DOI: 10.24996/ijs.2022.63.3.27

ISSN: 0067-2904

The Genetic Algorithm: A study survey

Rewayda Razaq Abo-Alsabeh Abdellah Salhi**
[1]*Department of Mathematical Sciences, Faculty of Computer Science and Mathematics, University of Kufa, Iraq*
[2]*Department of Mathematical Sciences, University of Essex, UK,*

Received: 5/11/2020 Accepted: 11/6/2021

Abstract
Genetic Algorithm (GA) is a population-based approach for optimization. It belongs to metaheuristic procedures that use population characteristics to guide the search. It maintains and improves multiple solutions which may produce a high-quality solution to an optimization problem. This study presents a comprehensive survey of GAs. We review and discuss genetic algorithms for new researchers in the field. We illustrate components of GA and view the main results on time complexity.

Keywords: Complexity; Crossover; Evolutionary algorithm; Genetic algorithm; Mutation; Selection

الخوارزمية الجينية: دراسة استقصائية

رويدة رزاق أبو السبح*, عبد الله صالحي
*قسم الرياضيات، كلية علوم الحاسوب والرياضيات, جامعة الكوفة، العراق
قسم الرياضيات،كلية الرياضيات, جامعة إسكس، المملكة المتحدة

الخلاصه
الخوارزميات الجينية (GA) هي نهج قائم على السكان للتحسين. إنها تنتمي إلى طريقة نهج الادلة العليا الذي تستخدم خصائص السكان لتوجيه البحث. وهي تحافظ على حلول متعددة وتحسنها والتي قد تنتج حلاً عالي الجودة لمشكلة التحسين .تقدم هذه الدراسة مسح شامل للخوارزميات الجينية. حيث نقوم بمراجعة ومناقشة الخوارزميات الجينية للباحثين الجدد في هذا المجال. نوضح المكونات التي تبني GAs ونعرض النتائج الرئيسية لوقت التعقيد.

1 Introduction

Since the 1960's there have been numerous attentions in mimicking living beings to develop strong algorithms for intractable optimization problems. Such techniques are called evolutionary computation. The most important algorithms in this type comprise evolution strategies provided by Rechenberg [1], and by Schwefel [2]; genetic algorithms (GAs) introduced by Holland [3]; genetic programming developed by Koza [4]; and evolutionary programming introduced by Fogel et al. [5].

Classical search methods are often suitable to solve optimization problems with small spaces in which they search a space of potential solutions. As each one of these methods is designed to efficiently solve only a specific type of problems, a major challenge arises when one algorithm is performed to solve many different problems. Also, these methods usually

*Email: ruwaida.mohsin@uokufa.edu.iq

converge to a locally optimum solution because they do not have a global perspective. Another challenge is their lack to be efficiently performed in parallel computing [6]. For these reasons, artificial intelligence techniques should be used for large spaces. GAs are stochastic algorithms that relied on the idea of Darwinian of survival of the fittest and genetic inheritance. Genetic algorithms are well known types of evolutionary computation approach as a strong and largely usable optimization and stochastic search techniques [7].

Their ability to make a good balance between exploring and exploiting the search space place them in the category of general use search methods. They have been successfully performed to large different optimization problems, because of their global perspective, simplicity, and inherent parallel working. Such problems include game playing, scheduling, wire routing, adaptive modelling, optimal control problems, traveling salesman problems, database query optimization, and transportation problems. See ([8- 14]).

The genetic algorithm belongs to the group of probabilistic algorithms, it varies from random algorithms. It is regarded as more robust than directed search methods because it joins stochastic search and elements of directed search. Another interesting feature is that the genetic algorithm applies a multi-directional search by preserving a population of potential solutions. It builds and exchanges the information between these directions while other methods handle only a single point of the search space [14].

GA is a stochastic approximate approach that does not impose assumptions on the components of the problem in hand to work. While GA does not guarantee the optimality of the solutions it generates, it often produces near optimal solutions in a reasonable time when exact methods require prohibitive amounts of time. GA is population based, i.e. it maintains a population of solutions from generation to generation, [15]. Fitness is the key criterion for ranking individuals and creating future generations. Some individuals (parents) are transformed through genetic operations to create new individuals (children) that are called offspring by merging parts from two parents and performing a mutation, which forms children by making changes in a single parent. In the final generation, the optimal solution is the individual with the best fitness value. A brief outline of the GA is as follows.

Algorithm 1 Pseudo-code of Genetic Algorithm

```
1: Randomly generate a population of individuals;
2: Evaluate the fitness of all population members, if termination criteria is met go to (5);
3: Create a new population by performing some genetic operator and a reproduction strategy
   to the individuals of the current population;
4: Go to (2);
5: The individual with best fitness is the candidate for an optimum solution. Stop.
```

2 Search space

Search strategies have two main properties: exploring the search (solution) space and exploiting the best solution [20, 21]. GAs basically do a blind search while they randomly search the landscapes. The eligible area of the solution space can be reached by employing the selection operators which guide the search. To improve the performance of GAs, a good balance between exploitation and exploration of the search space should take place. This can be obtained by carefully testing all the components of the genetic algorithm. Also, the algorithm should have a heuristic to encourage the performance.

The search space is the space of all feasible solutions. Relying on the definition of the problem, each potential solution can be assigned by measuring its fitness, in which each point represents a single potential solution in the search space. The GA searches the search space for the best solution that is represented by a single point among a number of potential solutions. The challenges in this are the suitable initial point and reaching the local minima.

The process of searching the space includes the initialization of the population and then creating new solutions until the stopping criteria is met. One purpose of the process is finding the global optima which can be never guaranteed, but the chance of finding a better solution in the next generation is always there. Another aim is faster convergence which is needed when the fitness function is costly to evaluate. Yet, the chance of reaching local solutions exists. Achieving a range of diverse with good solutions is another aim of the process.

3 Population

A population is a set of chromosomes (solutions). It includes a number of chromosomes for testing, the actual variables (phenotype parameters) that define the chromosomes, and search space information. The essential parts of the population that are used in the GAs are the population size and the initial population generation.

The size of the population is an essential parameter that plays an important role in the implementation of the GAs, it specifies the number of chromosomes in the population. Population size has been studied from different theoretical points of view. A larger size of population increases the number of fitness evaluations. In fact, too large a population would weaken the efficiency of the GA where finding the solution may cost a lot of time, while too small a population decreases the area for exploring the search space efficiently. Goldberg [16, 17] used the idea of schemata to consider the size of the population. His theory showed that the size of the population must grow as an exponential function of the chromosome length. But experimental proof by [18, 19] showed that Goldberg's suggestion is not necessary. The size of the population relies on the complexity of each problem. Choosing the initial population commonly assumed to be random. Another option is to seed the initial population with known good solutions.

