A Hybrid Approach: Genetic Algorithm and Hill Climbing for Binary Optimization

Adriana Vlad
Faculty of Computer Science
Computational Optimization and Artificial Intelligence
Iasi, Romania
adriana.vlad157@gmail.com

Andreea-Daniela Lupu
Faculty of Computer Science
Computational Optimization and Artificial Intelligence
Iasi, Romania
dandreaalupu@yahoo.com

***Abstract*—This paper explores the performance of a Genetic Algorithm (GA) enhanced with a Hill Climbing (HC) heuristic, tested on benchmark functions including Rastrigin's, Griewangk's, Rosenbrock's, and Michalewicz's. The study evaluates the hybrid GA-HC approach across three dimensions (2, 30, and 100) and compares it to the standard GA. The results indicate that the addition of Hill Climbing improves solution stability, particularly in higher-dimensional problems, but at the cost of increased computational time. While the Hill Climbing provides marginal improvements in lower dimensions, its effectiveness is more pronounced in complex optimization landscapes. This study suggests the potential of combining GA with other techniques to further enhance optimization performance and proposes future investigations into the refinement of Hill Climbing configurations for broader applicability.**

***Index Terms*—genetic algorithm, hill climbing, optimization, tournament selection, dynamic mutation rate**

I. Introduction

In recent years, Genetic Algorithms (GAs) have been widely applied in numerical optimization due to their ability to explore large and complex search spaces. In this work, a hybrid optimization approach that combines a Genetic Algorithm with a Hill Climbing (HC) heuristic is proposed to improve solution quality in solving multidimensional numerical optimization problems.

Specifically, a binary representation is utilized to address optimization challenges. A Genetic Algorithm and its hybridised Hill Climbing version are tested on well-known benchmark functions—Rastrigin's, Griewangk's, Rosenbrock's, and Michalewicz's—at dimensions of 2, 30, and 100; the goal is to find the minima of these functions.

The Hill Climbing heuristic from this paper follows a best-improvement approach, as bits are flipped iteratively in order to maximize the fitness function, until a local optimum is reached or a predefined number of steps, signifying the depth of the search, are completed. Hill Climbing has a chance to applied at the end of every 10th generation to the top 5% of the population, as well as 5% randomly selected members of the population, multiple times, across multiple generations, in a dynamic manner, making the number of allowed steps slightly increase as the generations progress and taking incline into consideration.

For the selection process the top 5% of the population is transferred over to the next population, the rest of it's members being chosen via Tournament selection, where a small subset of chromosomes, composed of 5 randomly chosen ones, competes to populate the next generation, further increasing selection pressure on higher-quality solutions.

After the tournament selection, the GA processes the population in chunks using parallelization, performing a three-cut-point crossover and adaptive mutation to enhance search performance. The crossover mechanism, specifically a three-cut-point crossover, enables greater diversity in offspring by swapping segments of three distinct points between two parent chromosomes. This approach promotes a balanced exploration of the search space by generating varied offspring configurations, helping to avoid premature convergence.

II. Solution Representation

In this GA, each solution, known as a *chromosome.x*, is encoded as a binary string. Each chromosome also includes a *chromosome.fitness* value, which represents how well the solution performs in relation to the optimization objective. This fitness value is derived from the function's calculated value, which is stored separately as `chromosome.f` within the chromosome structure. More specifically, since the goal is to minimize all the functions, the fitness is computed as $1/chromosone.f$. The only exception to this is the 4th function, which is negatively defined and thus requires further modification to obtain positive fitness values. For this we add 200 to $chromosome.f$ before computing the inverse.
The binary encoding of *chromosome.x* allows the search space to be discretized to a specified precision. Specifically, a solution's search space, defined by the interval $[a, b]$, is divided into $N = (b-a)\cdot 10^d$ equal subintervals, where d represents the desired decimal precision. To represent all possible $(b-a)\cdot 10^d$ values, we require n bits, where:

$$n = \lceil \log_2(N) \rceil \tag{1}$$

The total length of the bit string encoding a solution is thus the sum of bit lengths required for each parameter of the function being optimized.

Each chromosome's binary string is divided into segments, with each segment corresponding to a parameter in the optimization problem. In order to evaluate the solution (i.e., calculate its fitness), each binary segment has to be decoded into a real value first using the following transformation:

$$X_{\text{real}} = a + \text{decimal}(X_{\text{bits}}) \cdot \frac{(b-a)}{2^n - 1} \tag{2}$$

where:

- X_{real} is the decoded real-valued parameter.
- X_{bits} is the binary segment for the given parameter / dimension.
- $\text{decimal}(X_{\text{bits}})$ converts the binary sequence to its decimal equivalent.
- $2^n - 1$ is the maximum possible decimal value for n bits, ensuring a mapping over the interval $[a, b]$.

III. Genetic Algorithm

The genetic algorithm (GA) implemented in this experiment employs several standard evolutionary techniques to evolve a population towards an optimal solution. The key components are:

- **Three-Cut-Point Crossover:** This crossover technique is applied during the reproduction phase to combine genetic material from two parent chromosomes. In three-cut-point crossover, three random cut points are selected in the parent chromosomes, and the genetic material between these points is exchanged between the two parents, creating two offspring. This promotes genetic diversity and allows the algorithm to explore new regions of the solution space.
- **Mutation:** Mutation is applied to introduce small, random changes in a chromosome to maintain diversity and prevent the algorithm from prematurely converging to local optima. For each chromosome, mutation involves flipping some of the bits in its representation with a certain probability. This helps to explore regions that might not be reached through crossover alone, maintaining exploration of the search space. The mutation probability (pm) is initialized based on the population size to provide a greater starting chance of exploring new solutions in smaller populations.

  The mutation probability (pm) is dynamically adjusted throughout the genetic algorithm to balance exploration and exploitation. Over the course of 10 generations a variable called ($incline$) is adjusted to represent the sum of the differences between the standard deviation of the population in a generation and that of the former generation. Every 10th generation this value is used to adjust (pm) and is then reset to 0. If it is less than or equal to 0, the mutation rate is increased to promote exploration, but it is capped at a maximum value of 0.01. This is done because such an incline that the population's fitness is not progressing, and thus a higher (pm) might help create some changes in the population. Conversely, when the incline is positive, suggesting increasing fitness diversity, the mutation rate is decreased to encourage exploitation, with a lower bound of 0.00001. Additionally, a small reduction of 0.00005 is applied to the mutation rate at each adjustment, ensuring that the mutation probability still decreases over time and lets the solution converge in the end. This adaptive approach helps maintain a balance between exploring new solutions and exploiting promising ones as the algorithm progresses.

