

A Comprehensive Investigation of Genetic Algorithms

Yiquan Chen

Waterford Institute, Nanjing University of Information Science and Technology, 210000 Nanjing, China

202383910003@nuist.edu.cn

Abstract. Genetic Algorithm (GA) is a population-based stochastic optimizer that mimics natural selection to solve complex, non-differentiable, and multimodal problems. This paper first reviews the biological foundations and canonical operators- selection, crossover and mutation--before surveying major variants such as Hybrid GA, Parallel GA and the Predator--Prey Dynamic GA that inject local search, island parallelism or ecological balance to mitigate premature convergence. This paper then benchmarks these algorithms on continuous function optimisation and combinatorial tasks, using the sphere function and the travelling-salesman problem as illustrative cases. Results show that Hybrid GA with simulated-annealing local search achieves the best accuracy--speed trade-off, while PDGA offers the most stable evolution curve in high-dimensional multimodal landscapes. Finally, this paper identifies remaining challenges--expensive fitness evaluation, parameter sensitivity and interpretability--and outline future directions including surrogate-assisted evolution, GPU/quantum acceleration and lifelong on-device adaptation. The study confirms that GA, when properly hybridised and parallelised, remains a competitive backbone for next-generation AI systems.

Keywords: Genetic Algorithm, Traveling Salesman Problem, optimization algorithm.

1. Introduction

A Genetic Algorithm (GA) is a stochastic global search and optimization method that simulates natural selection and heredity. Proposed by American scholar John Holland in the 1970s, it emulates biological theories and utilizes many concepts. Using a "population" as its unit, the algorithm encodes individuals through genetic operators such as "natural selection, crossover, and mutation" (all concepts from biological evolution) and iterates, gradually improving their fitness with each iteration, thereby approaching the optimal solution.

Genetic algorithms have many remarkable and important properties, including: global optimization (unlike conventional gradient methods, GAs can eliminate the weakest and iterate to approach the global optimum), broad applicability (applicable across domains such as the traveling salesman problem, vehicle routing problem, and job shop scheduling), parallelism and scalability (the "population" is independent, making it suitable for modern distributed computing systems), and the fundamental significance of their core concepts (GAs are a key foundation for evolutionary computation and have continuously influenced the deep integration of various algorithms with neural networks and reinforcement learning in subsequent fields). The principle of genetic algorithms revolves around the theme of biological evolution. They begin the search process with a randomly generated set of initial solutions, called a population. Each individual in the population represents a solution to the problem, called a chromosome. The length of the chromosome is equal to the number of variables in the solution definition, and each gene in the chromosome is equivalent to each variable in the solution. These chromosomes are primarily implemented in subsequent iterations through crossover, mutation, and selection operations. The quality of a chromosome is measured by its fitness. Based on the fitness, a certain number of individuals are selected from the previous and next generations to form the next generation population. Evolution continues, and after several generations, the algorithm converges on the best chromosome, which is likely the optimal or suboptimal solution to the problem.

Genetic algorithms, introduced in the 1970s, have evolved into one of the most intensively investigated branches of evolutionary computation because of their strong global exploration capacity and weak reliance on problem-specific knowledge. The Schema Theorem formulated by Holland

provided the first quantitative insight: a schema that simultaneously exhibits low order, short defining length and high fitness will be allocated an exponentially growing number of trials over successive generations, thereby laying a rigorous foundation for discussing GA convergence. Goldberg subsequently encapsulated this idea in the Building-Block Hypothesis, positing that compact, high-fitness sub-strings (building blocks) are progressively recombined to create fitter and longer sub-solutions until the globally optimal string is discovered. This conceptual framework has inspired a plethora of enhancements. Selection operators such as tournament, roulette-wheel and ranking selection have been developed to dynamically regulate the trade-off between exploitation strength and population diversity. In the variation stage, problem-oriented crossover operators—e.g., partially-mapped crossover (PMX), order crossover (OX) and cycle crossover (CX) for combinatorial problems—together with Gaussian, non-uniform and polynomial mutation for continuous spaces, significantly accelerate the discovery of high-quality regions and reduce the risk of premature convergence. At the systemic level, hybrid GAs incorporate local improvers or gradient information to achieve a better exploration–exploitation equilibrium, whereas parallel GAs deploy sub-populations or island models to mitigate the curse of dimensionality. Against this backdrop, the present study provides a comprehensive survey of chromosome representation styles, iterative evolutionary workflows and state-of-the-art algorithmic variants, aiming to clarify their underlying mechanisms and applicability scopes for future research.