3.1 Encoding

Encoding is a task of the actual variable genes representation. It can be done using numbers, bits, arrays, trees, or any other objects. Representations for genetic and evolutionary algorithms are a major requirement. Using GAs for optimization problems needs representations for possible solutions, in which performing the GAs is impossible without them.

In 1899, Mendel distinguished that the full genetic information for an individual are stored by nature in pairwise alleles (values of a gene). This information that specifies the shape, appearance, and properties of an individual are stored by a number of chromosomes. This chromosome characterizes an individual by using genes (the values of those genes can take to alleles). He discovered that nature recognizes between the outward appearance of an individual and its genetic code. The outward appearance is represented by a phenotype and the chromosome that contains all the information is represented by a genotype.

According to this, in the GA approach, the representation of the variables of the problem can be separated. The actual variables (phenotype) in the original formula and the encoded representation of the variables (genotype). The GAs perform on these two types alternatively, the solution (phenotype) and the coding (genotype) spaces. In the GA, encoding a problem's solution into a string is a main matter. It has been studied from many sides, such that, when individuals are decoded into solutions, information are mapping from genotype space to phenotype space. Using genetic and evolutionary algorithms for solving optimization problems needs a decomposition of a genotype-phenotype function and phenotype fitness mapping. Suppose we have a genotype search space $\mathfrak{X}$ (this space can be either continuous or discrete), and a function $\mathfrak{f}: \mathfrak{X} \rightarrow \mathfrak{R}$, our problem is to find

$$\min_{\mathfrak{x}\in\mathfrak{X}} \mathfrak{f}(\mathfrak{x})$$

where $\mathfrak{f}$ is the objective (fitness) function, and $\mathfrak{x}$ is a vector of the decision variables. Notice, the crossover and mutation genetic operators are applied to this space. Let $\mathfrak{A}$ be the set made up of symbols of an alphabet. Suppose that a string where each component is a symbol from $\mathfrak{A}$. The vector $\mathfrak{x}$ can be represented by a string $\mathfrak{s}$, of length l that relies on the dimension of $\mathfrak{X}$ and $\mathfrak{A}$, by using the map

$$k: \mathfrak{A}^l \to \mathfrak{X} \tag{1}$$

Due to the fact that some strings under the mapping k may represent invalid individuals for the original formula, where k is preferred to be a bijection, we have to employ a search space $\mathfrak{S} \subseteq \mathfrak{A}^l$. Now the optimization problem becomes

$$\min_{\mathfrak{s}\in\mathfrak{S}} \mathfrak{g}(\mathfrak{s})$$

where the function $\mathfrak{g}(\mathfrak{s}) = \mathfrak{f}(k(\mathfrak{s}))$ [22]. For binary strings, the cardinality of $\mathfrak{X}$ is equal to two, where $\mathfrak{A} = \{0, 1\}$. With binary alphabet, 2^l different individuals can be encoded where $|\mathfrak{X}| = 2^l$, while when the cardinality higher than two we get $\mathfrak{A}^l$ different encoded possibilities where the size of the search space is $\mathfrak{X} = \mathfrak{A}^l$ [23]. After generating a set of strings that represent the population, the reproduction process transforms it into a new population by performing the operators: selection, crossover, and mutation. There are different ideas in this regard. Holland and Goldberg [3, 9] claimed that encoding the variable by binary strings is in some sense optimal, while Rechenberg, and Schwefel [1, 24, 22] directly worked with the original decision variables. The type of problem that is being investigated imposes the choice of encoding. Different kinds of encoding that used for different problems are as follows:

3.1.1 Binary encoding

For many reasons, this type of encoding is regarded as the most common one. The GA theory is relied on using fixed length, fixed order binary encodings. Holland concentrated in his work on this type of representation [3]. This theory is extended to nonbinary encodings, but it did not improve as the original GA theory. Another reason, the suitable parameter settings such as the mating and mutation probabilities for GA has been improved in the context of binary representations [25].

1	0	1	1	0	0	1	0	1	0	0

The genotype-phenotype mapping in the binary encodings relies on the optimization problem. The representation permits a direct and more natural encoding for many discrete optimization problems. For using binary representations for encoding integer problems, certain genotype-phenotype mappings are needed. Various kinds of binary representations for integers represent the integers (phenotypes) in a various way to the binary vectors (genotypes). The binary, gray, and unary coding are the most known representations [23].

3.1.2 Integer encoding

For combinatorial optimization problems, integer or literal encoding is regarded the best. As long as, the fundamental of these problems is the search for a combination of items subject to constraints or a best permutation. The best method of this kind of problems is literal permutation encoding. In integer encoding, the decision variable vector is represented by a string with a cardinality greater than two. A set of alphabet $\mathfrak{A}$ is used for the genotypes where the search space size is $|\mathfrak{X}| = \mathfrak{A}^l$ [23].

1	2	3	4	5	6	7	8	9	10	11

3.1.3 Real-number encoding

Using GA to solve any difficult problem needs presenting good encoding, in fact using the best encoding is almost equivalent to solving the problem itself. While integer and literal permutation representations perform well for combinatorial and permutation problems, real-valued representation can be used for a function optimization problem. It is better than gray or binary representation to perform for function optimizations and constrained optimizations [21]. As the form of the genotype space for this encoding is indistinguishable to the phenotype space, using beneficial techniques from appropriate methods can simply assist in forming efficient genetic operators.

Here, the search space is defined as $\Re^l$, where l is the individual length. Special genotype-phenotype mappings and real-valued vectors simply can represent some problems such as schedules, tours, trees, and other discrete problems. Notice, many experimental comparisons between this type and the binary or multiple-character provided better results for the first [26, 27, 25]. The work relies very much on the details of the GA being used and the problem.

3.1.4 Tree encodings

Koza [4] used tree encoding schemes to represent computer programs. In concepts, the size of any tree could be constructed by means of mating and mutating. These schemes have some benefits, such as letting the search space to be open-ended. But, unfortunately, this can cause some possible misleads, in which the trees can widely expand in an uncontrolled way, stop the construction of more structured, hierarchical elect solutions [25].

3.2 Properties of representation

Testing a new representation method is important to check the genetic searching effect by using it. Some properties are suggested to evaluate encoding an individual [28, 1] such as,

- **Legality**, there is a correspondence between any solution and its encoding permutation. This assures applying the existing genetic operators to the encoding.
- **Nonredundancy**, in which the mapping between encoding and solution is one to one.
- **Completeness**, where any solution has a corresponding encoding which assures that the genetic search accesses any point in the phenotype space.
- **Causality**, this property is associated with mapping from encoding space to solution space. Rechenberg [1] proposed it to evolution techniques where small changes in the encoding space relate to a mutation show small changes in the solution space.
- **Lamarckian property**, it focuses on whether or not one individual can move its features to the next population through genetic operators [29].

