen in the next generation. Therefore, it selects the area engaged over the gene on the wheel [31], [47]. The proportional roulette wheel strategy is illustrated in **Fig. 4.**

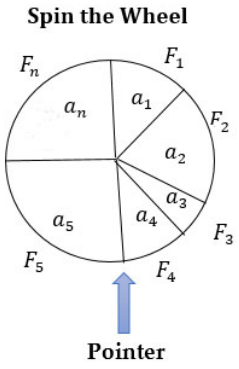


Fig. 4. Roulette wheel selection strategy [31], [47]

Following points are the key steps of the Roulette Wheel selection strategy.

I. Calculating the total of the fitness values for all genes in the population.
II. Computing the fitness value of each gene and the proportion of each gene's fitness value to the result fitness values of all genes in the whole population. The proportion denotes the probability of the gene to be chosen.
III. Partitioning the roulette wheel into segments based on the proportions obtained in the second step. Every segment represents a gene. The area of the segment is proportional to the gene's probability to be chosen.
IV. Spinning the wheel 'n' times, i.e., 'n' is the number of genes in the population. Therefore, when the spinning of Roulette-wheel stops, the segment on which the pointer indicates the corresponding gene being chosen.

Let's suppose that a population with size n, $P=\{a_1,a_2,a_3,...,a_n\}$, each a_i has the fitness value of $f(a_i)$, thus the probability of a_i being chosen can be calculated as follows.

$$P(a_i)=\frac{f(a_i)}{\sum_{j=1}^{n} f(a_j)}, i, j=1,2,...,n \quad (1)$$

Algorithm.2: Procedure for roulette wheel strategy [31]

```
1. Procedure Roulette_wheel_selection
2. While (Population size <Pop size do)
3.  Generate Pop size random number (r)
4.   Calculate Cumulative fitness, total fitness (P_i) and obtain sum of proportional fitness (sum)
5.    Spin the wheel pop size times
6.    If sum < r then
7.    Select the first gene (child), else select jth gene
8.     End if
9.   End while
10.  Return genes with fitness value proportional to the size of selected wheel segment
11. End Procedure
```

The main advantage of roulette wheel strategy is that this method never knocks off the genes in the population and provides an opportunity for all of the genes to be chosen. However, the proportionate selection has a few disadvantages. For instance, if an initial population includes one or more very appropriate but not the best ones and the remaining of the population are not fit, then the proper genes will be occupied the whole population and avoid the rest part of the population from exploring other suitable genes. Practically, it is very hard to use of roulette wheel selection on the problems of minimization whereby the fitness function for minimization must be transformed to maximization function as in the case of the Traveling Salesman Problem (TSP). The outline of the Roulette Wheel strategy is given by **Algorithm 2**. An example of a roulette wheel selection is written in MATLAB, which can be found in ref [50]. In general, proportionate reproduction refers to a group of selection strategies which select genes for reproduction according to their fitness values f. In these strategies, the $P(a_i)$ of a gene from the *ith* class in the generation is calculated by Eq.1. Various strategies have been introduced for sampling this probability distribution, consisting; roulette wheel selection [63], stochastic remainder selection [64], [65], and stochastic universal selection [66], [67].

2.2.2. Tournament Selection

Tournament selection is one of the most significant selection methods in the GAs due to having high effectivness and it is easy to implement by the existing platforms [41], [45]. In this strategy, (n) existing genes (parents) are chosen randomly from the larger population, and the picked genes compete with each other (dependent on the tournament size, commonly 2). The gene by the highest fitness is assigned as one of the next generation population. This strategy can control the selection pressure easily by altering the tournament size so that if the tournament size is larger than weak genes, then they have a smaller chance to be chosen. It also provides an opportunity for all genes to be chosen and it retains diversity, although preserving diversity may reduce the convergence speed. **Fig. 5** depicts the strategy of tournament selection, and, moreover, the outline of tournament selection is given by **Algorithm 3**. In practice, the tournament selection has low complexity and can work on parallel architectures [31], [43], [45].

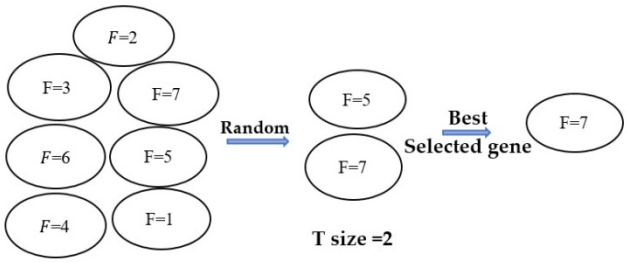


Fig. 5. The process of tournament selection strategy

In some cases, the reverse tournament selection is utilized in steady state GP where the gene by the worst fitness is picked to be exchanged by a newly generated gene (child). The tournament selection method provides a tradeoff to be considered among exploration and exploitation of the gene pool [1]. Let's assume that, k is equal to (10*N) in **Algorithm 1**.

Algorithm.3: Tournament selection [48], [49]

```
1. Function Tournament_Selection(Pop, k)
2. Best = null;
3.     For( i= 1; i <k; i++)
4.       Ind= pop [random (1,N)];
5.     if (Best=null) or (fitness=Ind) then
6.         Best=Ind;
7.          End if
7.       End for
14.  Return Best;
11. End function
```

An example of tournament selection is written in MATLAB, which can be found in ref [51].

2.2.3. Ranked Based Selection

The Ranking selection was presented by Baker to resolve the disadvantages of proportionate reproduction [44]. In this strategy, the genes are first ordered based on their fitness values and, afterward, the ranks are allocated to them. Best gene achieves rank 'N,' and the worst one

achieves rank '1'. Therefore, the selection probability is allocated linearly to the genes according to their ranks.

$$P_i = \frac{1}{N} n^- + (n^+ - n^-); i \in \{1, ..., N\} \quad (2)$$

The Eq.2. states that Pi is the probability of selection of the ith gene, and, $\frac{n^+}{N}$ is the selection probability of the best gene and, moreover, $\frac{n^-}{N}$ is the selection of probability of the worst gene. Each gene achieves a dissimilar rank even if their probabilities are equal. The Ranking process consists of two steps. In the first step, it orders the population according to the fitness values and in the second step, it allocates the ranks according to the corresponding fitness values to proportionate Selection. Rank based selection utilizes a function to map the indexes of genes in the sorted list to their selection probabilities. However, the mapping procedure could be non-linear (non-linear ranking) or linear (linear ranking), the goal of rank based selection has remained unchanged. The efficiency of the selection strategy depends on the mapping function. Practically, the mapping function includes a sort algorithm which takes O($n \log n$) computational cost. Thus, the computational complexity of the Ranking selection is O($n \log n$) + complexity of the selection (e.g., amounting between O(n) and O(n^2)) [31],[44]-[45]. The outline of the Ranked based strategy is given by **Algorithm. 4**.

Algorithm.4: Procedure for ranked based selection [31]

1. **Procedure** Ranked_Based_selection
2. **While (**Population size <Pop size **do)**
3. Sort population according to rank
4. Assign fitnesses to genes according to linear rank function
5. Generate Pop size random number (r)
6. Calculate Cumulative fitness, total fitness (P_i) and obtain sum of proportional fitness (sum)
7. Spin the wheel pop size times
8. **If** sum < r **then**
9. Select the first gene (child), else select jth gene
10. **End if**
11. **End while**
12. Return genes with fitness values proportional to the size of selected wheel segment
11. **End Procedure**

An example of ranked based selection is written in MATLAB, which can be found in ref [52].

2.2.4. Truncation Selection

For the first time, Muhlenbein has introduced the Truncation method to the domain of GAs [55]. Truncation is a selection strategy for choosing potential solutions by recombining of genes after the reproduction method. In this selection, the candidate genes are sorted by the fitness values, and some proportion, t, ($e.g. t = \frac{1}{2}, \frac{1}{3}, etc.$), of the fittest genes are chosen and reproduced $\frac{1}{t}$ times. The main advantage of the Truncation is that it is less sophisticated than other selection methods, and is not used frequently in practice. Moreover, due to the sorting process of the population, the Truncation strategy has a time complexity of $O(nlogn)$ [31], [53-55].

