Computer Modeling in Engineering & Sciences

Tech Science Press

https://doi.org/10.32604/cmes.2026.079859

REVIEW

A Review of Genetic Algorithms: Principles, Procedures, and Applications in Optimization

M. A. El-Shorbagy*

Department of Mathematics, College of Science and Humanities in Al-Kharj, Prince Sattam bin Abdulaziz University, Al-Kharj, Saudi Arabia

*Corresponding Author: M. A. El-Shorbagy. Email: mohammed_shorbagy@yahoo.com

Received: 29 January 2026; Accepted: 17 March 2026; Published: 27 April 2026

ABSTRACT: This paper provides a thorough examination of Genetic Algorithms (GAs), a category of evolutionary computation methods derived from the concepts of natural selection and genetics. The main concept and operational principle of GAs are elucidated, highlighting the evolution of populations of candidate solutions across multiple generations to get optimal or near-optimal solutions for complicated problems. The paper delineates the sequential phases of a conventional GA, encompassing problem formulation, solution encoding, initialization of population, fitness evaluation, selection, crossover, mutation, and termination criteria, so offering a coherent framework for comprehending the algorithm's functionality. Moreover, numerous prominent genetic operators, including crossover and mutation, are examined, highlighting their distinct forms and processes for fostering diversity and exploration within the search space. Also, the paper emphasizes the benefits of GAs, including their capacity to address nonlinear, multimodal, and high-dimensional optimization challenges without necessitating gradient information, along with their adaptability in resolving both continuous and discrete issues. The limitations and constraints of GAs, such as computing expense, parameter optimization, and the risk of premature convergence, are thoroughly analyzed. The paper examines various applications of GAs across fields, including engineering design, control systems, combinatorial optimization, machine learning, operations research, and multi-objective optimization, demonstrating the versatility and practical significance of this evolutionary method. This work establishes a robust basis for scholars and practitioners seeking to implement GAs in intricate optimization challenges. The review indicates that GAs have greatly progressed from Holland's original formulation to specialized variations, such as real-valued, permutation, and tree-based encodings, each tailored to certain issue categories. The critical study indicates that although classical GAs are proficient in global exploration, their hybridization with local search techniques (memetic algorithms), swarm intelligence (GA-PSO), and surrogate models significantly improves convergence time and solution accuracy. The study highlights ongoing research deficiencies, such as the disparity between theoretical convergence proofs and the actual performance of algorithms, as well as the necessity for systematic recommendations in the design of hybrid algorithms.

KEYWORDS: Genetic algorithm; evolutionary computation; global optimization; nonlinear optimization; computational intelligence; optimization techniques

1 Introduction

GAs are acknowledged as effective global optimization methods, especially adept at addressing nonlinear optimization challenges [1]. Numerous actual design and engineering optimization challenges cannot be efficiently resolved using conventional optimization techniques due to the presence of discontinuous, stochastic, highly nonlinear, or non-differentiable objective functions [2]. Classical optimization methods

frequently demonstrate inefficiency, high computational costs, and often converge to a local optimum close to the initial starting point. Conversely, GAs can surmount these obstacles as they do not depend on linearization assumptions or derivative calculations. Their primary benefit over conventional approaches is their global sampling capabilities, allowing for broader exploration of the search space instead of being limited to local areas [3].

Numerous extensive studies of GAs are present in the literature, each highlighting different aspects. Certain studies offer comprehensive summaries of essential GA techniques but fail to comprehensively categorize hybrid methodologies. Others focus predominantly on engineering and practical applications, providing minimal theoretical synthesis. Previous foundational reviews outline fundamental principles and classical formulations; nevertheless, they predate numerous current advancements in adaptive mechanisms and hybrid GAs. This review diverges from previous studies in three key aspects. Initially, it presents a comprehensive taxonomy that categorizes GA variations across many dimensions, such as representation schemes, operator adaptation strategies, and hybridization structures. Secondly, it offers a critical examination of the evolutionary advancement of GAs, mapping their evolution from traditional formulations to modern adaptive and hybrid models. Third, it provides a stringent conceptual definition of authentic algorithmic hybridization, distinctly differentiating substantive methodological integration from superficial or weakly connected combinations. This review complements prior surveys by offering historical continuity and methodological clarity for academics and practitioners aiming to comprehend and use contemporary advancements in GAs [4,5].

This review presents a new contribution via its systematic taxonomy that categorizes GA variants across several dimensions—representation, operators, and hybridization—offering researchers a formal framework for comprehending the links among various techniques. Furthermore, we present a stringent definition of authentic algorithmic hybridization that differentiates substantive integration from mere surface amalgamation, providing methodological clarity frequently absent in the literature.

Using concepts from genetics and natural selection, GAs perform stochastic searches and optimizations [6]. Instead of starting with a single point in the search space, as is the case with traditional deterministic search techniques, GAs work with a population of potential solutions. Starting with a population of randomly generated solutions (chromosomes) that fit within the constraints of the issue is the first step of the process. There is a potential solution for every chromosome [7].

Following the establishment of the first population, subsequent generations of solutions are generated through the utilization of genetic operators, which include:

- Selection identifies individuals according to their fitness values, favoring superior solutions.
- Crossover (recombination) merges segments of two parental chromosomes to generate novel offspring solutions.
- Mutation brings minor random mutations into chromosomes to maintain variety and avert premature convergence.

By repeatedly using these operators, the algorithm generates subsequent generations of chromosomes. The mean quality of solutions in each successive generation is generally superior to that of its predecessor, progressively steering the population toward optimal or near-optimal solutions [8]. The evolutionary process persists until a stopping requirement is satisfied, such as attaining a maximum number of generations or achieving a certain convergence threshold. Upon completion of the process, the highest-performing chromosomes from the last generation are presented as the solution set [9].

This work aims to elucidate GAs as a potent optimization method derived from natural evolution. The paper specifically intends to:

- Elucidate the core concept and operational concepts of GAs, demonstrating how populations of candidate solutions undergo evolutionary processes throughout generations to converge towards optimal answers.
- Elucidate the fundamental stages of a conventional GA, encompassing issue formulation, solution encoding, population initialization, fitness evaluation, mechanisms of selection, and the implementation of genetic operators, including crossover and mutation.
- Present and examine frequently employed genetic operators, emphasizing their function in enhancing diversity, navigating the search space, and facilitating convergence to optimal solutions.
- Examine the benefits and drawbacks of GAs, highlighting their capacity to resolve intricate, nonlinear, and multimodal issues, while also considering concerns such as computing expense, parameter optimization, and the risk of premature convergence.
- Examine the practical applications of GAs in various fields, such as engineering design, control systems, combinatorial optimization, machine learning, operations research, and multi-objective optimization, to illustrate their adaptability and significance in real-world scenarios.

This paper seeks to provide readers with both a theoretical framework and a practical viewpoint regarding the efficacy of GAs as potent instruments for addressing intricate real-world optimization challenges.

1.1 A Systematic Taxonomy of GAs

This paper employs a systematic taxonomy to classify GAs across several dimensions: representation scheme, evolutionary operators, population structure, and hybridization approach. GA variants are categorized into four principal classifications as follows:

1. Classical GAs: The standard model proposed by Holland [2], defined by binary encoding, proportional selection, single-point crossover, and bit-flip mutation. These constitute the basis upon which all future versions are constructed.
2. Representation-Based Variants: GAs customized for certain problem domains via specialized encoding methods, such as real-valued GAs for continuous optimization, permutation-based GAs for combinatorial challenges, and tree-based GAs (Genetic Programming) for program evolution.
3. Operator-Adaptive GAs: Algorithms that dynamically modify genetic operators or parameters during execution, encompassing self-adaptive GAs, feedback-based adaptive GAs, and parameter-less schemes.
4. Hybrid and Memetic Algorithms: Methodologies that amalgamate GAs with alternative optimization techniques, such as local search (memetic algorithms), swarm intelligence (GA-PSO), simulated annealing, and surrogate models.

This taxonomy offers a structured framework for comprehending the interrelations among various GA methodologies and directs the organization of this review. In the subsequent sections, we delineate the emergence of each branch of this taxonomy from the constraints of prior methodologies and how they collectively tackle the issues of contemporary optimization.

1.2 Origins and Evolutionary Development

GAs are based on John Holland and his colleagues' 1960s and 1970s University of Michigan research. Holland's main idea was to abstract natural evolution's selection, crossover, and mutation into computational operators for adaptive search. His 1975 book, "Adaptation in Natural and Artificial Systems," established the theoretical framework, including the Schema Theorem, which demonstrated how GAs automatically process and combine appropriate building elements.

GA evolution has numerous phases. Classical formulation began in the 1970s and 1980s. Holland formalized and Goldberg popularized the canonical GA, which had binary encoding, fitness-proportionate selection, and fixed crossover and mutation rates. Applications began with function optimization and minimal control. This time saw the first systematic parameter investigations, which set population sizing and operator probability for a generation of practitioners.

Specialization and encoding advancements characterized the 1980s–1990s second phase. AS researchers applied GAs to more diverse issues, binary encoding's limits became evident. This led to real-valued encoding for continuous optimization, permutation encoding for sequencing issues, and tree-based encoding for genetic programming. Each representation needed crossover and mutation operator advancements, resulting in problem-specific variations like Partially Mapped Crossover and Order Crossover for permutation issues.

The 1990s–2000s third phase concentrated on adaptive and self-adaptive GAs. The realization that static parameters limit GA performance across issue instances and search phases spurred adaptive control research. Early research on dynamic mutation rates led to sophisticated self-adaptive processes where chromosome properties evolve with solutions. Feedback-based techniques adjusted parameters based on population diversity or convergence indicators.

From the 2000s to the present, hybridization and memetic algorithms dominated the fourth phase. GAs thrive at global exploration but may struggle with local refining, leading to hybrid algorithms. Memetic algorithms enable local search within GA by combining evolutionary search with gradient-based or neighborhood search. Combining swarm intelligence, simulated annealing, and ant colony optimization has created strong hybrid solvers for challenging real-world problems.

From 2010 until the present, the fifth phase covers GA research advances. Modern research tackles high-dimensional, computationally demanding, dynamic optimization problems. Surrogate-assisted GAs approximate fitness functions with machine learning models, lowering computational cost. Modern hardware allows parallel and distributed implementations to scale to intractable issue sizes. Deep learning framework integration has advanced neural architecture search and automated machine learning.

This historical perspective shows a clear path from a single canonical algorithm to a broad family of specialized, adaptive, and hybrid algorithms that address specific shortcomings of their predecessors while maintaining evolutionary search principles.

2 Working Principles of GA

The operation of a GA is primarily rooted in Darwin's theory of survival of the fittest, which posits that more robust individuals within a population possess a greater likelihood of survival and reproduction [10]. In the realm of optimization, GA replicates this notion by progressively refining a population of potential solutions to achieve superior fitness values [11].

The primary elements of GAs consist of the chromosome, gene, population, fitness function, selection, crossover, and mutation [12]. A chromosome is an organized depiction of a potential solution, usually stored as a sequence of variables (genes). A population comprises a set of chromosomes that collectively embody various potential solutions to the optimization problem.

The GA process commences with the creation of an initial population, often generated randomly inside the problem's search space while adhering to boundary constraints. Every chromosome in this population is assessed by a fitness function that quantifies the quality of the solution it embodies. Certain chromosomes are picked as parents for the subsequent generation based on their fitness values. This is accomplished by the utilization of genetic operators [13]:

- Selection emulates natural selection by prioritizing chromosomes with superior fitness values for reproduction.
- Crossover (recombination): merges genetic material from two parental chromosomes to generate children, hence facilitating exploration of the search space.
- Mutation: introduces minor random alterations in chromosomes to sustain variety and avert premature convergence.

The progeny produced via crossover and mutation constitutes a new population that supplants the previous one. This evolutionary cycle of selection, crossing, mutation, and replacement is reiterated over multiple iterations (generations). With each iteration, the population is anticipated to progress toward superior solutions, ideally converging on the most "fit" individuals. The process persists until a specified termination criterion is satisfied, such as attaining a maximum number of generations, achieving an acceptable fitness level, or witnessing population convergence [14].

The fundamental technique of GA can be succinctly outlined as follows [15]:

1. Commence the population using a collection of randomly created chromosomes.
2. Assess the viability of each chromosome within the population.
3. Choose parent chromosomes according to their fitness values.
4. Implement crossover between parental pairs to produce progeny.
5. Implement mutation in progeny based on a specified mutation probability.
6. Establish a new population by substituting the previous one with the progeny.
7. Reiterate stages 2–6 until the termination requirement (e.g., maximum generations or an acceptable solution) is met.
8. Document the optimal chromosome(s) identified as the conclusive solution(s).

2.1 Classical GA Pseudocode and Complexity Analysis

Algorithm 1 presents the classical GA in pseudocode form, followed by variants and computational complexity analysis.

Algorithm 1: Classical GA pseudocode

Input: P → Population size
P_c → Crossover probability
P_m → Mutation probability
$Gmax$ → Maximum number of generations
Output: Best solution found
1: Generate initial population $P(0)$ of size P randomly
2: Evaluate fitness for each chromosome in $P(0)$
3: $t \leftarrow 0$
4: While ($t < Gmax$) and (termination criterion not satisfied) do
5: $t \leftarrow t + 1$
6: Select parent chromosomes from $P(t-1)$ based on fitness
7: Apply crossover with probability P_c to generate offspring
8: Apply mutation with probability P_m to offspring
9: Evaluate fitness of offspring
10: Form new population $P(t)$ using replacement strategy

(Continued)

Algorithm 1 (continued)

11: end while
12: Return the best chromosome found during evolution

Variants of the classical GA alter particular structural elements while maintaining the overarching evolutionary framework. In the steady-state GA, only a portion of people is substituted in each generation instead of reconstituting the complete population, which frequently facilitates more gradual convergence dynamics. The elitist GA implements a preservation mechanism that retains the top (k) individuals unaltered for the subsequent generation, thus protecting superior solutions from disturbance. The CHC (Cross-generational elitist selection, Heterogeneous recombination, and Cataclysmic mutation) version utilizes highly disruptive crossover operators, permitting recombination exclusively between sufficiently dissimilar parents according to a distance criterion, while excluding conventional mutation. The collapse of population diversity triggers a catastrophic reset mechanism that safeguards the optimal individual and reestablishes the surviving population, thus reinstating diversity and averting premature convergence. Conversely, adaptive GAs modify essential parameters—such as P_c and P_m—in real-time according to performance feedback, with the objective of optimizing the balance between exploration and exploitation over time.

2.2 Computational Complexity Analysis

Let P = population size, G = number of generations, and $f(n)$ = complexity of fitness evaluation for an n-dimensional problem. The time complexity of a classical GA is $O(G \times P \times f(n))$, as each generation requires evaluating all individuals. Selection (sorting or tournament) adds $O(P\,log\,P)$ per generation, and crossover/mutation add $O(P \times L)$ where L is chromosome length. For real-valued problems where $f(n)$ dominates, the GA complexity is effectively $O(G \times P \times f(n))$. Memory complexity is $O(P \times L)$ for storing the population. Table 1 provides a comparative examination of the computational time and memory complexity of prominent GA variants, emphasizing their structural distinctions and related trade-offs.

Table 1: Compares the complexity of GA variants.

GA Variant	Time Complexity	Memory Complexity	Key Trade-off
Canonical GA	$O\left(G \times P \times f\left(n\right)\right)$	$O\left(P \times L\right)$	Balanced exploration and exploitation
Steady-State GA	$O\left(G \times P \times f\left(n\right)\right)$	$O\left(P \times L\right)$	Faster convergence, reduced diversity
Island-Model GA	$O\left(\left(G \times P \times f\left(n\right)\right)/I\right)$	$O\left(P \times L\right)$	Parallel speedup, migration overhead
Memetic GA	$O\left(G \times P \times \left(f\left(n\right) + LS\right)\right)$	$O\left(P \times L\right)$	Higher precision, added local search cost
Surrogate-Assisted GA	$O\left(G \times P \times \left(\hat{f}\left(n\right) + f\left(n\right)/k\right)\right)$	$O\left(P \times L + modelcost\right)$	Reduced expensive evaluations, approximation error

Where LS is local search cost, I is number of islands, $\hat{f}(n)$ is surrogate evaluation cost, and k is frequency of high-fidelity evaluation. In addiritn, Table 2 provides a summary of the scaling behavior that is representative of several different GA variations across several rising issue dimensions.

Table 2: Scalability of GA variants with problem dimension.

