A Comparative Study of Genetic Algorithms, Hill Climbing and Simulated Annealing on Benchmark Optimization Problems

Albert Alexandru

December 2023

Abstract

This study was conducted to evaluate the efficacy of Genetic Algorithms (GAs) across four well-known functions (Rastrigin's, Schwefel's, Michalewicz's, and De Jong 1), comparing the outcomes with those of Hill Climbers (HC) and Simulated Annealing (SA) from a previous study.

The foundational GA employed key genetic operators: Mutation, Crossover, Selection, Elitism, and a modified selection pressure. Another section introduced improvements such as Grey Codes, adaptive operators, including adaptive mutation, adaptive crossover, and hypermutation.

Our findings highlight the GA's exceptional performance in optimizing complex functions, such as Rastrigin's, surpassing both HC and SA in mean results and time complexity. Further improvements through the adaptive operators and parameters enhanced the solution in certain cases.

However, for the simpler and convex function De Jong 1, the GA yielded inferior results compared to HC and SA, with HC providing the best results.

In summary, this study underscores the clear advantages of Genetic Algorithms over other heuristic methods, and revealing unexpected findings, such as the GA's ability to outpace SA in terms of speed.

Contents

1 Introduction

Optimization algorithms play a pivotal role in solving intricate problems across diverse domains, with Genetic Algorithms (GAs) standing out as robust tools in this landscape. This study aims to examine the Genetic Algorithm in four distinct functions: Schwefel, Rastrigin, Michalewicz, and De Jong 1. These functions, each characterized by unique features and challenges, offer a varied terrain for assessing the versatility and performance of Genetic Algorithms.

Genetic Algorithms, inspired by the principles of natural selection and genetics, employ evolutionary mechanisms to iteratively improve candidate solutions. This population-based approach allows for exploration of diverse regions within the solution space, making GAs particularly adept at handling complex, multi-modal problems.

On the other hand, Hill Climbing is a local search algorithm that iteratively moves towards the direction of increasing elevation in the solution space. Its simplicity and efficiency make it a valuable tool for navigating smooth, convex landscapes and finding local optima. However, its reliance on local information may limit its effectiveness in tackling more intricate optimization challenges.

Simulated Annealing, drawing inspiration from the annealing process in metallurgy, introduces a probabilistic element to the search for optimal solutions. By allowing occasional uphill moves with decreasing probability, Simulated Annealing aims to escape local optima and explore the solution space more comprehensively. This stochastic nature makes it well-suited for handling problems with complex and rugged landscapes.

All of these algorithms represent solutions as binary strings. Modifying such a solution involves a simple flip of a bit. The genetic algorithm encompasses multiple solutions, collectively forming a population. This is where genetic operators come into play, each tasked with flipping bits and generating new solutions in distinct ways. Mutation, with its random probability of flipping any bit, is designed for exploration, introducing new information which does not exist in the current generation.

In contrast, crossover engages in exploitation by combining existing information from the current generation. It achieves this by cutting two chromosomes at an equal point and swapping the information (bits) beyond that specific point.

Ultimately, the selection process stands as the most important component in any Genetic Algorithm. It determines which chromosomes or solutions possess the requisite traits to progress into the next generation. Further details on additional genetic operators will be explored in the upcoming "Methods" section.

2 Methods

In this section, we will discuss the genetic operators employed for implementing and optimizing the genetic algorithm, as well as explore the adaptive operators and other enhancements that have been attempted to improve the algorithm.

2.1 The Genetic Algorithm (GA)

The fundamental structure of the genetic algorithm is as follows:

```
$t \leftarrow 0$
Generate the starting population $P(t)$
Evaluate $P(t)$
**while** not StoppingCondition **do**
    $t \leftarrow t + 1$
    Select $P(t)$ from $P(t-1)$
    Mutate $P(t)$
    Crossover $P(t)$
    Evaluate $P(t)$
**end while**
```

2.1.1 Mutation

In the field of biology, a mutation signifies a modification in the nucleic acid sequence of an organism's genome, a virus, or extrachromosomal DNA. The genetic algorithm simplifies and emulates these processes using a basic unit: the bit. Unlike the intricate nature of mutations in biology, the genetic algorithm performs mutations by simply flipping a single bit—transitioning from 0 to 1 or from 1 to 0.

The mutation function operates by iterating through all the chromosomes and genes. For each gene, a random number is generated. If this number is smaller than the predetermined probability, a mutation occurs. When we will talk about mutation, the letter L will be used to represent the total length of the chromosome.

2.1.2 Crossover

In the genetic algorithm, crossover emulates the biological concept of mating. It involves taking two parent chromosomes and combining their genetic material to create one or more offspring chromosomes. Unlike the simplicity of bit flipping in mutation, crossover operates by exchanging entire segments or genes between parent chromosomes.