3.3 Infeasibility and illegality

Infeasibility means that a solution decoded from an individual locates out the feasible area of the problem. This arises from the essence of the constrained optimization problem. For some problems, the equalities and inequalities represent the fitness area. Penalty method is used to deal with infeasible chromosomes in such cases [30, 31, 21]. In these problems, the optimal solution occurs at the boundary between the infeasible and feasible regions. The purpose of the penalty method is to force the genetic to bring the optimal solution closer to both sides of the regions.

Illegality in a given problem means that the individual does not represent a solution. This case arises from the encoding techniques. It results from using problem with particular encodings in numerous combinatorial optimization problems. To convert an illegal individual to a valid one, repair techniques are used. For many combinatorial optimization problems repairing an illegal or infeasible chromosome is relatively simple. Actually, the repairing technique exceeds other strategies such as penalizing and rejecting strategies.

Breeding the population is the core of the GA, in which new and hopefully better solutions are created via the search process. Breeding includes three steps; selecting parents, mating them to create new children, replacing the parents with the children.

4 Fitness

Fitness function is regarded as the main idea of Darwinian evolution; the genetic algorithm’s direction progress relies on this function to improve the population. The ability of an individual to compete within a population of solutions is indicated by fitness. Goldberg [32] provided the fitness function as “some measure of profit, utility, or goodness that we want to maximize”. In a genetic algorithm, the competition between chromosomes relies on their performance in the domain of problem solutions. Chromosomes are given a fitness value that reflects their performance when applied to a problem. Thus the potential of each chromosome in the population can be tested.

5 Selection

It is the process of picking pair of individuals from a population to creating children for the next generation. It is a procedure of randomly choosing individuals relying on their fitness evaluation. Individuals with higher fitness value have more chances to be selected. Selection pressure improves population fitness through successive generations.

GA convergence is affected by the selection rate. If the pressure is too high, the GA may converge early to an incorrect solution. If the pressure is too low, the average of the convergence will be slow. Selection should be balanced with variation from crossover and mutation. Too weak selection cases too slow evolution. Too high selection, suboptimal highly good enough solutions dominate the population which decreases the diversity required for further improvement. Some types of the main selection methods are introduced below.

- **Roulette wheel selection**. Roulette wheel selection is used by Holland [3, 21]. The simple idea behind it is to find the selection rate for each individual proportional to the fitness value. This is implemented by dividing the individual's fitness by the fitness rate of the population. The selection process relies on twirling the wheel N time equal to the population size. On each spin, a single individual is chosen for parenthood. This method is easy to perform but noisy. The average of the evolution relies on the variance of the fitness values in the population. The roulette wheel method can be employed as follows;

1. Evaluate the probability p_i of selecting each chromosome in the population;
$p_i = \frac{f_i}{\sum_{j=1}^{n} f_j}$, where n is the population size, f_i is the fitness of each chromosome,
And $\sum_{j=1}^{n} f_j$ is the accumulative fitness of the population.
2. Compute the cumulative probability, $q_i = \sum_{j=1}^{i} p_j$ for each chromosome.
3. Choose a random number $k \in (0, 1]$ such that if $k < q_i$ then choose the first chromosome x_1, if not then choose the chromosome x_i such that $q_{i-1} < k < q_i$.

- **Stochastic uniform selection.** It selects parents for the next generation based on their fitness values. It is also known as the Stochastic Universal Sampling (SUS) which is an improvement on the Roulette Wheel selection, [33]. It is a single phase sampling procedure with zero bias and minimum spread. While the Roulette wheel selects several parents from the population by repeated random sampling, SUS uses a single random value to sample all of the parents by selecting them at evenly spaced intervals. The parents are mapped to adjacent sections of a line. Each parent's section is equal in size to its fitness value. Let N be the number of selections needed, equally spaced pointers N are placed over the line to select these parents. The distance between the pointers is $\frac{1}{N}$ and the location of the first pointer is provided by a uniformly randomly generated number in $\left[0, \frac{1}{N}\right]$, [34]. See Figure 1.

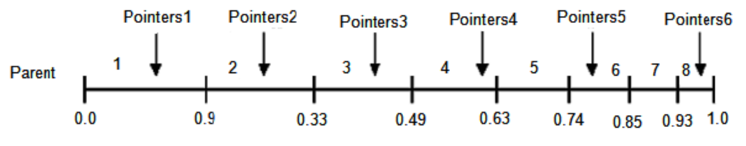

Figure 1- The Stochastic uniform selection

- **Remainder selection** allocates one parent deterministically from the integer part of each a scaled value of a chromosome and then employs the selection on the residual fractional part [35]. Chromosomes have a probability of selection based on their fitness, these fitness are computed by dividing chromosome fitness by average fitness.
- **Uniform selection** chooses chromosomes at random from a uniform distribution by applying the expectations and number of chromosomes [36].
- **Random selection.** This method randomly picks an individual from the population. On average, the roulette wheel selection is a little less disruptive than this type [37].
- **Rank selection.** This method ranks the individuals to find out the probability of survival. That's by sorting the population from finest to worst and assign the picking probability of an individual concerning the ranking but not its raw fitness [21].
- **Tournament selection**. It is randomly drawing a set of individuals then chooses the finest individual with the highest fitness value for parenthood [21, 37].
- Other types such as $(\mu + \lambda)$-selection and (μ, λ)-selection that introduced by Bäck, this method chooses the best individuals from parents and children [38, 21].
- **Elitist selection** proposed by Kenneth [39, 25], in addition to proportional selection, elitist maintains the best individual in the new population if it is not chosen by the proportional selection. The best individual or the few best individuals are copied to the next generation.
- **Block selection** and truncation selection methods rank all the chromosomes relating to their fitness value and choose the best as parents [40].
- **Boltzmann selection** is continuously changing the average of the selection relying on a current schedule [25].

6 Genetic operators

Searching the space for a solution is a solving method. There are two types of search: local and random search. Best solution is exploited by a local search which is eligible to climb up toward a local optimum. The solution can be explored by a random search which is eligible to get escape from a local optimum. Using the two types together provides a perfect search, but designing such a method for searching the space with suitable mechanisms unfortunately almost impossible.

The GAs are general search methods that provide a good balance between exploring and exploiting the search space by joining elements of directed and stochastic search. Designing the genetic operators is influenced by our conceptualization of the genetic search. Both the directed search and the random search capabilities are desirable in a search method. Cheng and Gen [21] proposed a genetic search that has two types of search capabilities. The crossover to explore the region beyond a local optimum by implementing a random search and improving the solution by using mutation to perform a local search. Here, the mutation acts a significant role as well as the crossover.