GA Variant	Asymptotic Complexity	n = 10	n = 30	n = 100	n = 1000	Approximate Scaling Behavior
Canonical GA	$O(G \times P \times f(n))$	1.0×	2.8×	9.5×	95×	$\sim O(n)$
Steady-State GA	$O(G \times P \times f(n))$	1.0×	2.6×	8.7×	88×	$\sim O(n)$ (slightly lower constant)
Island-Model GA	$O((G \times P \times f(n))/I)$	1.0×	2.8×	9.5×	95×	$\sim O(n)$ with I-fold speedup
Memetic GA	$O(G \times P \times (f(n) + LS))$	1.0×	3.2×	14×	180×	$\sim O(n^{1.2})$
Surrogate-Assisted GA	$O\left(G \times P \times \left(\hat{f}(n) + f(n)/k\right)\right)$	1.0×	2.1×	5.2×	28×	$\sim O(n^{0.7})$

Table 2 indicates that for low to intermediate dimensional issues ($n < 100$), both Canonical and Steady-State GAs demonstrate almost linear scaling behavior concerning problem dimension. While the Steady-State GA possesses the same asymptotic complexity as the Canonical GA, it may exhibit somewhat reduced practical computational cost owing to partial population replacement in each generation. Surrogate-assisted versions demonstrate significantly enhanced scalability in high-dimensional contexts, especially when fitness evaluation is computationally intensive, frequently leading to sublinear empirical scaling behavior. Parallel (Island) models maintain the same theoretical complexity as the Canonical GA while achieving significant runtime reductions in relation to the number of islands, rendering them appropriate for extensive optimization endeavors. Conversely, Memetic Algorithms that integrate local search methods typically demonstrate superlinear scaling due to the supplementary refinement costs, making them best suitable when elevated solution accuracy warrants greater CPU investment. As dimensionality increases significantly ($n > 1000$), processing requirements escalate markedly across all variants, underscoring the necessity of surrogate modeling, parallelization, or adaptive population management measures to provide practical scalability.

3 GA Procedure for Optimization Problems

The initial stage in employing a GA to address an optimization problem is to determine the representation of the design parameters for an individual [16]. This representation directly affects the efficacy of the search process, as it dictates the encoding, manipulation, and decoding of solutions into significant variables. In GA language, each solution is denoted as a chromosome, comprised of smaller components known as genes. The expression of genes and chromosomes is dictated by the encoding method [17].

Encoding techniques offer a systematic method for representing issue variables and may manifest in various ways, such as binary strings, arrays, numerical values, symbolic trees, or other specific data structures [18]. The selection of encoding is contingent upon the specific challenge and must be made judiciously to guarantee both representational correctness and operational efficiency in genetic processes (crossover and mutation).

Numerous encoding schemes have been suggested and effectively used for various optimization challenges over the years. The major approaches employed encompass [19]:

1. Binary Encoding: The conventional method in which variables are represented as sequences of binary digits (0 and 1 s). Each chromosome comprises a sequence of bits, facilitating the implementation of crossover and mutation procedures. This method is especially efficacious for discrete optimization challenges.

2. Permutation Encoding: Primarily utilized for ordering and sequencing challenges (e.g., the Traveling Salesman Problem). A chromosome is depicted as a series or permutation of integers, with each position representing a specific activity or city.
3. Value (Real-number) Encoding: Chromosomes are represented as arrays of real numbers rather than binary digits. This encoding is appropriate for continuous optimization problems, as it circumvents the precision challenges linked to binary representation.
4. Tree Encoding: Frequently utilized in Genetic Programming (GP), wherein chromosomes are depicted as trees with functions and terminals. Each tree structure represents a prospective solution program or mathematical expression.
5. Hybrid or Problem-specific Encoding: In certain instances, conventional encodings prove inefficient, prompting the creation of tailored encodings to address unique limitations or attributes of a problem (e.g., graph-based encodings for network design issues).

The selection of the encoding strategy is vital as it influences the representation of the search space and dictates the efficacy of GA operators in producing significant new solutions. An inadequate encoding selection may result in ineffective exploration and early convergence, whereas a judiciously selected encoding can markedly improve GA performance [20].

3.1 Encoding

GAs operate in two spaces: the coding space (genotype) and the solution space (phenotype). The genotype encodes the solution, while the phenotype represents its expressed form, obtained through a mapping process [21,22]. A chromosome is a linear structure composed of genes, which are the smallest units of information. The GA search is conducted in the genotype space, whereas evaluation and selection occur in the corresponding phenotype space (Fig. 1).

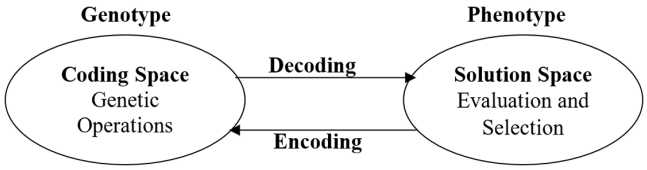

Figure 1: Encoding—Decoding method.

3.1.1 Real-Valued Encoding

In real-valued encoding, a chromosome is represented as a vector of real numbers, where each element directly corresponds to a design variable in the optimization problem [21–23]. This approach is particularly suitable for continuous optimization, as it preserves a direct mapping between variables and chromosome structure.

Unlike binary encoding, real-valued representation eliminates the need for encoding and decoding, reducing computational complexity and preventing precision loss. The chromosome length equals the number of design variables, leading to a compact and efficient representation that often improves convergence behavior.

Real-valued encoding is widely applied in structural design, parameter optimization, machine learning, and control systems, where decision variables are inherently continuous. Fig. 2 illustrates a chromosome represented in real-valued encoding.

Chromosome1	**1.2**	**0.43**	**3.01**	**0.89**	**10.2**
Chromosome2	**Aa**	**De**	**Et**	**Af**	**S**
Chromosome3	**Back**	**Back**	**Right**	**Left**	**Left**

Figure 2: Real-valued encoding.

3.1.2 Binary Encoding

Binary encoding is the predominant encoding method employed in GAs [24]. In binary encoding, each gene inside a chromosome is denoted as a bit (0 or 1), and the chromosome is represented as a sequence of these bits. This representation is suitable for issues where fitness depends on both the value and the inclusion or exclusion of specific elements [25].

In this approach, each gene is represented by a bit (0 or 1), forming a binary string. Binary encoding has numerous benefits, such as straightforward implementation, compatibility with conventional genetic operators (crossover and mutation), and appropriateness for discrete optimization challenges. Nonetheless, it may exhibit reduced efficiency for issues involving continuous variables, as precision is constrained by the length of the binary string. Fig. 3 depicts a chromosome represented by binary encoding.

Chromosome1	**1**	**0**	**1**	**0**	**1**	**0**	**0**	**1**
Chromosome2	**1**	**1**	**0**	**0**	**1**	**0**	**1**	**1**

Figure 3: Binary encoding.

The mathematical formulation of binary encoding enables the representation of continuous or discrete variables as binary strings appropriate for GAs. The binary encoding for the nth variable p_n can be articulated as follows [8]:

Define the variable p_n inside the constrained interval $\left[p_n^{min}, p_n^{max}\right]$. Let the binary string denote p_n possess a length of L_n bits. The value of p_n associated with a binary string can be calculated as:

$$p_n = p_n^{min} + \frac{\text{DecimalValue}\left(b_{n,1}b_{n,2}\ldots b_{n,L}\right)}{2^{L_n}-1} \cdot \left(p_n^{max} - p_n^{min}\right); \tag{1}$$

where, $b_{n,1}b_{n,2}\ldots b_{n,L}$ are the individual bits of the binary string representing p_n, $\text{DecimalValue}\left(\cdot\right)$ transforms the binary string into the equivalent decimal value, L_n the length of the binary string for the nth variable, and p_n^{min} and p_n^{max} are the lower and upper boundaries of the variable pn. This formula guarantees that the binary string precisely corresponds to the variable's feasible range, enabling the GA to investigate solutions inside the problem's domain. Modifying the string length L_n allows for the regulation of representation precision—longer strings produce greater resolution but raise computational complexity.

3.1.3 Permutation Encoding

Permutation encoding is appropriate for ordering, sequencing, routing, or scheduling problems, where the solution is represented as an arrangement of elements. A typical example is the Travelling Salesman Problem (TSP) [22], though this encoding can introduce additional complexity [26].

In the permutation encoding, each chromosome is represented as a sequence (permutation) of numbers, with each number denoting a specific element (e.g., a city in the TSP or a task in a scheduling problem). The defining feature of permutation encoding is the prohibition of repetition within a chromosome, guaranteeing that each element is represented exactly once in the sequence. Fig. 4 depicts a chromosome in permutation encoding.

Chromosome1	**3**	**4**	**2**	**7**	**1**	**5**	**6**	**8**
Chromosome2	**8**	**3**	**6**	**1**	**2**	**7**	**4**	**5**

Figure 4: Permutation encoding.

3.1.4 Tree Encoding

Tree encoding aimed at evolving programs, symbolic expressions, or decision rules instead of mere numerical solutions [27]. In this depiction, each chromosome is organized as a tree, with nodes symbolizing functions or operators, and leaves (terminals) denoting input variables or constants [21,22].

In tree encoding, chromosomes resemble syntax trees used in programming. Internal nodes represent operators (mathematical/logical) or functions (e.g., +, −, ×, ÷, AND, OR, sin, cos), while leaf nodes represent variables, parameters, or constants. For example, in symbolic regression, a tree chromosome may represent the expression x + (y × 3) (Fig. 5).

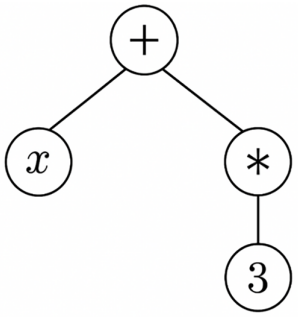

Figure 5: Chromosome representation using tree enoling.

Tree encoding allows exploration of complex solution spaces, with genetic operators applied directly to the tree: crossover exchanges subtrees between parents, and mutation replaces a node or subtree randomly. This makes it ideal for automatic program generation, symbolic regression, and expression evolution, though it can produce large, complex trees (“bloat”) that require additional management.

3.2 Initial Population

The GA commences with a set of chromosomes known as the starting population. Each chromosome consists of genes, which signify potential solutions to the issue [28]. These genes are generally produced randomly under defined boundary constraints to guarantee diversity in the search space. The caliber and variety of this initial population significantly influence the efficacy and convergence of the GA [29]. There are two distinct representation schemes:

- Real-valued representation—in which each gene is denoted by a real number, rendering it especially appropriate for optimization issues involving continuous variables.

- Binary encoding involves representing each gene as a binary digit (0 or 1). This method is frequently employed when the issue variables are discrete or when the objective function relies on combinations of binary states.

By integrating these methodologies, the GA can proficiently navigate both continuous and discrete solution spaces, hence augmenting its ability to address a diverse array of optimization challenges.

3.2.1 Initial Population for Real-Valued Encoding

The starting population in real-valued encoding is created as a collection of chromosomes, each of which is represented by a vector of real values. The optimization problem's design variables are represented by each gene in the chromosome. Every gene is initialized at random within its designated boundaries to guarantee viability.

A chromosome for a problem with n decision variables can be written as follows:

$$X_i = \left[x_{i1}, x_{i2}, x_{i3}, \ldots, x_{in}\right], i = 1, 2, \ldots, P; \tag{2}$$

where:

- X_i = the ith chromosome (candidate solution),
- x_{ij} = the value of the jth decision variable in chromosome i,
- n = number of decision variables,
- P = population size.

Each variable x_{ij} is generated randomly within its predefined range:

$$x_{ij} = x_j^{min} + r \cdot \left(x_j^{max} - x_j^{min}\right), r \in [0, 1]; \tag{3}$$

where x_j^{min} and x_j^{max} are the lower and upper bounds of the variable x_j, and r is a uniformly distributed random number in $[0, 1]$.

The initial population is represented as a matrix of dimensions $P \times n$, with each row denoting a chromosome and each column representing a decision variable.

3.2.2 Initial Population for Binary Encoding

In binary encoding, a chromosome is represented by a sequence of binary digits (0 and 1 s). Each gene within the chromosome is associated with a particular decision variable or a segment of its binary representation. The bit allocation for each variable is contingent upon the precision requirements of the solution and the range of the decision variables.

In a problem involving n decision variables, the ith chromosome is represented as follows:

$$X_i = \left[b_{i1}, b_{i2}, b_{i3}, \ldots, b_{im}\right], i = 1, 2, \ldots, P; \tag{4}$$

where:

- X_i = the ith chromosome,
- $b_{ij} \in \{0, 1\}$ = the jth bit of the chromosome,
- m = total chromosome length (sum of bits assigned to all variables),
- P = population size.

The association between a binary string and its respective real value is defined as follows:

$$x_j = x_j^{min} + \frac{D_j}{2^{L_j} - 1} \cdot \left(x_j^{max} - x_j^{min} \right); \tag{5}$$

where x_j^{min} and x_j^{max} are the lower and upper bounds of the variable x_j, L_j number of bits assigned to the variable x_j, and D_j decimal value of the binary substring representing x_j.

3.3 Fitness Evaluation

The initial step following the creation of the starting population is to compute the fitness value of each chromosome. The fitness function quantifies the extent to which an individual solution meets the optimization objective. Chromosomes exhibiting elevated fitness values signify more optimal solutions and are more probable to be chosen for reproduction in the subsequent generation.

In real-valued encoding, fitness evaluation is direct: each chromosome (a vector of real values) is inserted into the objective function to calculate its fitness value.

The evaluation method for alternative encoding schemes, including binary, permutation, or tree encoding [21], comprises two primary steps:

1. Analyzing the chromosome: The chromosome is initially converted into a set of significant decision variables.
2. Evaluating the decoded variables: After decoding the chromosome, the resultant values of the choice variables are inserted into the objective function.

The calculated objective function value is thereafter designated as the chromosome's fitness value or adjusted accordingly if the situation necessitates scaling for maximizing or minimizing.

During this procedure, each chromosome within the population is allocated a fitness value. Chromosomes exhibiting superior fitness are afforded an increased likelihood of selection for subsequent evolutionary processes, including selection, crossover, and mutation.

3.4 Create a New Population

After evaluation, a new population of chromosomes is obtained by applying genetic search operators one after another. The genetic search operators are selection, mutation, and crossover. The expected quality of a new population over all the chromosomes is better than that of the current population.

3.4.1 Selection

Selection is a crucial operator in GAs, determined by an evaluative criterion [30]. The most ideal chromosomes are picked for breeding and hybridization to generate a new generation that is anticipated to be better. During the selection phase, individuals are extracted from the sequence of chromosomes. The selection approach is the same for both real-valued representation and representation of encoding techniques. The selection is addressed with fitness that already comprises real values. The prevalent methods for chromosomal selection include roulette wheel, stochastic universal sampling, rank selection, tournament selection, and steady state selection [31,32].

(A) Roulette Wheel Selection

The parents' choice in the roulette wheel method is based on their fitness worth. Here we have the most popular approach to fitness-proportionate selection. Every chromosome has a certain area on the imaginary roulette wheel that is proportionate to its fitness value. Consequently, the fittest individuals have a better shot at selection and take up more space on the wheel. However, the contribution to maintaining population diversity occurs through the fact that less fit chromosomes can be selected in this method [31].

The selection process of the roulette wheel can be articulated as follows:

Step 1. The fitness values f_i for every individual $i$in the population are calculated.

Step 2. Calculate, for every individual, the selection probability p_i by dividing the individual chromosome's fitness f_i by the sum of fitness values $F = \sum_{i=1}^{P} (f_i)$ of the whole population P as:

$$p_i = \frac{f_i}{F}, \quad i = 1, 2, \ldots, P. \tag{6}$$

Step 3. Construct a cumulative probability distribution as $q_i = \sum_{j=1}^{i} (p_j) \; \forall \, i = 1, 2, \ldots, P$.

Step 4. Spin the roulette wheel: Generate a random number $r \in [0, 1]$.

Step 5. Select a chromosome: Select chromosome i such that $q_{i-1} < r < q_i$.

Step 6. Repeat: Continue the process until the desired number of parents is selected.

Fig. 6 depicts a roulette wheel representing five individuals with varying fitness levels. Each individual is allocated a sector (slice) of the wheel, with the dimensions of each sector being proportional to its fitness level. The likelihood of the pointer halting on a specific individual while the wheel is spun is contingent upon the dimensions of its segment.

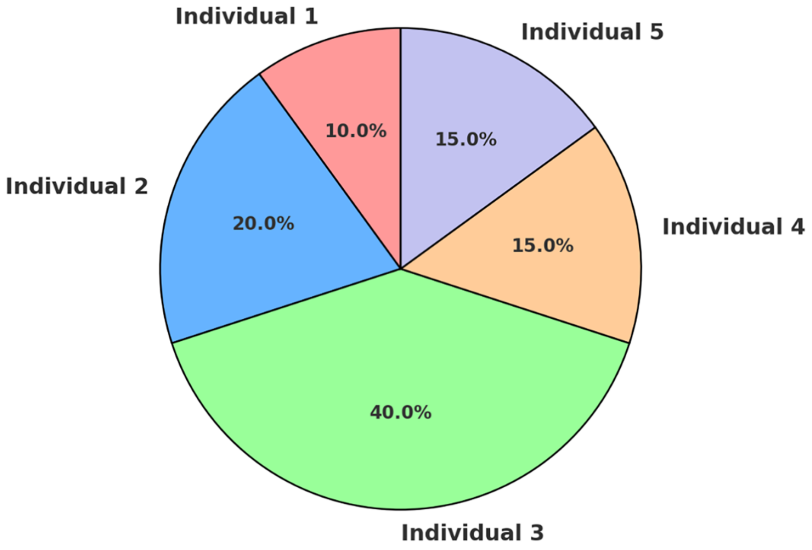

Figure 6: Roulette wheel selection.

