**A Hybrid Binary Genetic Algorithm with Gray Coding and Non-Uniform Mutation for Global Optimization**

**October 20, 2025**

**Abstract**

This report analyzes an advanced Hybrid Binary Genetic Algorithm (BGA) designed for complex global optimization problems. The algorithm introduces several key improvements over a standard BGA: (1) **Gray coding** to ensure locality and mitigate the "Hamming cliff" problem, (2) a **non-uniform mutation** operator that adaptively shifts from global exploration to local exploitation, (3) a **memetic component** via a local hill-climbing search on the best individual, and (4) a **stagnation reset** mechanism to maintain diversity in high-dimensional search spaces. The algorithm was evaluated on four standard benchmark functions (Rastrigin, Griewangk, Rosenbrock, Michalewicz) across three dimensionalities (2, 30, and 100). An extensive grid search, analyzing 360 parameter combinations with 30 independent runs each, was conducted. The results demonstrate that the hybrid approach is highly effective. The stagnation reset mechanism proved to be the fundamental driver of progress in high-dimensional search spaces. Notably, **Tournament selection** consistently outperformed Roulette Wheel selection, and the optimal initial mutation rate was found to be highly dependent on the problem's landscape.

**1. Introduction**

Global optimization, the task of finding the absolute minimum or maximum of a function over a given domain, is a fundamental challenge in science and engineering. Many real-world problems are characterized by non-linear, non-convex, and high-dimensional search spaces with numerous local optima. Traditional gradient-based methods often fail in this context, becoming trapped in the nearest local optimum.

Evolutionary Algorithms (EAs), particularly Genetic Algorithms (GAs), have emerged as robust metaheuristics for global optimization [1]. Binary Genetic Algorithms (BGAs), the classical implementation, encode solutions as binary strings. However, standard BGAs suffer from two primary drawbacks:

1. **Premature Convergence:** The population may lose diversity too quickly, converging on a suboptimal solution.
2. **The "Hamming Cliff":** In standard binary encoding, adjacent integers (e.g., 7 and 8) can have vastly different binary representations (0111 vs. 1000). A single-bit mutation can cause a massive leap in the phenotypic search space, disrupting the principle of locality (i.e., small genotypic changes should lead to small phenotypic changes) [2].

The algorithm under investigation, provided in Improved Binary GA Runner.py, is a hybrid BGA specifically engineered to address these weaknesses. It integrates four key strategies: Gray coding, non-uniform mutation, memetic local search, and stagnation-based re-initialization.

This report's objective is to formally describe this algorithm's architecture and evaluate its performance based on the aggregated results from an exhaustive experimental study, as documented in aggregated\_summary\_binary\_improved.csv and the provided performance graphs.

**2. Methodology**

The core of the investigation is the bga\_improved function. This function implements a generational GA with elitism, but its behavior is defined by the following specialized components.

**2.1. Encoding and Decoding: Gray Coding**

To solve the "Hamming cliff" problem, the algorithm eschews standard binary encoding. Instead, it uses **Gray coding** [2]. In a Gray code, the representations of any two consecutive integers differ by only a single bit. This property enforces locality, allowing the mutation operator to function as a minimal neighborhood search. The conversion from a Gray-coded integer to a standard binary integer is handled by the gray\_to\_binary\_vectorized function, which uses a series of bitwise XOR operations.

# Source: Improved Binary GA Runner.py

def gray\_to\_binary\_vectorized(gray\_codes):

""" Converts a matrix of integers in Gray code to standard binary integers. """

mask = gray\_codes >> 1

binary\_codes = gray\_codes.copy()

while np.any(mask):

binary\_codes ^= mask

mask >>= 1

return binary\_codes

**2.2. Genetic Operators**

* **Selection:** The experimental setup tests two methods: **Tournament Selection** (with k=3) and **Roulette Wheel Selection**.
* **Crossover:** A **Uniform Crossover** operator is used (crossover\_uniform\_vectorized).
* **Mutation: Non-Uniform Operator:** This is the algorithm's primary mechanism for balancing exploration and exploitation. The mutation rate is not static; it is calculated dynamically each generation based on the formula proposed by **Michalewicz** [3]: $r\_{mut} = r\_{final} + (r\_{initial} - r\_{final}) \times (1 - \frac{gen}{n\_{iter}})^{b}$ This operator begins with a high mutation rate ($r\_{initial}$) for global exploration and decays exponentially to a small floor rate ($r\_{final}$) for local exploitation and fine-tuning. The implementation calculates this effective\_r\_mut at the start of each generation.