Listing 1. Dynamic Mutation Rate Adjustment in Genetic Algorithm

```
void adjust_mutation_rate(double& pm, double
     incline, double inc) {
    // If the incline is less than or equal
        to 0, increase mutation rate
    if (incline <= 0.0) {
        pm = std::min(pm + 5 * inc, 0.01);
            // Cap the mutation rate at 0.01
    } else {
        pm = std::max(pm - 5 * inc, 0.00001)
            ;  // Lower bound at 0.00001
    }

    // Apply a small reduction to mutation
        rate to avoid too large mutation
    pm = std::max(pm - 0.00005, 0.00001);
        // Ensure mutation rate doesn't go
        below 0.00001
}
```

These operators—crossover and mutation—work together to explore and exploit the solution space effectively, allowing the algorithm to progressively refine its population over generations.

IV. Hill Climbing and other optimizations

In the experiments conducted in this paper, both with and without hill climbing, the results were analyzed to assess the impact of hill climbing on the optimization process. Whilst the application of hill climbing to the population introduces a stronger focus on exploitation, potentially limiting exploration, other optimization techniques mentioned in this section remain relevant and contribute to the overall efficiency of the algorithm.

A. Hill Climbing

Note on Hill Climbing Implementation:
The way Hill Climbing was applied in this algorithm was determined through experimentation. This includes several key components:

- *Limiting Hill Climbing search by imposing a maximum number of steps. This allows the program to not use the more costly Hill Climbing algorithm to find the perfect local minimum for each chromosome it is applied to, instead allowing the faster Genetic Algorithm to find improvements as well. This limit also helps avoid premature convergence.*
- *A dynamic number of steps/depth for Hill Climbing, which increases as the generations progress (e.g., $1 + \frac{g}{50}$).*

- *Application of Hill Climbing to the top 5% of the population in each generation to optimize the best current potential solutions.*
- *Application of Hill Climbing on an additional 5% of the population, chosen at random, to explore other local minima as well.*
- *Usage of Hill Climbing only when incline is positive, as to not interfere with the implications of negative incline: when the population is stuck and (pm) is increased to combat that issue, Hill Climbing could hinder the exploration too much and is thus avoided.*

Generally, Hill Climbing means that for each generation, the algorithm iteratively improves the fitness of the current best chromosome by flipping its bits. A neighbour chromosome is created by copying the current chromosome, then each bit in the chromosome is flipped one by one. After each flip, the fitness of the modified chromosome is computed. If the new chromosome has a better fitness than the original, it replaces the original chromosome. This process continues until no further improvement is found or the maximum number of steps is reached.

After tournament selection, three-cut-point-crossover and mutation, Hill Climbing is applied if g is divisible by 10, where g represents the current generation number, and if the incline factor is positive.

The incline factor is calculated based on the current and previous standard deviations of the fitness. If the incline factor is positive, it means that the standard deviation of the fitness has increased compared to the previous generation. This could indicate that the population has become more diverse (perhaps because of the introduction of new individuals or more variation in fitness between them), so refining the best individuals with Hill Climbing seems natural.

Specific to this implementation, every time Hill Climbing is applied, the number of steps is dynamically adjusted in each generation, becoming slightly larger as the current generation progresses. Increasing the number of steps in the Hill Climbing algorithm along with the generations allows the search process to become more thorough over time. Early generations explore quickly with fewer steps, while later generations refine the solution with more steps. The parameter $1 + \frac{g}{50}$, gradually increases the allowed steps as the number of generations increases.

The first 5% of individuals are typically high-performing, so applying a growing number of hill climbing steps here could push them towards even finer refinements, thus improving the final results.

The randomly selected 5% allows the algorithm to find other local minima than the ones currently at the top of the population, decreasing the likelihood of convergence of the entire population in only one, non-global, minimum.

B. Number of bits per dimension

Additionally, to represent each dimension of the solution space with a specific precision, we calculate the number of bits required for encoding based on the range of values for each test function. The number of bits per dimension is determined by the following steps:

1) **Retrieving Function Boundaries**: A function is used to retrieve the minimum and maximum values of the domain, $[\min_x, \max_x]$, for the specified optimization function. This defines the search interval for that dimension.
2) **Calculating Discretization**: Based on the desired precision, the number of discrete values needed to span the interval is determined $[\min_x, \max_x]$. This is done by calculating:

$$N = (\max_x - \min_x) \times 10^{\text{precision}}$$

where precision represents the required number of decimal places, in our case 5.
3) **Computing the Number of Bits Needed**: To represent all N discrete values in binary, the number of bits is calculated after this formula n :

$$n = \lceil \log_2(N) \rceil$$

which gives the minimum number of bits needed to encode each dimension with the desired precision.

This ensures that each dimension can be encoded in binary with sufficient precision, allowing accurate conversion between binary-encoded solutions and their corresponding real values within the search space. In also ensures the minimum sufficient number of bits is used, allowing the program to run as fast as possible while still conforming to the desired precision.

C. Thread Pool

After tournament selection, the entire population is processed in chunks using a thread pool, which enables parallelization of the evolutionary operations. Each chunk of the population consists a pair of chromosomes, and for each pair, genetic operations such as three-cut-point-crossover and mutation are applied. The thread pool manages the parallel execution of these operations, ensuring that multiple pairs are processed simultaneously, and that the processing of new pair is started as soon as another finishes and a thread is made available, which significantly speeds up the computation.

A similar approach is used to handle the computationally expensive Hill Climbing algorithm. A thread pool is utilized to handle the optimization of the top 5%, each chromosome in it's own thread, as soon as one is available. After all of them are finished, random indices for 5% of the population are selected and then processed in much the same way.

D. Stopping the algorithm earlier if arriving at the global minima

The program may be stopped earlier by the programmer, via signals, if the GA arrives at the global minima. More specifically, this can be done by pressing $Ctrl + C$.

V. Minimizing Generations with Larger Population

Another optimization strategy focuses working with a larger population size, thus allowing the result to be reach in fewer generations. The reasoning behind this approach is that although, with a slight change to other parameters, the same results could be achieved with a smaller population in more generations, experiments showed it was still overall slower than the presented implementation. This larger population benefits from parallel processing through a Thread Pool, which helps mitigate the time complexity introduced by the population size, while the steps done in the main thread lead to a higher final time when more generations were used. As a benefit, the increased population size also ensures a more diverse and representative search, which is critical for finding optimal or near-optimal solutions efficiently.

The balance between population size and number of generations that worked best ended up being: population of $150 \cdot$num_dims, where num_dims is the number of dimensions in the problem, and 500 generations.

VI. Experimental setup

Crossing probability: 0.8.