2.2.5. Exponential Selection

This method is also a type of rank based strategy (different from linear ranking selection) in a way that the probabilities in this strategy are exponentially calculated. The base of the exponent is C, where $0 < C < 1$.

$$P_i = \frac{C^{N-i}}{\sum_{j=1}^{N} C^{N-j}}, i, j \in \{1, ..., N\} \quad (3)$$

Here, the $\sum_{j=1}^{N} C^{N-j}$ normalizes probabilities to guarantee that $\sum_{i=1}^{N} P_i = 1$.

The outline of both algorithms the linear ranking and the exponential ranking is similar together, but the difference is in the calculation of probabilities. Also, it also allocates rank 'N' to the best gene, and rank '1' to the worst one [45], [46], [61]. Therefore, the total time complexity of GAs on exponentially scaled problems is "quadratic" or O(n^2) [62]. The rate of selection adjusts the population precentage which permits to reproduce in each generation. For proportionate, rank based, tournament, and truncation selection, it is often '1' so that all genes possess a chance of reproducing no matter whether it is small, but smaller values are also feasible so that only the top X% are qualified to reproduce. If the selection is elitist, then some percentage of the fittest genes will be ensured inclusion in the next generation. The literature consist of many (parent) selection strategies not completely categorized in the above, but almost most of the selection strategies inspired from those five majors to select the fittest genes in the existing GAs.

2.3. Crossover Operators

Crossover or recombination of genes is one of the key genetic operators which merges program structures during the evolutionary process called building blocks (BBs), and it has changed the rule in practically all the GP's associated researches after including as the primary operator in the GAs [56]. Commonly, after two genes are picked from the population, the basic crossover or standard one randomly chooses a node in each child tree excluding the root of the tree. Afterward, it swaps the two subtrees rooted with the picked nodes (named crossover points) concerning the two parent trees to reproduce two new genes (children). The recombination of genes which randomly selects the crossover points and ignores the semantics of the parents, moreover, it can frequently disorder valuable building blocks of tree structures. To solve this problem, much research has been introduced for improving the standard crossover operators [46], [56-60]. The rest of this section describes four popular crossover operations in the GAs.

2.3.1 Single or One-Point Crossover

One-point crossover is one of the simplest and elementary crossover operators which often used in the GAs. This method includes selecting a gene randomly to cut the parent genes through two new generation. For example, the parent1 ($p1$) and parent2 ($p2$) of length (l), and, in addition, a random number between (l) and ($l-1$) is chosen. Each parent is shifted into $p1_{left}$, $p1_{right}$, $p2_{left}$ and $p2_{right}$. The children are joined into $p1_{left}|p2_{right}$, and $p2_{left}|p1_{right}$, as depicted in **Fig. 6** [68-70].

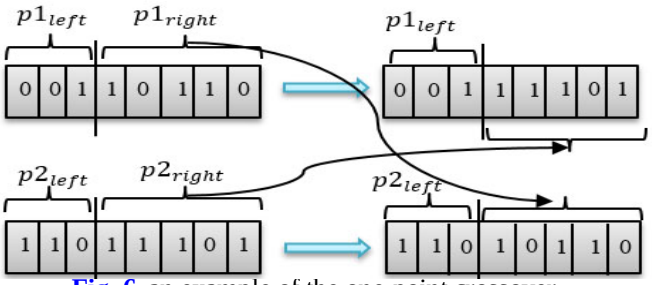


Fig. 6. an example of the one-point crossover

2.3.2. N-Point Crossover

This operator includes dividing the parents into N segments and then joins their points to reproduce a new child. These points are chosen similar to that of one-point crossover, which here instead of one pint, N cut points randomly will be selected from two parents at the same locations [68-69]. **Fig. 7** depicts an example of n-point crossover by n=2.

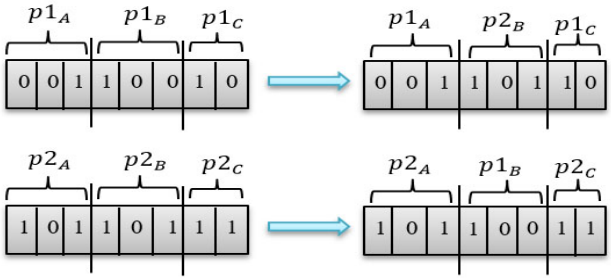
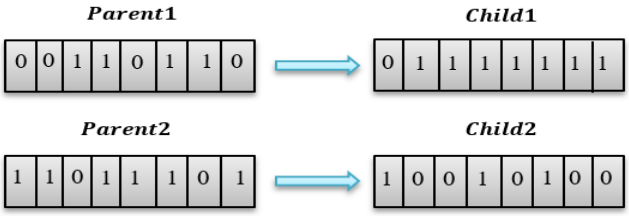


Fig. 7. an example of N-point crossover

2.3.3. Uniform Crossover

Uniform crossover determines which parents can be used for reproducing a new gene by uniformity in combining the bits of both parents. In other words, it operates by exchanging bits from the parents into a new child based on a probability value or a uniform random number p (between 0 to 1). Practically, the p value determines which child can use l^{th}genomes from the parent1 or parent2. Let's suppose that, a genotype with length L is given as depicted in **Fig. 8**, then L random numbers are obtained from a uniform probability distribution, while each value is between '0' and '1'. First, the P-values are calculated and, in addition, the operator checks each value and if it be less than the parameter p (usually 0.5), then the gene is picked from the parent1, otherwise, it is chosen from the parent2 [68-70].



P-Values = [0.36, 0.65, 0.24, 0.46, 0.89, 0.63, 0.12, 0.55]

Fig. 8. An example of uniform crossover with genotype (L= 8) and 8 values

2.3.4. Flat (BLX) or Discrete Crossover

Flat Crossover also applies the random numbers to reproduce one child from two parents.This operator functions the same as the uniform crossover, but the random numbers should be a subset of having the minimum and maximum of the genes. Generally, flat crossovers are employed in real-coded GAs [68-72].

$Parent1 = \left(x_{1.1},...,x_{1,n}\right)$

$Parent2 = \left(x_{2.1},...,x_{2,n}\right)$

and a vector of random values $r = \left(r_1,...,r_n\right)$

The $\text{Child1} = \left(x_1^1,...,x_n^1\right)$ can be calculated a vector of linear combinations by Eq.4.

$(for \ \ all, i = 1,...,n)$

$$x_i^1 = r_i x_{1,i} + (1 - r_i)x_i, i = 1,...,n \qquad (4)$$

and, so on.

For more information about the crossover operators, we suggest the readers to review Ref [72].

2.4. Mutation Operators

Mutation is the process of randomly changing a part of the genetic material (genotype) of one selected parent to produce a new genotype. In the GAs, the mutation occurs when the recombination of two parents is done and, then it alters with a small probability. The variation between the mutation and recombination is that the recombination applies two parents to reproduce a new child whereas the mutation only focuses on a parent and alters its genotype to form the new child. Various mutation operators are employed in the GPs recently that several will be described below [68], [69], [73], [74].

2.4.1. Bitwise or Binary Representations

This type of mutation operators works based on flipping (0 to 1, or 1 to 0) a small part of genotype. For example, a binary representation is given in **Fig. 9**, a sequence of bits (0's and 1's), with length L, and a probability P_m, afterward, the operator considers each gene separately and flips each bit if the generated P-value is less than the P_m value. Therefore, on average, the number of mutations for a genotype with length is equal to $L \times P_m$.

Fig. 9. an example of bitwise mutation, bits '3' and '8' are mutated in the new child [68], [75].

2.4.2. Integer Representations

Creep mutation and random reversing are two types of mutation operators which are applied when the encoding procudure utilizes an integer representation. A probability P_m is employed to determine how many mutations can be occurred and it is applied to a specific

gene. In the random reversing, each gene is permitted to be modified from a list of feasible values relying upon the probability P_m. Commonly, this operator is chosen where the list of the encoded values are original values. In other words, the creep mutation is applied for basic characteristics and functions based on changing a small value on each genotype by probability p (i.e., the value could be either negative or positive), that more information about these types of mutation operators could be found in Ref [75], [76].