Genetic operators implement a random search but cannot assure improving children. A simple GA uses three operators in many practical problems which produces good results, they are; Reproduction, Crossover, and Mutation.

6.1 Reproduction

In GA this operation consists of two steps: selecting a single individual, and then copying it without alteration. This operator copies the chromosomes relying on their fitness values. Chromosomes with a higher value have a higher chance to provide one or more children in the next generation. This operator can be performed in many ways; the simplest way is using a biased roulette wheel in which each chromosome has a roulette wheel slot sized in proportion to its fitness [9].

6.2 Crossover

The execution of a genetic system strongly relies on this operator which is performed to the mating pool to enrich the population with better chromosomes. It is performed after the selection (reproduction) process. It recombines pair of chromosomes with a given probability to produce a new child [37]. The two parents that are randomly selected for mating are chosen with a probability p_c. If $k \le p_c$ for a random number k then the parents are recombined, else, if $k > p_c$then the two children are simply copies of their parents [9]. Some types of crossover operators are:

- **One point, and two point crossover operators**. These are the most well-known and easiest methods for recombining chromosomes. In one point crossover, to produce two children we choose two chromosomes and pick a single point on each of them which split both into two strands, then replace the two pieces. If some of the points are repeated, the validation of the chromosome will not be affected. In the second kind, we pick two points on the chromosome (say i and j) which splits it into three parts. Genes numbered less than or equal to i are chosen from the first chromosome. Genes numbered from $i + 1$ to j are chosen from the second chromosome. Then genes numbered greater than j are chosen from the first chromosome. Then the algorithm concatenates these genes to form a single chromosome. This idea can be extended to k-point crossover.

Table 1-Two points crossover for binary numbers

Parent1	1	1	0	0	0	1	0	1	0	0
Parent2	1	1	1	0	1	1	0	1	0	1

Which produce

Child1	1	1	0	0	1	1	0	1	0	0
Child2	1	1	1	0	0	1	0	1	0	1

- **Scattered operator.** It generates a random binary chromosome called a mask. It then chooses the genes that equal to 1 from the first parent, and the genes that equal 0 from the second parent, and combines them to form the child. Reversing the roles of the parents will produce the second child. [41].
- **Heuristic operator.** It generates children that are laid at random on the line including the parents, away from the parents who are of better fitness value and in a direction away from the parents of poor fitness value [21]. The children are found by the following equations, where $b \in (0, 1)$

Child1 = Best Parent + b * (Best Parent - Worst Parent)
Child2 = Best Parent.

Table 2-Scattered crossover for binary numbers

Parent1	1	1	0	0	0	1	0	1	0	0
Parent2	1	1	1	0	1	1	0	1	0	1
Mask	0	1	1	0	0	1	0	1	0	1
Child1	1	1	0	0	1	1	0	1	0	0
Child2	1	1	1	0	0	1	0	1	0	1

- **Arithmetic operator**. It uniformly generates children that on the line between the parents from the random arithmetic mean of both of them [21]. The new child is computed using the following equations. For a random weighting factor d.

Child1 = $d *$ Parent1 + $(1 - d) *$ Parent2

Child2 = $(1-d)*$ Parent1 $+ d *$ Parent2

These types of crossover are not suitable for search problems with a chromosome of integer numbers such as the travelling salesman problem.

- **Order Crossover.** This operation begins by choosing a substring (subchromosome) of random size from one of the two parents randomly, then copying it into its corresponding location in the child. The genes in the second parent that appeared in the substring are deleted to prevent repeating the gene. In the end, place the genes into the unfixed location of the child from left to right relying on the order of the sequence. See Figure 2, we create Child1 by choosing a substring from Parent1 (1, 10, 8) then delete this substring from Parent2 and add the rest (3, 6, 5, 2, 4, 7, 9) to Child1. The same for Child2, we choose (3, 6, 5) from Parent2 then delete it from Parent1 and add the rest (1, 10, 8, 7, 2, 4, 9) to Child2.

Table 3- Order crossover

Parent1	1	10	8	7	5	3	6	2	4	9
Parent2	3	6	5	2	4	8	7	9	1	10
Child1	**1**	**10**	**8**	3	6	5	2	4	7	9
Child2	**3**	**6**	**5**	1	10	8	7	2	4	9

- **Uniform order-based crossover.** Here, two parents are randomly chosen and a random binary string is created. First, child1 chromosome is filled by taking genes from the first parent that are in the 1's location of the string. The rest of the genes of this parent that correspond to zeros in the string is sorted in the same order as they appear in the second parent. The sorted genes list is used to fill the 0's location in child1. Child2 is generated in the same way.

Table 4- Uniform order-based crossover

Parent1	1	10	8	7	5	3	6	2	4	9
Parent2	3	6	5	2	4	8	7	9	1	10
Binary string	0	1	1	0	0	1	0	1	0	1
Child1	6	10	8	5	4	3	7	2	1	9
Child2	1	6	5	7	3	8	2	9	4	10

6.3 Mutation

This operator plays a secondary role in the GAs operation, which causes random changes in different chromosomes. It protects the search space from losing the diversity in genetic information that results from the crossover operation [9]. It enables GA to avoid being trapped in local optima. This is implemented by changing one or more genes. There are many types of mutation that can be suitable for various representations. Simple mutation can be used for binary representation where the gene can be converted via a small probability. The probability is the reciprocal of the length of the chromosome. Some of these mutations are:

1. **Flipping.** It converts 0 to 1 and 1 to 0 relied on a mutation string created.
2. **Interchanging.** It selects two locations at random then interchanges the bits in these locations.
3. **Reversing.** It mutates the chromosome by selecting a random location and exchanges it by the bits beside this location.

Some other mutations that are employed for other types of representations are (*Uniform*, *Gaussian*, and *Adaptivefeasible*).

- **Gaussian mutation**. It has been provided in Evolution Strategy by Rechenberg in 1973, [42], to create a new offspring, a Gaussian element is added to the chromosome or parent. It is picked from a Gaussian distribution with a mean 0. Gaussian mutation can control the variance of the population fitness of each generation according to [43].

- **Uniform mutation.** It is a process of two steps. First, the algorithm chooses a fraction of the chromosome entries of an individual for mutation. These entries have a similar probability as the mutation rate of being mutated. Second, the entry is changed by the algorithm by choosing a random number from the range of the entry [21].
- **Adaptive feasible mutation.** It randomly generates directions that are adaptive for the last successful or unsuccessful generations. Along each direction, a step length is selected such that bounds and linear constraints are satisfied [44].

7 Convergence criteria

Genetic algorithms are stochastic search procedures that can be implemented forever. A stop criterion is necessary for practice. Some approaches to terminate search in GAs are as follows [37].