The graphic clearly indicates that Individual 3 occupies the most substantial segment of the wheel. Consequently, it possesses a higher likelihood of being chosen relative to the other persons. This illustrates the principle of fitness-proportionate selection: the greater the fitness of an individual, the higher the probability of its selection as a parent for the subsequent generation.

(B) Stochastic Universal Sampling (SUS) Selection

Stochastic Universal Sampling (SUS) was proposed by Baker (1987) [32] as an enhancement to the roulette wheel selection technique. Like roulette wheel selection, SUS allocates slots to individuals in accordance with their fitness values. Instead of using a single pointer to choose individuals, as in roulette wheel selection, SUS employs N equally spaced pointers, with N representing the desired number of selections, often corresponding to the population size.

The process operates as outlined below:

- The overall fitness of the population is computed, and individuals are allocated proportional positions on a line, like a roulette wheel.
- A random number r is produced within the interval $[0, \frac{1}{N}]$. This value establishes the location of the initial pointer.
- The remaining $N - 1$ pointers are positioned at uniform intervals of $\frac{1}{N}$ along the line.
- All candidates who meet these criteria are selected concurrently.

This technique guarantees a more equitable distribution of selection pressure throughout the population. In contrast to roulette wheel selection, which may exhibit stochastic fluctuations leading to bias towards specific individuals, Stochastic Universal Sampling (SUS) offers a more reliable and equitable depiction of selection possibilities.

Benefits of SUS Selection:

- Ensures a selection result that aligns more closely with the anticipated proportion.
- Minimizes stochastic mistakes associated with roulette wheel selection.
- Guarantees equitable and uniform sampling throughout the population.
- Facilitates the choosing of multiple options with a single rotation of the wheel.

Consequently, SUS selection enhances the dependability of fitness-proportionate selection while maintaining population variety [31,32]. Fig. 7 illustrates Stochastic Universal Sampling (SUS) Selection, wherein the line is segmented into proportionate slots for five individuals, accompanied by evenly spaced red indicators demonstrating the process of several selections in a single rotation.

Figure 7: Stochastic Universal Sampling (SUS) selection.

(C) Rank selection method

Roulette wheel selection may become ineffective when the fitness values of chromosomes exhibit significant variation. In these instances, the individual exhibiting the best fitness typically prevails in the selection process, whereas other individuals possess minimal likelihood of getting selected. This results in the algorithm's premature convergence.

The rank selection approach resolves this issue by allocating probability according to the rank of individuals instead of their absolute fitness ratings. The protocol is as follows:

The population is initially organized based on fitness values.

- Each individual is allocated a rank, with rank N designated for the best chromosome and rank 1 for the worst.
- The chance of selection for each chromosome is assigned in proportion to its rank instead of its absolute fitness.
- Individuals are chosen based on these probabilities, guaranteeing that even less robust individuals maintain a reasonable opportunity for selection.

This approach inhibits quick convergence to a suboptimal solution and preserves variation throughout the population. While convergence may occur at a slower pace, the likelihood of premature stagnation is diminished.

Depiction:

- Fig. 8A: Roulette Wheel Selection—When the optimal individual possesses 80% of the fitness share, it occupies 80% of the wheel, significantly diminishing the probability of selection for other individuals.
- Fig. 8B: Rank Selection—The top individual is allocated a selection probability commensurate with its rank (e.g., 45%), whereas the remaining individuals are granted more equitable probabilities.

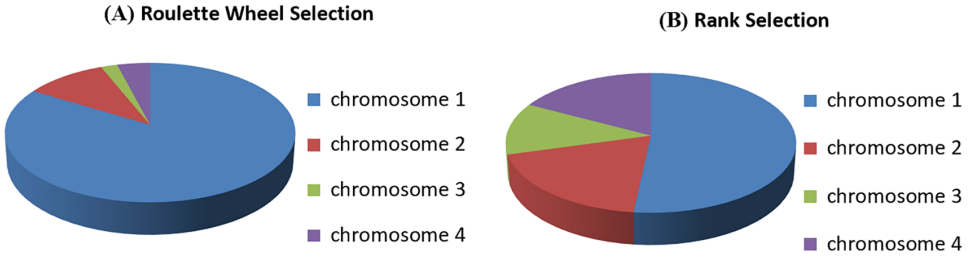

Figure 8: Rank selection.

Consequently, rank selection allocates selection probabilities more uniformly throughout the population, mitigating bias towards a singularly fit individual and promoting exploration of the solution space [32].

(D) Tournament Selection

Tournament selection was proposed by Goldberg (1989) [7] as a straightforward yet efficient selection technique in GAs. The procedure commences with the random selection of a subset of t people from the population. This cohort of t individuals constitutes a tournament, wherein the individual exhibiting the best fitness is proclaimed the victor and designated as a parent. The procedure can be encapsulated as follows:

- Select t individuals randomly from the population.
- Evaluate their fitness values.
- Choose the individual exhibiting the highest fitness to serve as a parent.
- Continue the procedure until the requisite number of parents is selected.

The parameter t, referred to as the tournament size, regulates the selection pressure:

- An increased tournament size (t) enhances the likelihood of selecting superior individuals, resulting in expedited convergence but less variety.
- A reduced tournament size preserves greater diversity but hinders convergence.

Consequently, the tournament size establishes an equilibrium during the search process between exploration and exploitation.

Nonetheless, tournament selection can become computationally intensive when the population size is substantial, as numerous tournaments are required to choose an adequate number of parents for reproduction [32].

Fig. 9 illustrates the Tournament Selection Example; the blue boxes represent individuals along with their fitness values, the red lines delineate the tournament participants, and the gold box indicates the winner (the individual with the highest fitness among the selected).

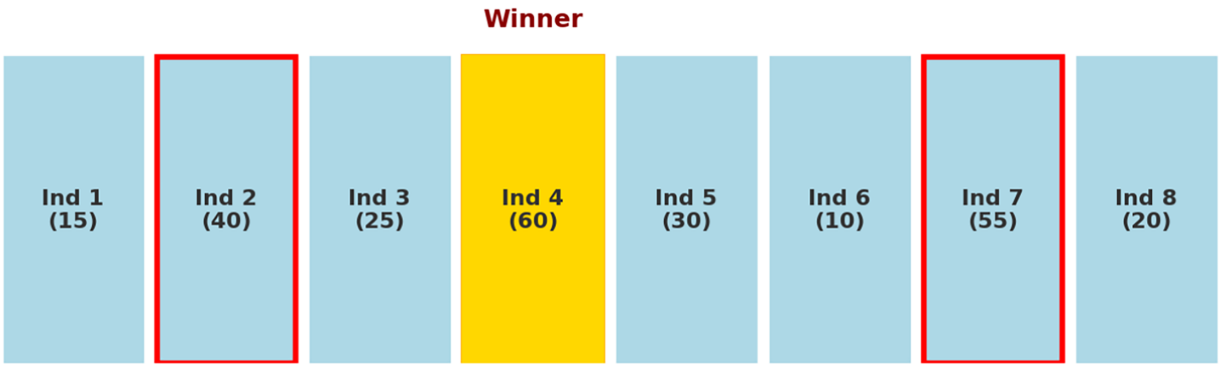

Figure 9: Tournament selection example.

(E) Steady-state selection

Steady-state selection is a GA replacement approach in which only a limited number of people are substituted for each generation, rather than creating an entirely new population. Generally, A limited number of parents (often two) are chosen from the population. Then the crossover and mutation result in the production of offspring. Finally, the least favorable members in the population are supplanted by the new progeny.

This methodology guarantees that the majority of the population persists into the subsequent generation, effective solutions endure for an extended duration, and diversity is preserved, averting early convergence [31–33].

3.4.2 Crossover

Crossover is crucial genetic operators. It combines two chromosomes to produce children with those characteristics. Segment exchange between parent strings generates new strings. Starting crossover requires selecting a random cut point. The offspring is produced by combining segments from one parent to the left of the cut-point and the other to the right. Choosing the right crossover is critical to fixing an issue [34]. This section discusses crossovers between different encoding systems and optimization difficulties.

(A) Single Point Crossover

Single-point crossover is a prominent GA crossover method. This method randomly selects a chromosomal cut spot. Segments from two parents create offspring. The first set of offspring uses the segment to the left of the cut-point from one parent and the segment to the right from the other. The next set reverses this technique.

This strategy preserves genetic variety by giving all offspring a mix of genes from both parents. Cut point position greatly affects operator efficacy. A good placement can yield fitter offspring, while a bad cut-point can reduce solution quality. Despite this limitation, single-point crossover is a simple and effective

technique for genetic variety [35]. Figs. 10–13 show single-point crossover in real-valued, binary, valued, and tree encodings, respectively.

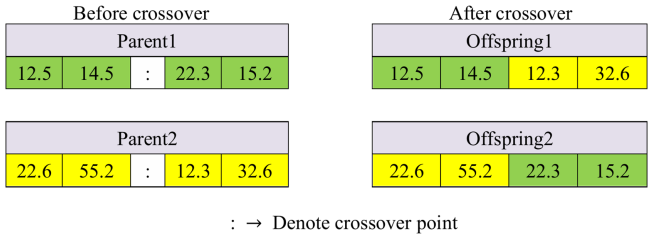

Figure 10: Single-point crossover, real-valued encoding.

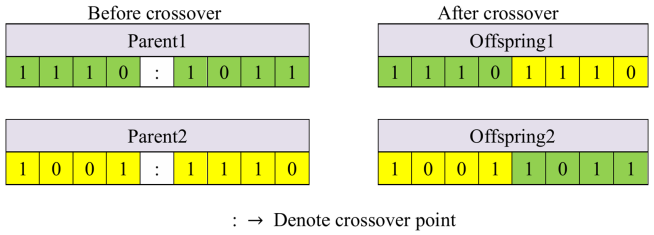

Figure 11: Single-point crossover, binary encoding.

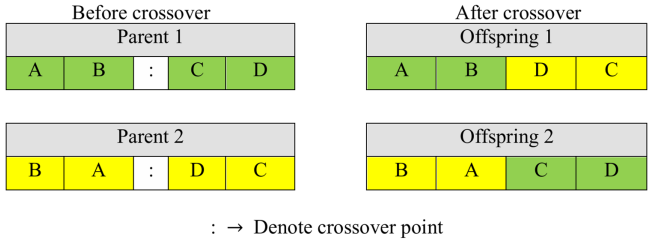

Figure 12: Single-point crossover, value encoding.

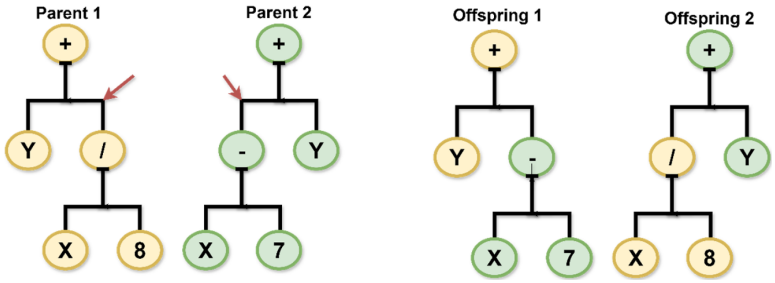

Figure 13: Single-point crossover, tree encoding crossover.

(B) N-Point Crossover

De Jong's 1975 N-point crossover expands the GAs' single-point crossover technique [36]. This approach generates children by picking numerous cut locations along parent chromosomes and exchanging segments

between them. Despite numerous cutting points, it follows single-point crossover: mixing genetic material from two parents to create new solutions.

Both parent chromosomes exchange segments between two specified loci in the two-point crossover. Increased cut points divide the chromosome into segments, which may disrupt well-adapted building blocks, subsets of genes that improve solution quality. Adding cut points may increase population diversity, but it may also destroy key building blocks, lowering algorithm performance. In N-point crossover, exploration (diversity) and exploitation (effective genetic structures) are trade-offs. Depending on the problem, choose the number of cut points carefully. Figs. 14 and 15 show real-valued and binary two-point crossover, respectively.

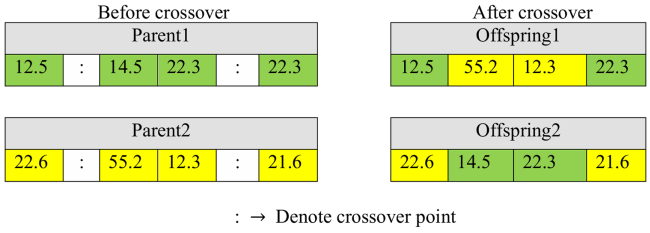

Figure 14: Two-point crossover, real-valued encoding.

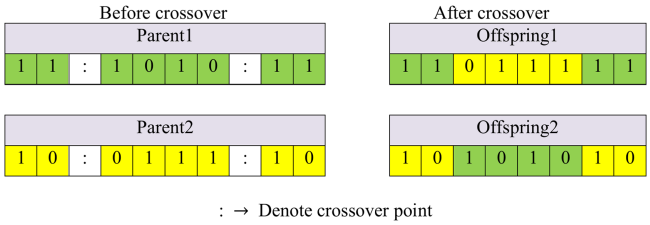

Figure 15: Two-point crossover, binary encoding.

(C) Uniform Crossover

Uniform crossover, a GA recombination method, does not segment parent chromosomes. Use a binary crossover mask the same length as the parent chromosomes. Every parent pair receives a random mask.

The crossover mask's matching bit selects each offspring gene from one of the parents during uniform crossover. Choose the gene from the first parent if the mask bit is 1 and from the second parent if it is 0. Thus, the offspring inherits genes from both parents, boosting genetic diversity while retaining their advantages. Figs. 16 and 17 show uniform crossover in real-valued and binary encoding, respectively [37].

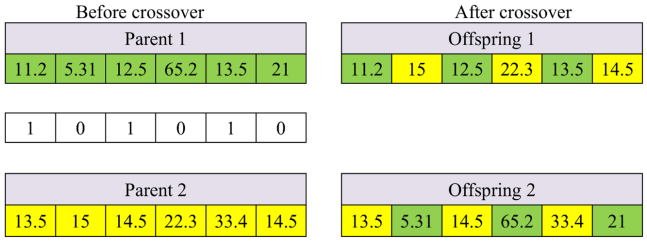

Figure 16: Uniform crossover, real-valued encoding.

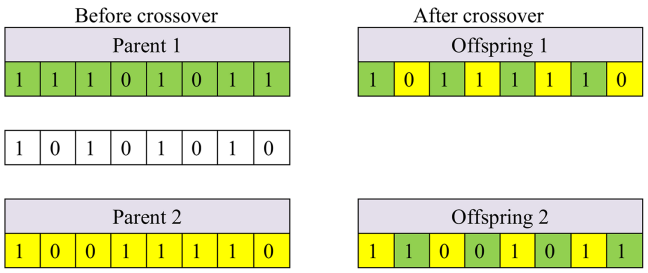

Figure 17: Uniform crossover, binary encoding.

(D) Three Parents Crossover

Randomly selecting three parent chromosomes from the population starts the three-parent crossover. If both parents have the same gene at a gene site, the baby inherits it. If the first two parents' genes differ, the offspring will inherit the third parent's gene at that place.

This crossover strategy works well for binary-encoded chromosomes because it keeps the first two parents' traits while adding the third parent's. This method may increase offspring variety and GA search efficiency by integrating data from three sources.

(E) Arithmetic

Arithmetic crossover is a real-valued (continuous) recombination operator. This approach produces offspring by linearly combining two parent chromosomes. The offspring gene for each chromosome gene is a weighted average of the parents' genes.

To calculate the offspring gene y, use the formula: $y = \alpha x_1 + (1 - \alpha)\, x_2$, where α is a random value between 0 and 1 indicating the contribution of each parent. Each chromosomal gene undergoes this procedure separately. Arithmetic crossover merges genetic material to create kids with features from both parents while conserving values within parental genes. Continuous search space optimization issues benefit from this method, which simplifies solution space research. This arithmetic crossover uses AND/OR operators (Fig. 18).

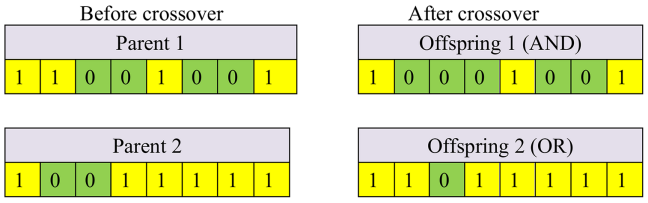

Figure 18: Arithmetic crossover, binary encoding.

(F) Partially mapped Crossover

Partially Mapped Crossover (PMX) is a popular crossover operator for permutation encoding, especially in the Travelling Salesman Problem. Goldberg and Lingle introduced PMX [38]. This crossover approach maintains gene positions on parental chromosomes while ensuring offspring permutations.