The crossover process works by randomly selecting a crossover point along the chromosomes. The genetic material beyond this point is swapped between the parents, generating new offspring chromosomes. This mechanism introduces diversity and the potential for novel combinations of genetic information in the algorithm's population, aiding in the exploitation of the current material.

Three distinct crossover methods were explored: fitness-based crossover, where only the children were retained; random crossover with the selection of the two best individuals out of four; and random crossover with the retention of only the offspring. Surprisingly, the latter approach appeared to yield the best results, possibly due to the algorithm requiring increased exploration, since the number of chromosomes kept as elites was very high, inhibiting natural exploration.

Furthermore, experimentation extended to the crossover probability, revealing that the most favorable outcomes were achieved with a 100% crossover probability.

2.1.3 Selection

In the genetic algorithm, the Wheel of Fortune selection method introduces a level of randomness and chance inspired by nature. Instead of directly choosing the best chromosomes, this operator employs a metaphorical spinning wheel to determine the probability of selecting each chromosome for the next generation. Think of each chromosome as a segment on this wheel, and the size of the segment is directly proportional to the chromosome's fitness or desirability.

During the selection process, the wheel spins, and where it stops dictates which chromosomes are chosen to form the next generation. Fitter chromosomes occupy more significant portions of the wheel, increasing their likelihood of being selected, while less fit ones have smaller segments and a lower probability of being chosen.

This approach introduces a stochastic element into the genetic algorithm, bringing variability to the selection. By simulating the element of chance, the Wheel of Fortune operator enhances the algorithm's exploration of diverse genetic material. This variability fosters the creation of novel offspring, contributing to the algorithm's ability to discover optimal solutions by encouraging the exploration of a broad solution space.

Moreover, a modified variant of this operator was employed, allowing us to regulate the size of the slices in the Wheel of Fortune by manipulating the selection pressure. This was achieved by scaling all fitness values to the [1, 2] interval and subsequently elevating them to a specified power. For instance, within the base interval of 1 to 2, the best individual possesses twice the likelihood of being selected compared to the worst one. Raising fitness values to the power of 2 enhances this discrepancy, making the best individual four times more likely to be chosen.

Henceforth, when referring to 'using a selection pressure of n', it implies that fitness values were initially scaled to the [1, 2] interval and subsequently raised to the power of n.

In our experimental endeavors, we applied a low selection pressure of less than 1, signifying that suboptimal fitness solutions had a more favorable chance of being selected. Given the substantial size of the elite population, this strategy could potentially have benefited the algorithm by introducing greater novelty into the population pool, akin to the effects observed in crossover scenarios (with maximal cx probability).

2.1.4 Elitism

In the context of genetic algorithms, elitism is a concept inspired by the evolutionary principle of natural selection. It involves preserving the best-performing individuals from one generation to the next, ensuring their direct passage to the subsequent population. This mechanism mirrors the biological concept of 'survival of the fittest.'

Unlike the intricate interplay of various factors influencing survival in nature, elitism in genetic algorithms simplifies the process by selecting individuals based on their fitness scores. The fittest individuals, often a predetermined percentage of the population, are directly carried over to the next generation without undergoing any genetic modifications.

This retention of elite individuals serves to maintain highly successful traits within the population, preventing the loss of beneficial genetic material. By safeguarding the best performers, elitism enhances the efficiency of the genetic algorithm in converging towards optimal solutions over successive generations.

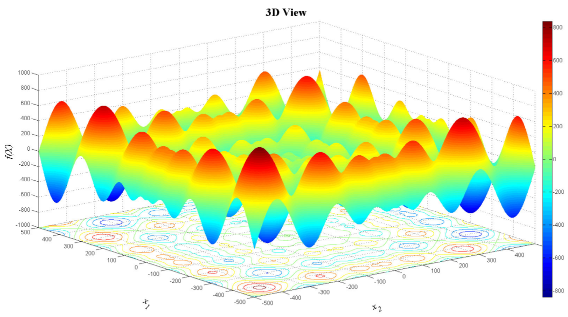
In our experiments, elitism played a pivotal role, frequently preserving over 80% of the population. This operates more like the culling of marginal, weaker solutions rather than merely saving the best. Importantly, this 'elite' population was not excluded from the mutation process.