Precision: 5.

Tournament size: 5.

Population Size: $150 \cdot$ num_dims.

Generations: 500. However, sometimes the process was stopped earlier if the global minimum was found before completing the full set of generations.

VII. Test functions

A. *Rastrigin's function*

Rastrigin's function is defined as:

$$f(\mathbf{x}) = A \cdot n + \sum_{i=1}^{n} \left(x_i^2 - A \cdot \cos(2\pi x_i)\right)$$

where $A = 10$, and n is the number of dimensions.

Solution: The global minimum is at $\mathbf{x} = 0$, where $f(\mathbf{x}) = 0$.

Minima: The function has a large number of local minima, making it challenging for optimization algorithms. The global minimum is at $\mathbf{x} = 0$, where all dimensions are zero.

TABLE I
Genetic Algorithm Results Without Hill Climbing on Rastrigin's

Dim	Mean	Std. Dev.	Best	AVG(Best)	AVG(Worst)	Time
2	0.47070	2.25444	0.00000	0.00000	15.18295	1.64s
30	19.18259	18.05826	0.00029	0.00226	74.05379	44.05s
100	8.76336	10.78267	0.995022	2.81607	95.64831	462.26s

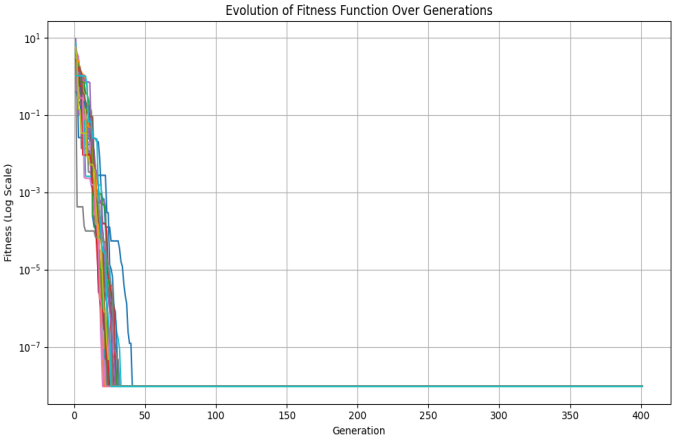

Fig. 1. Rastrigin function without HC on dimension 2.

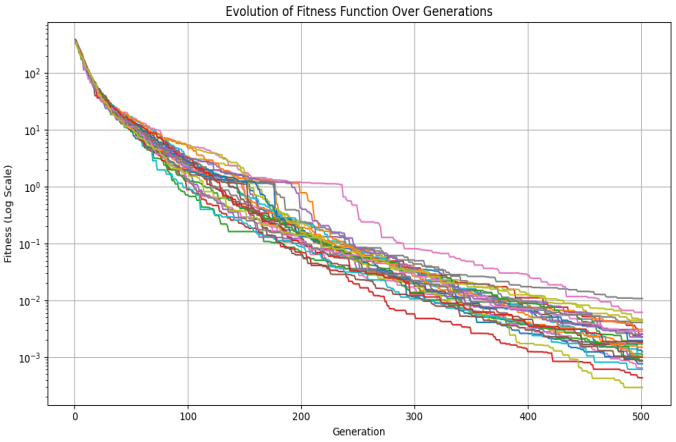

Fig. 2. Rastrigin function without HC on dimension 30.

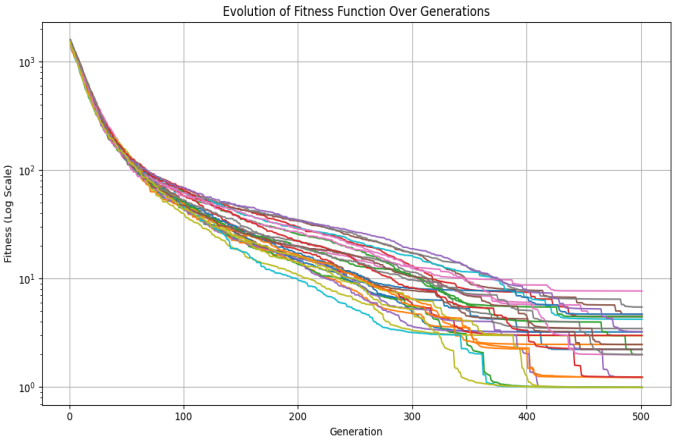

Fig. 3. Rastrigin function without HC on dimension 100.

TABLE II
GENETIC ALGORITHM RESULTS WITH HILL CLIMBING ON RASTRIGIN'S

Dim	Mean	Std. Dev.	Best	AVG(Best)	AVG(Worst)	Time
2	0.57188	2.44318	0.00000	0.00000	15.21133	2.12s
30	10.52149	12.77225	0.00000	0.00000	46.63114	66.81s
100	102.04804	44.027991	0.00000	1.42432	189.10202	462.26s

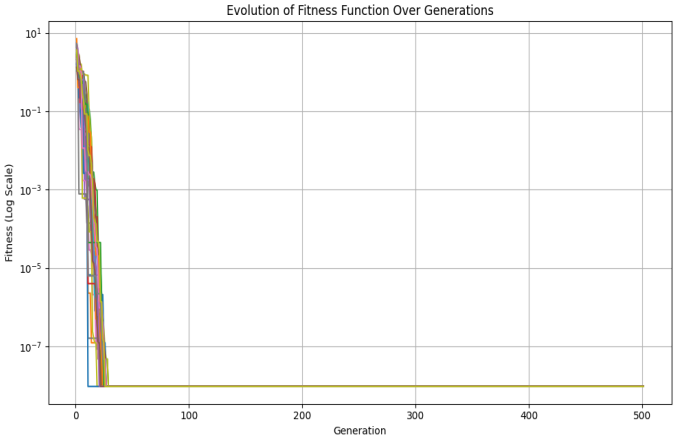

Fig. 4. Rastrigin function with HC on dimension 2.

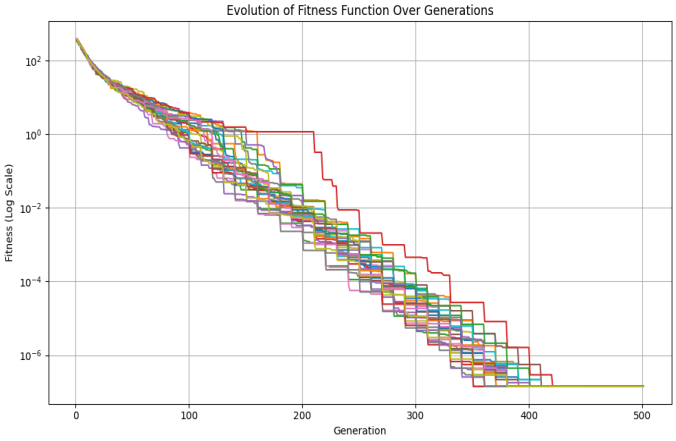

Fig. 5. Rastrigin function with HC on dimension 30.