PMX selects two parent chromosomes and randomly assigns two crossing spots. One progenitor passes on the genes between these two loci to the progeny. To preserve permutation structure and avoid gene duplication, position-by-position mapping populates the remaining genes. By mapping each gene precisely once in the progeny, PMX is ideal for combinatorial optimization problems where element sequence is critical. Fig. 19 shows a permutation-encoded partial mapped crossover.

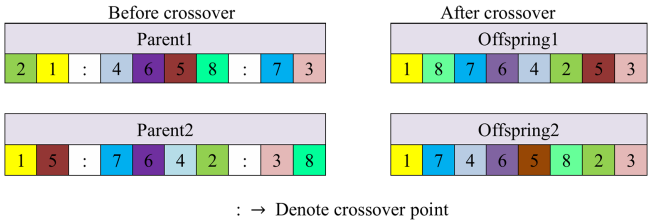

Figure 19: Partially mapped crossover, permutation encoding.

(G) Order Crossover (OX)

Order Crossover (OX), described by Davis [39], is typical for permutation-encoding chromosomes. Instead of fixing gene sites, OX maintains gene relative order.

Choose two crossover spots on the parent chromosomes to start OX. The offspring inherit the genes between these two locations directly from the primary parent. Deriving the remaining genes from the second parent preserves their relative order and excludes genes from the copied section. This "slide" method ensures that the offspring retains maximal secondary parent ordering information and maintains a legal gene permutation.

Combinatorial optimization problems like the traveling salesman problem, where element sequencing is crucial for superior solutions, use Order Crossover extensively. Fig. 20 shows Order Crossover.

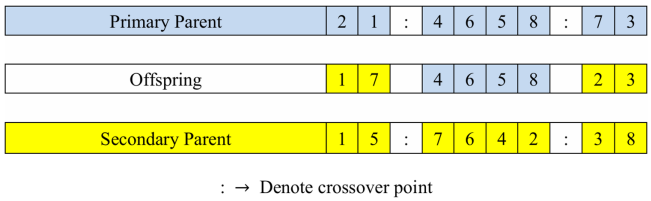

Figure 20: Illustration of the order crossover (OX).

(H) Cycle Crossover (CX)

Cycle Crossover (CX) is a permutation-encoding recombination operator. The basic idea is that each gene in the offspring must come from one parent, ensuring true permutations.

CX is about recognizing positional cycles between parents. The procedure begins with Parent 1's first gene. We then look for the gene at the Parent 2 locus and trace it back to Parent 1. Continue alternating between parents until the cycle returns to Parent 1's gene. The genes collected this way form a cycle.

Parent 1 replicates all cycle components, whereas Parent 2 fills the remaining spots to create the first offspring. In the second offspring, Parent 2 derives the cycle elements while Parent 1 populates the remaining slots. Thus, CX produces two offspring that follow the parents' cycle. Fig. 21 shows Cycle Crossover.

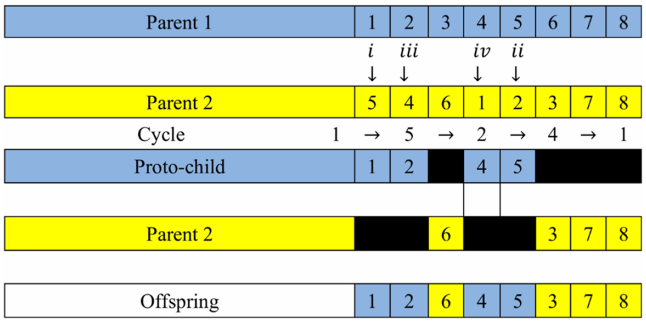

Figure 21: Illustration of the Cycle Crossover (CX).

3.4.3 Mutation

Premature convergence is the most significant challenge that the majority of optimization approaches face. When highly suited parent chromosomes in a population create many offspring that are identical to each other during the early stages of evolution, this phenomenon is known as premature convergence [40,41]. It is not likely that the crossover action of GA will produce children that are significantly different from their parents in this scenario. Mutation is a frequent operator that seeks to locate new points in the search space to examine in order to assist in the preservation of diversity within the population. In the process of selecting a chromosome for mutation, the values of certain sites within the chromosome are altered in a completely random manner. The purpose of this article is to discuss several sorts of mutations and to propose some of them for real-valued representation and encoding approaches [42].

(A) Insert Mutation

Insert mutation is a frequently employed mutation operator in permutation encoding, particularly in combinatorial optimization challenges like the Travelling Salesman Problem. The process operates in the following manner:

- Select two gene loci randomly from the chromosome.
- Eliminate the gene at the second designated location.
- Position this gene directly after the initial selected gene, adjusting the intermediary elements accordingly to facilitate the modification.

This mutation operator is advantageous as it maintains most of the original order and adjacent information within the chromosome, while simultaneously introducing diversity into the population [43]. By marginally modifying the sequence, insert mutation investigates novel permutations without significantly compromising the integrity of effective solutions. Fig. 22 illustrates the Insert Mutation.

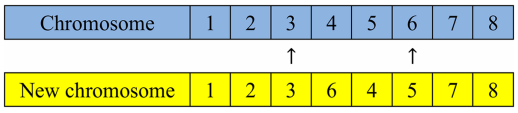

Figure 22: Illustration of the insert mutation.

(B) Inversion Mutation

Inversion Mutation is a mutation operator used in GAs, particularly when using permutation encoding (as in the Traveling Salesman Problem) [44]. The method goes as follows:

- Select two chromosomal locations (genes) at random.

- Select the substring between these two locations.
- Reverse the order of genes in the substring.
- Insert the inverted substring back into the chromosome.

This operator has the advantage of assisting in the maintenance of building blocks (subsequences) of good solutions, as well as introducing diversity without significantly disturbing the chromosome structure. However, it can greatly disturb order information and may not be effective if the problem is not order-sensitive. Fig. 23 illustrates the Insert Mutation.

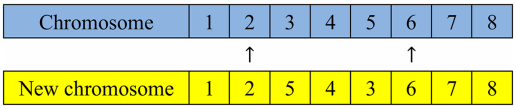

Figure 23: Illustration of the inversion mutation.

(C) Scramble Mutation

Scramble Mutation is another mutation operator used in GAs that encodes permutations (such as routing or sequencing problems) [6]. The method is as follows:

- Select two chromosomal locations at random.
- Select the substring between those two locations.
- Randomly shuffle (scramble) the genes inside the substring.
- Insert the scrambled substring back into the chromosome.

This operator has the virtue of providing high diversity by disturbing orders within a subsequence, and it is useful for avoiding local optima. However, it can be excessively destructive, destroying useful building blocks, necessitating a compromise with less disruptive mutations such as inversion. Fig. 24 illustrates the Scramble Mutation.

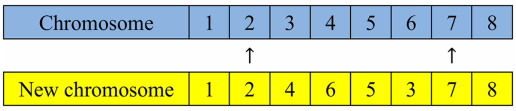

Figure 24: Illustration of the scramble mutation.

(D) Swap Mutation

Swap Mutation is among the most elementary mutation operators in GAs, particularly in the context of permutation encoding, as the Traveling Salesman Problem (TSP) [6]. The procedure is as follows:

- Randomly choose two genes from the chromosome.
- Exchange their locations.
- Preserve the remainder of the chromosome unaltered.

This operator possesses numerous advantages, including ease of implementation, Low disruption, and preservation of much of the original structure. However, it offers restricted exploration of the search space and may require integration with additional mutation operators (such as inversion or scrambling) to enhance diversity. Fig. 25 illustrates a swap mutation with permutation encoding, while Fig. 26 illustrates a swap mutation with tree encoding.

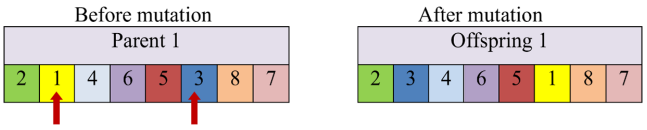

Figure 25: Swap mutation, permutation encoding.

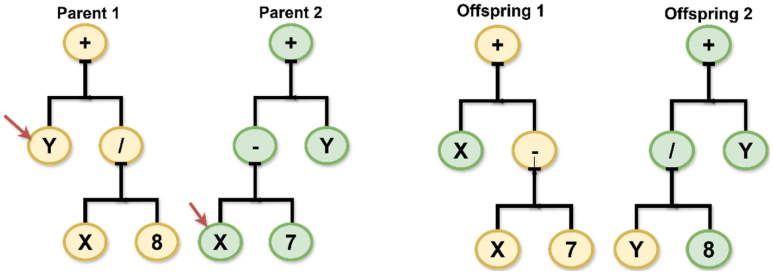

Figure 26: Swap mutation, tree encoding mutation.

(E) Flip Mutation

Flip mutation is a mutation operator frequently employed in binary-encoded chromosomes inside GAs. The objective is to enhance genetic variety by modifying specific segments inside a parental chromosome [42]. The protocol is as follows:

- Selection of Parent Chromosome: A parent chromosome is chosen from the population. It is denoted as a sequence of 0 and 1 s.
- Mutation Chromosome Generation: A mutation chromosome is generated randomly, with each bit signifying whether the matching bit in the parent will be inverted.
- Bit Flipping: Each bit designated as 1 in the mutation chromosome results in the inversion of the corresponding parent bit ($0 \rightarrow 1$ or $1 \rightarrow 0$).
- Outcome: The offspring's chromosome is generated, exhibiting minor deviations from the progenitor to preserve genetic variety within the community.

Fig. 27 depicts a flip mutation utilizing binary encoding.

Parent Chromosome	1	1	0	0	1	0	0	1
Mutation Chromosome	0	0	1	0	1	0	1	0
offspring's chromosome	1	1	1	0	0	0	1	1

Figure 27: Flip mutation, binary encoding.

(F) Interchanging Mutation

Interchanging mutation is a mutation operator frequently employed in permutation encoding, when the sequence is significant, as in scheduling or routing challenges. This procedure involves selecting two random sites inside the chromosome and interchanging the genes at those positions to generate a new child [6].

This operator offers several advantages, including ease of implementation, preservation of solution feasibility in permutation-based problems, and the introduction of minor variants while maintaining the integrity of the majority of the chromosome. Fig. 28 illustrates the Interchanging mutation in real-valued encoding.

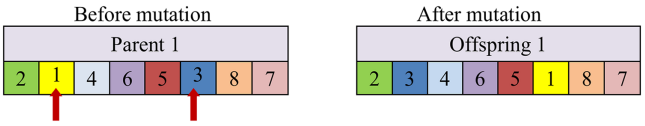

Figure 28: Interchanging mutation, real-valued encoding.

(G) Uniform Mutation

GAs that work with real or integer values make use of the uniform mutation operator. This operator substitutes a new randomly determined value within the gene's permitted range (lower and upper boundaries) for the original value, as opposed to binary encoding (bit flipping) or permutation encoding (position swapping).

One of the many benefits of this operator is that it prevents premature convergence by substituting new values for old ones. Another advantage is that it is easy to construct and is useful for exploring new parts of the search space [42].

(H) Creep Mutation

Creep mutation is a mutation operator employed in real-valued GAs. Creep mutation differs from uniform mutation by making a minor change (either an increment or a decrement) to the value of a chosen gene, rather than replacing it with a completely random value from the range. This gradual transition facilitates a more regulated investigation of the search space.

This operator offers several advantages, including a balance between exploration and exploitation. More effective in refining solutions than uniform mutation and diminishes the likelihood of significant disruptive alterations [6].

3.5 Repair

The primary concept of the chromosome-repairing scheme involves converting infeasible solutions into feasible ones using targeted repairing strategies [45]. Any chromosome that denotes an infeasible solution is termed an infeasible chromosome. Repairing a chromosome involves transforming an infeasible chromosome into a feasible one through the application of an appropriate repair procedure. The selected strategy frequently relies on the presence of a deterministic repair mechanism. Numerous repair methods have been examined, with the following three being the most notable [46]:

(A) Lamarckian Approach

The Lamarckian approach involves the genetic modification of an infeasible solution to render it feasible. Upon repair, the viable chromosome substitutes the infeasible one within the population and remains engaged in subsequent evolutionary processes. This method is straightforward and guarantees the retention of only viable solutions within the evolutionary framework [47].

(B) Baldwinian Approach

The Baldwinian approach represents a less destructive methodology that integrates learning and evolution. This method involves repairing infeasible solutions solely during evaluation, without making permanent alterations to the population. The repaired version is utilized to compute fitness, whereas the original chromosome remains unaltered. Empirical and analytical studies demonstrate that this approach decreases the convergence speed of the evolutionary algorithm while enhancing the likelihood of achieving the global optimum through the preservation of diversity [48].

(C) Annealing Approach

The repair strategy based on annealing is derived from the principles of simulated annealing. This method temporarily accepts infeasible solutions with a specified probability. In the initial phases of the algorithm, the likelihood of accepting infeasible solutions is comparatively elevated, but it diminishes gradually over time. An infeasible solution, if rejected, is rectified through a specified procedure. This probabilistic mechanism facilitates exploration within the search space before progressively concentrating on feasibility and optimization.

In each of the three approaches, repairing a solution generally entails exchanging the elements that contravene the problem constraints [49].

3.6 Elitism and Population Update (Migration)

During this step, the newly produced offspring are included in the current population, and a new population is formed by choosing the people who are the most physically fit from both the parent group and the offspring group. This mechanism ensures the retention of optimal solutions during the evolutionary process [50].

To improve performance, the principle of elitism is utilized, whereby the top chromosomes of the current generation are retained unchanged for the subsequent generation. The algorithm ensures that the quality of solutions remains stable across generations by reserving elite individuals while still promoting diversity and exploration within the rest of the population.

3.7 Termination Assessment

The termination condition delineates the cessation criterion of the GA. The algorithm concludes when one of the subsequent requirements is met:

- The algorithm terminates upon reaching a predetermined number of generations, irrespective of the discovery of the best solution.
- Population convergence refers to the phenomenon wherein all individuals within a population become identical or almost indistinguishable. At this juncture, crossover procedures have no additional impact, as they are incapable of generating new variations within the population.

In the absence of both circumstances, the algorithm will proceed to build a new population through the processes of selection, crossover, mutation, and replacement until the termination criteria are fulfilled.

4 GA Parameters

The fundamental variables in GA are the probability of crossover, mutation, and population size [8].

4.1 Crossover Probability (P_c)

The ratio of the number of offspring produced by the crossover operator in each generation, expressed as a percentage of the overall population size, is the definition of the crossover probability. P_c is the symbol that is used to indicate the crossing probability. When P_c is 100%, then every single child is the consequence of a crossover. If it is zero percent, a new generation is formed entirely from identical copies of chromosomes from the population that came before it.

The process of crossover is carried out with the assumption that the chromosomes that are produced will have characteristics that are superior to those of the population that is already there. There is a significant relationship between the magnitude of the crossover probability and the efficiency of GAs. A larger crossover

probability makes it easier to explore a wider universe of potential solutions and reduces the possibility of arriving at a solution that is less than ideal. On the other hand, when the crossover probability is very high, a significant amount of computing effort is wasted on portions of the solution space that do not show any signs of being promising [21].

4.2 Mutation Probability (P_m)

In the absence of mutation, offspring are produced by the population that is generated by crossover, and they do not undergo any changes. These chromosomes change because of the mutation process. The mutation probability is defined as the ratio of the number of offspring produced by the mutation operator in each generation to the total number of offspring in the population. This ratio is calculated by every generation. P_m is the symbol that corresponds to the mutation probability. If P_m is 100%, then every chromosome will be altered; however, if it is 0%, then there will be no chromosomal changes.

To avoid GAs from convergently reaching a local optimal solution, mutation is an extremely important factor. There is a correlation between the value of mutation probability and the efficacy of GAs. Excessively low Pm levels result in the inactivation of many advantageous genes. In contrast, if the value is very high, the offspring may have a lessened similarity to the parents, the algorithm will be unable to learn from the search history, and there will be a substantial amount of random disturbance [22].

4.3 Population Size

The population size is a basic parameter in GA. It ascertains the quantity of chromosomes (solutions) in each generation.

A requisite minimum of chromosomes is essential for the algorithm to successfully traverse the search space and converge on the ideal solution. A diminutive population size restricts the GA's crossover opportunities, resulting in less exploration of the search space and an increased likelihood of premature convergence.

Conversely, an excessively large population size substantially escalates the computational expenses. Research indicates that above a specific threshold—primarily determined by the encoding method and the problem's characteristics—utilizing excessively large populations does not enhance performance or expedite convergence relative to a moderately sized population [51].

Consequently, the selection of population size must reconcile diversity (to investigate the search space) and efficiency (to minimize computational burden).

4.4 Parameter Sensitivity Analysis

The main parameters of a GA have a significant impact on its performance. The crossover probability (P_c) influences the recombination of solutions; a low P_c (<0.4) hinders development because of insufficient investigation, but a very high P_c (>0.95) might upset well-designed solution structures. An intermediate range ($P_c = 0.7 - 0.9$) allows for effective search by striking a balance between exploration and exploitation.