1. No change in fitness, if there is no change in the value of the population's fitness for a particular number of generations, the algorithm terminates.
2. Still time limit, if there is no progress in the objective function within a period of time, the genetic algorithm will stop.
3. Still generations, if there is no progress in the objective function for a certain number of successive generations, the algorithm terminates.
4. Maximum generations, when a particular number of generations are reached, the algorithm terminates.
5. Elapsed time, when a certain time passes the genetic process is ended.

8 On GA complexity

8.1 Some interesting general results

The time complexity analysis and the computation time theory of Evolutionary Algorithms (EAs) is a fundamental subject. It displays the expected number of generations to reach the optimal solution [45- 48]. The theoretical results regarding time complexity are relatively few although the EAs have a lot of applications in discrete optimization [49, 46, 50]. In previous work most of the time complexity focuses on easy cases or provided $(1 + 1)^*$-EAs without crossover. Some general outcomes are covered in this survey.

- He and Yao [51] suggested a general EA that employs a finite population, crossover, mutation, selection. For a problem the cost which is at least exponential time (in problem size n), they proposed some drift conditions such that an EA will cost no more than polynomial time (in problem size n) to solve the problem.

Instead of working on a specific EA, they provided a general theory for a class of EAs. Their theory has been improved via drift analysis which is a useful procedure for analysing random sequences [52]. This theory can find the first hitting time (the stopping time on the optimum) by estimating the drift of a random sequence which is simpler than estimating the first hitting time directly. Their simple proposal starts with modelling the evolution of an EA population as a random sequence, like a Markov chain. The EA consists of a population of multiple chromosomes with the crossover and mutation operators. The drift of the sequence is analyzed to and from the optimum value. Subject to various drift conditions, different bounds are found on the first hitting time. Some of these conditions lead the random sequence to drift in the direction of the optimum, whereas other conditions drift the sequence away from the optimum.

The result of the theory is applied to some classical combinatorial optimization problems such as (the subset sum problem). A simple description of the EAs and drift analysis with some results can be as follows: Find max$\{f(y); y \in V\}$ for a function $f(y)$ and a finite state space V, $y \in V$. Suppose $f_{max} = f(y^*)$ where y^* is one state with the optimum value, the EA used for solving the problem is as follows:

---

*(1+1)-EAs is a population size of 1.

1. Generate heuristically or randomly an initial population of $2N$ chromosomes, represented By $\lambda_0 = (y_1, \dots, y_{2N})$ Let $i \leftarrow 0$ and N an integer number, $N > 0$. Define $f(\lambda_i) = \max\{f(y_i);\ y_i \in \lambda_i\}$ for any population λ_i.
2. Generate a new population of offspring using the crossover and mutation operators or any other operators and denote it $\lambda_{i+1/2}$.
3. Select and reproduce $2N$ chromosomes from populations $\lambda_{i+1/2}$. and λ_i and find a new intermediate one λ_{i+V}.
4. Stop if $f(\lambda_{i+V}) = f_{max}$; else let $\lambda_{i+1} = \lambda_{i+V}$ and $i \leftarrow i+1$, and go to second step 2.

This EA is nearer to evolutionary programming and the evolution strategies [2] than to GAs [9] with respect to applying the crossover and/or mutation before the selection operator. The drift analysis is described as follows: Let $d(y, y^*)$ be the distance between a point y and the optimum y^*. In case there is more than one optimal value, then the distance between the chromosome y^* and the optimal set V^* is suggested as $d(y, V^*) = \min\{d(y, y^*) : y^* \in V^*\}$, simply noted as $d(y)$. The distance from a given population $Y = \{y_1, \dots, y_{2N}\}$ to the optimum is

$$d(Y) = \min\{d(y) : y \in Y\} \tag{2}$$

The EA generates a random sequence $\{d(\lambda_i); i = 0, 1, 2, \dots\}$ which can be modelled by a homogeneous Markov chain. The drift of this sequence at time i is described as

$$\Delta(d(\lambda_i)) = d(\lambda_{i+1}) - d(\lambda_i)$$

The time of the EA to stop is defined as $t = \min\ \{i : d(\lambda_i) = 0\}$ which represents the first hitting time on the optimum value. Their work [51] tested the relationship between the problem size n and the predicted first hitting time t. In fact, they estimated the first hitting time $E(t)$ with respect to different conditions. To this end, suppose a deterministic algorithm is used to solve an optimization problem with the distance d between the optimum and the starting solution. We require at most d/Δ time iterations to reach the optimum when the drift towards the optimum is greater than Δ at each iteration. Many results were provided for some problems with polynomial time. The problems are;

1. The subset sum problem is NP-complete. They found the first hitting time $h(n) = O(n^2)$ where $h(n)$ is a polynomial of problem size n.
2. The linear functions which are defined as; A function $f: V \to R$ is linear if $f(y) = u_0 + \sum_{j=1}^{n} u_j v_j$ where $u_j \in R$. It is proved that it requires an average $O(n\ ln\ n)$ steps to reach the optimum.
3. Pesudo-modular functions; An example of this function is

$$f(y) = \sum_{j=1}^{n} \prod_{k=1}^{j} v_k \tag{3}$$

The expected first hitting time of the EA for the function is $E|t| = n^2(e-1)$.

4. Unimax functions; A function $f: V \to R$ is unimax if the $y^* \in V$ is the only locally maximal point. One of the known unimax problems is the long path problem. The path Ln for an odd number n is defined by recursion with $L_1 = \{0, 1\}$ as the base path. The length of the paths is set out by the recurrence equations $|L_1| = 2, |L_{n+2}| = 2|L_n| + 1$, and its solution is $|L_n| = 3.2^{\frac{n-1}{2}} - 1$ for odd $n \gg 1$. Given y as a point on the path $|L_n|$, the fitness function

$$f(y) = -\left(3.2^{\frac{n-1}{2}} - 2\right) + \begin{cases} Pos(y), & if\ Pos(y) \geq 0, \\ -\sum_{j=1}^{n} v_j, & otherwise. \end{cases} \quad (4)$$

where $Pos(y)$ is the location of y on the path that is numbered from 0 to $3.2^{\frac{n-1}{2}} - 2$ $Pos(y)$is equal -1 for a point not belonging to the path. The expected hitting time of the EA of this function is $E[t] = O(n^3)$.

5. Almost positive functions; Let $f: V \to R$ be a function, it says to be almost positive if the coefficients of all nonlinear terms are non-negative [47]. An example of the function where the distance function can be defined as $d(y) = \sum_{j=1}^{n} |v_j - 1|$.

$$f(y) = n - \sum_{j=1}^{n} v_j + (n+1) \prod_{j=1}^{n} v_j \quad (5)$$

The expected first-hitting time of the EA for the almost positive function is
$E[t] = \Omega(n^n)$.