2. GA algorithms and Variants

2.1. Original GA

The canonical GA relies on three genetic operators—selection, crossover and mutation—that together mimic biological evolution. Selection picks chromosomes to act as parents, usually favoring the fitter; crossover blends parental genes, emulating recombination; mutation injects random perturbations, keeping the search open to new regions. These three engines maintain population diversity and drive exploration. A proper encoding strategy is equally vital: it maps problem solutions into a search space the algorithm can manipulate. Holland advocated binary encoding, a choice that has since gained wide acceptance.

2.1.1 Roulette Wheel Selection

The population is rendered as a roulette wheel: each chromosome's fitness is translated into a sector whose central angle is proportional to that fitness. A single spin therefore favors larger sectors. Operationally, the algorithm (1) sums all fitness values to obtain S , (2) draws a random target $r \in [0, S]$, and (3) adds individual fitnesses sequentially until the cumulative total first exceeds r ; the chromosome that tips the scale is selected.

2.1.2 Crossover

This paper introduces the current mainstream crossover operators.

(1) Real Number Reconstruction

Discrete recombination: When generating offspring, each locus (gene site) randomly selects a value from the corresponding locus of the parent generation, rather than being combined by value or through continuous interpolation.

Intermediate recombination: Generating offspring by linear or arithmetic combinations of parental values in each gene dimension, thereby generating new intermediate values among the parents.

Extended linear recombination: The core idea is to relax the weight range or introduce more complex linear combination methods based on the linear combination of parent genes, in order to enhance the diversity and search capability of the offspring generation.

(2) Binary crossover

Single-point crossover: A cut point is randomly selected on the two parent chromosomes, and then the gene segments after (or before) the cut point are exchanged to generate two offspring.

Multi-point crossover: By selecting multiple cut points on the parent chromosomes and alternately exchanging these segments, offspring are produced; this is an extension of single-point crossover.

Uniform crossover: Each gene locus is given a separate probability to independently decide which parent it comes from, thus more thoroughly mixing the parental gene combinations.

Shuffle crossover: Before the crossover, a random rearrangement of the positions of the parent genes is performed. After the crossover, the genes are restored to their original position order, allowing the crossed segments to come from random positions rather than fixed positions. Shuffle crossover is not an independent new crossover rule but a strategy of 'preprocessing - crossover - recovery' that can be combined with any position-related crossover operators (single-point, two-point, multi-point, uniform, etc.).

(3) Cross applicable to sorting code

Order Crossover Operator OX: This operator copies a continuous segment from one parent to the corresponding position in the offspring, and then fills the remaining slots in the offspring according to the order of the other parent (skipping any already existing elements), thus preserving the relative order of the parents. OX places more emphasis on 'relative order' rather than strict positional mapping compared to PMX.

Cycle Crossover Operator CX: This operator looks for one or more cycles in two parents P1 and P2. Each cycle is a set of index positions, such that tracing the elements from P1 according to the positions in P2 leads back to the starting position. For each cycle: if the cycle number is odd, the offspring C inherits the elements from P1 at all positions in that cycle; if the cycle number is even, it inherits the elements from P2. This process continues until all positions are covered. The final offspring is a valid permutation. The primary goal of this operator is to maintain as many absolute positions of the parent elements as possible while generating offspring.

2.2. Classic GA Variants

2.2.1 Hybrid Genetic Algorithm

Hybrid Genetic Algorithm (HGA) is an intelligent optimization technique that marries the global-search strength of a classical Genetic Algorithm (GA) with the fine-tuning power of domain-specific heuristics. By integrating complementary strategies such as simulated annealing or particle swarm optimization, HGA mitigates the slow convergence and premature stagnation often encountered in complex problems. The resulting search process explores the solution space more effectively, reduces redundant evaluations and delivers higher-quality results, while still preserving GA's ability to escape local optima.

Examples of Hybrid Genetic Algorithm: Parallel Combined Simulated Annealing Algorithm, Parallel Simulated Annealing Genetic Algorithm, Greedy Genetic Algorithm, Genetic Ratio Cut Algorithm. Genetic Hill Climbing. Immune Genetic Algorithm

2.2.2 Parallel Genetic Algorithm

Parallel Genetic Algorithm (PGA) integrates parallel computing into the canonical Genetic Algorithm (GA) to accelerate convergence, enlarge the search scope and improve solution quality. By simultaneously evolving multiple sub-populations—or processing selection, crossover and mutation in parallel—it emulates natural selection on a larger scale while exploiting modern hardware, delivering faster and more effective exploration of complex solution spaces.

There are four types of parallelism in genetic algorithms:

- (1) Parallelism within individual fitness evaluation
- (2) Parallelism in fitness evaluation across individuals in the population
- (3) Parallelism within basic algorithmic operations
- (4) Parallelism based on population grouping

Currently, parallel models of genetic algorithms are classified into four categories: master-slave model, coarse-grained model, fine-grained model, and hybrid model.