Careful monitoring of the mutation probability (P_m) is also necessary. While high Pm (>0.1) promotes random behavior that prevents convergence, very low P_m ($<1/L$, where L is the chromosome length) lowers variety and may cause premature convergence. Keeping mutations within the ideal range ($P_m = 1/L$ to $2/L$) protects variety without compromising viable solutions.

To balance computational cost and diversity, population size (P) is crucial. While very large populations ($P > 100n$) result in decreasing returns and increased computation, small populations ($P < 20$ for low-dimensional issues, $P < 10n$ for high-dimensional problems) run the danger of premature convergence. Choosing a size within the ideal range ($P \approx 10n - 50n$) guarantees a successful search at a fair cost.

There are intricate interactions between these parameters. Adjusting crossover, mutation, and population size based on the kind of problem—for example, lowering PC with greater PM for multimodal problems or raising P_c with lower P_m for unimodal problems—can greatly enhance performance. Moreover, adaptive parameter control techniques that dynamically modify these values in response to feedback improve the efficacy and efficiency of algorithms.

5 Applications for GAs

5.1 Transition from Theory to Practice

The prior sections have delineated the theoretical underpinnings of the proposed framework, outlining its mathematical architecture, algorithmic processes, convergence attributes, and computational features. These analyses elucidate how fundamental principles regulate search behavior, the balance between exploration and exploitation, parameter sensitivity, and solution quality. The theoretical insights elucidate the strengths and limitations of the strategy across various problem landscapes, including limited, nonlinear, and high-dimensional optimization scenarios.

The discourse on model formulation and operator design illustrates the role of structural components in enhancing robustness and adaptability. The convergence study offers a conceptual insight into stability and performance patterns, whilst complexity concerns delineate the computing viability of large-scale implementations. These theoretical findings not only validate the methodological selections but also set expectations about efficiency, scalability, and reliability.

This section shifts from theoretical concepts to practical applications by analyzing the manifestation of these features in real-world scenarios. The practical applications support the theoretical assertions, demonstrate the algorithm's adaptability across several domains, and furnish empirical evidence of its efficacy. The study transitions from conceptual formulation to applied performance evaluation in a systematic and logically related fashion.

5.2 Mathematical and Function Optimization

GAs have been extensively utilized in mathematical and function optimization, especially for nonlinear, multimodal, and discontinuous problems where gradient-based approaches frequently prove ineffective. By representing variables in binary or real formats, GAs can efficiently navigate extensive search spaces and circumvent entrapment in local minima. They are particularly adept at benchmark functions like the Rosenbrock and Ackley functions, in addition to practical engineering challenges such as aerodynamic form optimization [52–55].

5.3 Combinatorial Optimization

In combinatorial optimization, GAs have exhibited robust efficacy in managing extensive and intricate discrete solution spaces. Prominent applications encompass the Travelling Salesman Problem (TSP), the Vehicle Routing Problem (VRP), the Knapsack Problem, and Job-Shop Scheduling [7]. Utilizing permutation encoding and problem-specific operators, GAs maintain solution feasibility while effectively seeking high-quality solutions [6].

5.4 Engineering Design Optimization

Engineering design optimization constitutes a significant area of GA applications. In structural and civil engineering, GAs have been employed for the optimization of size and topology in trusses and beams [56]. Aerospace engineering utilizes them for aerodynamic wing and aircraft trajectory design [57], whereas

electrical and electronic engineering applies them in VLSI circuit layout and antenna design [58]. Their capacity to address intricate, nonlinear, and simulation-based issues renders them formidable instruments, despite the persistent challenge of computing expense.

5.5 Control Systems Optimization

GAs are extensively utilized in the optimization of control systems for the calibration of controllers, including PID, adaptive, and fuzzy controllers. They have been employed to optimize parameters for multivariable systems, robotic trajectory planning, and load frequency regulation in power systems [25]. In contrast to traditional tuning approaches, GAs do not necessitate gradient information, rendering them particularly efficient for nonlinear or uncertain systems. Nevertheless, fitness evaluation is resource-intensive due to its reliance on system simulations.

5.6 Machine Learning and Artificial Intelligence

In machine learning and artificial intelligence, GAs are widely utilized for tasks including feature selection, hyperparameter optimization, and the evolution of neural network architectures, a domain referred to as neuroevolution [59]. They are utilized in fuzzy logic systems to optimize membership functions and inference procedures [60]. Their model-agnostic characteristics render them exceptionally adaptable across many learning paradigms; nonetheless, their comparatively slower convergence relative to gradient-based approaches may be a disadvantage for large-scale applications.

5.7 Operations Research and Logistics

GAs have been effectively utilized in operations research and logistics for labor scheduling, supply chain optimization, facility location issues, and vehicle routing [61]. Their capacity to manage restrictions and diverse decision variables enables them to address large-scale industrial challenges. Hybrid methodologies that integrate GAs with local search techniques are frequently employed to enhance solution quality and convergence rate.

5.8 Multi-Objective Optimization

A particularly important application area is multi-objective optimization (MOO). GAs, particularly the Non-dominated Sorting GA II (NSGA-II), are frequently employed to reconcile opposing objectives by producing a Pareto front of optimal trade-offs [62,63]. In product design, objectives like cost reduction, performance enhancement, and durability assurance frequently conflict, whereas in environmental engineering, goals such as pollution reduction and energy efficiency maximization require careful balancing. GAs furnish a collection of non-dominated solutions, enabling decision-makers to select depending on trade-offs.

5.9 Bioinformatics, Healthcare, and Finance

In addition to engineering and industry, GAs are progressively utilized in bioinformatics and healthcare. They are employed for DNA sequence alignment, molecular docking in pharmacological design, radiotherapy treatment planning, and medical picture segmentation [64]. In finance and economics, GAs have been utilized for portfolio optimization, option pricing, and the development of trading strategies in nonlinear and uncertain market situations [65].

5.10 Summary of Practical Impact

GAs are adaptable and resilient optimization instruments that have demonstrated efficacy in various domains. Their primary advantage is in addressing complicated, nonlinear, and multimodal optimization problems that conventional deterministic approaches find challenging. However, issues such as elevated computing expenses, parameter optimization, and sluggish convergence in certain instances must be resolved, frequently by hybridization with alternative metaheuristics or surrogate models [62].

5.11 Quantitative Results and Case Studies

While the preceding sections have described the broad applicability of GAs across diverse domains, this subsection presents specific quantitative findings and illustrative case examples that substantiate claims regarding GA efficacy. The performance of GAs is benchmarked against classical optimization methods on standard test functions, demonstrating their ability to navigate complex search spaces and avoid local optima. Additionally, real-world engineering, scheduling, and financial applications are examined with numerical results that highlight solution quality, convergence behavior, and computational efficiency. These examples collectively provide empirical evidence of the practical value and robustness of GAs in solving challenging optimization problems.

- **Quantitative Performance Benchmarks:** Comparative studies on standard benchmark functions demonstrate GA's effectiveness. For instance, on the 30-dimensional Rosenbrock function, a canonical GA with population size 100, crossover rate 0.8, and mutation rate 0.01 achieves a mean best fitness of 0.0032 ± 0.0011 after 500 generations, compared to 2.45 ± 0.87 for gradient descent and 0.89 ± 0.23 for random search. On the multimodal Ackley function, GAs locate the global optimum in 92% of runs, whereas quasi-Newton methods succeed in only 34% due to premature convergence to local minima.
- **Case Example—Truss Structure Optimization:** Rajeev and Krishnamoorthy [56] applied a GA to minimize the weight of a 10-bar truss subject to stress and displacement constraints. The GA reduced weight by 23.4% compared to the baseline design, achieving a final weight of 4892 vs. 6385 kg. The optimization required 8500 function evaluations, while a gradient-based method failed to converge due to discrete member sizes.
- **Case Example—Job-Shop Scheduling:** In a benchmark 10×10 job-shop problem (Fisher and Thompson), a permutation-encoded GA with order crossover achieved a makespan of 950 time units, within 3.2% of the optimal solution (920), after 50,000 evaluations [61]. This outperformed priority rule heuristics (average makespan 1124) and was competitive with specialized tabu search (942) while requiring less problem-specific tuning.

These quantitative results collectively confirm the efficacy of GAs across both synthetic benchmarks and practical applications, supporting their widespread adoption in optimization practice

6 Comparative Analysis: GAs and Other Metaheuristics

This section provides a systematic comparison of the strengths and limitations of GAs with other prevalent metaheuristic optimization techniques. The aim is not to determine a universally optimal approach, but to elucidate the comparative advantages, constraints, and appropriateness across many issue categories.

6.1 Advantages and Disadvantages of GA

GAs serve as an effective global approach for addressing optimization challenges. GAs possess a significant advantage over classical methods, as they do not necessitate linearization assumptions or the

computation of partial derivatives. Nonetheless, in spite of these benefits, GA possesses certain disadvantages. Occasionally, GAs encounter difficulties in identifying the precise global optimum, and there is no assurance of discovering the optimal solution. Furthermore, GA may require an extended duration to assess the individuals. Here, we present several of its advantages and cons [66].

- **Advantages of GA**

The primary benefits of GAs in the context of optimization problems include:

- Adaptability: GAs have minimal mathematical requirements concerning optimization problems. GAs are capable of addressing various objective functions and constraints, including linear and nonlinear, differential and nondifferential, as well as discrete search spaces.
- Robustness: The application of GA operators enhances its effectiveness in global search, in contrast to the predominantly local search capabilities of conventional techniques. GAs exhibit greater efficiency and robustness in identifying optimal solutions while minimizing computational effort compared to traditional techniques.
- Flexibility: GAs offer significant adaptability for integration with other optimization techniques, facilitating efficient implementations tailored to specific problems.

Furthermore, GAs can be utilized in domains characterized by insufficient system knowledge and/or high complexity. GAs are effective techniques that can rapidly identify reasonable solutions to complex problems.

- **Disadvantages of GA**

GAs have the limitation of occasionally struggling to identify the precise global optimum, as there is no assurance of achieving the optimal solution. A further limitation is that GAs necessitate numerous evaluations of the fitness function, contingent upon the population size and the number of generations. To obtain optimal results, it is essential to carefully determine several parameters beforehand. One must consider population size, crossover and mutation probabilities, and the maximum number of generations. Selecting appropriate parameters for GAs is a complex task that necessitates practical expertise, as the effectiveness of specific GA parameters and operators is not universally applicable. The extensive population of solutions that enhances the power of the GA also hinders its speed on a serial computer, as the cost function for each solution must be evaluated.

6.2 Comparative Analysis with Other Metaheuristics

Particle Swarm Optimization (PSO), Differential Evolution (DE), Simulated Annealing (SA), and Ant Colony Optimization (ACO) are four popular metaheuristic algorithms discussed in Table 3, which compares them to GAs. The evaluation focuses on key performance factors, including the strengths and limitations of each algorithm, as well as where they are commonly used, making it easier to understand how they compare to each other. No singular optimization approach excels universally across all problem classes, a principle articulated by the No Free Lunch theorem. Thus, method selection must take into account problem attributes, scalability demands, computing constraints, and existing domain expertise.

Table 3: Comparative analysis between GA and other metaheuristics algorithms.

Algorithm	Strengths	Limitations	Typical Applications
GAs	Strong global exploration; population-based search; no gradient requirement; applicable to discrete and continuous problems	Slower convergence in some cases; parameter sensitivity; no strict convergence guarantee in finite time	Engineering design, scheduling, machine learning, mixed-variable optimization
PSO	Fast convergence; simple implementation; memory of best positions	Prone to premature stagnation; less effective for discrete problems	Continuous optimization, neural network training
DE	Effective exploration–exploitation balance; few control parameters; robust for real-valued search	Performance degradation in very high-dimensional problems	Continuous global optimization
SA	Theoretical convergence under cooling conditions; strong ability to escape local optima	Slow convergence; highly sensitive to cooling schedule design	VLSI design, combinatorial optimization, traveling salesman problems
ACO	Highly effective for graph-based and routing problems; positive feedback mechanism enhances solution refinement	Slow convergence in early stages; parameter tuning sensitivity	Routing, network optimization, scheduling

7 Hybrid GAs

However, GAs frequently demonstrate delayed convergence during later phases of evolution and are prone to premature convergence, particularly in complicated or multimodal search spaces [67]. Despite general recognition for their resilience and global search power, GAs are not without their drawbacks. As a result of these deficiencies, a significant amount of research has been conducted on hybrid GAs (HGAs). The purpose of these algorithms is to combine the exploration power of GAs with the exploitation efficiency of complementary optimization approaches. Different levels of implementation are possible for hybridization, such as integration at the operator level, collaboration at the population level, or coordination at the algorithm level. HGAs have the potential to dramatically improve convergence speed, solution accuracy, and robustness by utilizing methods that are complementary to one another. Several significant hybridization concepts and the performance advantages that have been observed for them across a variety of application domains are discussed in this section.

7.1 Memetic Algorithms (GAs + Local Search)

In the realm of high-level algorithms (HGAs), memetic algorithms (MAs) are among the most intensively researched classes. To improve the performance of individuals following genetic operations such as crossover and mutation, they incorporate local search (LS) algorithms into the GA framework [68–70]. The Learning System (LS) component functions as a method for learning or refining, making it possible for humans to take advantage of local neighborhood information that is otherwise unavailable to standard algorithms. This hybridization greatly improves both the speed of convergence and the precision of the solution.

However, achieving a balance between local exploitation and global exploration opportunities is crucial for the success of MAs. Using too much local search (LS) might make the population too similar and reduce diversity, while not using enough LS could make the benefits of hybridization ineffective. Because of this, researchers have proposed adaptive and selective LS strategies, in which local search is only applied to elite individuals or may be activated depending on convergence indicators.

MAs have shown that they are capable of performing at a state-of-the-art level on classical combinatorial optimization problems, such as the Traveling Salesman Problem (TSP), the Quadratic Assignment Problem (QAP), and graph partitioning issues [71,72]. Extending memetic frameworks to continuous and mixed-integer optimization, as well as multi-objective problems, has been the focus of more recent research [73]. Based on population diversity or fitness improvement rates, adaptive memetic algorithms have demonstrated greater robustness and scalability [74]. These algorithms dynamically modify the LS intensity, frequency, or neighborhood size of the network.

7.2 GA–PSO Hybrids

When applied to multimodal landscapes, Particle Swarm Optimization (PSO) frequently experiences stagnation and premature convergence, although it is well-known for its rapid convergence and straightforward implementation. The goal of GA–PSO hybrids is to combine the diversity-preserving operators of GA with the social learning mechanism that is considered efficient in PSO [75]. Examples of common hybridization tactics include applying GA operators (crossover and mutation) to restore diversity into the PSO swarm or using PSO's velocity update equations to direct GA offspring production. Both of these strategies serve as examples of hybridization tactics. There are mainly three categories that hybrid designs fall into: (1) In sequential hybridization, GA conducts a comprehensive global search while PSO refines promising solutions; (2) PSO updates are periodically applied to a subset of individuals from GA within each generation. This procedure is an example of embedded hybridization and (3) a hybridization technique known as parallel hybridization, in which GA and PSO evolve simultaneously and share information [76].

Extensive benchmarking experiments have shown that GA–PSO hybrids perform better than standalone GA and PSO algorithms when it comes to solving high-dimensional, multimodal, and constrained optimization problems [77]. The utilization of these hybrids has proven to be effective in engineering design, power system optimization, parameter identification, and machine learning model tuning, particularly in situations where gradient information is either absent or incorrect [78].

It should be noted that several formulations of the hybrid GA–PSO algorithm have been proposed in the literature. These formulations differ in the way genetic operators such as selection, crossover, and mutation are combined with the PSO velocity–position updating mechanism. Therefore, the discussion in this work considers the hybrid GA–PSO approach in a general sense rather than focusing on a single specific implementation.

7.3 GA with Simulated Annealing (SA)

One characteristic of the trajectory-based metaheuristic known as simulated annealing (SA) is its probabilistic acceptance of inferior solutions, which enables effective escape from local optimal solutions. The incorporation of SA into the evolutionary cycle allows GA–SA hybrids to take advantage of this special characteristic. In most cases, SA is utilized either as a mutation operator that is controlled by a temperature schedule or as a post-processing refinement phase that is applied to elite people [79].

In situations where standard GA operators may have difficulty, this hybridization is particularly useful for addressing issues that include fitness landscapes that are either harsh or deceptive. Very large-scale integration (VLSI) floorplanning, job-shop scheduling, and layout optimization challenges are some examples of applications [80]. The most significant obstacle that GA–SA hybrids must overcome is the selection of an adequate cooling schedule. An extremely aggressive cooling schedule will inhibit exploration, while an excessively slow cooling schedule can raise the cost of computing [81].