- He and Yao [48] again estimated the complexity time of the EAs drift analysis. They discussed drift conditions that are employed to find the lower and upper bounds of the first hitting times. Also, a new general classification of easy and hard problems for EAs relies on this analysis are provided. Their EA is proposed to solve maximization problems. Let E be the set of all populations, and let a random variable λ_ι that its values from E be the ι^{th} population of generation. Let E^* be the set of populations with the optimum, to test the distance of the population say y to E^* a distance function is used. The distance is defined in many ways like Hamming distance or $G(y) = \min\{|f(y) - f(z)|; z \in E^*\}$ or $G(y) = 0$ if $y \in E^*$ and $G(y) = 1$ if $y \notin E^*$, where y and z are any population in E.

To find the EAs computation time, Markov chain and supermartingale* are provided as mathematical models for the EAs. Markov chain is used to model the sequence of random variables $\{\lambda_\iota; \iota = 0, 1, 2, \ldots\}$. To find the gain of a population towards the optimal solution, the one-step mean drift at λ_ι generation for a given distance function is needed. It is defined as

$$E[G(\lambda_\iota) - G(\lambda_{\iota+1})\ \lambda_\iota = y] \coloneqq G(y) - \sum_{z \in E} P(y, z; \iota) G(z)$$

where $P(y, z; \iota)$ is the transition probability and $y, z \in E$ are populations.

$$P(y, z; \iota) \coloneqq P(\lambda_{\iota+1} = \mathrm{z} | \lambda_\iota = \mathrm{y})$$

If this drift is positive then the population converges to the optimal solution and if it is negative then it diverges far from it. For the EA, number of generations to reach optimum (first hitting time), is defined as $t \coloneqq \min\{\iota \geq 0;\ \lambda_\iota \in E^*\}$. The drift analysis that is applied for specific EAs is shown in the following two cases.

Analysis of a (1+1)-EA for linear functions

This problem can be solved by a (1+1)-EA with mutation and selection with two different distance functions [48]. The first function with a binary string $y = (v_1, \ldots, v_n)$ which is

$$d(y) = 4(n-1)^2 \sum_{j=1}^{n} |v_j - 1| \quad (6)$$

Which produces first hitting time $E[t|\lambda_0] = O(n^3)$. This result can be tightened further by using the second distance

$$d(y) = n \ln(1 + \sum_{j=1}^{n/2} s|v_j - 1| + \sum_{j=\frac{n}{2}+1}^{n} |v_j - 1|), \quad (7)$$

where $s(1 < s \leq 2)$ and it gives a first hitting time $E[t|\lambda_0] = O(n \ln n)$.

Analysis of a $(n + n)$-EA for ONEMAX problem

This problem can be solved by (1 + 1)-EA in time $O(n \ln n)$. He and Yao solved by $(n +$

[*]Supermartingale is a method to model EAs to find the convergence of non-elitist selection strategy.

$n)$-EA with mutation and selection only, where n is the length of the string. They found an optimal solution equal to $E[t|\lambda_0] = O(n \ln n)$.

This result can be changed into $E[t|\lambda_0] = O(n)$ with a different distance function; Hamming distance between a chromosome and the optimum value. He and Yao separated the optimization problems into two types based on the average number of generations required to find the solution.

1. **Easy class**; given EA, the average number of generations required by the EA to solve a problem is polynomial in the problem size. The sufficient and necessary conditions for this class are as follows.

Theorem 8.1. *[48] Given an EA which can be modelled by a homogeneous absorbing Markov chain, a problem belongs to the easy class iff there exists a distance function $G(y)$ such that; for a polynomial $h_1(n)$ in the problem size n, $G(y) \leq h_1(n)$, and for any population λ_ι at generation ι with $G(\lambda_\iota) > 0$, the one-step mean drift satisfies*

$$E[G(\lambda_\iota) - G(\lambda_{\iota+1}) | \lambda_\iota] \geq b_{low}, \tag{8}$$

where $b_{low} > 0$ is a lower bound constant.

2. **Hard class**; given EA, the average number of generations required by the EA to solve a problem is exponential in the problem size.

Theorem 8.2. [48] *Given an EA which can be modelled by a homogeneous absorbing Markov chain, a problem belongs to the hard class iff there exists a distance function $G(y)$ such that; for some population $y \in E$ and an exponential $h_2(n)$ in the problem size n, G(y) satisfies $G(y) \geq h_2(n)$, and for any population λ_ι at generation ι with $G(\lambda_\iota) > 0$, the one-step mean drift satisfies*

$$E[G(\lambda_\iota) - G(\lambda_{\iota+1}) | \lambda_\iota] \leq b_{up}, \tag{9}$$

where b_{up} is a positive upper bound constant.

- Chen et al [53], analyzed the time complexity of the population-based evolutionary algorithms on unimodal problems by improving a new general method relying on EAs models and some common[well-known] concepts, like the supermartinle [52, 51, 48, 54], the Markov chain model [55, 50, 56, 57], and the takeover time which provided by Goldberg and Deb [32]. They joined the idea of the overtake time to EAs and drift analysis to prove that the $(N + N)$-EA with truncation selection (or two-tournament selection) and the bitwise mutation requires $O(n \ln n + n \ln n / N)$ and $O(n \ln n + n^2 / N)$ generations to obtain the global optimum of the ONEMAX and LEADINGONES problems, consequently, where N is a number of individuals and n is the length of the string. Instead of using only a selection operator, they generalized the takeover time for the EAs with mutation operators.

- Thierens et al, [58] analyzed the time complexity of convergence of the BinInt problem which provided by Rudnick [59]. It is called the sequential convergence (*domino convergence*) because it is similar to a falling row of domino stones. They provided that the time complexity of domino convergence is exponential $O(2^n)$ for proportionate selection and linear in number of building blocks $O(n)$ for selection algorithms with constant selection intensity like (tournament or truncation selection). These results were compared with the previous results for ONEMAX problem, where n is the chromosome length. Their convergence behaviour differs from the ONEMAX or Bit-Counting problem which is the reason of choosing it as a prototype example. Their analysis provided that the ONEMAX problem is of order $O(n\ ln\ n)$ for proportionate selection and $O(\sqrt{n})$ for selection algorithms with constant selection intensity.