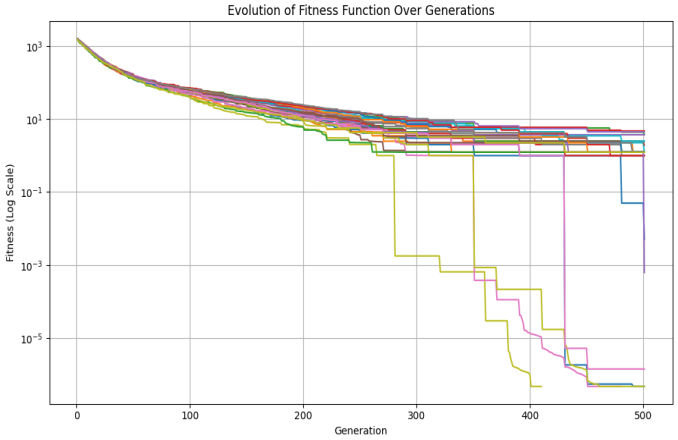

Fig. 6. Rastrigin function with HC on dimension 100.

B. *Griewank's Function*

Griewank's function is defined as:

$$f(\mathbf{x}) = 1 + \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right)$$

Solution: The global minimum is at $\mathbf{x} = 0$, where $f(\mathbf{x}) = 0$.

Minima: The function has a relatively smooth landscape with a single global minimum. However, the presence of oscillating cosine terms introduces a complexity that can make it tricky for optimization methods to converge to the global optimum.

TABLE III
GENETIC ALGORITHM RESULTS WITHOUT HILL CLIMBING ON GRIEWANGK'S

Dim	Mean	Std. Dev.	Best	AVG(Best)	AVG(Worst)	Time
2	0.12168	0.43887	0.00000	0.00786	3.23258	1.63s
30	18.22615	31.97513	0.00038	0.00554	130.97456	29.46s
100	3.84867	16.34738	0.00000	0.00026	185.7073	300.64s

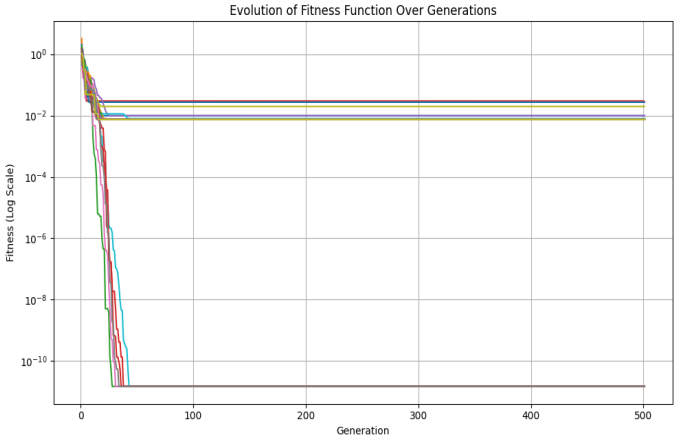

Fig. 7. Griewangk function without HC on dimension 2.

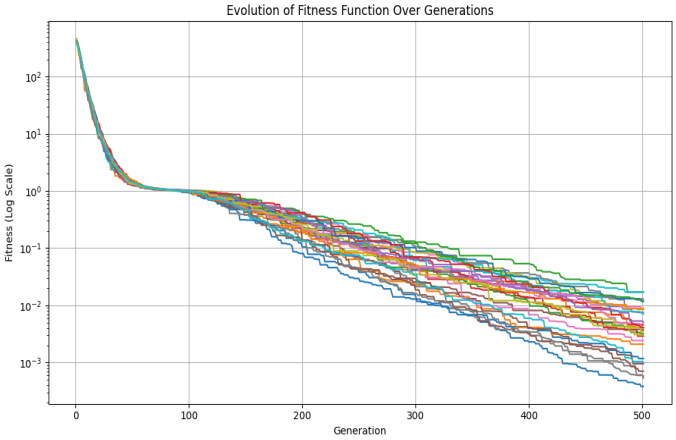

Fig. 8. Griewangk function without HC on dimension 30.

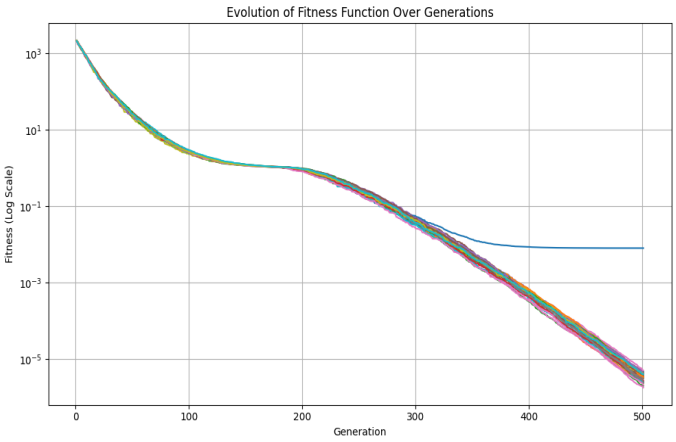

Fig. 9. Griewangk function without HC on dimension 100.

TABLE IV
GENETIC ALGORITHM RESULTS WITH HILL CLIMBING ON GRIEWANGK'S

Dim	Mean	Std. Dev.	Best	AVG(Best)	AVG(Worst)	Time
2	0.14684	0.59445	0.00000	0.00732	4.81149	1.69s
30	15.72083	28.71170	0.00000	0.00041	100.90	57.38s
100	3.13013	11.96393	0.00000	0.00000	165.16862	803.23s

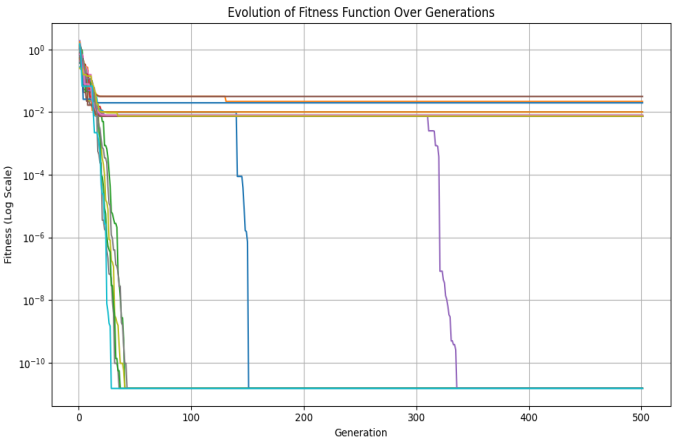

Fig. 10. Griewangk function with HC on dimension 2.