# Source: Improved Binary GA Runner.py

# Inside the main generational loop...

progress = gen / n\_iter

effective\_r\_mut = r\_mut\_final + (r\_mut\_initial - r\_mut\_final) \* (1 - progress) \*\* decay\_factor

**2.3. Hybridization and Diversification**

The algorithm is a hybrid framework that incorporates two additional strategies.

* **Memetic Component (Hill Climbing):** After the evolutionary cycle, the single best individual is subjected to a stochastic hill-climbing search (hill\_climbing\_gray). This function tests a limited number of single-bit flips in the elite chromosome, immediately accepting any change that improves the score. This combination of a population-based global search (GA) with an individual local search classifies this as a **Memetic Algorithm (MA)** [4].
* **Stagnation Reset:** For high-dimensional problems (D=30 and D=100), if the best score does not improve for 50 consecutive generations, the population is re-initialized, preserving only a 10% elite. This re-introduces diversity to escape local optima.

# Source: Improved Binary GA Runner.py

# Inside the main generational loop...

if stagnation\_limit is not None and stagnation\_counter >= stagnation\_limit:

stagnation\_counter = 0

n\_elite = int(0.1 \* n\_pop)

sorted\_indices = np.argsort(scores)

elite\_pop = pop[sorted\_indices[:n\_elite]]

new\_individuals = np.random.randint(0, 2, size=(n\_pop - n\_elite, chromosome\_len))

pop = np.vstack([elite\_pop, new\_individuals])

**2.4. Experimental Setup**

The algorithm's performance was benchmarked using a comprehensive grid search on four functions chosen for their diverse topologies:

1. **Rastrigin** [5]: Highly multimodal, a test of exploration.
2. **Griewangk** [6]: Multimodal with a global funnel structure.
3. **Rosenbrock** [7]: Unimodal with a narrow, parabolic valley, a test of exploitation.
4. **Michalewicz** [3]: Multimodal with steep ridges and valleys.

The parameter grid included variations in dimensions (2, 30, 100), selection method, crossover rate (0.7, 0.8, 0.9), and mutation multiplier (1.0, 2.0, 4.0). Each of the 360 unique experiments was repeated 30 times for statistical validity.

**3. Results and Discussion**

The aggregated results and performance graphs provide clear insights into the algorithm's behavior, robustness, and scalability.

**3.1. The Stagnation-Reset Search Dynamic**

For all high-dimensional problems (D=30 and D=100), the stagnation reset mechanism was the fundamental driver of progress. Instead of a smooth convergence, the algorithm exhibits a characteristic "stair-step" pattern. It explores, finds a promising region, stagnates on a local optimum, and then re-initializes the population, allowing it to "jump" to a new, better region.

This behavior is clearly visible in the convergence plots for the 100-dimension Rastrigin and Rosenbrock functions (Figure 1). For Rastrigin, a highly multimodal function, the resets allow for broad exploration. For Rosenbrock, a unimodal function with a difficult valley, the resets help the algorithm escape the valley walls and re-attempt to find the minimum. This shows the mechanism is a versatile and essential strategy, not an anomaly.

*[Insert Figure 1 here: A 1x2 panel showing both Top1\_Rastrigin\_C0.9\_M1.0\_conv.png and Top1\_Rosenbrock\_C0.9\_M4.0\_conv.png]*

**3.2. Impact of Selection Method**

Across almost all functions and dimensionalities, **Tournament selection consistently and significantly outperformed Roulette Wheel selection**, as evidenced by the mean final scores in the aggregated\_summary\_binary\_improved.csv data.

A direct visual comparison on the 100-D Rastrigin function (Figure 2) illustrates this. While both methods exhibit the stagnation-reset dynamic, the run using Tournament selection achieves a noticeably better final score. This is likely because Tournament selection's lower selection pressure helps maintain population diversity for longer, which is crucial for multimodal problems.

*[Insert Figure 2 here: A 1x2 panel showing Conv\_Rastrigin\_D100\_Tournament\_C0.9\_M1.0.png and Conv\_Rastrigin\_D100\_Roulette\_C0.9\_M1.0.png]*

**3.3. Impact of Non-Uniform Mutation**

The optimal initial mutation rate (controlled by the multiplier, M) was highly function-dependent.

* **Rosenbrock** strongly favored the **highest mutation multiplier (M=4.0)**, as seen in its top-performing runs. The aggressive initial exploration is crucial for locating the narrow valley in the vast search space.
* **Rastrigin and Michalewicz** performed best with a **low mutation multiplier (M=1.0)**. For these functions, excessive mutation appears to prevent the algorithm from effectively exploiting the basins of attraction of the best local optima.
* **Griewangk** performed best with a **medium multiplier (M=2.0)**, balancing the need to explore past local optima while exploiting the function's global funnel structure.