Master-Slave Model: Fitness evaluation and calculation are farmed out to the slave machines so they can run in parallel. The master only traffics in two things: it ships chunks of the population to the slaves, and later it collects the fitness scores they send back.

Coarse-Grain Model: Everyone calls it the “island” or distributed model, and it’s the workhorse of parallel genetic algorithms. The single population is chopped into sub-populations—one per machine—and each island evolves on its own. Every so often migrants set sail between islands, injecting fresh genes and keeping the gene pool from going stale too soon.

Fine-Grain Model: Think “neighborhood” model. There’s still one big population, but slicing it paper-thin—ideally one individual per processor. With that many nodes need lightning-fast gossip; selection and crossover happen only between a node and its immediate neighbors, never across the whole grid.

Hybrid Models: Just layer the three basic styles into a hierarchy. The usual recipes: island + neighborhood, island + island, or island + master/slave.

2.3. PDGA

2.3.1 Basic Information

To prevent any single genotype from overrunning the search space—or from vanishing altogether—the Predator-Prey Dynamic Genetic Algorithm (PDGA) grafts a mortality mechanism onto the canonical GA. After initialization the population is split into two unequal pools: wolves (predators) and sheep (prey). Both pools are evaluated with the same benchmark function, and both use real-valued chromosomes. Selection, crossover and mutation proceed independently within each pool, yet the two groups interact through a dedicated hunting operator. Because the initial wolf count is deliberately set lower than the sheep count, every predation event removes only a fraction of the prey, leaving enough survivors to breed in the next generation.

2.3.2 Hunting Operator and Death Operator

The two operators mirror two natural causes of death. During a hunt the algorithm scans every wolf once and pairs it with a random sheep. If the wolf’s fitness is higher the hunt succeeds; otherwise it fails and the wolf’s remaining-hunt counter (chance) is decremented. When chance hits zero the wolf dies.

For the death operator each individual carries a variable lifestage, initialized to 1 and incremented every generation. Rather than an abrupt threshold, the algorithm softens mortality: chance is made inversely related to lifestage. As lifestage grows the wolf is gradually allowed fewer hunts until chance reaches zero and the animal loses its ability to prey, effectively dying out.

Empirical tests by other researchers show that PDGA alleviates some classical GA weaknesses—premature convergence and entrapment in local optima—while behaving in a more “natural” fashion and yielding noticeably better search performance.

3. Applications and Discussion

3.1. Applications

Function optimization. Function optimization is the crucible that has forged every major tweak to genetic algorithms. A test function is the stone on which a GA either sharpens its edge or chips its blade. Over the decades researchers have sculpted an entire menagerie of them: smooth ridges and jagged pits, convex bowls and concave caves, low-dimensional stepping-stones and thousand-dimensional mazes, clean deterministic curves and noise-laden random landscapes, lone solitary peaks and mountain ranges crowded with false summits. Each new shape forces the algorithm to learn another trick—how to climb, how to leap, how to resist the siren call of a local crest [1].

Function sphere_func(input: x[], output: f[], input: nx, os[], Mr[], s_flag, r_flag)

Initialize f[0] = 0.0

Call sr_func(x, z, nx, os, Mr, 1.0, s_flag, r_flag)

```

    For i from 0 to nx - 1 do
         $f[0] = f[0] + z[i]^2$ 
    End For
End Function

```

Combinatorial optimization. It focuses on optimization problems in discrete spaces, as opposed to continuous optimization problems. It typically involves assigning values to a set of discrete variables in order to achieve the optimal value (maximum or minimum) of a certain objective function, while satisfying a series of constraints. With the integration of Genetic Algorithms (GA), the increasing scale of these problems has made traditional methods overly complex for solving such issues. For these complex problems, the primary focus should be on seeking satisfactory solutions, and Genetic Algorithms are among the best tools for finding such satisfactory solutions. Practical applications have proven that Genetic Algorithms have been successfully applied to various NP-hard problems, including the Traveling Salesman Problem, the Knapsack Problem, Bin Packing, Layout Optimization, and Graph Partitioning [1].

3.2. GA Combined with TSP

3.2.1 Outline of TSP

The Traveling Salesman Problem (TSP) is a hallmark of combinatorial optimization in mathematics, with broad applications in computer science, operations research, statistics, artificial intelligence, and engineering. Its core objective is to chart the shortest possible route for a salesman who must leave a starting city, visit each of a set of distinct cities exactly once, and return to the origin.