2.2 Fine-tuning the GA for every function

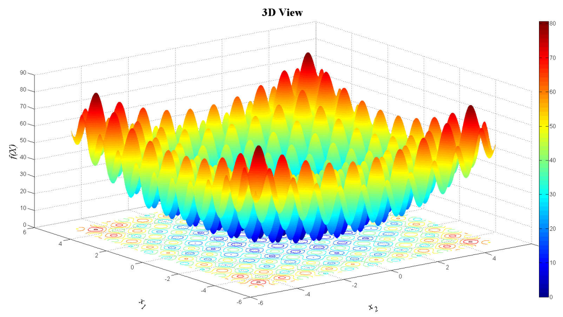
This section involved extensive experimentation, with frequent revisits to specific functions and the exploration of new parameters following improvements in others. Utilizing implementations featuring a relatively low number of generations allowed us to conduct a substantial number of experiments. Through this process, we discovered that distinct parameters proved optimal for various functions, yet certain things remained consistent.

We set the crossover probability at 100%, as revealed in the adaptive crossover probability section to be optimal. Additionally, a distinct GA adjustment involved keeping a large elite population (around 80%), effectively eliminating weaker solutions rather than preserving the best.

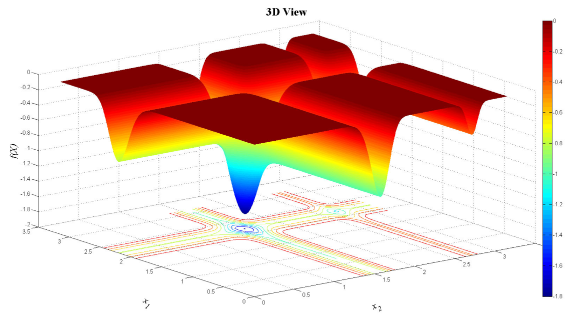
The experiments were conducted on four well-known benchmark functions: Schwefel's, Rastrigin's, Michalewicz's, and De Jong 1.



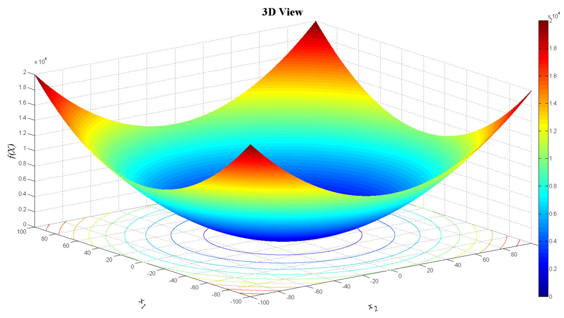
(a) Schwefel's Function.



(b) Rastrigin's function.



(c) Michalewicz's function.



(d) De Jong 1 function.

2.2.1 Schwefel's Function

$$f(x) = -\sum_{i=1}^{d} x_i \sin(\sqrt{|x_i|}),\ \ x_i \in [-500, 500]$$

Initially, we conducted experiments and discovered that, for Michalewicz's function, an exceptionally high selection pressure of 10 or even 15 yielded decent results. This indicated that the GA quickly found a satisfactory solution, and the heightened selection pressure aided swift convergence.

However, as our experimentation progressed, we realized that the opposite approach was more effective—optimal results were achieved by decreasing the selection pressure and modifying other parameters accordingly. When optimizing for Schwefel's function with a high selection pressure, we encountered poor results. Additionally, we observed that the algorithm converged to an optimal solution rapidly, rendering a high number of generations unnecessary. Gradually reducing it from 1000 to 100 still produced good results.

We noted that lower crossover probabilities led to inferior results, so we incrementally increased the crossover probability until reaching 100%. This worked well in conjunction with saving the top 50% of the population (elitism) before applying crossover. This also, in time, reached a proportion of around 80%.

The mutation probability was also adjusted, increased from $\frac{1}{L}$ to $\frac{5}{L}$, and then lowered to $\frac{2}{L}$, with the latter proving to be an optimal parameter based on our experiments. Further experimentation revealed that increasing the population size was beneficial. Maintaining a larger population, around 200, yielded better solutions compared to increasing the number of generations.

Therefore, the population size was kept at 200, and the number of generations was set to where the solution did not improve any further, around 200.

- Summary of the parameters used:
  - Nr. of Generations = 200
  - Population Size = 200
  - Elite Pool = 170
  - Mutation Prob. = $\frac{2}{L}$
  - Crossover Prob. = 100%
  - $\epsilon = 10^{-3}$

2.2.2 Rastrigin Function

$$f(x) = A \cdot n + \sum_{i=1}^{n} \left[x_i^2 - A \cdot cos(2\pi x_i)\right],\ A = 10,\ x_i \in [-5.12, 5.15]$$

We began experimenting with the Rastrigin function, using the same (initial) approach we applied to Schwefel's. The initial results in the 30-dimensional space were: Mean value: 25.33, Average time: 0.64 seconds.