- Fernando et al, [60] introduced experimental and theoretical analysis of the GAs complexity time on problems exponentially scaled building blocks. This work relies on the previous one of Thierens et al, but here for the building blocks instead of the case of single genes. They found an overall quadratic time complexity in terms of the evolution of the fitness function under idealized situations as they suggested perfect building block mixing. For the case of the building blocks uniformly scaling, where the time complexity of the GA with perfect mixing is $O(m)$ where the population size and convergence time grow linearly with $\sqrt{m}$ and for the building blocks, exponential scaling is $O(m^2)$. Integer m is a number of building blocks.
- Rylander, [61] illustrated the Minimum Chromosome Length (MCL) method to compute genetic algorithm time complexity of problems. Using this approach shows the possibility of finding time complexity of problems based GAs. This approach relies on the search space growth rate as a function of the input problem size. They proved two particular cases empirically and defined a new complexity class NPG (the class of problems that can be solved by GA with cost more polynomial time). The worst case complexity of the problem for a GA can be bounded by the MCL growth rate. The problem belongs to the class PO (the optimization equivalent of P) if the MCL grows slowly enough. Conversely, it belongs to the class of NPO (the optimization equivalent of NP) as the MCL growth rate will be no more than linear where searching the space does not grow faster than exponentially [62].

9 Conclusion

Currently, large companies use GA to optimize problems that concern schedules and designing. GAs are very common for optimization [63, 64, 65], which is different from classical optimization approaches. It uses the coding of the problem's parameters instead of the parameters themselves. It implies a probabilistic transition function. This approach may not find the optimum because either the algorithm converges fast and stop before getting the optimum or the algorithm is far from the optimum solution. The ability of GAs to explore and exploit together and successfully apply to real-life problems confirms that GAs are robust and powerful optimization approaches. This survey introduced the basic idea of the GAs. The historical improvement of the evolutionary computation is viewed. The survey included the main concepts of GAs, GA operators, and convergence criteria. Also, some interesting results and theorems on time complexity were provided.

References

[1] I. Rechenberg, "Evolutions strategie: Optimierung technischer systeme nach prinzipien der biologischen evolution," *Frommann-Holzboog Verlag, Stuttgart (2nd edition 1993),* 1973.

[2] H.-P. Schwefel, "Evolution and Optimum Seeking", Wiley, New York, NY," 1995.

[3] J. H. Holland, *Adaptation in natural and artificial systems.* Ann Arbor MI: University of Michigan Press, 1975.

[4] J. R. Koza, *Genetic programming: On the programming of computers by means of natural selection,* Cambridge: MIT Press, vol. 1, 1992.

[5] L. J. Fogel, A. J. Owens, and M. J. Walsh, *Artificial Intelligence Through Simulated Evolution,* Wiley, New York.

[6] M. A. Iquebal, "Genetic algorithms and their applications: An overview," *Ind. J. Agric. Sci,* vol. 79, pp. 399-401, 2009.

[7] L. Davis and M. Steenstrup, Genetic algorithms and simulated annealing: An overview, ch. 1, pp. 1-11, 1987.

[8] L. Booker, "Intelligent behaviour as an adaptation to the task environment (doctoral dissertation, technical report no. 243. ann arbor: University of Michigan, logic of computers group)," *Dissertations Abstracts International,* vol. 43, no.2, 1982.

[9] D. E. Goldberg et al., *Genetic algorithms in search optimization and machine learning*. Addison-Wesley Reading Menlo Park, 1989.

[10] J. J. Grefenstette, (Editor), *Proceedings of the Second International Conference on Genetic Algorithms, Lawrence Erlbaum Associates, Hillsdale, NJ.* 1987.

[11] Z. Michalewicz, J. B. Krawczyk, M. Kazemi, and C. Z. Janikow, "Genetic algorithms and optimal control problems," in *Decision and Control, 1990. Proceedings of the 29th IEEE Conference on,* pp. 1664-1666, IEEE, 1990.

[12] J. J. Grefenstette, "Incorporating problem specific knowledge into genetic algorithms," *Genetic algorithms and simulated annealing,* vol. 4.

[13] G. A. Vignaux, and Z. Michalewicz, "A genetic algorithm for the linear transportation problem," *Systems, Man and Cybernetics, IEEE Transactions on,* vol. 21, no. 2, pp. 445-452, 1991.

[14] Z. Michalewicz, *Genetic algorithms+ data structures= evolution programs*. Springer Science & Business Media, 1992.

[15] A. Salhi, H. Glaser, and D. De Roure, "Parallel implementation of a genetic programming based tool for symbolic regression," *Information Processing Letters*, vol. 66, no. 6, pp. 299-307, 1998.

[16] D. E. Goldberg, *Optimal initial population size for binary-coded genetic algorithms. Clearinghouse for Genetic Algorithms,* Department of Engineering Mechanics, University of Alabama, 1985.

[17] D. E. Goldberg, "Sizing populations for serial and parallel genetic algorithms," in *Proceedings of the 3rd International Conference on Genetic Algorithms,* pp. 70-79.

[18] J. J. Grefenstette, "Optimization of control parameters for genetic algorithms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 16.

[19] J. D. Schaffer, R. A. Caruana, L. J. Eshelman, and R. Das, "A study of control parameters affecting online performance of genetic algorithms for function optimization," in *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 51-60, Morgan Kaufmann, Los Altos, CA, 1989.

[20] L. Booker, "Improving search in genetic algorithms," *Genetic algorithms and simulated annealing, Morgan Kaufmann, San Francisco*, pp. 61-73.

[21] M. Gen, and R. Cheng, *Genetic algorithms and engineering optimization*, vol. 7. John Wiley & Sons, 2000.

[22] F. W. Glover, and G. A. Kochenberger, *Handbook of metaheuristics,* vol. 57. Springer Science & Business Media, 2006.

[23] R. Franz, "Representations for genetic and evolutionary algorithms", 2006.

[24] H.-P. Schwefel, *Numerische optimierung von computer-modellen mittels der evolutions strategie,* vol. 1. Birkhäuser, Basel Switzerland, 1977.

[25] M. Mitchell, *An introduction to genetic algorithms*. MIT press, 1998.

[26] C. Z. Janikow, and Z. Michalewicz, "An experimental comparison of binary and floating point representations in genetic algorithms.," in *ICGA,* pp. 31-36, 1991.

[27] A. H. Wright, "Genetic algorithms for real parameter optimization. In G. Rawlins, ed.," *Foundations of Genetic Algorithms. Morgan Kaufmann,* 1991.

[28] S. Kobayashi, "Foundations of genetic algorithms and its applications," *Communications of ORSJ,* vol. 45, pp. 256-261, 1993. (in Japanese).

[29] R. Cheng, M. Gen, and Y. Tsujimura, "A tutorial survey of job-shop scheduling problems using genetic algorithms--I. Representation, Computers & Industrial Engineering, vol. 30, no. 4, pp. 983-997, 1996.

[30] Z. Michalewicz, "A survey of constraint handling techniques in evolutionary computation methods.," *Evolutionary Programming,* vol. 4, pp. 135-155, 1995.

[31] M. Gen and R. Cheng, "A survey of penalty techniques in genetic algorithms," in *Evolutionary Computation, Proceedings of IEEE International Conference on*, pp. 804-809, IEEE, 1996.