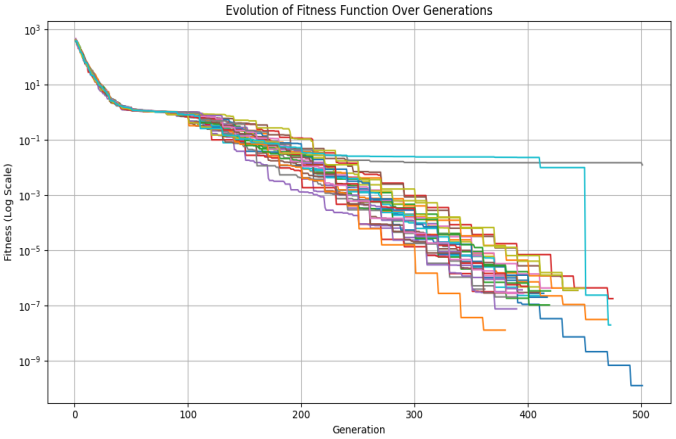

Fig. 11. Griewangk function with HC on dimension 30.

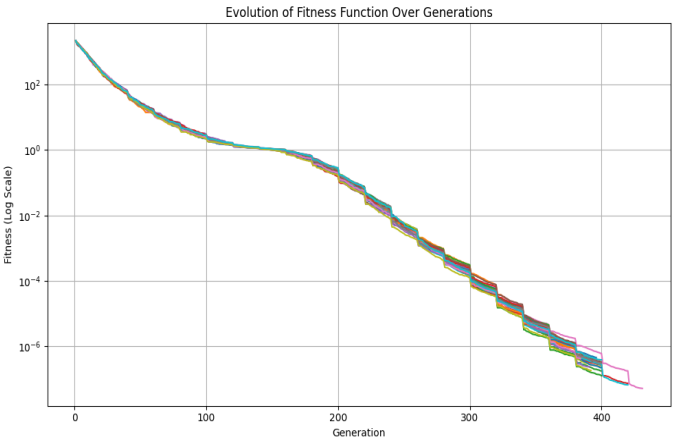

Fig. 12. Griewangk function with HC on dimension 100.

C. *Rosenbrock's function*

Rosenbrock's function is often called the "Rosenbrock's valley" or "banana function" due to its distinctive shape. It is defined as:

$$f(\mathbf{x}) = \sum_{i=1}^{n-1} \left(100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2\right)$$

Solution: The global minimum is at $\mathbf{x} = (1, 1, \ldots, 1)$, where $f(\mathbf{x}) = 0$.

Minima: The function has a narrow, curved valley containing the global minimum, with many local minima along the valley. Optimization algorithms must carefully navigate this valley to find the global minimum.

TABLE V
GENETIC ALGORITHM RESULTS WITHOUT HILL CLIMBING ON ROSENBROCK'S

Dim	Mean	Std. Dev.	Best	AVG(Best)	AVG(Worst)	Time
2	0.16722	1.02722	0.00000	0.00032	10.25536	1.61s
30	294.30116	643.08563	23.23250	25.60002	2664.602	23.40s
100	2021.62245	1620.2605	88.7279	95.24692	6403.07893	221.94s

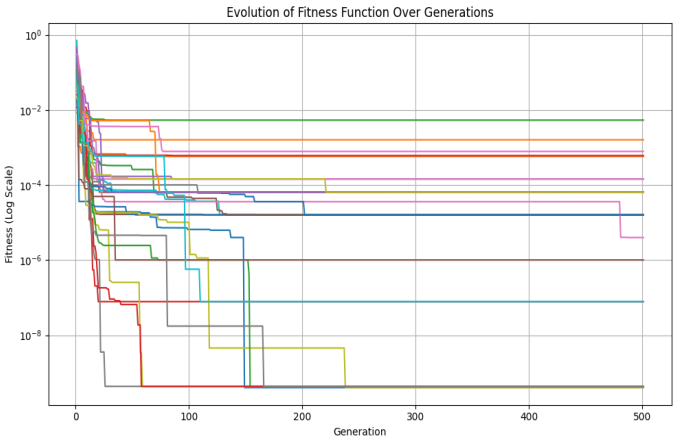

Fig. 13. Rosenbrock function without HC on dimension 2.

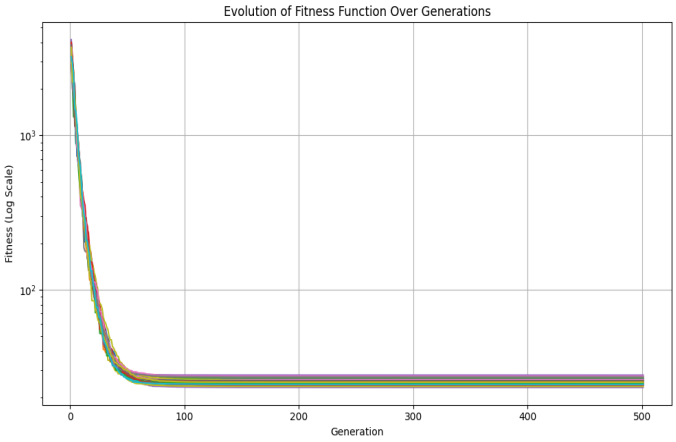

Fig. 14. Rosenbrock function without HC on dimension 30.

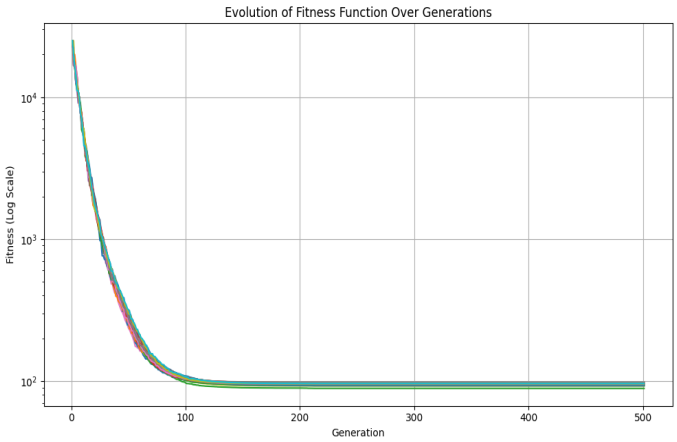

Fig. 15. Rosenbrock function without HC on dimension 100.