2.4.3. Permutation Representations

In permutation representations, if a specific gene is mutated autonomously, then it might lead to duplication of genotype problems. For instance, a city tour with a genotype of {5,2,3,1,4} is given, and the mutation operator alters the third gene to 5, then the result will be {5,2,5,1,4}; therefore, there is no city tour '3' in the genotype. The result of permutation confirms that it never gets to visit city '3' while visits city '5' twice. During the permutation of a genotype, the main point is, the operator should keep the same values and does not present, delete or replicate any specific genotype. There are different mutation operators that function based on permutation representations which are discussed in the following points [68], [76].

- Swap Mutation: it operates by randomly choosing two genes in the genotype and exchanges the selected genes of the parent. **Fig. 10** depicts an example of the swap mutation operator [68], [76].

Fig. 10. Swap mutation, it swapped '1' and '7'.

- Insert Mutation: it functions by randomly choosing two genes in the genotype and shifts other genes to the next position index plus for the other genes [68], [76].

Fig. 11. an example of insert mutation.

As depicted in **Fig. 11**, the genes '2' and '6' get selected, as well as, '6' is hosted in the next of '2' and so on (e.g., for 3, 4, 5).

- Scramble Mutation: this operator acts by picking a part of the genotype and randomly scrambles the selected genes [68], [76].

Fig. 12. an example of scramble mutation

As shown in **Fig. 12**, the selected part of the parent is '4' to '7', afterward, it scrambles the genes to create the new genotype.

- Inverse Mutation: it also works by randomly picking a part of the genotype so that reverses the order of genes. [68], [76].

Fig. 13. an example of inverse mutation

As depicted in **Fig. 13**, the selected part of the parent is '3' to '6', then, the order is reversed.

3. Various Types of Genetic Programming

During the past three decades, there have been done many types of research to progress the GPs in different applications, that can be classified in eight major types: including Tree-based GP, Stack-based GP, Linear GP, Extended Compact GP, Grammatical Evolution GP, as following types. In practice, almost all the various types of GPs have the same structure as depicted in **Fig. 1**, and different operators (e.g., selection, crossover and mutation).

3.1. Tree-based Genetic Programming (TGP)

As we have already explained above, the tree-based GP was the first type in that the programs are represented in tree structures which are evaluated recursively to generate the resulting multivariate expressions. In the tree-based GP, the basic nomenclature determines that a tree node (or node) is an operator (e.g., *, /, +, -, etc.) and a terminal node (or leaf) is a variable (e.g., a, b, c, d, etc.), [1-5], [77]. Lisp was the first programming language applied to tree-based GP due to having the same structure and similarities with the trees. However, many other languages such as C++, Java, and Python have been utilized to advance the tree-based GP applications [78]. An example of tree-based GP designed for simulating the evolutionary processes in the biological world with two types of species. This program is written in Java language which can be found on GitHub ref [79]. **Table 2** depicts some applications of the tree-based GP.

Table. 2. Some existing applications of the Tree-based GPs

Refrences	Scientific Area	Goal of Application
[1-5], [11]	➢ Biological and Genomic	DNA Expression, SNP analysis, Epistasis analysis, Cancer gene expression, Gene annotation, and Molecular structure optimization, etc.
[7], [103]	➢ Scientific, Statistical and Numerical Computing	Quantom Computing, Solving Complex Optimization Problems, search problems, etc.
[107]	➢ Mobile Communication Infrastructures	QoS routing, Communication Scaduling, etc.
[108], [109]	➢ Transportation Technology	Non-linear transportation, Transportation planning, costs, Multi-stage supply chain networks, etc.
[12]-[104-106]	➢ Physical of Materials	Solid state physics: electronic and other properties, Designe, Optical properties, and Spectroscopy (*), etc.
[6], [110]	➢ Image Processing	Building Blocks, Face Recognition, and Pattern Recognition, Classification, etc.

3.2. Stack-based Genetic Programming (SGP)

In this type of GPs, the programs execute on a stack-based virtual machine. In other words, the programs in the evolving population are represented in a stack-based programming language. Commonly, the specific languages differ between systems, but most are similar to FORTH insofar as programs are formed by instructions that obtain arguments from the data stacks and push results back on those data stacks again. In the Push family of languages, which were created specifically for the GP, a separate stack is presented for each data type, and, in addition, the program's code can manipulate itself on the data stacks and consequently performed. Depending on the genetic operators used and the specific language, a stack-based GP can provide a variety of advantages over tree-based GP. These may consist bloat-free crossover and mutation operators, improvements or simplifications to the handling of the multiple data types, execution tracing, programs with loops that produce accurate outputs even when terminated prematurely, parallelism, the evolution of arbitrary control structures, and automatic simplification of evolved programs [78], [80]. **Table 3** depicts some applications of the SGP. A Python-based environment and stack-based language for genetic programming can be found in Ref [93].

Table. 3. Some applications of the Stack-based GP

Refrences	Scientific Area	Goal of Application
[80], [114]	➢ Automated design and Program Synthesis-analysis	Benchmark problems, Automatic Programming, etc.
[94], [95], [113]	➢ High Performance Computing	Parallel Computing, Vector Processing, GPU processing, etc.
[111], [112]	➢ Electronic Circuit Design	Micro architectural and instruction design, Memory Management overhead, etc.

3.3. Linear Genetic Programming (LGP)

Linear GP is a variant of the GPs wherein the programs in a population are expressed as a series of instructions from powerful programming language or machine code. The graph-structured data flow which occurs from several usages of register contents and the presence of structurally non-effective code (introns) are two main variations of the linear GP from the more common TGP. In the LGP, a linear tree is a program which consists of a variable number of unary operations and a single terminal. In addition, the linear GP varies from the binary string GAs since a population may include programs with different lengths and there may be more than two types of operations or more than two types of terminals. Basically, the LGP programs are expressed by a linear order of instructions, and they are simpler to read and operate on than their tree-based counterparts [81]. **Table 4** lists some applications of the LGP.

Table. 4. Some applications of the LGP

Refrences	Scientific Area	Goal of Application
[81]	➢ Data Mining and Knowledge Discovery	Time Prediction, and Classification Problems, etc.
[81]	➢ Signal Processing and Image Processing	Time series prediction, Control problems, etc.
[115]	➢ Hydrological phenomena.	Prediction model for the river, Streamflow prediction, etc.

An example of the LGP is written in Java for solving regression problems, that can be found in Ref [82].

3.4. Grammatical Evolution Genetic Programming (GEGP)

Grammatical Evolution (GE) works based on the grammar strucure which joins principles from molecular biology to the symbolic power of formal grammars. GE's rich modularity provides specific adaptability, making it possible to apply alternative search procedures, whether deterministic, evolutionary or some other methods. Moreover, it radically manipulates its behavior by only altering the grammar supplied. As grammar is employed for expressing the structures which are produced by the GE, it is trivial to change the output structures by only adopting the plain text grammar. This feature is one of the primary merits which makes the GE method so appealing. The genotype or phenotype (e.g., is a part of genotype) mapping indicates that in lieu of operating particularly on solution trees, as in the standard GP, the GE permits search operators to be executed on the genotype (e.g., binary or integer genes), moreover, partially resulting phenotypes, and the wholly formed phenotypic derivation trees themselves. One of the advantages of GE is that this mapping explains the use of search to various programming languages and other structures [83], [100], [116], [117]. **Table 5** summarizes some applications of the GEGP.

Table. 5. Some applications of the GEGP

Refrences	Scientific Area	Goal of Application
[116]	➢ Automated Programming	Automatic generation of benchmarks for plagiarism detection tools
[117]	➢ Biological and Genomic	Petri net modeling of high-order genetic systems using grammatical evolution.
[100], [121]	➢ Financial Modeling	Predicting corporate bankruptcy , bond credit ratings, Forecasting stock indices, etc.

An example of GEGP is written in Java that can be found in Ref [84].