This confirms the non-uniform mutation strategy is effective, but requires tuning to the problem's landscape.

**3.4. Algorithm Scalability: A Case Study on Rastrigin**

The set of plots for the Rastrigin function at 2, 30, and 100 dimensions vividly illustrates the "curse of dimensionality."

The convergence plots (Figure 3) show a dramatic shift in behavior. The 2-D problem is solved trivially in under 20 generations. The 30-D problem requires the stagnation reset mechanism to find a near-optimal solution. The 100-D problem is characterized by frequent resets and fails to reach the global optimum, highlighting the exponential increase in difficulty.

*[Insert Figure 3 here: A 1x3 panel showing Scale\_Conv\_Rastrigin\_D2\_Tournament.png, Scale\_Conv\_Rastrigin\_D30\_Tournament.png, and Scale\_Conv\_Rastrigin\_D100\_Tournament.png]*

The boxplots of the final scores (Figure 4) demonstrate the corresponding decline in consistency. At D=2, every run finds the exact global minimum. At D=30, consistency is still extremely high. At D=100, the variance increases significantly, showing that while the algorithm is still making progress, the outcome of any single run is far less predictable.

*[Insert Figure 4 here: A 1x3 panel showing Scale\_Box\_Rastrigin\_D2\_Tournament.png, Scale\_Box\_Rastrigin\_D30\_Tournament.png, and Scale\_Box\_Rastrigin\_D100\_Tournament.png]*

**3.5. Robustness and Consistency**

Despite the challenges of high dimensionality, the algorithm proved to be remarkably robust. The boxplots for the four 100-dimension benchmark functions (Figure 5), using one of their best-performing parameter sets, show tight interquartile ranges. This indicates that the algorithm's performance is reliable and not the product of chance. Even when it does not find the global optimum (as in Rastrigin D-100), it consistently finds solutions of a similar, predictable quality across multiple independent runs. The extremely compressed boxes for Griewangk and Michalewicz demonstrate exceptional stability on those functions.

*[Insert Figure 5 here: A 2x2 panel showing Box\_Rastrigin\_D100\_Tournament\_C0.9\_M1.0.png, Box\_Rosenbrock\_D100\_Tournament\_C0.9\_M4.0.png, Box\_Griewangk\_D100\_Tournament\_C0.9\_M2.0.png, and Box\_Michalewicz\_D100\_Tournament\_C0.9\_M1.0.png]*

**4. Conclusion**

The analyzed algorithm is a sophisticated, high-performance memetic framework that successfully overcomes the primary weaknesses of standard Binary GAs. The strategic combination of Gray coding, a memetic hill-climbing stage, non-uniform mutation, and a stagnation-driven reset mechanism creates a powerful and robust optimizer.

The key findings are:

1. The **stagnation reset mechanism** is the primary driver of progress in high-dimensional search spaces, creating a distinctive and effective "stair-step" convergence pattern.
2. **Tournament Selection** is demonstrably more robust and reliable than Roulette Wheel selection for these complex problems, likely due to its ability to preserve diversity.
3. The **Non-Uniform Mutation** operator is a powerful tool for balancing exploration and exploitation, though its optimal initial rate is highly dependent on the topology of the specific problem.
4. The algorithm demonstrates **excellent scalability** up to moderately high dimensions (D=30) but, like all optimizers, faces significant challenges in very high-dimensional (D=100) and highly multimodal search spaces.

This hybrid algorithm represents a well-engineered and effective approach to global optimization, showcasing the synergistic benefits of combining multiple heuristic strategies.

**5. References**

[1] J. H. Holland, *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: University of Michigan Press, 1975.

[2] L. D. Whitley, "A genetic algorithm tutorial," *Statistics and Computing*, vol. 4, no. 2, pp. 65–85, 1994.

[3] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*. Berlin, Heidelberg: Springer-Verlag, 1992.

[4] P. Moscato, "On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms," *Caltech Concurrent Computation Program, C3P Report 826*, 1989.

[5] L. A. Rastrigin, "Systems of extremal control," Mir, Moscow, 1974.

[6] A. O. Griewank, "Generalized descent for global optimization," *Journal of Optimization Theory and Applications*, vol. 34, no. 1, pp. 11–39, 1981.

[7] H. H. Rosenbrock, "An automatic method for finding the greatest or least value of a function," *The Computer Journal*, vol. 3, no. 3, pp. 175–184, 1960.