TABLE VI
GENETIC ALGORITHM RESULTS WITH HILL CLIMBING ON ROSENBROCK'S

Dim	Mean	Std. Dev.	Best	AVG(Best)	AVG(Worst)	Time
2	0.13664	0.82661	0.00000	0.00014	8.44961	1.66s
30	102.33747	252.01284	21.68510	25.23125	1527.61606	47.37s
100	1720.18480	1492.24736	89.96610	94.41777	5497.15710	916.96s

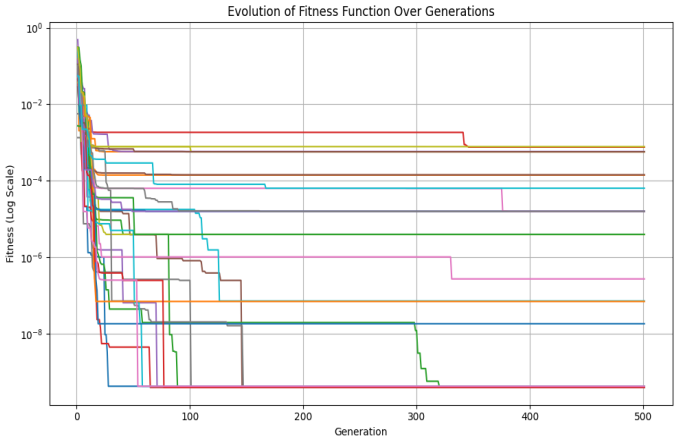

Fig. 16. Rosenbrock function with HC on dimension 2.

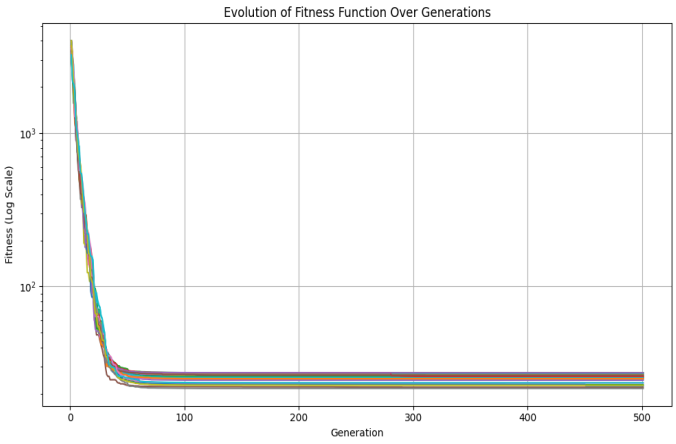

Fig. 17. Rosenbrock function with HC on dimension 30.

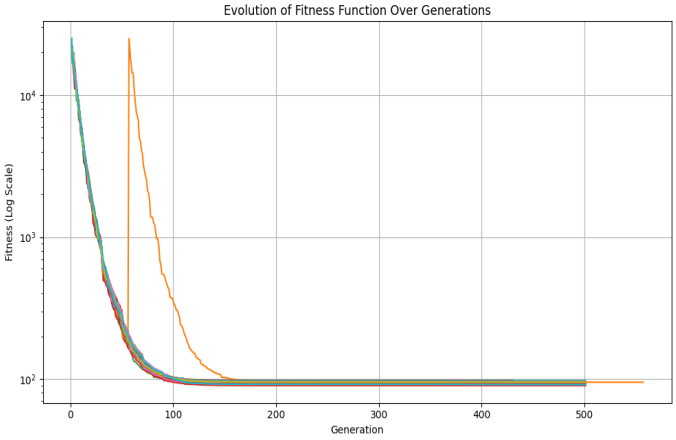

Fig. 18. Rosenbrock function with HC on dimension 100.

According to these results, Hill Climbing provides consistency, but not improvement. Therefore for this function also we attempted a different approach: lower population ($100 * dims$), higher initial pm (0.08) an upper limit for pm (0.1), more generations (1500) and more steps for Hill Climbing ($50 + g/50$). While this provided better results for dimension 30, the increase in time complexity made testing for dimension 100 difficult. The average time for a test is about 10 times that of the original approach. What we can say about dimension 100 is that these exact values did not seem to provide the same improvement as they did for dimension 30 in what testing we were able to do. Still, we would like to share these results, as they indicate that a high mutation rate and more thorough Hill Climbing can lead to better final results, albeit in many more generations and much more time. In addition, we have reason to believe that the solution can still be improved after 1500 generations, although the process becomes much slower. As we can see from the graph, there are increasingly long periods of stagnation.

TABLE VII
GENETIC ALGORITHM RESULTS WITH IMPROVED HILL CLIMBING ON ROSENBROCK'S

Dim	Mean	Std. Dev.	Best	AVG(Best)	AVG(Worst)	Time
30	4882.52953	2170.85052	0.68314	8.280043	9893.247795	424.90s

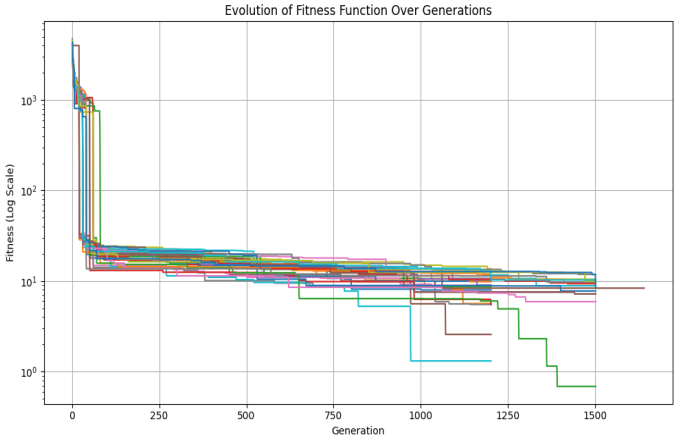

Fig. 19. Rosenbrock function with HC on dimension 30, improvement.

D. Michalewicz's function

The Michalewicz function is known for its multiple local minima. It is defined as:

$$f(\mathbf{x}) = -\sum_{i=1}^{n} \sin(x_i) \cdot \sin^{2m}\left(\frac{ix_i^2}{\pi}\right)$$

where m is typically set to 10, and x_i is in the range $[0, \pi]$.

Solution: The global minimum depends on the value of n, where n is the dimension. Generally speaking it is slightly bigger than -n. For dimensions 2, 30 and 100 the minimum is -1.8013, -29.6308839 and estimated to be approximately -99.56129, respectively.