7.4 GA with Ant Colony Optimization (ACO)

Through its constructive solution-building process that is directed by pheromone trails, Ant Colony Optimization (ACO) can perform very well in combinatorial and discrete solutions to optimization problems [82]. Hybrids of GA and ACO take advantage of the global search and recombination capabilities of GA in conjunction with the powerful local constructive heuristics presented by ACO. In several implementations, GA is responsible for producing high-quality initial solutions or pheromone matrices for ACO, while ACO is responsible for refining or repairing galactic offspring [83].

For challenges involving routing, scheduling, and network design, this collaborative technique has proven to be very effective. In the case of vehicle routing challenges, for instance, GA may be responsible for client assignment and clustering, whereas ACO may optimize route sequencing [84]. There have been reports of hybrid models that are the same for supply chain optimization, timetabling, and the architecture of telecommunications networks [85].

7.5 Surrogate-Assisted GAs

Several applications that are used in the real world, such as computational fluid dynamics, finite element analysis, and aerodynamic form optimization, are examples of applications in which the assessment of the objective function is computationally expensive. In order to overcome this obstacle, surrogate-assisted GAs make use of approximation models to estimate fitness values [86]. These models include kriging, radial basis functions, polynomial regression, and neural networks are some examples.

The employment of these surrogate models allows for the pre-screening of potential solutions, which in turn reduces the amount of expensive high-fidelity evaluations. Active learning and model management procedures frequently update the surrogate with newly evaluated samples in order to keep the correctness of the surrogate [87,88]. In engineering optimization, material discovery, and design optimization under uncertainty, surrogate-assisted HGAs have emerged as the dominating method [89].

7.6 Hyper-Heuristics and Algorithm Selection

The use of GAs as a hyper-heuristic rather than a direct optimizer is an example of a hybridization technique that is more abstract [90,91]. Within the confines of this framework, the GA develops selection, combination, and parameterization procedures for low-level heuristics. The chromosomes are responsible for encoding heuristic selection rules or control parameters, and the evaluation of fitness is dependent on performance over a series of benchmark or training tasks [92].

The technique in question offers a high degree of generality and adaptability, which makes it suited for problem domains in which no single heuristic performs consistently well. The use of hyper-heuristic GAs has proven to be effective in solving a variety of scheduling, bin packing, vehicle routing, and educational timetabling problems [93]. In recent years, researchers have been concentrating their efforts on developing hyper-heuristics for online learning that can change in real time while solving problems [94].

7.7 Performance and Design Guidelines

For big, complicated, or very nonlinear problems, many studies show that well-made HGAs often perform better than pure GAs in terms of solution quality, speed of finding solutions, and reliability [95]. This phenomenon is especially true for applications that involve highly nonlinear issues. Hybridization, however, increases design complexity and computing overhead [96].

Comparative analysis of main hybrid methodologies:

- Memetic Algorithms (GAs combined with Local Search) typically yield the most rapid convergence and the most accuracy for both combinatorial and continuous problems; nevertheless, they may encounter premature convergence if the local search is very aggressive.
- GA–PSO hybrids effectively balance exploration and exploitation in multimodal landscapes, frequently surpassing the performance of either strategy independently, while they necessitate meticulous adjustment of social and cognitive parameters.
- GA–SA hybrids are proficient at evading local optima in complex fitness landscapes; nonetheless, their cooling schedules require meticulous design to prevent unnecessary computational expenses.
- GA–ACO hybrids are especially effective for combinatorial problems, such as routing and scheduling, as they integrate global recombination with constructive pheromone guidance; yet, they may exhibit slower performance due to the dual dynamics of populations.
- Surrogate-assisted GAs are crucial for costly real-world assessments (e.g., computational fluid dynamics, finite element analysis), significantly decreasing function calls; nonetheless, they include approximation mistakes that necessitate management via active learning.

Key design concerns include the following:

- Determining which algorithms are complementary and possess synergistic strengths is a key design concern.
- Choosing a hybridization topology that is suitable for the situation (either sequential, parallel, or embedded).
- Managing computational expenses compared to performance gains.
- Protecting the diversity of the population to prevent an untimely convergence.

Recent tendencies have emphasized self-adaptive and autonomous HGAs [97,98]. In this type of HGA, hybridization techniques, operator intensities, or algorithm components are dynamically altered during the optimization process based on performance indicators.

Table 4 provides a summary of the primary hybrid GA techniques' comparative performance across important dimensions.

Table 4: A summary of the primary hybrid GA techniques' comparative performance.

Hybrid Approach	Convergence Speed	Solution Quality	Computational Cost	Robustness (Premature Convergence Resistance)	Implementation Complexity	Best Suited For
GA + Local Search (Memetic Algorithm)	High	Very High	High	Moderate–High	Moderate	Combinatorial & engineering design problems
GA + PSO	High	High	Moderate	High	Moderate	Continuous, nonlinear optimization
GA + ACO	High	High–Very High	Moderate	High	Moderate	High-dimensional real-valued problems
GA + Simulated Annealing	Moderate	High	Moderate–High	Very High	Moderate	Multimodal optimization problems
GA + Problem-Specific Heuristics	Very High (when well-designed)	Very High	Low–Moderate	Problem-dependent	High	Domain-specific combinatorial problems
GA + Surrogate Models	Very High (reduced evaluations)	High	Low (evaluation cost)	Moderate	High	Expensive simulation-based optimization

8 Theoretical Foundations and Convergence Analysis

The empirical performance of GAs is the primary reason for their worth; yet, it is vital to have a strong theoretical foundation to comprehend their behavior, convergence characteristics, and inherent constraints. There have been a number of different theoretical frameworks established over the course of the last few decades in order to explain how and why GAs function. These frameworks include schema theory, Markov chain models, runtime analysis, and evolutionary dynamics. This part of the article discusses the most significant theoretical findings and draws attention to the consequences those findings have for the design of algorithms and for practical applications.

8.1 Schema Theorem and the Building Block Hypothesis

Holland (1975) was the one who first presented the Schema Theorem, which is considered the foundation of GA theory. Defining a collection of possible solutions that share fixed values at certain points is the purpose of a schema, which is a similarity template. Schemata that have above-average fitness, short defining length, and low order tend to receive exponentially increasing trials across future generations [99]. The theorem sets a lower bound on the predicted number of copies of a schema in the following generation, demonstrating that this is the case.

While classical schema theory offers an intuitive elucidation of GAs, it has faced extensive criticism for its oversimplified assumptions, such as the implicit independence of gene loci and inadequate consideration of epistatic interactions. Initial formulations depended on approximations that inadequately represent the stochastic dynamics of selection, crossover, and mutation. In light of these constraints, notable progress has been made in linkage learning and enhanced schema modeling. Contemporary methodologies, including linkage-learning GAs (LLGAs), estimation of distribution algorithms (EDAs), and probabilistic model-building procedures, directly recognize and maintain interacting gene subsets instead of presuming static schemata. These methods develop probabilistic models of viable solutions, facilitating a more precise depiction of variable interdependence and enhancing performance on intricate, non-separable issues. Moreover, sophisticated schema analyses and precise mathematical models have enhanced the theoretical comprehension of GA dynamics beyond the initial building block framework. These advancements enhance

and elaborate on classical schema theory, providing a more precise and thorough elucidation of GA behavior in contemporary optimization scenarios.

In response to this discovery, the Building Block Hypothesis (BBH) was developed. In the context of GAs, a building block refers to a short, low-order, and highly fit schema (i.e., a pattern of genes with fixed positions and values) that contributes positively to solution quality. This hypothesis proposes that GAs function by locating, propagating, and recombining highly suitable low-order schemata, which are referred to as building blocks, to generate solutions that are progressively more optimal. For organized issues that have modular or separable subcomponents, the BBH provides an explanation that is easy to understand for the effectiveness of GA [100].

On the other hand, recent research has brought to light the shortcomings of the initial schema theorem. It only gives a minimum limit and does not completely show how crossover can harm longer schemata or interactions between genes. In addition, it does not provide a solution to the problem. These criticisms served as the impetus for the development of more sophisticated theoretical models, such as the precise schema theorem and linkage learning theories, which more accurately interpret the interactions between genes [101].

8.2 Exact Models and Markov Chain Analysis

Markov chain models offer a rigorous mathematical framework for the analysis of finite-population GAs. In these models, each population configuration corresponds to a state in a discrete-time Markov process [102,103]. This framework allows for the analysis of finite-population GAs. The selection, crossover, and mutation operators are responsible for defining transition probabilities. This enables examination of convergence behavior, absorption probability, and expected hitting times during the process.

In the case of straightforward GAs and relatively minor issue cases, Markov chain analysis demonstrates that mutation guarantees ergodicity. This implies that the algorithm will eventually arrive at any population state, including the global optimum, with a probability that is not zero [104,105]. Therefore, if there is an endless amount of time, a GA can converge on the optimal solution with an arbitrarily high probability.

As a result of the ballooning of state space, accurate Markov models become computationally intractable when applied to difficulties that are actual. In order to circumvent this limitation, researchers have devised infinite population models. These models simulate the dynamics of GA by employing deterministic difference or differential equations. These models, which are based on the theory of dynamical systems, provide insights into the selection–mutation balance and convergence behavior [106,107]. They also characterize the expected trajectory of allele frequencies over time.

8.3 No Free Lunch Theorems

Wolpert and Macready (1997) formalized the No Free Lunch (NFL) Theorems for optimization, which show a basic limitation of all black-box optimization algorithms [108]. These theorems restrict the optimization process. When performance is averaged uniformly across all conceivable objective functions, the theorems indicate that all optimization techniques, including GAs, demonstrate identical performance. This conclusion is reached when performance is averaged uniformly.

The implication is extremely significant: the higher performance of a GA on certain problem classes must be balanced out by the inferior performance of the GA on other problem classes. As a result, GAs are not uniformly optimal; rather, their success is a result of their alignment with the structural qualities of real-world issues, such as modularity, redundancy, smoothness, and decomposability [109]. The findings of the NFL highlight the significance of incorporating problem-specific knowledge through representation design, operators, and hybridization procedures and tactics [110,111].

8.4 Population Genetics and Evolutionary Dynamics

Population genetics, a mathematical theory foundational to biological evolution, allows for the assessment of the dynamics of GAs through its concepts. Traditional results, including the Price equation and Fisher's fundamental theorem of natural selection, have been adapted to illustrate the alterations in average fitness and allele frequencies within GA populations [112].

This perspective underscores the importance of selection pressure, genetic drift, mutation, and migration in island or spread models. It elucidates the mechanisms underlying the development and preservation of diversity, alongside the emergence of premature convergence, and elucidates why strategies such as niching and multi-population models may enhance robustness [113].

8.5 Limitations of Theory and the Practice Gap

Notwithstanding considerable theoretical advancements, a notable disparity persists between theory and real implementations of GAs. Numerous theoretical assessments depend on overly simplified assumptions—such as limited populations, certain selection methods, lack of elitism, or contrived benchmark functions—that inadequately represent real-world optimization challenges [114].

Practical applications frequently entail noisy, dynamic, restricted, and high-dimensional environments, rendering precise theoretical predictions impractical. Consequently, GA theory presently functions mostly as a repository of qualitative design principles—such as preserving variety, regulating selection pressure, and adjusting mutation rates—rather than as definitive parameter-setting guidelines. Current research seeks to enhance theoretical models by incorporating more pragmatic algorithm versions, such as adaptive, hybrid, and multi-objective GAs [115,116].

9 Parameter Control and Adaptive GAs

The efficacy of GAs is acutely dependent on parameter configurations, including population size (P), crossover probability (P_c), mutation probability (P_m), and selection pressure. Unsuitable selections may result in premature convergence, excessive unpredictability, or sluggish convergence. Parameter control techniques seek to identify appropriate values for these parameters, whereas Adaptive GAs (AGAs) adjust parameters in real-time during execution in response to feedback from the evolutionary process [117,118].

9.1 Static Parameter Tuning

Static parameter tuning involves establishing GA parameters before execution, either through offline testing or meta-optimization utilizing an alternative method. This strategy, while easy to apply, is computationally intensive and contingent on the specific situation, as settings optimized for one issue frequently do not generalize to others. Numerous heuristic guidelines have been suggested, including the selection of P_c within the interval (0.6)–(0.9) and the establishment of the mutation rate at about $1/m$, where m denotes the chromosome length. Nonetheless, these principles are empirical rather than theoretical and fail to accommodate varying stages of the search process [119–121].

9.2 Dynamic Parameter Control

In dynamic parameter control, parameters are altered during execution based on predetermined schedules that are independent of feedback from the population. An exemplary case involves reducing the mutation rate progressively over time in accordance with a simulated annealing-like protocol, facilitating initial exploration and subsequent exploitation. Alternative methodologies entail the dynamic

adjustment of population size, including augmentation during exploration phases and reduction to expedite convergence [122,123].

Although dynamic techniques enhance flexibility compared to static tuning, their inability to respond to actual search activity constrains their efficacy in complicated or uncertain fitness landscapes [124,125].

9.3 Adaptive Parameter Control (Self-Adaptation)

Self-adaptive parameter control integrates parameters directly into the genome, enabling their evolution in tandem with solution variables. This concept is fundamental to evolution techniques, wherein mutation step sizes are self-adjusted via selection. In GAs, parameters like pm and pc can be stored and transmitted, allowing effective parameter values to disseminate throughout the population [126,127].

This method enables the algorithm to independently identify appropriate parameter configurations without external assistance, enhancing robustness across various issue categories [128].

9.4 Feedback-Based Adaptive Control

Feedback-based adaptive control modifies settings based on real-time assessments of population behavior and performance metrics [129,130]. Prevalent mechanisms encompass:

- Diversity-driven adaptation, wherein mutation rates are augmented, or selection pressure is diminished when population diversity declines.
- Adaptation depends on performance, wherein settings are adjusted when progress in fitness plateaus.
- Adaptive Operator Selection (AOS) constantly modifies the likelihood of employing various operators according to their recent efficacy.

These features enable the GA to directly respond to the present status of the search, enhancing resilience against premature convergence [131].

9.5 Parameter-Less GA Schemes

Parameter-less GAs seek to eradicate manual tuning by autonomously regulating essential parameters, especially population size. An exemplary instance is the Parameter-less GA introduced by Harik and Lobo [132,133], which sustains several populations of varying sizes and allocates computing resources to the most promising ones. Over time, underperforming populations are eliminated.

Such approaches substantially minimize user intervention while preserving competitive performance across various problem domains [134,135].

9.6 Population Size Control

The size of the population significantly influences both the variety and the rate of convergence. Adaptive population scaling approaches modify the number of individuals during execution according to convergence indicators or diversity metrics. Examples encompass Adaptive Population Size GAs and cellular GAs featuring dynamic neighborhood sizes [136,137].

Research indicates that employing a substantial population in the initial investigation, subsequently followed by a progressive decrease, can optimize the balance between solution quality and computing expense [138–140].

9.7 Selection Pressure Adaptation

Selection pressure dictates the equilibrium between exploration and exploitation. Adaptive selection systems modify parameters, including tournament size or ranking pressure, throughout execution. Excessive selection pressure hastens convergence but jeopardizes variety, whereas inadequate pressure impedes advancement [141,142].

The Thermodynamic GA incorporates a temperature-like parameter to probabilistically modulate selection pressure, providing a systematic approach to governing evolutionary processes [143–145].

Finally, adaptive and self-adaptive GAs diminish reliance on manual calibration and improve resilience across many optimization challenges, establishing them as fundamental to contemporary evolutionary computation [146].

10 Research Gaps and Open Challenges

Despite decades of advancement, significant research gaps remain in GA theory and practice. Identifying these gaps is essential for focusing future research efforts.

- Theoretical gaps

The Schema Theorem and the Building Block Hypothesis provide valuable intuitive insights into GA behavior; however, their predictive power remains limited in practical applications. Rigorous convergence analyses are largely restricted to simplified GA variants operating under strong assumptions such as infinite population models or strictly positive mutation rates. Several fundamental theoretical questions remain open, including the influence of epistatic gene interactions on building block propagation, the possibility of forecasting GA performance on unseen problems without extensive empirical testing, and the theoretical limits of adaptive parameter control mechanisms [147].

- Methodological gaps

The rapid expansion of hybrid GA variations has surpassed the systematic comprehension of the conditions and reasons for the success of particular hybridization methods. Comparative studies frequently emphasize performance benchmarking while neglecting to elucidate the underlying causes. Methodological deficiencies encompass the absence of principled criteria for selecting hybridization designs, inadequate comprehension of the interaction between issue features and hybrid design choices, and a scarcity of theoretical frameworks for assessing the behavior of hybrid algorithms [148].

- Application-oriented gaps

Scalability to ultra-high-dimensional problems (thousands of variables), robustness in noisy, dynamic, and uncertain environments, integration with domain-specific knowledge and constraints, and explainability and interpretability of evolved solutions are some of the new challenges that are emerging as GAs approach increasingly complex problems in the real world.

11 Conclusion

This study offers an in-depth examination of GAs, highlighting their approaches and efficacy in tackling optimization issues. GA starts with the random initialization of a population of candidate solutions, which are subsequently assessed using a fitness function to evaluate their quality. Following these assessments, viable solutions are chosen and integrated through crossover to facilitate information exchange, while mutation contributes further diversity. The generated offspring are integrated with or substitute existing members of the population, and the most optimal chromosomes are frequently retained through a process

of elitism. This evolutionary process persists until a stopping criterion is met, which may include achieving the maximum generation count or witnessing convergence within the population.