After tweaking the code a bit, playing with the number of generations and reducing the mutation rate, we saw much better results. However, there was a big difference between runs. One run gave us results under 0.01, and the next one shot up to 20. It seemed like the algorithm was close to finding the global minimas but might be getting stuck somewhere.

Just like with Schwefel's, increasing the population size seemed to lead to better final results. So, we set the population at 200, with 170 individuals saved as elites. After a few experiments, we discovered that higher mutation rates resulted in better outcomes. We gradually increased it to $\frac{12}{L}$, which might sound like a lot, but considering a low ϵ value of 10^{-5}, the long individual chromosome length of 810, the actual mutation rate is around 1.5%.

The combination of reducing the selection pressure (using a value of 0.6) played a key role in achieving the best results in our experiments. Since the outcomes were pretty good, we will conduct experiments in the 50-dimensional and 100-dimensional spaces.

- Summary of the parameters used:
  - Nr. of Generations = 220

- Population Size = 200
- Elite Pool = 170
- Mutation Prob. = $\frac{12}{L}$
- Crossover Prob. = 100%
- Selection Pressure = 0.6
- $\epsilon = 10^{-5}$

2.2.3 Michalewicz's Function

$$f(\mathbf{x}) = -\sum_{i=1}^{d} \sin(x_i) \sin^{2m}\left(\frac{ix_i^2}{\pi}\right),\ x_i \in [0, \pi]$$

Starting off with the (initial) Rastrigin parameters, the initial average was -20. Quickly, we dialed down the mutation rate to $\frac{3}{L}$, resulting in an average of -25. The fine-tuning tweaks that worked well for Rastrigin also proved effective here. So, we settled on a population of 200, with 160 individuals as elites, maintaining a selection pressure of 0.6.

Interestingly, unlike Rastrigin's setup, we discovered that a high mutation rate messed up the results and completely wrecked the good solutions. We gradually lowered the mutation rate to $\frac{0.1}{L}$. Even though the epsilon value was higher (10^{-3}, leading to a chromosome length of 598), the mutation probability was way lower, around 0.000167%.

- Summary of the parameters used:
  - Nr. of Generations = 200
  - Population Size = 200
  - Elite Pool = 170
  - Mutation Prob. = $\frac{0.1}{L}$
  - Crossover Prob. = 100%
  - Selection Pressure = 0.6
  - $\epsilon = 10^{-4}$

2.2.4 De Jong 1 Function

$$f(x) = \sum_{i=1}^{d} x_i^2, \quad x_i \in [-5.12, 5.12]$$

Given the simplicity and convex nature of this function, a genetic algorithm is notably unsuitable. Consequently, only a minimal number of experiments were conducted.

- Summary of the parameters used:
  - Nr. of Generations = 100
  - Population Size = 100
  - Elite Pool = 170

- Mutation Prob. $= \frac{0.1}{L}$
- Crossover Prob. $= 100\%$
- Selection Pressure $= 0.1$
- $\epsilon = 10^{-4}$

2.3 Optimizing the GA

2.3.1 Grey Code

In the context of genetic algorithms, Grey codes are often used to represent binary-coded individuals in a way that minimizes the impact of mutations on the chromosome. Grey coding is a binary encoding scheme in which two consecutive numbers in the sequence differ in only one bit.

In standard binary encoding, a small change in the binary representation of a number can result in a large change in the decoded value. This can be problematic in genetic algorithms, where small changes in the binary representation should ideally correspond to small changes in the solution value. But, with using the current implementation of the GA, this was not the case at all.

The implementation itself was fairly straightforward; only the decoding function underwent modification.

$decode\_binary\_string(string)$ became $decode\_binary\_string(gray\_to\_binary(string))$.

2.3.2 Adaptive Mutation Probability

In genetic algorithms, adaptive mutation probability refers to the dynamic adjustment of the mutation rate during the evolutionary process based on the performance and characteristics of the population.

The mutation probability determines the likelihood of a mutation occurring for each gene in an individual's chromosome. A fixed mutation rate means that the probability remains constant throughout the evolution, which might not be optimal for all stages of the optimization process.

Adaptive mutation probability addresses this limitation by allowing the mutation rate to change dynamically in response to the evolving population. The idea is to fine-tune the balance between exploration and exploitation during different phases of the algorithm.

To achieve this, the following formula was used: $f(x) = \frac{initial\_mutation\_probability}{1.0 + decay\_factor \times current\_generation}$

2.3.3 Hypermutation

Hypermutation in genetic algorithms refers to a mutation strategy where the mutation rate is significantly increased for a specific subset of the population or during a certain phase of the evolutionary process. The later was the one implemented here. In the event that a specific number of generations passed without improvement, the entire population underwent a process with a significantly elevated mutation rate.