Minima: This function has multiple local minima and a single global minimum. Its highly irregular shape with many valleys makes it a challenging problem for optimization algorithms.

TABLE VIII
GENETIC ALGORITHM RESULTS WITHOUT HILL CLIMBING ON MICHALEWICZ'S

Dim	Mean	Std. Dev.	Best	AVG(Best)	AVG(Worst)	Time
2	-1.78875	0.06474	-1.8013	-1.80130	-1.29309	1.63s
30	-28.33641	0.90449	-29.5832	-29.48719	-25.75987	41.51s
100	-97.27816	0.74192	-98.7833	-98.08184	-93.02301	213.67s

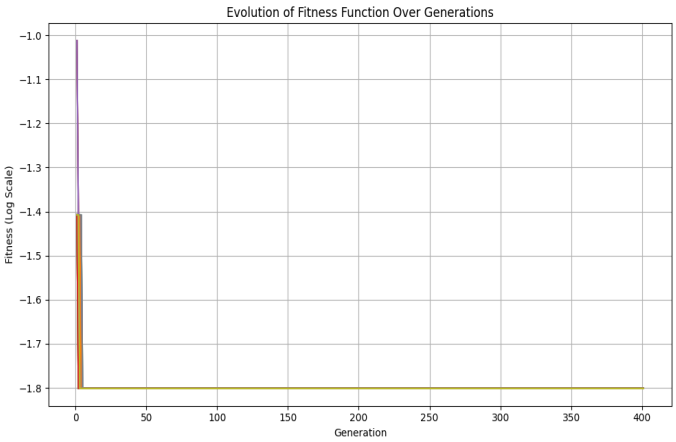

Fig. 20. Michalewicz function without HC on dimension 2.

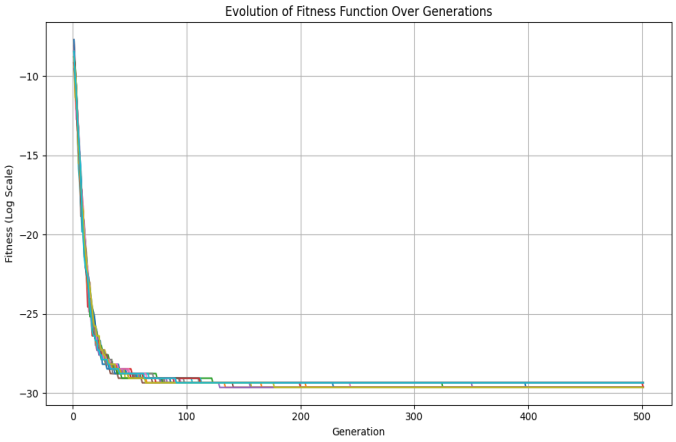

Fig. 21. Michalewicz function without HC on dimension 30.

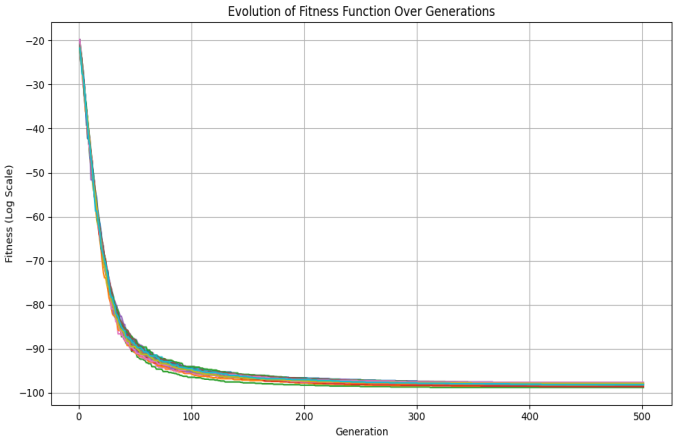

Fig. 22. Michalewicz function without HC on dimension 100.

TABLE IX
GENETIC ALGORITHM RESULTS WITH HILL CLIMBING ON MICHALEWICZ'S

Dim	Mean	Std. Dev.	Best	AVG(Best)	AVG(Worst)	Time
2	-1.79007	0.05810	-1.80130	-1.80130	-1.36050	2.15s
30	-29.19724	0.45247	-29.60860	-29.53697	-27.90870	67.84s
100	-91.76821	2.66067	-98.86280	-98.47335	-86.88900	2011.93s

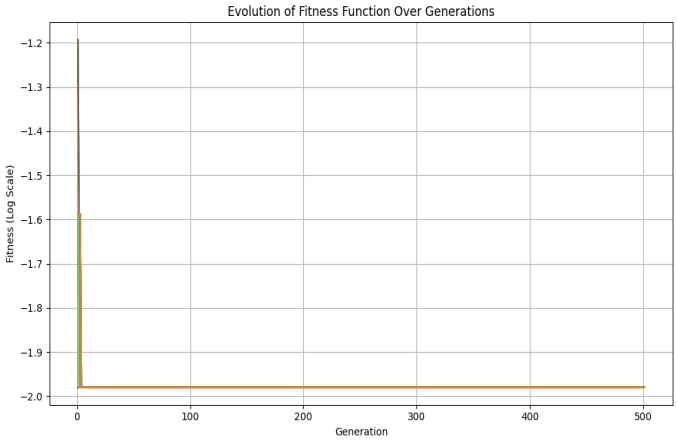

Fig. 23. Michalewicz function with HC on dimension 2.

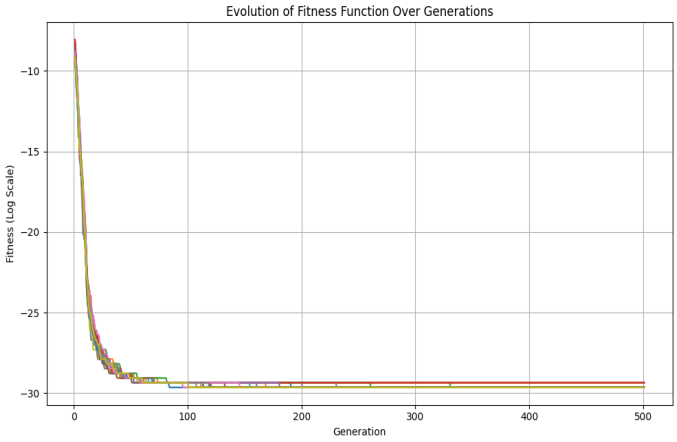

Fig. 24. Michalewicz function with Hc on dimension 30.