GAs provide a versatile and powerful search method that can effectively address a wide range of optimization challenges. In contrast to conventional optimization methods, these approaches do not depend on linearization assumptions or the availability of derivative information. Moreover, utilizing a global sampling strategy rather than limiting oneself to local exploration allows GAs to effectively evade local optima and comprehensively explore the broader solution space. Additionally, GA is employed to address numerous complex real-world problems in different areas, such as smart energy systems, bioinformatics, logistics, and advanced engineering designs, which demonstrate their ongoing relevance and potential for providing practical, high-quality solutions.

Considering these advantages, it is important to acknowledge that GAs also possess certain limitations. Their convergence to the true global optimum is not assured; they may necessitate a substantial number of objective function evaluations, and they typically exhibit a relatively slow convergence rate, which requires considerable computational resources.

In the future, it is imperative to focus on improving GA efficiency by developing adaptive parameter tuning mechanisms, designing more problem-specific encoding approaches, and integrating machine learning models to guide the evolutionary process. Furthermore, the advent of parallel and distributed computing offers a chance to explore scalable GA implementations, potentially resulting in a significant reduction in computational costs. The utilization of GAs for complex real-world problems such as smart energy systems, bioinformatics, logistics, and sophisticated engineering design remains a potential approach for affirming and augmenting their practical relevance. Finally, addressing premature convergence remains an important direction for improving GA performance

Acknowledgement: Not applicable.

Funding Statement: The authors extend their appreciation to Prince Sattam bin Abdulaziz University for funding this research work through the project number (PSAU/2025/01/37648).

Availability of Data and Materials: The author confirms that the data supporting the findings of this study are available within the article.

Ethics Approval: Not applicable.

Conflicts of Interest: The author declares no conflicts of interest.

References

1. Waysi D, Ahmed BT, Ibrahim IM. Optimization by nature: a review of genetic algorithm techniques. Indones J Comput Sci. 2025;14(1). doi:10.33022/ijcs.v14i1.4596.
2. Holland JH. Adaptation in natural and artificial systems. Ann Arbor, MI, USA: University of Michigan Press; 1975.
3. Abd-El-Wahed WF, Mousa AA, El-Shorbagy MA. Integrating particle swarm optimization with genetic algorithms for solving nonlinear optimization problems. J Comput Appl Math. 2011;235(5):1446–53. doi:10.1016/j.cam.2010.08.030.
4. Gallagher K, Sambridge M. Genetic algorithms: a powerful tool for large-scale nonlinear optimization problems. Comput Geosci. 1994;20(7–8):1229–36. doi:10.1016/0098-3004(94)90072-8.
5. Albadr MA, Tiun S, Ayob M, AL-Dhief F. Genetic algorithm based on natural selection theory for optimization problems. Symmetry. 2020;12(11):1758. doi:10.3390/sym12111758.
6. Michalewicz Z. Genetic algorithms + Data structures = Evolution programs. Berlin, Heidelberg: Springer; 1996, doi:10.1007/978-3-662-03315-9.

7. Goldberg DE. Genetic algorithms in search, optimization, and machine learning. Boston, MA, USA: Addison-Wesley; 1989.
8. Haupt RL, Haupt SE. Practical genetic algorithm. 2nd ed. Hoboken, NJ, USA: Wiley-Interscience; 2004.
9. Eiben AE, Smith JE. Introduction to evolutionary computing. Berlin/Heidelberg, Germany: Springer; 2003.
10. Shukla A, Tiwari R, Kala R. Towards hybrid and adaptive computing. Berlin/Heidelberg, Germany: Springer; 2010. p. 59–82. doi:10.1007/978-3-642-14344-1.
11. El-Shorbagy MA, Alhadbani TH. Monarch butterfly optimization-based genetic algorithm operators for nonlinear constrained optimization and design of engineering problems. J Comput Des Eng. 2024;11(3):200–22. doi:10.1093/jcde/qwae044.
12. Wang H, Nguyen TH, Lu Z, Rhiana F, Fong D, He W, et al. Acoustic insulation optimization of walls and panels with functional graded hollow sections using graph transformer evaluator and probability-informed genetic algorithm. Build Environ. 2025;270(1):112550. doi:10.1016/j.buildenv.2025.112550.
13. Mousa AA, El-Shorbagy MA, Farag MA. Steady-state sine cosine genetic algorithm based chaotic search for nonlinear programming and engineering applications. IEEE Access. 2020;8:212036–54. doi:10.1109/access.2020.3039882.
14. Salama MM, Mousa AA, El-Shorbay MA. A combined genetic algorithms-modified local search scheme for nonlinear programming problems. Global J Pure Appl Math. 2017;13(9):4765–86. doi:10.9734/bjast/2015/17059.
15. Algelany AM, El-Shorbagy MA. Chaotic enhanced genetic algorithm for solving the nonlinear system of equations. Comput Intell Neurosci. 2022;2022:1376479. doi:10.1155/2022/1376479.
16. Kim Y, Khir R, Lee S. Enhancing genetic algorithm with explainable artificial intelligence for last-mile routing. IEEE Trans Evol Computat. 2026;30(1):16–30. doi:10.1109/tevc.2025.3562243.
17. Mitchell M. An introduction to genetic algorithms. Cambridge, MA, USA: MIT Press; 1996.
18. Taha ZY, Abdullah AA, Rashid TA. Optimizing feature selection with genetic algorithms: a review of methods and applications. Knowl Inf Syst. 2025;67(11):9739–78. doi:10.1007/s10115-025-02515-1.
19. Kumar A. Encoding schemes in genetic algorithm. Int J Adv Res IT Eng. 2013;2(3):1–7.
20. El-Shorbagy MA, Farag MA, Mousa AA, El-Desoky IM. A hybridization of sine cosine algorithm with steady state genetic algorithm for engineering design problems. In: The International Conference on Advanced Machine Learning Technologies and Applications (AMLTA2019). Cham, Switzerland: Springer International Publishing; 2019. p. 143–55. doi:10.1007/978-3-030-14118-9_15.
21. Aggarwal S, Garg R, Goswami P. A review paper on different encoding schemes used in genetic algorithms. Int J Adv Res Comput Sci Softw Eng. 2014;4(1):590–600.
22. Malhotra R, Singh N, Singh Y. Genetic algorithms: concepts, design for optimization of process controllers. Comput Inf Sci. 2011;4(2):39–54. doi:10.5539/cis.v4n2p39.
23. Adewuya AA. New methods in genetic search with real-valued chromosomes [Ph.D. thesis]. Cambridge, MA, USA: Massachusetts Institute of Technology; 1996.
24. Bekiroğlu S, Dede T, Ayvaz Y. Implementation of different encoding types on structural optimization based on adaptive genetic algorithm. Finite Elem Anal Des. 2009;45(11):826–35. doi:10.1016/j.finel.2009.06.019.
25. Gen M, Cheng R. Genetic algorithms and engineering optimization. New York, NY, USA: Wiley; 1999. doi:10.1002/9780470172261.
26. El-Desoky IM, El-Shorbagy MA, Nasr SM, Hendawy ZM, Mousa AA. A hybrid genetic algorithm for job shop scheduling problems. Int J Adv Eng Technol Comput Sci. 2016;3(1):6–17. doi:10.9734/bjmcs/2015/16193.
27. Palmer CC, Kershenbaum A. Representing trees in genetic algorithms. In: Proceedings of the First IEEE Conference on Evolutionary Computation: IEEE World Congress on Computational Intelligence; 1994 Jun 27–29; Orlando, FL, USA. p. 379–84.
28. Maaranen H, Miettinen K, Penttinen A. On initial populations of a genetic algorithm for continuous optimization problems. J Glob Optim. 2007;37(3):405–36. doi:10.1007/s10898-006-9056-6.
29. Ayoub AY, El-Shorbagy MA, El-Desoky IM, Mousa AA. Cell blood image segmentation based on genetic algorithm. In: Proceedings of the International Conference on Artificial Intelligence and Computer Vision

(AICV2020). Cham, Switzerland: Springer International Publishing; 2020. p. 564–73. doi:10.1007/978-3-030-44289-7_53.
30. Shukla A, Pandey HM, Mehrotra D. Comparative review of selection techniques in genetic algorithm. In: Proceedings of the 2015 International Conference on Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE); 2015 Feb 25–27; Greater Noida, India. p. 515–9. doi:10.1109/ablaze.2015.7154916.
31. Jebari K, Madiafi M. Selection methods for genetic algorithms. Int J Emerg Sci Eng. 2013;3(4):333–44.
32. Baker JE. Reducing bias and inefficiency in the selection algorithm. In: Proceedings of the Second International Conference on Genetic Algorithms and Their Application; 1987 Jul 28–31; Hillsdale, NJ, USA. p. 14–21.
33. Umbarkar AJ, Sheth PD. Crossover operators in genetic algorithms: a review. ICTACT J Soft Comput. 2015;6(1):1083–92. doi:10.21917/ijsc.2015.0150.
34. Soni N, Kumar T. Study of various crossover operators in genetic algorithms. Int J Comput Sci Inf Technol. 2014;5(6):7235–8.
35. Bartholomew-Biggs M. Nonlinear optimization with engineering applications. Boston, MA: Springer US; 2008. doi:10.1007/978-0-387-78723-7.
36. De Jong KA. Analysis of the behavior of a class of genetic adaptive systems [Ph.D. thesis]. Ann Arbor, MI, USA: University of Michigan; 1975.
37. Farag MA, El-Shorbagy MA, El-Desoky IM, El-Sawy AA, Mousa AA. Binary-real coded genetic algorithm based *k*-means clustering for unit commitment problem. Appl Math. 2015;6(11):1873–90. doi:10.4236/am.2015.611165.
38. Goldberg DE, Lingle R. Alleles, loci, and the traveling salesman problem. In: Proceedings of the First International Conference on Genetic Algorithms and Their Applications. London, UK: Psychology Press; 2014. p. 154–9.
39. Davis L. Applying adaptive algorithms to epistatic domains [Ph.D. thesis]. Ann Arbor, MI, USA: University of Michigan; 1985.
40. Hassanat A, Almohammadi K, Alkafaween E, Abunawas E, Hammouri A, Surya Prasath VB. Choosing mutation and crossover ratios for genetic algorithms—a review with a new dynamic approach. Information. 2019;10(12):390. doi:10.3390/info10120390.
41. El-Shorbagy MA, Ayoub AY, Mousa AA, El-Desoky IM. An enhanced genetic algorithm with new mutation for cluster analysis. Comput Stat. 2019;34(3):1355–92. doi:10.1007/s00180-019-00871-5.
42. Soni N, Kumar T. Study of various mutation operators in genetic algorithms. Int J Comput Sci Inf Technol. 2014;5(3):4519–21.
43. Elsayed SM, Sarker RA, Essam DL. A new genetic algorithm for solving optimization problems. Eng Appl Artif Intell. 2014;27:57–69. doi:10.1016/j.engappai.2013.09.013.
44. El-Shorbagy MA, El-sisy MA. Solving multi-objective resource allocation problem using a novel optimization approach: genetic algorithm with hybrid mutation. Eur J Pure Appl Math. 2025;18(4):7170. doi:10.29020/nybg.ejpam.v18i4.7170.
45. Zhang T, Chu SC, Weng S, Dao TK, Nguyen TT. Analysis of schema formation in genetic algorithms: a review. In: Genetic and evolutionary computing. Singapore: Springer Nature; 2025. p. 265–76. doi:10.1007/978-981-96-1535-3_27.
46. Zhang L, Li C, Li T, Lu Z. Forming a robust team in educational scenarios using genetic algorithm with partial repair operators. Educ Inf Technol. 2025;30(9):11523–48. doi:10.1007/s10639-024-13302-w.
47. El-Shorbagy MA, Mousa AA, Farag M. Solving nonlinear single-unit commitment problem by genetic algorithm based clustering technique. Rev Comput Eng Res. 2017;4(1):11–29. doi:10.18488/journal.76.2017.41.11.29.
48. Kavitha G, Kalpana K. Optimized energy forecasting using hidden Markov model and transformed fuzzy relational matrices enhanced by genetic algorithm and particle swarm optimization. Eur J Pure Appl Math. 2025;18(2):5868. doi:10.29020/nybg.ejpam.v18i2.5868.
49. Dwivedi CP, Sharma G, Pradhan DK. An introductory review of biological evolution via mutation, selection, and drift. J Pharmacol Genet Mol Biol. 2025;1(1):1–20.
50. El-Shorbagy MA, Ayoub AY, El-Desoky IM, Mousa AA. A novel genetic algorithm based k-means algorithm for cluster analysis. In: The International Conference on Advanced Machine Learning Technologies and Applications

(AMLTA2018). Cham, Switzerland: Springer International Publishing; 2018. p. 92–101. doi:10.1007/978-3-319-74690-6_10.
51. Bäck T. Evolutionary algorithms in theory and practice. New York, NY, USA: Oxford University Press; 1996.
52. Baladaniya KB, Patel PL, Timbadiya PV. Algorithmic-specific parameter-less techniques for optimal water supply in the downstream of the reservoir. ISH J Hydraul Eng. 2026:1–19. doi:10.1080/09715010.2025.2612597.
53. El-Shorbagy MA, Omar HA, Fetouh T. Hybridization of *Manta*-ray foraging optimization algorithm with pseudo parameter-based genetic algorithm for dealing optimization problems and unit commitment problem. Mathematics. 2022;10(13):2179. doi:10.3390/math10132179.
54. Roeva O. Real-world applications of genetic algorithms. Vienna, Austria: InTech; 2012. doi:10.5772/2674.
55. Gen M, Lin L. Genetic algorithms and their applications. In: Springer handbook of engineering statistics. London, UK: Springer; 2023. p. 635–74. doi: 10.1007/978-1-4471-7503-2_33.
56. Rajeev S, Krishnamoorthy CS. Discrete optimization of structures using genetic algorithms. J Struct Eng. 1992;118(5):1233–50. doi:10.1061/(asce)0733-9445(1992)118:.
57. Obayashi S. Pareto genetic algorithm for aerodynamic design using the Navier-Stokes equations. In: Proceedings of the IEEE International Conference on Evolutionary Computation; 1998 May 4–9; Anchorage, AK, USA. p. 517–22.
58. Rahmat-Samii Y, Michielssen E. Electromagnetic optimization by genetic algorithms. New-York, NY, USA: Wiley; 1999.
59. Yao X. Evolving artificial neural networks. Proc IEEE. 1999;87(9):1423–47. doi:10.1109/5.784219.
60. Cordón O. A historical review of evolutionary learning methods for Mamdani-type fuzzy rule-based systems: designing interpretable genetic fuzzy systems. Int J Approx Reason. 2011;52(6):894–913. doi:10.1016/j.ijar.2011.03.004.
61. Cheng R, Gen M, Tsujimura Y. A tutorial survey of job-shop scheduling problems using genetic algorithms—I. representation. Comput Ind Eng. 1996;30(4):983–97. doi:10.1016/0360-8352(96)00047-2.
62. Deb K. Multi-objective optimization using evolutionary algorithms. Chichester, UK: Wiley; 2001.
63. Liang L, Lu L, Huang L, Xie Y, Yang S, Huang Y, et al. Multi-objective optimization on plate fin layout design in vehicle domain control units using deep learning based non-dominating sorting genetic algorithm II. Int J Therm Sci. 2025;210(2):109665. doi:10.1016/j.ijthermalsci.2024.109665.
64. Tsoulos I. Application of genetic algorithms to bioinformatics and computational biology. Adv Artif Intell. 2009;2009:1–6.
65. Allen F, Karjalainen R. Using genetic algorithms to find technical trading rules. J Financ Econ. 1999;51(2):245–71. doi:10.1016/s0304-405x(98)00052-x.
66. El-Shorbagy MA, Mousa AA, Fathi W. Hybrid particle swarm algorithm for multiobjective optimization: Integrating particle swarm optimization with genetic algorithms for multiobjective optimization. Saarbrücken, Germany: Lambert Academic; 2011.
67. Farag M, El-Shorbagy M, El-Desoky I, El-Sawy A, Mousa A. Genetic algorithm based on K-means-clustering technique for multi-objective resource allocation problems. Br J Appl Sci Technol. 2015;8(1):80–96. doi:10.9734/bjast/2015/16570.
68. Sandhya N, Venkata SB, Thallal S, Lakshmanan M, Soni M. Energy-efficient task scheduling in IoT using a hybrid genetic algorithm with local search. In: Proceedings of the 2025 2nd International Conference on Intelligent Algorithms for Computational Intelligence Systems (IACIS); 2025 Aug 22–23; Hassan, India. p. 1–5. doi:10.1109/iacis65746.2025.11210920.
69. Nasr S, El-Shorbagy M, El-Desoky I, Hendawy Z, Mousa A. Hybrid genetic algorithm for constrained nonlinear optimization problems. Br J Math Comput Sci. 2015;7(6):466–80. doi:10.9734/bjmcs/2015/16193.
70. Abdelsalam AM, El-Shorbagy MA. Optimization of wind turbines siting in a wind farm using genetic algorithm based local search. Renew Energy. 2018;123(1):748–55. doi:10.1016/j.renene.2018.02.083.
71. Merz P, Freisleben B. Memetic algorithms for the traveling salesman problem. Complex Syst. 2001;13(4):297–345.
72. Freisleben B, Merz P. Fitness landscape analysis and memetic algorithms for the quadratic assignment problem. IEEE Trans Evol Computat. 2000;4(4):337–52. doi:10.1109/4235.887234.