The idea behind hypermutation is to inject additional diversity into the population by increasing the likelihood of more drastic genetic changes. This can help escape local optima and promote exploration in regions of the search space that might not be thoroughly explored under normal mutation rates.

2.3.4 Adaptive Crossover Probability

Adaptive crossover in genetic algorithms refers to the dynamic adjustment of the crossover rate during the evolutionary process based on the performance and characteristics of the population.

The motivation behind adaptive crossover is to allow the genetic algorithm to adapt its exploration-exploitation strategy as the optimization process progresses. By dynamically changing the crossover rate, the algorithm can strike a balance between exploration (creating diverse solutions) and exploitation (refining promising solutions) at different stages of the evolution.

Given that our base algorithm operates best with a 100% mutation probability, we don't anticipate good results. Nevertheless, for the sake of experimentation, we are going to try it.

The following formula was used:

$f(x) = initial\_crossover\_probability \times (1.0 + growth\_factor \times current\_generation)$

3 Experimental Setup

These functions were run across different dimensional spaces to assess their adaptability and to properly compare the GA with HC and SA. Specifically, experiments were conducted in 5, 10, and 30 dimensions, with additional investigation in 50 dimensions for Schwefel's and Rastrigin functions, and in 100 dimensions for the Rastrigin function.

The parameters for the functions were detailed in the Method section and will be further analyzed in the Results section; hence, there is no need to repeat them here.

Binary strings were generated by initializing a Mersenne Twister pseudo number generator with a seed obtained from 'std::random_device', thereby harnessing system entropy for a more robust and unpredictable initialization.

For the basic GA, a total of 30 iterations were performed for each experiment. For subsequent adaptations, such as Grey Codes and hypermutation, a partial sample of 10 iterations was used for these experiments.

4 Results

4.1 Results for the 5-dimensional space

Function	**Mean**	**Median**	**Avg Time**	**Min**	**Max**	**Std Dev**	**IQR**
Schwefel	-2094.48686	-2094.69986	0.34510	-2094.80829	-2091.94870	0.62925	0.16483
Rastrigin	0.00000	0.00000	0.38767	0.00000	0.00000	0.00000	0.00000
Michalewicz	-4.67807	-4.68538	0.30425	-4.68761	-4.63899	0.01456	0.01096
De Jong 1	0.00152	0.00001	0.08344	0.00000	0.02570	0.00483	0.00011

Table 1: Summary of GA's performance on the 5-dimensional space

4.2 Results for the 10-dimensional space

Function	Mean	Median	Avg Time	Min	Max	Std Dev	IQR
Schwefel	-4188.04494	-4189.04443	0.64248	-4189.40001	-4180.49877	1.88838	1.67474
Rastrigin	0.00000	0.00000	0.67728	0.00000	0.00000	0.00000	0.00000
Michalewicz	-9.59664	-9.60914	0.51245	-9.65972	-9.46019	0.05289	0.08161
De Jong 1	0.01307	0.00749	0.14926	0.00020	0.05916	0.01605	0.01402

Table 2: Summary of GA's performance on the 10-dimensional space

4.3 Results for the 30-dimensional space, and more

Function	Mean	Median	Avg Time	Min	Max	Std Dev	IQR
Schwefel	-12491.13136	-12502.16711	1.76852	-12532.39661	-12404.42769	33.42110	30.03230
Rastrigin (30D)	0.00000	0.00000	1.89477	0.00000	0.00000	0.00000	0.00000
Rastrigin (50D, 300G)	1.39916	0.00954	4.19949	0.00010	8.93273	2.32014	1.36438
Rastrigin (100D, 100G)	594.43319	593.42997	6.42556	542.25228	652.53824	25.62625	27.50680
Rastrigin (100D, 200G)	220.99934	220.54736	12.55925	138.56118	270.22917	29.27293	37.69567
Rastrigin (100D, 300G)	40.88256	38.66026	18.64221	17.42299	68.70603	11.87162	10.87535
Rastrigin (100D, 400G)	5.16085	5.19433	25.32102	0.72563	12.93982	3.05321	3.70861
Rastrigin (100D, 500G)	2.74712	1.68539	30.99845	0.04044	11.78433	2.74757	3.64248
Rastrigin (100D, 1000G)	2.16654	1.23623	61.76511	0.00003	19.32462	3.75308	3.70861
Michalewicz	-28.72712	-28.76918	1.40788	-29.14783	-28.29714	0.21092	0.32845
De Jong 1	1.65646	1.49006	0.40939	0.78183	3.87855	0.71848	0.96648

Table 3: Summary of GA's performance on the higher dimensional space

4.4 Optimizing the GA

Since genetic algorithms (GAs) yield excellent results, especially in lower-dimensional spaces, there's no need to evaluate them on simpler complexities. Consequently, all experiments will be conducted in the 30-dimensional space.