3.5. Extended Compact Genetic Programming (ECGP)

Extended Compact GP (ECGP) works based on a key idea which the selection of a proper probability distribution is equal to linkage learning. The quantity of a reasonable distribution is measured based on minimum description length (MDL) models. The key idea of MDL models is that given all things are equivalent, simpler distributions are greater than the complex ones. The limitation of MDL assesses both inaccurate and complex models, whereby leading to an optimal probability distribution. Therefore, the restriction of MDL reformulates the problem

of obtaining a proper distribution as an optimization problem which reduces both the probability model and the population representation [85], [86]. In practice, the ECGA could solve complex problems in the binary domain. Moreover, it is accurate and reliable, due to having the ability of identifying building blocks, but several difficulties are experienced when we directly employ the ECGA to problems in the integer domain [87]. **Table 6** summarizes some applications of the ECGP.

Table 6. Some applications of the ECGP

Refrences	Scientific Area	Goal of Application
[96]	➢ Transmission Power Systems	solving optimum allocation of power quality monitors
[97]	➢ System on Chip in the Nanoscale Technologies	efficient routing algorithm for Network-on-Chip
[98]	➢ Performance and Memory Space Optimization	Memory saving optimization with limitted hardware

An example of ECGP is written in MATLAB which can be found in Ref [88].

3.6. Cartesian Genetic Programming (CGP)

Cartesian is a highly effective and flexible form of GP which encodes a graph illustration of a computer program. The CGP assigns computational structures (e.g., computer programs, mathematical equations, circuits, etc.) as a string of integers. The assigned integers, known as genes specify the operations of nodes in the graph, the links between nodes, the links to inputs and places in the graph where nodes obtain their input. Practically, employing a graph representation is very flexible as many computational structures could be expressed as graphs. A excellent example of this is artificial neural networks (ANNs) that could be easily encoded in CGP. Generally, the CGP obtains proper solutions very efficiently in a few evaluations. However, it employs many generations and utilizes extremely small populations (e.g., typically 5), where it is the best one from the previous generation [89]. Embedded CGP (E-CGP) is an extension of the directed graph based CGP, that is able of automatically obtaining, expanding and re-using partial solutions in the form of modules. The E-CGP results have shown that it is more computationally effective than the CGP on developing solutions to a range of problems [118].

Table 7. Some applications of the CGP & E-CGP

Refrences	Scientific Area	Goal of Application
[118]	➢ High Performance Computing	Lawnmower and Hierarchical-if-and-only-if (H-IFF) Problems
[119], [125]	➢ Digital Circuits	Improving the evolvability of digital multipliers, Optimization of combinational Circuits, etc.
[89]	➢ Control Engineering	Automatic visual defect detection, Process Control, etc.

Table 7 summarizes some applications of the CGP & E-CGP. An example of CGP is written in Java which can be found in Ref [90].

3.7. Probabilistic Incremental Program Evolution (PIPE) GP

Probabilistic incremental program evolution (PIPE) is an efficient type of automatic programming. The PIPE combines probability vector coding of program instructions, population-based incremental learning, and tree-coded programs to provide practical solutions, i.e., similar those applied in some variants of GP. Moreover, it iteratively produces progressive populations of operative programs based on an adaptive probability distribution over all feasible programs such that each iteration employs the best one to improve the distribution. Therefore, PIPE stochastically creates better and better programs. Since the distribution improvements rely only upon the best solution of the current population, PIPE could assess program populations efficiently when the aim is to find a program by the minimum runtime [91]. **Table 8** summarizes some applications of the PIPE.

Table. 8. Some applications of the PIPE

Refrences	Scientific Area	Goal of Application
[99]	➢ Machine learning	Learning speedup by evaluating programes on parallel (if the idea is to discover programs by minimal runtime)
[99]	➢ Multigene Tasks	Automatoc Task Decomposition, solving tasks with high algorithmic complexity
[99]	➢ Long or Shorter Time Lag Challenge	finding solutions by classifying all the sequences of the training data

An example of the PIPE is written in Ruby which can be found in Ref [92].

3.8. Strongly-Typed Genetic Programming (STGP)

Basically, the standard form of GP has the limitation, is identified as “closure,” i.e. that all variables, arguments, constants for terminals, and values returned from terminals must be of the same data types. In this case, while the programs manage several data types and include terminals devised to work on specific data types, it could propel to redundant large search times or unnecessarily poor generalization efficiency [20], [100]. To address this deficiency, Montana was proposed an improved version of GP called “Strongly typed genetic programming (STGP)” which applies data type constraints and whose use of the generic terminals. In the STGP, every terminal has a type, and each function has types for each of its arguments and a type for its return value [4]. Moreover, it makes the STGP more potent than other techniques to type constraint enforcement. Therefore, it is able to solve a wide variety of moderately difficult problems concerning several data types [29], [101]. **Table 9** summarizes some applications of the STGP. An example of STGP is written in Python which can be found in Ref [102].

Table 9. Some applications of the STGP

Refrences	Scientific Area	Goal of Application
[20], [29], [123]	➢ Knowledge discovery	Huge data classification, and knowledge extraction, Game playing, etc.
[20], [29]	➢ Data Mining	Extracting hidden patterns, Classification rules, etc.,
[20], [29]	➢ Data Center/Server farm	Data Integration, Data Cleaning, Data Transformation, etc.

3.9. Advantages and Disadvantages GPs

In this subsection, we summarize some advantages and disadvantages for various types of GPs with respect to the evaluated Algorithmic Complexity (AC) on complex problems in the existing literature. Since the genetic operators for different types of GPs are different from each other, we have no common criteria to compare the performance of them together. Moreover, we have to mention that all the listed advantages and disadvantages discovered from the mentioned references as depicted in **Table 10**.

4. Suggestions for the Future Works

The GP is a very powerful and flexible programming technique that could be employed in various ways to solve complex problems in different scientific areas such as computer science, biology, and chemistry, transportation engineering, financial engineering, etc. In this section, we suggest some directions aimed at guiding researchers on the best options to employ various types of GPs depending on the characteristics of the applications. However, we have to notice that these guidelines are general and empirically obtained rules of thumb; these suggestions must not be considered rigidly or dogmatically.

- With regard to the implementation of various types of GPs, we have summarized some applications and open-source examples for each type of GP separately. It provides a useful set of information about various types of GPs that the beginners can easily find them and may apply the source codes for further works.
- One of the most important decisions to be taken when considering the application of GP to a specific area is the related characteristics to be considered. For example, if the researchers aim to use a type of GP for vector processing with low complexity, then, the best option is to employ the Stack-based GP by using a related crossover and mutation operators.

Table 10. advantages and disadvantages of various types of GPs

Type of GP	Advantages	Disadvantages
Tree-based (TGP) [77], [100], [103]	➢ Higher-order functions are a powerful addition to the TGP which enables the evolution of programs with greater than constant-time complexity	➢ Closure (having the same data type between operators and terminals), which causes to increase the AC in the multiple data type problems ➢ High AC in the Lawnmower and H-IFF problems
Stack-based (SGP) [80], [94], [95], [113], [114], [120]	➢ High performance on symbolic regression problem ➢ Low AC (outperforms the TGP) ➢ Efficient performance in parallel computing	➢ Inefficient performance where long programs (variables) are pushed in the stack on limited resources systems ➢ It can only be implemented on stack support languages.
Linear (LGP) [81]	➢ High flexibility (e.g., allows more freedom on the internal representation) ➢ Low AC ➢ Allowing a more efficient evaluation of programs	➢ Higher compiler overhead than the TGP
Grammatical Evolution (GEGP) [78], [83], [100], [121], [124]	➢ The flexibility of language choice that it allows (e.g. the user could output algorithm in any language and utilize a compiler for that language to write an executable code to calculate a fitness). ➢ Low AC, delineating the search space	➢ High AC in the Travelling Salesman problem

	obviously and avoiding unproductive search in infeasible regions, ➢ Low AC and high predictive accuracy in financial problems	
Extended Compact (ECGP) [96], [97], [98], [122]	➢ Low AC and better solutions for the economic dispatch problem (outperforms the TGP) ➢ Creating offspring (child) without disrupting linkage groups of decision variables.	➢ ECGP can handle only binary variables.
Cartesian (CGP) Embedded E-CGP [89], [118], [119], [125]	➢ CGP has low AC (better runtime) in the Lawnmower and H-IFF problems (outperforms the TGP and SGP) ➢ E-CGP is more efficient than CGP in the difficult problems	➢ CGP and its derivatives suffer from over-fitting on the training data when applied to series forecasting
PIPE [99]	➢ By several time steps between a relevant input and the corresponding output, it could outperform even the best neural network algorithms (Long Short-Term Memory) ➢ It can solve difficult tasks (low AC) in an acceptable time ➢ Low AC in the long time lag task	➢ High AC in the shorter time lag task
Strongly Typed (STGP) [4], [20], [29], [123], [126]	➢ Low AC in Multi data type problems ➢ Low AC in Game playing problem ➢ Producing more accurate solutions in classification problems	➢ High AC or more training time in Classification problems