[32] D. E. Goldberg and K. Deb, "A comparative analysis of selection schemes used in genetic algorithms," *Foundations of Genetic Algorithms. San Matteo in Genetic Algorithms*, vol. 1, pp. 69-93, 1991.

[33] J. E. Baker, "Reducing bias and inefficiency in the selection algorithm," in *Proceedings of the second international conference on genetic algorithms*, vol. 206, pp. 14-21, 1987.

[34] T. Pencheva, K. Atanassov, and A. Shannon, "Modelling of a stochastic universal sampling selection operator in genetic algorithms using generalized nets," in *Proceedings of the Tenth International Workshop on Generalized Nets, Sofia*, pp. 1-7, 2009.

[35] E. Cantu-Paz, *Efficient and accurate parallel genetic algorithms*, vol. 1. Springer Science & Business Media, 2000.

[36] T. Ray, and K. S. Won, "An evolutionary algorithm for constrained bi-objective optimization using radial slots," in *International Conference on Knowledge-Based and Intelligent Information and Engineering Systems*, pp. 49-56, Springer, 2005.

[37] G. Chartier, "Introduction to genetic algorithms", 2008.

[38] T. Back, "Selective pressure in evolutionary algorithms: A characterization of selection mechanisms," in *Evolutionary Computation, IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on*, pp. 57-62, IEEE, 1994.

[39] K. DeJong, "An analysis of the behavior of a class of genetic adaptive systems," *Ph.D. Thesis, University of Michigan*, 1975.

[40] D. Thierens, and D. Goldberg, "Convergence models of genetic algorithm selection schemes," in *International Conference on Parallel Problem Solving from Nature*, pp. 119-129, Springer, 1994.

[41] R. Chakrabortty, and M. Hasin "Solving an aggregate production planning problem by using multi-objective genetic algorithm (MOGA) approach," *International Journal of Industrial Engineering Computations*, vol. 4, no. 1, pp. 1-12, 2013.

[42] S. Blum, R. Puisa, J. Riedel, and M. Wintermantel, "Adaptive mutation strategies for evolutionary algorithms," in *The Annual Conference: EVEN at Weimarer Optimierungsund Stochastiktage*, vol. 2, 2001.

[43] Y. Wu, *Software engineering and knowledge engineering: theory and practice*, vol.2. Springer Science & Business Media, 2012.

[44] P. Vasant, "Hybrid mesh adaptive direct search and genetic algorithms techniques for industrial production systems," *Archives of Control Sciences*, vol. 21, no. 3, pp. 299-312, 2011.

[45] H.-G. Beyer, H.-P. Schwefel, and I. Wegener, "How to analyse evolutionary algorithms," *Theoretical Computer Science*, vol. 287, no. 1, pp. 101-130, 2002.

[46] A. E. Eiben, and G. Rudolph, "Theory of evolutionary algorithms: a bird's eye view," *Theoretical Computer Science*, vol. 229, no. 1/2, pp. 3-9, 1999.

[47] G. Rudolph, "Finite markov chain results in evolutionary computation: a tour d'horizon," *Fundamenta Informaticae*, vol. 35, no.1-4, pp. 67-89, 1998.

[48] J. He and X. Yao, "A study of drift analysis for estimating computation time of evolutionary algorithms," *Natural Computing*, vol. 3, no. 1, pp. 21-35, 2004.

[49] S. Droste, T. Jansen, and I. Wegener, "On the analysis of the (1+ 1) evolutionary algorithm," *Theoretical Computer Science*, vol. 276, no. 1-2, pp. 51-81, 2002.

[50] J. He and X. Yao, “Towards an analytic framework for analysing the computation time of evolutionary algorithms,” *Artificial Intelligence*, vol. 145, no. 1/2, pp. 59-97, 2003.

[51] J. He and X. Yao, “Drift analysis and average time complexity of evolutionary algorithms, “*Artificial Intelligence,* vol. 127, no. 1, pp. 57-85, 2001.

[52] B. Hajek, “Hitting-time and occupation-time bounds implied by drift analysis with applications,” *Adv. Appl. Prob,* vol. 14, no. 3, pp. 502-525, 1982.

[53] T. Chen, J. He, G. Sun, G. Chen, and X. Yao, “A new approach for analyzing average time complexity of population-based evolutionary algorithms on unimodal problems,” *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 39, no. 5, pp. 1092-1106, 2009.

[54] P. S. Oliveto, and C. Witt, “Simplified drift analysis for proving lower bounds in evolutionary computation,” in *International Conference on Parallel Problem Solving from Nature, Dortmund, Germany*, pp. 82-91, Springer, 2008.

[55] J. Garnier, L. Kallel, and M. Schoenauer,”Rigorous hitting times for binary mutations,” *Evolutionary Computation*, vol. 7, no. 2, pp. 173-203, 1999.

[56] J. Suzuki, “A markov chain analysis on simple genetic algorithms,” *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 25, no. 4, pp. 655-659, 1995.

[57] J. Suzuki, “A further result on the markov chain model of genetic algorithms and its application to a simulated annealing-like strategy,” *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 28, no. 1, pp. 95-102., 1998.

[58] D. Thierens, D. E. Goldberg, and A. G. Pereira, “Domino convergence, drift, and the temporal-salience structure of problems,” in *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence. The 1998 IEEE International Conference on*, pp. 535-540, IEEE, 1998.

[59] W. M. Rudnick, “Genetic algorithms and fitness variance with an application to the automated design of artificial neural networks. PhD thesis, Oregon Graduate Institute of Science and Technology”, 1992.

[60] F. G. Lobo, and D. E. Golberg, and M. Pelikan, “Time complexity of genetic algorithms on exponentially scaled problems,” in *Proceedings of the 2nd Annual Conference on Genetic and Evolutionary Computation*, pp. 151-158, Morgan Kaufmann Publishers Inc., 2000.

[61] B. Rylander, *Computational Complexity and the Genetic Algorithm. PhD* thesis, University of Idaho, 2001.

[62] J. L. Balcázar, J. Diaz and J. Gabarró, “Structural complexity Theory I, volume 11 of eatcs monographs on theoretical computer science, Springer-Verag”, 1988.

[63] R. Abo-Alsabeh and A. Salhi, "An Evolutionary Approach to Constructing the Minimum Volume Ellipsoid Containing a Set of Points and the Maximum Volume Ellipsoid Embedded in a Set of Points." *Journal of Physics: Conference Series*. Vol. 1530. No. 1, pp. 012087, IOP Publishing, 2020.

[64] S. Muhammad, A. Salhi, and E. S. Fraga, "The plant propagation algorithm: modifications and implementation." *arXiv preprint arXiv:1412.4290*, 2014.

[65] A. Salhi, and E. S. Fraga, "Nature-inspired optimisation approaches and the new plant propagation algorithm." K2-1, 2011.