4.5 Grey Codes

Schwefel's Function:

Experiment 1: In this initial experiment, we ran the genetic algorithm with parameters identical to those of the simple genetic algorithm. This setup proved to be very bad, yielding a mean value around -7600. In contrast, the original GA achieved a significantly superior result, surpassing -12500.

Experiment 2: After extensive parameter tuning, including reducing the population kept as elites from 150 to 100, mutation from $\frac{2}{L}$ to $\frac{0.1}{L}$, and adjusting selection pressure to 0.5, improvements were observed. However, due to the substantial modifications, drawing conclusive insights would not be correct.

Based on the results of these experiments, it becomes evident that Grey Codes are not a suitable implementation for this genetic algorithm, as they led to a suboptimal outcome.

Rastrigin's Function:

Experiment 1: Once again, in the first experiment, the parameters mirrored those of the original algorithm. Surprisingly, the results were incredibly poor. The implementation of Grey Code is known to enhance the genetic algorithm for minimizing Rastrigin's function. However, contrary to expectations, this did not happen.

Experiment 2: Lowering the mutation rate from $\frac{12}{L}$ to $\frac{0.1}{L}$ improved the results, but they remained incomparable to the initial implementation.

Michalewicz's Function:

Experiment 1: The incorporation of Grey Code slightly worsened the mean, possibly within the margin of error given the limited 10 runs.

Experiment 2: Increasing the selection pressure to 3 and the number of iterations to 300 resulted in marginal improvement.

De Jong 1 Function:

Experiment 1: Retaining the same parameters as the old genetic algorithm, a slightly worse mean was observed, though within the margin of error.

Overall, the best that the Grey Code implementation achieved was a slight worsening of the results. This could be attributed to the fast convergence and extensive exploration inherent in the base genetic algorithm. From these results, it can be assumed that this genetic algorithm works more effectively with the arrangement of Hamming walls resulting from the binary representation.

Function	Exp	Mean	Median	Avg Time	Min	Max	Std Dev	IQR
Schwefel (GC)	1	-7660.67486	-7597.71214	1.40572	-8397.56943	-7174.26922	292.20216	1223.30021
Schwefel (GC, Tuning)	2	-10569.35466	-10517.85667	1.42756	-11293.29367	-9681.64198	418.86236	2611.65169
Rastrigin (GC)	1	83.08436	83.27138	2.98636	74.69249	89.29061	3.73941	14.17246
Rastrigin (GC, Tuning)	2	9.58837	9.43744	3.09604	6.00034	14.96166	2.22113	3.68932
Michalewicz (GC)	1	-28.04116	-28.01475	1.71476	-28.58922	-27.58218	0.27574	0.40936
Michalewicz (GC, Tuning)	2	-28.45547	-28.49581	2.55877	-28.83386	-28.11156	0.22238	0.27430
De Jong 1 (GC)	1	2.04529	1.94364	0.53780	1.10183	2.93772	0.54025	1.83589

Table 4: Summary of Grey Codes Experiments

4.6 Adaptive Mutation Probability

Schwefel's Function:

Experiment 1: We started with an initial mutation probability of $\frac{6}{L}$, deviating from the standard $\frac{2}{L}$ used in the initial Genetic Algorithm (GA). This seemed to help, improving the mean, which was a little bit under -12500, to nearly reach the global minima.

Experiment 2: In this experiment, we wanted to observe the outcome of using a non-adaptive mutation probability of $\frac{6}{L}$. The results are noticeably worse.

Experiment 3, 4: When running the GA in the more complex 50-dimensional space, even more favorable results were observed, demonstrating a clear improvement in the function value.

Rastrigin's Function:

Since the basic GA yielded very good results in the 30-dimensional space, the study extended to the 100-dimensional space for the implementation of the adaptive mutation probability.

Experiment 1: For Rastrigin's function, a 30-dimensional run with an initial mutation rate of $\frac{16}{L}$ and a decay factor of 0.01 yielded no significant changes.

Experiment 2: Without adaptive mutation and with a fixed mutation probability value of $\frac{16}{L}$, issues arise, and the mean deteriorates significantly.

Experiment 3 : In this run, we wanted to observe how the generic GA is handling the 100-dimensional space. Because the 100-dimensional space is vastly more complex, we increased the generation number to 400.

Experiment 4 : In this experiment, we implemented the adaptive mutation probability on the previous parameters. The initial mutation rate was set to $\frac{20}{L}$.

Experiment 5: Here, we aimed to observe how the basic GA would manage the function with the mutation rate employed in the adaptive GA. The outcomes were notably unfavorable, signaling that the mutation rate is evidently too high to be used for all of the generations.