Note Low AC: "low runtime or better efficiency," High AC: "high runtime or low efficiency"

- In some occasions, efficiency and accuracy are the two most significant factors that specify the effectiveness of the GP. For example, in financial prediction domains, a slight increase in predictive accuracy can indicate a higher income percentage. The use of an automatic programming such as GEGP can provide more predictive accuracy and appropriate expression in the financial domain. The GEGPs are often utilized also in applications related to estimating forecasting stock indices, bond credit ratings, corporate bankruptcy, etc. The reason is that the expression given by the GEGPs is very akin to the kind of mathematical operations and financial predictions usually used in the financial systems.
- One of the main disadvantages of the GP is its high training time, that embitters when combined with the need for dealing with the huge dataset often found in classification problems. It is necessary to investigate into the potentialities available to perform the GP training as efficient as possible, like distributed and parallel GP or a combination of two types of GPs.
- To sum up, which direction is suitable for applying the GP? We cannot present an accurate or perfect answer to this question. The researchers should take into account many considerations like various advantages and disadvantages of GPs, together with the guidelines that we have collected. Also, they should consider whether the GP approaches could be appropriate or not for the problem at hand. When the researcher figures out that some of the merits of GP can give a valuable benefit or fits naturally to the specific needs and attributes of the problem at issue; therefore, a proper type of GP should probably be given a try.

5. Conclusions

This survey provides a comprehensive review of various aspects of GP. First of all, we have overviewed a standard framework of GP including, key steps, selection strategies, crossover and mutation operators. Secondly, we have categorized various type of GP techniques, and their applications among some example source codes and, moreover, we summarized some advantages and disadvantages of various types of GPs. Finally, we suggested some of the guidelines and directions that could merit further attention in future works. GP is still an efficient evolutionary algorithm can be desirable to obtain the best solution from the problems of the emerging field of next-generation sequencing. It is obvious that GP is still a growing field of research, whose practitioners are still investigating its potentialities and limitations.

References

[1] W. B. Langdon, A. Qureshi, "Genetic Programming- Computers Using "Natural Selection" to Generate Programs," *Genetic Programming and Data Structures*, *The Springer International Series in Engineering and Computer Science*, Springer, Boston, 1998. Article (CrossRef Link).

[2] J. H. Holland, "Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology," *Control and Artificial Intelligence*, MIT Press Cambridge, MA, USA, ISBN:0262082136, 1992.

[3] Uiterwijk, J. W. H. M., van den Herik, H. J., and Allis, L. V., "A knowledge-based approach to connect-four," In David Levy and Don Beals, editors, Heuristic Programming in Artificial Intelligence: The First Computer Olympiad, Ellis Harwood; John Wiley, 1989.

[4] R. Poli, W. B. Longdon, N. F. McPhee, and J. R. Koza, "A Field Guide to Genetic Programming," *Evolutionary Computation*, published by Springer, ISBN:1409200736 9781409200734, 2008.

[5] Z. Zhi-hui, L. Xiao_Feng, G. Yue-Jiao, and Z. Jun, "Cloud Computing Resource Scheduling and a Survey of Its Evolutionary Approaches," *ACM Computing Surveys*, vol.47, no.4, pp. 1- 33, Article 63, 2015. Article (CrossRef Link).

[6] F. Dai, Y. Fujihara, and N. Kushida, A Survey of Face Recognition by Genetic Programming, Nova Sciecne Publisher Inc., New York, 2011. Article (CrossRef Link).

[7] I. Boussaïd, J. Lepagnot, and P. Siarry, "A survey on optimization metaheuristics," *Information Sciences*, Vol. 327, pp. 82-117, July. 2013. Article (CrossRef Link).

[8] Knysh, D. S., & Kureichik, V. M., "Parallel Genetic Algorithms: A Survey and Problem State of the Art," *Journal of Computer and Systems Sciences International*, 49(4), 579–589, 2010. Article (CrossRef Link).

[9] J. R. Koza, "Survey of genetic algorithms and genetic programming," in *Proc. of WESCON/'95 Conference record IEEE*, 1995. Article (CrossRef Link).

[10] M. Srinivas and L. M. Patnaik, "Genetic algorithms: a survey," *IEEE Journal & Magazines Computer*, Vol. 27, pp. 17-26, 1994. Article (CrossRef Link).

[11] M.W. Khan, M. Alam, "A survey of application: Genomics and genetic programming," *a new frontier, Genomics*, Vol. 100, pp. 65–71, 2012. Article (CrossRef Link).

[12] W. Paszkowicz, "Genetic Algorithms, a Nature-Inspired Tool: Survey of Applications in Materials Science and Related Fields," *Materials and Manufacturing Processes*, Vol. 24, pp. 174-197, 2013. Article (CrossRef Link).

[13] T. Li, G. Shao, W. Zuo, and S. Huang, "Genetic Algorithm for Building Optimization - State-of-the-Art Survey," in *Proc. of ICMLC 2017 Proceedings of the 9th International Conference on Machine Learning and Computing*, 2017. Article (CrossRef Link).

[14] M. Paulinas, and A. Ušinskas, "A Survey of Genetic Algorithms Applications for Image Enhancement and Segmentation," *Information Technology and Control*, vol. 36, pp. 278-284, 2007.

[15] H. Zhao, "a multi-objective genetic programming approach to developing Pareto optimal decision trees," *Decision Support Systems*, vol. 43, pp. 809-826, 2007. Article (CrossRef Link).