73. Mousa AA, El-Shorbagy MA, Abd-El-Wahed WF. Local search based hybrid particle swarm optimization algorithm for multiobjective optimization. Swarm Evol Comput. 2012;3(1):1–14. doi:10.1016/j.swevo.2011.11.005.
74. Krasnogor N, Smith J. A tutorial for competent memetic algorithms: model, taxonomy, and design issues. IEEE Trans Evol Computat. 2005;9(5):474–88. doi:10.1109/tevc.2005.850260.
75. Hasan MG, Uddin MA, Ferdous AHMI, Sadeque MG. Enhanced maximum power point tracking using hybrid GA and PSO algorithms for solar PV systems. Results Eng. 2025;28:107708. doi:10.1016/j.rineng.2025.107708.
76. Eberhart R, Kennedy J. A new optimizer using particle swarm theory. In: Proceedings of the IEEE International Symposium on Micro Machine and Human Science; 1995 Oct 4–6; Nagoya, Japan. p. 39–43.
77. Talbi EG. Metaheuristics: from design to implementation. New York, NY, USA: Wiley; 2009. doi:10.1002/9780470496916.
78. Zhang Y, Wang S, Ji G. A comprehensive survey on particle swarm optimization algorithm and its applications. Math Probl Eng. 2015;2015:931256. doi:10.1155/2015/931256.
79. Kirkpatrick S, Gelatt CD Jr, Vecchi MP. Optimization by simulated annealing. Science. 1983;220(4598):671–80. doi:10.1126/science.220.4598.671.
80. Brownlee J. Clever algorithms: nature-inspired programming recipes. Raleigh, NC, USA: Lulu Press; 2011.
81. Sergeev M. Hybrid adaptive cooling schedules in simulated annealing for TSP. A performance study [Ph.D. thesis]. Amsterdam, The Netherlands: Vrije Universiteit Amsterdam; 2025.
82. Dorigo M, Stützle T. Ant colony optimization. Cambridge, MA, USA: MIT Press; 2004.
83. Heng H, Rahiman W. ACO-GA-based optimization to enhance global path planning for autonomous navigation in grid environments. IEEE Trans Evol Comput. 2025;30(1):226–40. doi:10.1109/tevc.2025.3543401.
84. Blum C, Roli A. Metaheuristics in combinatorial optimization: overview and conceptual comparison. ACM Comput Surv. 2003;35(3):268–308. doi:10.1145/937503.937505.
85. Agrawal A, Pal AK. Adaptive hybrid genetic-ant colony optimization for dynamic self-healing and network performance optimization in 5G/6G networks. Procedia Comput Sci. 2025;252(3):404–13. doi:10.1016/j.procs.2024.12.041.
86. Jin Y. A comprehensive survey of fitness approximation in evolutionary computation. Soft Comput. 2005;9(1):3–12. doi:10.1007/s00500-003-0328-5.
87. Wang Q, Li H, Zhang W, Zhang Y, Gong D, Chu F, et al. Surrogate-assisted evolutionary algorithm with adaptive local region search for high-dimensional expensive multi-objective optimization problems. Swarm Evol Comput. 2026;100(3):102232. doi:10.1016/j.swevo.2025.102232.
88. Pekel E. A hybrid surrogate-assisted evolutionary algorithm for technician routing and scheduling problems. Swarm Evol Comput. 2026;100(7):102280. doi:10.1016/j.swevo.2025.102280.
89. Jin Y, Olhofer M, Sendhoff B. A framework for evolutionary optimization with approximate fitness functions. IEEE Trans Evol Computat. 2002;6(5):481–94. doi:10.1109/tevc.2002.800884.
90. Đurasević M, Gil Gala FJ, Jakobović D, Mei Y. Genetic programming as a hyper-heuristic for solving combinatorial optimisation problems. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion; 2025 Jul 14–18; NH Malaga Hotel Malaga, Spain. p. 1070–94. doi:10.1145/3712255.3716516.
91. Juárez J, Falcón-Cardona JG, Ortiz-Bayliss JC. A steady state micro genetic algorithm for hyper-heuristic generation in one-dimensional Bin packing. Sci Rep. 2025;15(1):27220. doi:10.1038/s41598-025-10790-9.
92. Zhu Z, Zhou Y, Luo Q. A hyper-heuristic algorithm based on genetic and greedy strategy for university course scheduling problem. Appl Soft Comput. 2026;186(3):114150. doi:10.1016/j.asoc.2025.114150.
93. Burke EK, Hyde M, Kendall G, Ochoa G, Özcan E, Woodward JR. A classification of hyper-heuristic approaches. In: Handbook of metaheuristics. Boston, MA, USA: Springer; 2010. p. 449–68. doi:10.1007/978-1-4419-1665-5_15.
94. Burke EK, Kendall G, Soubeiga E. A tabu-search hyperheuristic for timetabling and rostering. J Heuristics. 2003;9(6):451–70. doi:10.1023/B:HEUR.0000012446.94732.b6.
95. El-Mihoub TA, Hopgood AA, Nolle L, Battersby A. Hybrid genetic algorithms: a review. Eng Lett. 2006;13(2):124–37.
96. Karaboga D, Akay B. A comparative study of Artificial Bee Colony algorithm. Appl Math Comput. 2009;214(1):108–32. doi:10.1016/j.amc.2009.03.090.

97. Sayah A, Aqil S, Lahby M. Hybrid metaheuristics-driven distributed task scheduling for latency-sensitive edge data processing. In: Proceedings of the 2025 12th IFIP International Conference on New Technologies, Mobility and Security (NTMS); 2025 Jun 18–20; Paris, France. p. 129–35. doi:10.1109/ntms65597.2025.11076755.
98. Sadrani M, Tirachini A, Antoniou C. Bus scheduling with heterogeneous fleets: formulation and hybrid metaheuristic algorithms. Expert Syst Appl. 2025;263(1):125720. doi:10.1016/j.eswa.2024.125720.
99. Juliany J, Vose MD. The genetic algorithm fractal. Evol Comput. 1994;2(2):165–80. doi:10.1162/evco.1994.2.2.165.
100. El-Shorbagy MA, El-Refaey AM. Hybridization of grasshopper optimization algorithm with genetic algorithm for solving system of non-linear equations. IEEE Access. 2020;8:220944–61. doi:10.1109/access.2020.3043029.
101. Hassen OA, Majeed HL, Hussein MA, Darwish SM, Al-Boridi O. Quantum machine learning for video compression: an optimal video frames compression model using qutrits quantum genetic algorithm for video multicast over the Internet. J Cybersecur Inf Manag. 2025;15(2):43–64. doi:10.54216/jcim.150205.
102. Nix AE, Vose MD. Modeling genetic algorithms with Markov chains. Ann Math Artif Intell. 1992;5(1):79–88. doi:10.1007/BF01530781.
103. Suzuki J. A Markov chain analysis on simple genetic algorithms. IEEE Trans Syst Man Cybern. 1995;25(4):655–9. doi:10.1109/21.370197.
104. Rudolph G. Convergence analysis of canonical genetic algorithms. IEEE Trans Neural Netw. 1994;5(1):96–101. doi:10.1109/72.265964.
105. Vose MD. The simple genetic algorithm: foundations and theory. Cambridge, MA, USA: MIT Press; 1999.
106. Peiravi A, Nourelfath M, Zanjani MK. Redundancy strategies assessment and optimization of k-out-of-n systems based on Markov chains and genetic algorithms. Reliab Eng Syst Saf. 2022;221(02):108277. doi:10.1016/j.ress.2021.108277.
107. Al Malki A, Rizk MM, El-Shorbagy MA, Mousa AA. Identifying the most significant solutions from Pareto front using hybrid genetic k-means approach. Int J Appl Eng Res. 2016;11(14):8298–311.
108. Wolpert DH, Macready WG. No free lunch theorems for optimization. IEEE Trans Evol Computat. 1997;1(1):67–82. doi:10.1109/4235.585893.
109. English TM. Evaluation of evolutionary and genetic optimizers: no free lunch. In: Evolutionary programming. Cambridge, MA, USA: MIT Press; 1996. p. 163–9.
110. Adam SP, Alexandropoulos SN, Pardalos PM, Vrahatis MN. No free lunch theorem: a review. In: Approximation and optimization. Cham, Switzerland: Springer International Publishing; 2019. p. 57–82. doi:10.1007/978-3-030-12767-1_5.
111. Wolpert DH. What is important about the No free lunch theorems? In: Black box optimization, machine learning, and no-free lunch theorems. Cham, Switzerland: Springer International Publishing; 2021. p. 373–88. doi:10.1007/978-3-030-66515-9_13.
112. Price GR. Selection and covariance. Nature. 1970;227(5257):520–1. doi:10.1038/227520a0.
113. Al Malki A, Rizk MM, El-Shorbagy MA, Mousa AA. Hybrid genetic algorithm with K-means for clustering problems. Open J Optim. 2016;5(2):71–83. doi:10.4236/ojop.2016.52009.
114. Wegener I. Towards a theory of randomized search heuristics. In: Mathematical foundations of computer science. Berlin/Heidelberg, Germany: Springer; 2003. p. 125–41. doi:10.1007/978-3-540-45138-9_7.
115. Qi X, Palmieri F. Theoretical analysis of evolutionary algorithms with an infinite population size in continuous space. Part I: basic properties of selection and mutation. IEEE Trans Neural Netw. 1994;5(1):102–19. doi:10.1109/72.265965.
116. Bodenhofer U. Genetic algorithms: theory and applications. Lecture Notes, Fuzzy Logic Laboratorium Linz-Hagenberg, Winter, 2004. 2003 [cited 2026 Jan 1]. Available from: http://www.flll.jku.at/div/teaching/Ga/notes.pdf.
117. Grefenstette J. Optimization of control parameters for genetic algorithms. IEEE Trans Syst Man Cybern. 1986;16(1):122–8. doi:10.1109/tsmc.1986.289288.
118. Smith JE, Fogarty TC. Operator and parameter adaptation in genetic algorithms. Soft Comput. 1997;1(2):81–7. doi:10.1007/s005000050009.

119. Doerr B, Doerr C. Optimal static and self-adjusting parameter choices for the (1+(λ, λ)) genetic algorithm. Algorithmica. 2018;80(5):1658–709. doi:10.1007/s00453-017-0354-9.
120. Angelova M, Pencheva T. Tuning genetic algorithm parameters to improve convergence time. Int J Chem Eng. 2011;2011(1):646917. doi:10.1155/2011/646917.
121. El-Shorbagy MA, El-Refaey AM. A hybrid genetic-firefly algorithm for engineering design problems. J Comput Des Eng. 2022;9(2):706–30. doi:10.1093/jcde/qwac013.
122. Fogarty TC. Varying the probability of mutation in the genetic algorithm. In: Proceedings of the 3rd International Conference on Genetic Algorithms (ICGA); 1989 Jun 4–7; San Francisco, CA, USA. p. 104–9.
123. Madebo NW. Enhancing intelligent control strategies for UAVs: a comparative analysis of fuzzy logic, fuzzy PID, and GA-optimized fuzzy PID controllers. IEEE Access. 2025;13(3):16548–63. doi:10.1109/access.2025.3532743.
124. Ma ZS, Krings AW. Dynamic populations in genetic algorithms. In: Proceedings of the 2008 ACM Symposium on Applied Computing; 2008 Mar 16–20; Fortaleza, Brazil. p. 1807–11. doi:10.1145/1363686.1364119.
125. Abdelkhalik O, Gad A. Dynamic-size multiple populations genetic algorithm for multigravity-assist trajectory optimization. J Guid Control Dyn. 2012;35(2):520–9. doi:10.2514/1.54330.
126. Tripathy A. Numerical optimization of computer models. J Oper Res Soc. 1982;33(12):1166. doi:10.1057/jors.1982.238.
127. Bäck T. April. Self-adaptation in genetic algorithms. In: Proceedings of the First European Conference on Artificial Life. Cambridge, MA, USA: MIT Press; 1992. p. 263–71.
128. Eiben AE, Hinterding R, Michalewicz Z. Parameter control in evolutionary algorithms. IEEE Trans Evol Computat. 1999;3(2):124–41. doi:10.1109/4235.771166.
129. Srinivas M, Patnaik LM. Adaptive probabilities of crossover and mutation in genetic algorithms. IEEE Trans Syst Man Cybern. 1994;24(4):656–67. doi:10.1109/21.286385.
130. Dozier G, Homaifar A, Tunstel E, Battle D. An introduction to evolutionary computation. In: Intelligent control systems using soft computing methodologies. Boca Raton, FL, USA: CRC Press; 2001. p. 365–80.
131. Fialho Á. Adaptive operator selection for optimization [Ph.D. thesis]. Paris, France: Université Paris Sud; 2010.
132. Harik GR, Lobo FG. A parameter-less genetic algorithm. GECCO. 1999;99:258–67.
133. Lobo FG, Goldberg DE. The parameter-less genetic algorithm in practice. Inf Sci. 2004;167(1–4):217–32. doi:10.1016/j.ins.2003.03.029.
134. Papa G. Parameter-less algorithm for evolutionary-based optimization: for continuous and combinatorial problems. Comput Optim Appl. 2013;56(1):209–29. doi:10.1007/s10589-013-9565-4.
135. Omar HA, El-Shorbagy MA. Modified grasshopper optimization algorithm-based genetic algorithm for global optimization problems: the system of nonlinear equations case study. Soft Comput. 2022;26(18):9229–45. doi:10.1007/s00500-022-07219-0.
136. Smith JE, Fogarty TC. Self adaptation of mutation rates in a steady state genetic algorithm. In: Proceedings of the IEEE International Conference on Evolutionary Computation; 1996 May 20–22; Nagoya, Japan. p. 318–23.
137. Arabas J, Kozakiewicz M. Adaptive population size for genetic algorithms. In: Proceedings of the IEEE Congress on Evolutionary Computation 2001; 2001 May 27–30; Seoul, Republic of Korea. p. 103–10.
138. Bäck T, Fogel DB, Michalewicz Z. Evolutionary computation 1. Bristol, UK: Institute of Physics Publishing; 2000.
139. Odetayo MO. Optimal population size for genetic algorithms: an investigation. In: IEE colloquium on genetic algorithms for control systems engineering. Washington, DC, USA: IET; 1993.
140. Ran X, Suyaroj N, Tepsan W, Lei M, Ma H, Zhou X, et al. A novel fuzzy system-based genetic algorithm for trajectory segment generation in urban global positioning system. J Adv Res. 2026;81(15):469–80. doi:10.1016/j.jare.2025.06.007.
141. Goldberg DE, Deb K. A comparative analysis of selection schemes. In: Foundations of genetic algorithms. San Francisco, CA, USA: Morgan Kaufmann; 1991. p. 69–93.
142. Blickle T, Thiele L. A comparison of selection schemes used in genetic algorithms. Zurich, Switzerland: ETH Zurich; 1995. Technical Report.
143. Mahfoud SW. Niching methods for genetic algorithms [Ph.D. thesis]. Urbana, IL, USA: University of Illinois, USA; 1995.

144. Terauchi A, Mori N. Evolutionary approach for autoaugment using the thermodynamical genetic algorithm. Proc AAAI Conf Artif Intell. 2021;35(11):9851–8. doi:10.1609/aaai.v35i11.17184.
145. Atashkari K, Nariman-Zadeh N, Pilechi A, Jamali A, Yao X. Thermodynamic Pareto optimization of turbojet engines using multi-objective genetic algorithms. Int J Therm Sci. 2005;44(11):1061–71. doi:10.1016/j.ijthermalsci.2005.03.016.
146. Kelly JD Jr, Davis L. A hybrid genetic algorithm for classification. In: Proceedings of the International Joint Conference on Artificial Intelligence; 1991 Aug 24–30; Sydney, Australia. p. 645–50.
147. Alkhrijah Y, Fahim M, Usman SM, Mehmood Q, Khalid S, Alawad MA, et al. Quantum genetic algorithm based ensemble learning for detection of atrial fibrillation using ECG signals. Comput Model Eng Sci. 2025;145(2):2339–55. doi:10.32604/cmes.2025.071512.
148. AlSalman H, Alfakih T, Al-Rakhami M, Hassan MM, Alabrah A. A genetic algorithm-based optimized transfer learning approach for breast cancer diagnosis. Comput Model Eng Sci. 2024;141(3):2575–608. doi:10.32604/cmes.2024.055011.