Experiment 6: In this case, our goal was to investigate the impact of providing the GA with a larger population to work with. A comparison of these results with the initial GA, which had 500 individuals and 300 generations, and took about the same time to run, showed a clear improvement.

Michalewicz's Function:

Experiment 1: Modifying the base algorithm to run with 300 generations and reducing ϵ from 10^{-3} to 10^{-4} almost reached the 29 marks. With these parameters, we will now implement Adaptive Mutation Probability.

Michalewicz's function appears to be highly sensitive to high mutation rates, as deduced from the tests we conducted. Therefore, careful management of the mutation is crucial to achieve good results in this implementation.

Experiment 2: Setting the initial mutation probability to 1/L and the decay factor to 0.01 seemed to achieve no discernible improvements.

Experiment 3: After tweaking the decay factor to 0.05, the mean was able to reach the 29 marks.

Experiment 4: Setting the initial mutation probability at $\frac{10}{L}$ and the decay factor at 0.01 seems to have a significant impact, resulting in the loss of the best solutions.

Experiment 5: Here, it appears that the lower the mutation probability, the better the results. Other factors have a more substantial impact. Increasing the population from 200 to 300 (with the elite population at 270) ensured that the worst solution remained above 29.

De Jong 1 Function:

Experiment 1: For De Jong 1, changing the mutation probability from $\frac{0.1}{L}$ to $\frac{4}{L}$ and setting the decay factor seemed to worsen the solution.

Similar to Michalewicz's case, lower mutation rates appear to yield better solutions.

Function	Exp	Mean	Median	Avg Time	Min	Max	Std Dev	IQR
Schwefels (30D, Adaptive)	1	-12549.49157	-12551.95975	1.73418	-12565.77601	-12515.09874	10.73967	14.85258
Schwefels (30D, N-Adaptive)	2	-12129.97808	-12124.58043	1.74463	-12019.85308	-12225.58194	54.96911	92.66999
Schwefels (50D, N-Adaptive)	3	-20087.42917	-20064.04351	3.40301	-19979.95848	-20228.60641	85.11614	162.78926
Schwefels (50D, Adaptive)	4	-20680.50192	-20688.97239	3.18826	-20577.89300	-20807.56495	69.59678	95.35209
Rastrigin (30D, Adaptive)	1	0.00001	0.00000	1.69121	0.00000	0.00002	0.00001	0.00001
Rastrigin (30D, N-Adaptive)	2	11.72548	0.77967	1.70573	0.00000	28.92471	14.04658	28.92421
Rastrigin (100D, 200P, 400G)	3	81.71010	90.71728	14.75465	40.48837	108.47913	24.07054	34.94537
Rastrigin (100D, 20/L, Adaptive)	4	32.47406	34.00086	12.44049	20.08983	41.19293	6.27962	7.11983
Rastrigin (100D, 20/L, N-Adaptive)	5	191.06746	186.38600	10.87349	65.71944	276.50338	60.21105	86.69125
Rastrigin (300P, 250E)	6	13.83697	12.57061	16.38436	7.26504	27.90947	5.82226	6.97227
Michalewicz (300G, 0.0001 epsilon)	1	-28.95868	-29.00586	2.28528	-29.25452	-28.57723	0.25126	0.55055
Michalewicz (1/L, 0.01)	2	-28.85605	-28.90714	2.26907	-29.29027	-28.33648	0.26704	0.33774
Michalewicz (1/L, 0.05)	3	-29.05626	-29.08548	2.26564	-29.21044	-28.75106	0.12668	0.15061
Michalewicz (10/L, 0.01)	4	-27.02518	-26.98028	2.21572	-27.33101	-26.75917	0.21036	0.46939
Michalewicz (300P, 270E)	5	-29.22104	-29.21943	3.37338	-29.37970	-29.08078	0.08921	0.08971
De Jong 1 (4/L, 0.01)	1	2.53227	2.09160	0.30504	1.16342	4.66436	1.09341	1.90970

Table 5: Summary of Adaptive Mutation Probability Experiments

This approach was effective in balancing exploration and exploitation. Initially, when the population was diverse, a higher mutation probability encouraged exploration. As the algorithm progressed, and the population converged, the mutation probability decreased, focusing more on exploitation to refine the solutions.

4.7 Hypermutation

Schwefel's Function:

Experiment 1: Since the algorithm uses a low number of generations, indicating that it converges quite quickly, there is a low chance for hypermutation to occur with the initial number of generations. Therefore, for the first experiment, the number of generations has been increased to 300.

Experiment 2: With hypermutation implemented (triggered after 30 generations without improvement with a 10% probability), the results did not show any improvement.