[16] B. Can, and C. Heavey, "A comparison of genetic programming and artificial neural networks in meta modeling of discrete-event simulation models," *Computers &Operations Research*, vol. 39, pp.424–436, 2012. Article (CrossRef Link).
[17] E. ALBA, J. M. TROYA, "A Survey of Parallel Distributed Genetic Algorithms," *Journal Complexity*, vol. 4, pp. 31–52, 1999.
[18] E. Cantú-Paz, A Survey of Parallel Genetic Algorithms, University of Illinois at Urbana-Champaign, USA, 1998.
[19] L. Vanneschi, M. Castelli, S. Silva, "A survey of semantic methods in genetic programming," *Genetic Programming and Evolvable Machines*, vol. 15, pp. 195–214, 2014. Article (CrossRef Link).
[20] W. B. Langdon et al., "Genetic Programming: An Introduction and Tutorial, with a Survey of Techniques and Applications," *Studies in Computational Intelligence*, vol. 115, pp. 927–1028, 2008. Article (CrossRef Link).
[21] G. Folino, C. Pizzuti, and G. Spezzano "A Scalable Cellular Implementation of Parallel Genetic Programming," *IEEE Transactions on Evolutionary Computation*, 7, pp. vol. 37-53, 2003.
[22] P. G. Espejo, S. Ventura, and F. A. Herrera, "Survey on the Application of Genetic Programming to Classification," *IEEE Transactions on Systems, Man, And Cybernetics-Part C: APPLICATIONS AND REVIEWS*, vol. 40, pp. 121-144, 2010. Article (CrossRef Link).
[23] A. N. Shakarneh, "A Review of Genetic Algorithm Optimization: Operations and Applications to Water Pipeline Systems," *International Journal of Mathematical and Computational Sciences*, vol. 7, pp. 1782-1788, 2013. Article (CrossRef Link).
[24] M. Harman, W. B. Langdon, and W. Weimer, "Genetic Programming for Reverse Engineering," in *Proc. of WCRE (WCRE), 20th Working Conference on Reverse Engineering*, pp. 1782-1788, 2013. Article (CrossRef Link).
[25] L. Lijia, and X. Yu, "A new two-stage genetic programming classification algorithm and its applications," *Transactions of the Institute of Measurement and Control*, pp. 1-19, 2017. Article (CrossRef Link).
[26] A. K. Swian and A. M. S. Zalzala, "An Overview of Genetic Programming: Current Trends and Applications," *SCSE Research Report No*.732, *Department of Automatic Control and systems*, The University of Sheffield, 1998.
[27] M. A. Iqubal,"Genetic Algorithms and Their Applications: and Overview," *White Paper*, I.A.S.R.I., Library Avenue, New Delhi-110012.
[28] C. Qing-Shan et al., "A modified genetic programming for behavior scoring problem," *IEEE Symposium on Computational Intelligence and Data Mining*, pp. 535–539, 2007. Article (CrossRef Link).
[29] S. Sakprasat, and M.C. Sinclair, "Classification rule mining for automatic credit approval using genetic programming," *IEEE Congress on Evolutionary Computation*, pp. 548–555, 2007. Article (CrossRef Link).
[30] A. L. Garcia-Almanza and E. P. K. Tsang, "Evolving decision rules to predict investment opportunities," *International Journal of Automation and Computing*, vol. 5, pp. 22–31. 2008. Article (CrossRef Link).
[31] N. M. Razali, and J. Geraghty, "Genetic algorithm performance with different selection strategies in solving TSP," *Proceedings of the World Congress on Engineering*, London, UK, 2011.
[32] M. C. Vargas et al., "Analysis of X-ray diffraction data using a hybrid stochastic optimization method," *Journal of Physics A: Mathematical and General*, vol. 35, pp.3865–3876, 2002. Article (CrossRef Link).
[33] Banzhaf, W.; Lasarczyk, C., "Genetic programming of an algorithmic chemistry," *Genetic Programming Theory and Practice II*, pp.175-190, 2005. Article (CrossRef Link).
[34] B. Can, and C. Heavey, "Comparison of experimental designs for simulation-based symbolic regression of manufacturing systems," *Computers & Industrial Engineering*, vol. 61, pp. 447-462, 2013. Article (CrossRef Link).
[35] K. Y. Chan, C. K. Kwong, and T. C. Fogarty, "Modeling manufacturing processes using a genetic programming-based fuzzy regression with detection of outliers," *Information Science*, vol. 180,

pp.506–518, 2010. Article (CrossRef Link).
[36] T. Blickle and L. A. Thiele, "A comparison of selection schemes used in evolutionary algorithms," *Evolutionary Computation*, vol. 4, pp. 361–94, 1996. Article (CrossRef Link).
[37] O. Giustolisi et al., "An evolutionary multi objective strategy for the effective management of groundwater resources," *Water Resources Research*, vol.44, no.1, 2008. Article (CrossRef Link).
[38] F. E. B Otero, et al., "Genetic programming for attribute construction in data mining," in *Proc. of Lecture Notes in Computer Science, Genetic Programming, 6th European Conference, EuroGP 2003*, Essex, UK, April 14-16, 2003. Proceedings, 2003, Article (CrossRef Link).
[39] C. S. Greene, and J. H. Moore, "Solving complex problems in human genetics using GP: challenges and opportunities," *ACM SIGEVOlution*, vol.3, pp. 2-8, 2008. Article (CrossRef Link).
[40] R. Poli,and N. F. McPhee., "Parsimony Pressure Made Easy: Solving the Problem of Bloat in GP," *Theory and Principled Methods for the Design of Metaheuristics*, 2014. Article (CrossRef Link).
[41] D. E. Goldberg, and K. Deb, "comparative analysis of selection schemes used in genetic algorithms," in: G.J.E. Rawlins (Ed.), *Foundations of Genetic Algorithms*, Morgan Kaufmann, Los Altos, pp. 69–93, 1991. Article (CrossRef Link).
[42] D. B. Fogel, Handbook of Evolutionary Computation, *IOP Publishing Ltd. and Oxford University Press*, 1997.
[43] Blickle, T., Thiele, L., "A Comparison of Selection Schemes used in Genetic Algorithms," *TIK-Report*, Zurich, 1995.
[44] J. E. Baker, "Adaptive selection methods for genetic algorithm," in *Proc. of an International Conference on Genetic Algorithms and Their Applications*, pp. 100-111, 1995.
[45] A. Shukla, H. M. Pandey and D. Mehrotra, "Comparative Review of Selection Techniques in Genetic Algorithm," *International Conference on Futuristic trend in Computational Analysis and Knowledge Management*, pp. 515-519, 2015. Article (CrossRef Link).
[46] H. Xie, "An Analysis of Selection in Genetic Programming," *PhD thesis, the Victoria University of Wellington*, 2008.
[47] J. Zhong, et al., "Comparison of performance between different selection strategies on simple genetic algorithms," in *Proc. of Computational Intelligence for Modelling, Control and Automation, International Conference on Intelligent Agents, Web Technologies and Internet Commerce*, Vol. 2, 2005. Article (CrossRef Link).
[48] The general framework of Genetic Algorithm, Tournament selection pseudo code, last seen [11/08/2017], available: https://cstheory.stackexchange.com/questions/14758/tournament-selection-in-genetic-algorithms
[49] Algorithmic Trading program, that uses Genetic Programming and Genetic Algorithms to predict stock prices., https://github.com/giladbi/algorithmic-trading
[50] Roulette wheel selection algorithm, last seen, [11/08/2017], Available: https://www.mathworks.com/matlabcentral/answers/69881-roulette-algorithm-probability-loop?requestedDomain=www.mathworks.com
[51] Tournament selection pseudo code, last seen [11/08/2017], https://stackoverflow.com/questions/31933784/tournament-selection-in-genetic-algorithm
[52] Ranked based selection pseudo code, last seen [11/08/2017], https://stackoverflow.com/questions/13659815/ranking-selection-in-genetic-algorithm-code
[53] Bulmer, M.G., The Mathematical Theory of Quantitative Genetics, Clarendon press, Oxford, 1980.
[54] J.F. Crow, and M. Kimura, "An Introduction to Population Genetics Theory," *Harper and Raw*, New York, 1980.
[55] H. Muhlenbein, and D. Schlierkamp-Voosen, "Predictive models for the breeder genetic algorithm," *Evolutionary Computation*, 1993. Article (CrossRef Link).
[56] J. R. KOZA,"Genetic Programming- On the Programming of Computers by Means of Natural Selection," *MIT Press, Cambridge*, 1992. Article (CrossRef Link).
[57] J. R. KOZA, "A response to the ML-95 paper entitled Hill climbing beats genetic search on a Boolean circuit synthesis of Koza's," *International Machine Learning Conference in Tahoe City, California*, USA, 1995. Article (CrossRef Link).