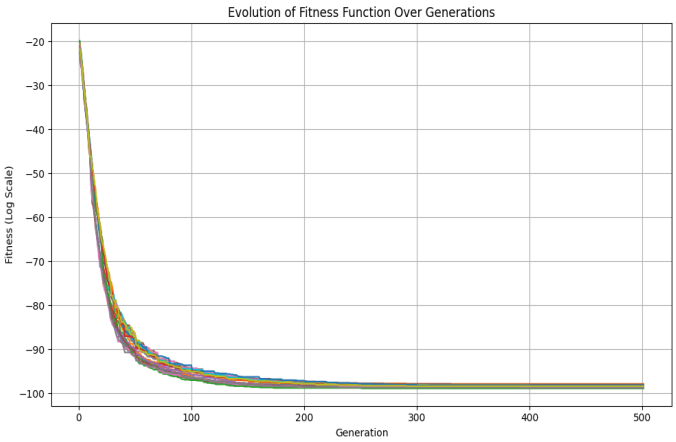

Fig. 25. Michalewicz function with HC on dimension 100.

VIII. Comparisons

In the case of Rastrigin's function:

- With Hill Climbing, the best results at each dimension are quite similar, but the mean performance is generally better than without Hill Climbing, especially in high dimensions.
- The time to complete the computation is higher with Hill Climbing at higher dimensions (30 and 100), suggesting the addition of the Hill Climbing increases the complexity.
- The worst values in all dimensions are lower without Hill Climbing.

For Griewank's function:

- The addition of Hill Climbing results in a more stable performance, with the best values being very close to zero in all cases.
- The worst results are better with Hill Climbing, particularly at higher dimensions.
- The time required for computation is generally higher with Hill Climbing, especially for higher dimensions (100), suggesting it improves the algorithm's ability to find better solutions at the cost of computation time.

For Rosenbrock's function:

- The mean and worst values are significantly improved with Hill Climbing, particularly at higher dimensions.
- The computation time increases with Hill Climbing, but it results in better optimization performance at the cost of more time.
- The Hill Climbing helps the algorithm to better navigate the narrow valley, resulting in a higher likelihood of reaching the global minimum.

For Michalewicz's function:

- In the 2-dimensional case, the mean improved slightly when the Hill Climbing was introduced, from -1.78875 to -1.79007. The average best solution remained the same at -1.80130, but the avergae worst solution improved from -1.29309 to -1.36050. The time taken increased slightly, from 1.63s to 2.15s.
- For the 30-dimensional case, the mean decreased from -28.33641 to -29.19724, and the standard deviation improved from 0.90449 to 0.45247, indicating faster convergence. The average best solution improved slightly from -29.48719 to -29.53697, while the average worst solution worsened from -25.75987 to -27.90870. The time taken almost doubled, from 41.51s to 67.84s.
- In the 100-dimensional case, the mean increased from -97.27816 to -91.76821, and the standard deviation increased significantly from 0.74192 to 2.66067, indicating more fluctuation in results. The average best solution improved from -98.08184 to -98.47335, while the average worst solution increased from -93.02301 to -86.88900. The time taken skyrocketed from 213.67s to 2011.93s, which is a significant increase due to the Hill Climbing's additional refinement.

IX. Conclusions

The Genetic Algorithm (GA) with Hill Climbing outperforms the standard GA in higher-dimensional problems, providing more stable and consistent results, but also faster loss of diversity in most cases and a significant increase in time complexity. In lower dimensions, the Hill Climbing offers only marginal improvements, suggesting it is more effective for complex landscapes. However, the computational cost increases significantly with dimensionality, limiting scalability.

Future Research:

- Exploration of hybrid algorithms combining GA with other optimization techniques such as as simulated annealing or particle swarm optimization for better performance in high-dimensional spaces.
- Further experimentation with different configurations of the Hill Climbing algorithm to enhance its performance across a wider range of problem domains.
- Specifically regarding the secondary approach to Hill Climbing for Rosenbrock's function, further testing on possible parameters to achieve similar improvements for dimension 100 as we did for 30.

References

[1] Hussain, A., Ashas, H., Shahid, A., Qureshi, S., Karrila, S. (2024). Hybrid Approach Involving Genetic Algorithm and Hill Climbing to Resolve the Timetable Scheduling for a University. In: Arai, K. (eds) Advances in Information and Communication. FICC 2024. Lecture Notes in Networks and Systems, vol 919. Springer, Cham. https://doi.org/10.1007/978-3-031-53960-2_6

[2] J. . -M. Renders and H. Bersini, "Hybridizing genetic algorithms with hill-climbing methods for global optimization: two possible ways," Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence, Orlando, FL, USA, 1994, pp. 312-317 vol.1, doi: 10.1109/ICEC.1994.349948.

[3] Krentel, M., & Reinelt, G. (2020). Heuristic Optimization Algorithms for Solving 3D Packing Problems: An Empirical Comparison. arXiv preprint arXiv:2003.09867. https://arxiv.org/pdf/2003.09867

[4] Mitchell, Melanie & Holland, John & Forrest, Stephanie. (1994). When Will a Genetic Algorithm Outperform Hill Climbing?. Advances in neural information processing systems 6.

[5] Wadhwa, A. (2017). Performance analysis of selection schemes in genetic algorithm for solving optimization problem using De jong's function1. Int. J. Recent Innov. Trends Comput. Commun, 5, 563-569.

[6] Chawla, G., & Bala, M. Y. (2014). Solving Optimization Problem By Hybrid Genetic Algorithm Using Hill Climbing In Replacement Operator. Journal of Recent Research Aspects, 2(4), 73-78.
[7] A. Shukla, H. M. Pandey and D. Mehrotra, "Comparative review of selection techniques in genetic algorithm," 2015 International Conference on Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE), Greater Noida, India, 2015, pp. 515-519, doi: 10.1109/ABLAZE.2015.7154916.
[8] Miller, B. L., & Goldberg, D. E. (1995). Genetic algorithms, tournament selection, and the effects of noise. Complex systems, 9(3), 193-212.
[9] Srinivas, M., & Patnaik, L. M. (1994). Adaptive probabilities of crossover and mutation in genetic algorithms. IEEE Transactions on Systems, Man, and Cybernetics, 24(4), 656-667.
[10] Kramer, O., & Kramer, O. (2017). Genetic algorithms (pp. 11-19). Springer International Publishing.
[11] Juneja, K., & Gill, N. S. (2013). Optimization of dejong function using ga under different selection algorithms. International Journal of Computer Applications, 64(7).
[12] De Jong, K. A., & Spears, W. M. (1990, October). An analysis of the interacting roles of population size and crossover in genetic algorithms. In International Conference on Parallel Problem Solving from Nature (pp. 38-47). Berlin, Heidelberg: Springer Berlin Heidelberg.