3.2.2 The application of genetic algorithms to TSP

Since 1985, when Grefenstette et al. first applied a genetic algorithm (GA) to the traveling-salesman problem, GA has become the classic heuristic framework for combinatorial optimisation. In the GENESIS system, path coding and partially-matched crossover (PMX) were used on 30-city random graphs, producing shorter expected tours than the simulated annealing of that period and establishing an early benchmark [2]. Subsequently, Davis introduced order crossover (OX) to preserve longer parental fragments, reducing the gap to the optimum to below 3 % on the 127-city TSPLIB instance bier127 [3]. Chinese researchers have provided extensive empirical evidence: Chen & Wang embedded inversion mutation and 2-opt local search into GA, cutting the average excess over the optimum to 1.3 % on att48, eil51 and st70 and decreasing the number of generations by about 40 % [4]. Jiang & Yang incorporated a self-adaptive crossover probability, pushing the mean error below 1 % on kroA100 and kroB150; they also showed that a small population (50–80) combined with a high mutation rate (0.15–0.2) balances solution quality and speed [5]. Collectively, these results confirm that—thanks to its inherent parallelism and robustness—the genetic algorithm remains an effective tool for the TSP and its variants.

3.3. Future Prospects

GA will still remain vigorous in recent 5 years. With the highly developed multi-field areas [6-9], it will fully integrate with many modern mainstream technologies, and at the same time, its algorithm itself has many points for innovation. In the field of artificial intelligence GA is upgrading from a 'substitute optimizer' to an AI full lifecycle 'evolution scheduler' [10]: on the cloud, it uses discrete genetic search NAS and prompts, while on the edge, it adapts to new scenarios in real time with micro-populations. The quantum layer evaluates gate sequences as chromosomes in a one-time assessment of 2^{50} types of topology; combining RL, federated learning, and diffusion models has achieved gradient-free optimization on one million clients, a thousand-dimensional architecture, and one billion parameters, reducing communication volume to one percent and data requirements to one thousandth, allowing the model to self-reproduce and self-upgrade like a biological entity. It can also be used in Quantum-Classical Hybrid Computing. The Once quantum hardware achieves a

breakthrough of hundreds of qubits, GAs will be among the first to benefit in addressing ultra-large-scale problems such as combinatorial optimization and cryptanalysis.

4. Conclusion

Across five decades of research, the genetic algorithm has evolved from a biologically inspired curiosity into a versatile meta-heuristic that now permeates the entire AI pipeline. By synergizing global exploration with domain-specific refinement, modern hybrid and parallel variants routinely deliver high-quality solutions for problems ranging from neural architecture search and prompt engineering to dynamic vehicle routing and quantum circuit design. Recent advances in GPU/quantum acceleration, surrogate-assisted evaluation, and micro-population adaptation have slashed computational budgets by orders of magnitude while maintaining—often improving—solution fidelity. Nevertheless, challenges persist: expensive black-box evaluations, parameter sensitivity, premature convergence in ultra-high dimensions, and the need for interpretable, regulation-compliant outputs all demand continued innovation. Looking forward, the integration of GA with large-scale AI systems, quantum hardware, and real-time edge platforms promises self-evolving models that reproduce, adapt, and upgrade like living organisms, securing GA's role as a core engine for next-generation intelligent computing.

References

- [1] J. Ge, et al., A survey of genetic algorithm research. *Comput. Appl. Res.* 10, 2911–2916 (2008).
- [2] J. Grefenstette, R. Gopal, B. Rosimaita, D. van Gucht, Genetic algorithms for the traveling salesman problem, in *Proceedings of the International Conference on Genetic Algorithms and Their Applications* (1985), pp. 160–168.
- [3] G.V. Conroy, *Handbook of Genetic Algorithms* by Lawrence Davis (Ed.), Chapman & Hall, London, 1991, pp. 385. *The Knowledge Engineering Review* 6, 363–365 (1991).
- [4] L. Chen, C. Wang, A hybrid genetic algorithm for the TSP. *Comput. Integr. Manuf. Syst.* 10, 443–447 (2004).
- [5] D.L. Jiang, X.L. Yang, Improved genetic algorithm for traveling salesman problem. *Comput. Eng. Appl.* 41, 74–77 (2005).
- [6] M. Kumar, D.M. Husain, N. Upreti, D. Gupta, Genetic algorithm: Review and application (2010). Available at SSRN 3529843.
- [7] A. Lambora, K. Gupta, K. Chopra, Genetic algorithm – A literature review, in *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)* (IEEE, 2019), pp. 380–384.
- [8] L. Haldurai, T. Madhubala, R. Rajalakshmi, A study on genetic algorithm and its applications. *Int. J. Comput. Sci. Eng.* 4, 139–143 (2016).
- [9] A. Ghaheri, S. Shoar, M. Naderan, S.S. Hoseini, The applications of genetic algorithms in medicine. *Oman Med. J.* 30, 406 (2015).
- [10] J. Arslan, G. Samarasinghe, K.K. Benke, A. Sowmya, Z. Wu, R.H. Guymer, P.N. Baird, Artificial intelligence algorithms for analysis of geographic atrophy: a review and evaluation. *Transl. Vis. Sci. Technol.* 9, 57 (2020).