[58] J. R. KOZA et al., "Genetic Programming III: Darwinian Invention and Problem Solving," 1st ed. Morgan Kaufmann, 1999.
[59] J. R. KOZA et al., "Genetic programming IV: Routine Human-Competitive Machine Intelligence," *Kluwer Academic Publishers Norwell*, MA, USA, 2003.
[60] W. B. LANGDON, "Size Fair and Homologous Tree Crossovers for Tree Genetic Programming," in *Proc. of the Genetic and Evolutionary Computation Conference*, vol. 2, pp. 95–129, 2000. Article (CrossRef Link).
[61] D. WHITELY, "The genitor algorithm and selection pressure: Why rank based allocation of reproductive trials is best," in *Proc. of the 3rd International Conference on Genetic Algorithms*, J. D. Schaffer, Ed., Morgan Kaufmann Publishers, pp.116–121, 1989.
[62] F. G. Lobo, D. E. Goldberg, and M. Pelikan, "Time complexity of genetic algorithms on exponentially scaled problems," in *Proc. of GECCO'00 Proceedings of the 2nd Annual Conference on Genetic and Evolutionary Computation*, pp.151-158, 2000.
[63] K. A. Jong, "An analysis of the behavior of a class of genetic adaptive systems," (Doctoral dissertation, University of Michigan). Dissertation Abstracts International, (University Microfilms No. 76-9381), 1975.
[64] L. B. Booker, "Intelligent behavior as an adaptation to the task environment," (Doctoral dissertation, Technical Report No. 243, Ann Arbor: University of Michigan, Logic of Computers Group). Dissertation Abstracts International, 43(2), 469B. (University Microfilms No. 8214966), 1982.
[65] A. Brindle, Genetic algorithms for function optimization (Doctoral dissertation and Technical Report TR81-2). Edmonton: University of Alberta, Department of Computer Science, 1981.
[66] J. E. Baker, "Reducing bias and inefficiency in the selection algorithm," Proceedings of the Second International Conference on Genetic Algorithms, pp.14-21, 1987.
[67] J. J. Grefenstette, and J. E. Baker, "How genetic algorithms work: A critical look at implicit parallelism," in *Proc. of the Third International Conference on Genetic Algorithms*, pp.20-27, 1989. Article (CrossRef Link).
[68] R. Raghavjee, and N. Pillay, "Using Genetic Algorithms to the South African School Timetabling Problems," *Second World Congress on Nature and Biologically Inspired Computing (NaBIC)*, 2010. Article (CrossRef Link).
[69] S. G. V. Kumar, and R. Panneerselvam, "A Study of Crossover Operators for Genetic Algorithms to Solve VRP and its Variants and New Sinusoidal Motion Crossover Operator," *International Journal of Computational Intelligence Research*, vol.13, no.7, pp. 1717-1733, 2017. Article (CrossRef Link).
[70] A. J. Umbarkar, and P. D. Sheth, "CROSSOVER OPERATORS IN GENETIC ALGORITHMS: A REVIEW, *ICTACT Journal on Soft Computing*, vol.6, pp.1083-1092, 2015. Article (CrossRef Link).
[71] S. M. Lim, et al., "Crossover and Mutation Operators of Genetic Algorithms," *International Journal of Machine Learning and Computing*, vol.7, pp.9-12, 2017. Article (CrossRef Link).
[72] J. MAGALHÃES-MENDES, "The role of genetic crossover operators in project scheduling under multiple renewable resources constraints," *Recent Advances in Applied and Theoretical Mathematics*, pp.216-221, 2014.
[73] C. Contreras-Bolton, and V. Parada, "Automatic Combination of Operators in a Genetic Algorithm to Solve the Traveling Salesman Problem," *PLoS ONE*, 10, 2015. Article (CrossRef Link).
[74] E. Osaba et al. "Automatic Combination of Operators in a Genetic Algorithm to Solve the Traveling Salesman Problem," *The Scientific World Journal, Article ID 154676*, 2014. Article (CrossRef Link).
[75] A. E. Eiben, and J. E. Smith, Introduction to Evolutionary Computing, Springer, New York, 2 editions, 2003. Article (CrossRef Link).
[76] N. Soni, and T. Kumar, "Study of Various Mutation Operators in Genetic Algorithms," *International Journal of Computer Science and Information Technologies*, vol. 5, pp.4519-4521. 2014.

[77] A. Agapitos et al., "Recursion in tree-based genetic programming," *Genetic Programming and Evolvable Machines*, vol.18, no.2, pp.149-183. 2017. Article (CrossRef Link).
[78] Tree based Genetic Programming, last seen [11/13/2017], http://geneticprogramming.com/about-gp/tree-based-gp/
[79] An example of Tree-based GP, last seen [11/13/2017], https://github.com/halucinka/Serengeti-World-Genetic-Programming
[80] T. Helmuth, and L. Spector, "General Program Synthesis Benchmark Suite," in *Proc. of GECCO '15: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pp. 1039-1046, 2015. Article (CrossRef Link).
[81] M. F. Brameier, and W. Banzhaf, "Linear Genetic Programming," *Genetic and Evolutionary Computation, Springer-Verlag* US, 2008. Article (CrossRef Link).
[82] A linear GP library is written in Java, last seen [11/13/2017], https://github.com/rishavray/LinearGP
[83] M. O'Neill, E. Hemberg and C. Gilligan, "GEVA - Grammatical Evolution in Java, Linear Genetic Programming," *Natural Computing Research & Applications Group, University College Dublin*, 2011. Article (CrossRef Link).
[84] GEVA is an implementation (source code) of Grammatical Evolution in Java developed at UCD's, last seen [11/14/2017], http://ncra.ucd.ie/GEVA.html
[85] G. Harik, "Linkage Learning via Probabilistic Modeling in the ECGA," IlliGAL Report No. 99010, University of Illinois at Urbana-Champaign, Urbana, IL, 1999. Article (CrossRef Link).
[86] K. Sastry and D. E. Goldberg, Probabilistic Model Building and Competent Genetic Programming. In: Riolo R., Worzel B. (eds) *Genetic Programming Theory and Practice. Genetic Programming Series*, 6, Springer, Boston, MA, 2003. Article (CrossRef Link).
[87] H. Ping-Chu, C. Ying-Ping, "iECGA, Integer Extended Compact Genetic Algorithm," NCLab Report No. NCL-TR-2006005, Natural Computing Laboratory (NCLab), Department of Computer Science, National Chiao Tung University, 2006.
[88] An example of ECGP (source code), last seen [11/15/2017], https://jp.mathworks.com/matlabcentral/fileexchange/32576-extended-compact-genetic-algorithm
[89] J. F. Miller, Cartesian Genetic Programming, Natural Computing Series, © Springer-Verlag Berlin Heidelberg, 2011. Article (CrossRef Link).
[90] A simple implementation of CGP (source code), last seen [11/15/2017], https://github.com/hopple/gp4j
[91] R. Sdustowicz,and J. Schmidhuber, "Probabilistic incremental program evolution," Evolutionary Computation, vol.5, pp.123-141, 2007. Article (CrossRef Link).
[92] An example of PIPE (source code), [11/15/2017], http://www.cleveralgorithms.com/nature-inspired/probabilistic/pbil.html
[93] A Python-based environment and stack-based GP (source code), last seen [11/15/2017], https://github.com/logicalzero/gplab
[94] K. Stoffel, and L. Spector, "High-Performance, Parallel, Stack-Based Genetic Programming. In Koza, John R., Goldberg, David E., Fogel, David B., and Riolo, Rick L., "Genetic Programming," in *Proc. of the First Annual Conference*, pp.224-229. Cambridge, MA: The MIT Press, 1996.
[95] D. M. Chitty, "Faster GPU-based genetic programming using a two-dimensional stack," *Soft Computing*, vol.21, pp.3859-3787, 2017. Article (CrossRef Link).
[96] H. M. G. C. Branco, et al, Extended compact genetic algorithm applied for optimum allocation of power quality monitors in transmission systems," *Power and Energy Society General Meeting*, IEEE, 2011. Article (CrossRef Link).
[97] Z. Emrani, and K. Mohammadi, "A technique for NoC routing based on extended compact genetic optimization algorithm," in *Proc. of 19th Iranian Conference on Power Electrical Engineering (ICEE*), 2011.
[98] G. Lacca, "Memory-Saving Optimization Algorithms for Systems with Limited Hardware," *N: ISBN:978-951-39-4538-1, Copyright ©, by University of Jyväskylä*, 2011.

[99] R. Salustowicz, "Probabilistic Incremental Program Evolution," *IDSIA: Istituto Dalle Molle di Studi sull'Intelligenza Articiale in Lugano*, Switzerland. PhD dissertation, 2003. Article (CrossRef Link).
[100] A. Brabazon, and M. O'Neill, "Diagnosing Corporate Stablity Using Grammatical Evolution," *International Journal of Applied Mathematics and Computer Science*, vol.14, pp.363–374, 2004. Article (CrossRef Link).
[101] D. J. Montana, "Strongly Typed Genetic Programming," BBN Technical Report #7866, Cambridge, 1996. Article (CrossRef Link).
[102] A strongly-typed genetic programming framework for Python, last seen [11/13/2017], https://github.com/hchasestevens/monkeys
[103] Leier A., Banzhaf W., "Evolving Hogg's Quantum Algorithm Using Linear-Tree GP," *Cantú-Paz E. et al. (eds) Genetic and Evolutionary Computation — GECCO 2003. GECCO 2003. Lecture Notes in Computer Science*, vol 2723. Springer, Berlin, Heidelberg, 2003. Article (CrossRef Link).
[104] Djuriši´c, A.B.; Bundaleski, N.K.; Li, E.H., "The design of reflective filters based on AlxGa1−xN multilayers," *Semiconductor Science and Technology*, 91–97, 2001. Article (CrossRef Link).
[105] Ahonen, H.; de Souza, Jr. P.A.; Garg, V.K., "A genetic algorithm for fitting Lorentzian line shapes in Mössbauer spectra," *Nuclear Instruments and Methods in Physics Research B*, 124, 633–638, 1997. Article (CrossRef Link).
[106] Thalken, J.; Haas, S.; Levi, A.F.J., "Synthesis for semiconductor device design," *Journal of Applied Physics*, 98, 044508-1-8, 2005. Article (CrossRef Link).
[107] Maniscalco, V., Greco Polito, S., "Binary and m-ary encoding in applications of tree-based genetic algorithms for QoS routing," *Intagliata, A. Soft Comput.*, 18: 1705, 2014. Article (CrossRef Link).
[108] Yao, MJ. & Hsu, HW., "A new spanning tree-based genetic algorithm for the design of multi-stage supply chain networks with nonlinear transportation costs," *Optimization Engineering*, 10: 219, 2009. Article (CrossRef Link).
[109] K. Antony Arokia Durai Raj, Chandrasekharan Rajendran, "A genetic algorithm for solving the fixed-charge transportation model: Two-stage problem," *Journal of Computers and Operations Research archive*, Vol.39 (9), pp.2016-2032, 2012. Article (CrossRef Link).
[110] Liu K., Tong M., Xie S., Zeng Z., "Fusing Decision Trees Based on Genetic Programming for Classification of Microarray Datasets," *In: Huang DS., Jo KH., Wang L. (eds) Intelligent Computing Methodologies. ICIC 2014. Lecture Notes in Computer Science*, vol. 8589, 2014. Article (CrossRef Link).
[111] Heywood M.I., Zincir-Heywood A.N., "Register Based Genetic Programming on FPGA Computing Platforms," In: Poli R., Banzhaf W., Langdon W.B., Miller J., Nordin P., Fogarty T.C. (eds) Genetic Programming. EuroGP 2000, *Lecture Notes in Computer Science*, vol. 1802. Springer, 2000. Article (CrossRef Link).
[112] Martin, Peter. "A hardware implementation of a genetic programming system using FPGAs and Handel-C," *Genetic Programming and Evolvable Machines*, Vol. 2(4), pp.317-343, 2004. Article (CrossRef Link).
[113] Chitty, Darren M. "Faster GPU-based genetic programming using a two-dimensional stack," *Soft Computing*, vol. 21 (14), pp.3859-3878, 2017. Article (CrossRef Link).
[114] Kim, Kyung Joong, "Automatic python programming using stack-based genetic programming," in *Proc. of the 14th annual conference companion on Genetic and evolutionary computation*, ACM, 2012. Article (CrossRef Link).
[115] Mehr, Ali Danandeh, Ercan Kahya, and Cahit Yerdelen, "Linear genetic programming application for successive-station monthly streamflow prediction," *Computers & Geosciences*, vol.70, pp. 63-72, 2014. Article (CrossRef Link).
[116] Cebrian M., Alfonseca M., Ortega A., "Automatic generation of benchmarks for plagiarism detection tools using grammatical evolution," in *Proc. of GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, Vol. 2, pp. 2253-2253, 2007. Article (CrossRef Link).

[117] Moore J.M., Hahn L.W., “Petri net modeling of high-order genetic systems using grammatical evolution,” *BioSystems*, vol.72, pp.177-186, 2003. Article (CrossRef Link).
[118] Walker, James Alfred, and Julian Francis Miller, "Embedded Cartesian genetic programming and the lawnmower and hierarchical-if-and-only-if problems," in *Proc. of the 8th annual conference on Genetic and evolutionary computation, ACM*, 2006. Article (CrossRef Link).
[119] Walker, James Alfred, and Julian Francis Miller, "Improving the evaluability of digital multipliers using embedded Cartesian genetic programming and product reduction," in *Proc. of International Conference on Evolvable Systems*, Springer, Berlin, Heidelberg, 2005. Article (CrossRef Link).
[120] T. Perkis, “Stack-Based Genetic Programming,” in *Proc. of IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on Evolutionary Computation*, 1994. Article (CrossRef Link).
[121] Mckay, Robert I., et al., "Grammar-based genetic programming: a survey," *Genetic Programming and Evolvable Machines*, 11.3-4, 365-396, 2010. Article (CrossRef Link).
[122] Chen, Ying-ping, and Chao-Hong Chen, "Enabling the extended compact genetic algorithm for real-parameter optimization by using adaptive discretization," *Evolutionary Computation*, vol.18, pp.199-228, 2010. Article (CrossRef Link).
[123] Jia, Baozhu, and Marc Ebner, "A strongly typed GP-based video game player," in *Proc. of Computational Intelligence and Games (CIG), 2015 IEEE Conference on. IEEE*, 2015. Article (CrossRef Link).
[124] N. Miguel, “Genetic Algorithms using Grammatical Evolution,” *Master Thesis*, University of Limerick, 2006. http://hdl.handle.net/10197/8262
[125] Turner, Andrew James, and Julian Francis Miller, "Recurrent Cartesian Genetic Programming of Artificial Neural Networks," *Genetic Programming and Evolvable Machines*, vol.18, pp.185-212, 2017. Article (CrossRef Link).
[126] Loveard, Thomas, and Victor Ciesielski. "Representing classification problems in genetic programming," *Evolutionary Computation, 2001. Proceedings of the 2001 Congress on*. Vol. 2. IEEE, 2001. Article (CrossRef Link).

Milad Taleby Ahvanooey received the BSc in Software Engineering from UAST, Semnan, Iran, in 2012, and his MSc in Computer Engineering from IAU Science & Research, Tehran, Iran, in 2014. He is currently pursuing towards Ph.D. in Computer Science at Nanjing University of Science and Technology, Nanjing, China. From Sep, 2014 to Jun, 2016, he was a lecturer with the School of Mathematics and Computer Science at Damghan University, Iran. His research interests include modern coding theory, text mining, text hiding, and genetic programming. He is also an external reviewer of various international journals including the IEEE Access, the Computers in Human Behavior, and the KSII Transactions on Internet and Information Systems.

Qianmu Li received the BSc and PhD degrees from Nanjing University of Science and Technology, China, in 2001 and 2005, respectively. He is a professor with the School of Computer Science and Engineering, Nanjing University of Science and Technology, China. His research interests include information security, computing system management, and data mining. He received the China Network and Information Security Outstanding Talent Award and multiple Education Ministry Science and Technology Awards. He is the author/co-author of more than 100 high indexed (SCIE/E-SCI/EI) Journal/Conference papers, and eight books.

Ming Wu received his B.E. degree from Nanjing University of Science and Technology, China, in 2014. Currently, he is a Ph.D. candidate in the School of Computer Science and Engineering at Nanjing University of Science and Technology, China. He is now working towards his researches at Florida International University, the USA as a visiting scholar supported by China Scholarship Council. His research interests are crowdsourcing in machine learning and data mining.

Shuo Wang received her BSc from Nanjing University of Science and Technology, China, in 2016. Currently, she is working towards her Ph.D. in the School of Computer Science and Engineering at Nanjing University of Science and Technology, China. Her research interests are data mining, text processing, graph theory and social network.