Experiment 3, 4: Expanding the dimensional space to 50 and increasing the generation number to 500 did not result in any noticeable improvement. Experiment 4 was conducted without using hypermutation, and the results are almost identical.

Experiment 5: Decreasing the number of generations without improvement to 10, in which case the mutation is applied with a 1% probability, seemed to worsen the results.

Rastrigin's Function:

Experiment 1: Implementing hypermutation with a 20% probability when there were 50 consecutive generations without improvement seemed to make no change in the mean, probably because it had no chance to occur in the first place.

Experiment 2: Hypermutation doesn't seem to affect Rastrigin's function in any way, and since the results are already at the global minima, we are going to study the 100-dimensional space. The baseline for this experiment is set as follows: 100 dimensions, 300 individuals (of which 250 are kept as elites), and 400 generations.

Experiment 3: In this experiment, we implemented hypermutation in an identical way to the first case.

Experiment 4: Decreasing the number of generations without improvement to 30 led to catastrophic results.

Experiment 5: However, reverting to 50 generations without improvement and lowering the mutation probability to 10% resulted in a significantly better solution for the mean.

Michalewicz's Function:

Experiment 1: Due to Michalewicz's function sensitivity to mutation, a 5% mutation rate after 20 generations without improvement was employed. Results were satisfactory, improving the mean.

Experiment 2: Increasing the number of generations to 1000 did not lead to further improvement in the best solution, indicating that multiple occurrences of hypermutation were not able to improve the solution any further.

De Jong 1 Function:

Experiment 1: This run was conducted with the base algorithm, but with 300 generations, providing a baseline for comparison with the modified algorithm.

Experiment 2: Resetting genes with a 10% probability after 50 generations without improvement yielded potentially improved results but remained within the margin of error.

Function	Exp	Mean	Median	Avg Time	Min	Max	Std Dev	IQR
Schwefel (300G)	1	-12534.66462	-12537.96727	2.65641	-12563.29343	-12488.35548	21.11800	29.72077
Schwefel (Hyper.)	2	-12533.13575	-12537.57740	2.61711	-12556.73428	-12491.90807	19.51070	27.45293
Schwefel (50D, Hyper.)	3	-20728.61846	-20732.36485	7.06629	-20854.86985	-20558.09004	102.69025	166.05591
Schwefel (50D, No Hyper.)	4	-20729.00992	-20758.07340	9.31334	-20848.78490	-20510.87863	104.29685	170.97881
Schwefel (Modified Hyper.)	5	-12506.42816	-12507.97789	2.59581	-12536.33097	-12460.43929	19.19411	17.07797
Rastrigin (30D, Hyper.)	1	0.00000	0.00000	1.86797	0.00000	0.00000	0.00000	0.00000
Rastrigin (100D, No Hyper.)	2	36.86554	41.91842	16.51944	17.62986	53.80910	12.57460	14.43357
Rastrigin (100D, Hyper. 1)	3	33.79648	31.07135	16.55701	19.48260	59.07355	11.78938	17.55987
Rastrigin (100D, Hyper. 2)	4	111.70942	86.99667	16.16218	49.77933	246.82345	69.10669	2.04325
Rastrigin (100D, Hyper. 3)	5	23.23175	20.82644	16.12806	13.63185	40.11209	9.07807	5.87949
Michalewicz (Hyper. 1)	1	-29.00307	-29.02051	2.26844	-29.19692	-28.71920	0.14450	0.26012
Michalewicz (Hyper. 1, 1000G)	2	-29.00164	-29.04846	7.48562	-29.27581	-28.60653	0.20656	0.30939
De Jong 1 (300G, No Hyper.)	1	1.22713	1.17075	0.87475	0.48904	2.29289	0.60358	0.92181
De Jong 1 (300G, Hyper.)	2	0.72895	0.51046	0.87433	0.29261	1.70257	0.44983	0.54178

Table 6: Summary of Hypermutation Experiments

4.8 Adaptive Crossover Probability

Schwefel's Function:

Experiment 1: Initiating the GA with a crossover probability of 50% and a growth factor of 0.01 yielded average results.

Experiment 2: In contrast, employing a non-adaptive crossover probability of 50% did not yield any advantages over the basic implementation.

Experiment 3: Revisiting the base GA performances without any adaptive crossover seemed to work best.

Rastrigin's Function:

Experiment 1: Setting the initial crossover rate to 10% and the growth factor to 0.01 resulted in catastrophic outcomes, emphasizing the importance of exploitation, especially at the beginning.

Experiment 2: Increasing the crossover probability to 50% improved results but was still inferior to the non-adaptive approach.

Michalewicz's Function:

Experiment 1: Initiating with a crossover probability of 10% and a growth factor of 0.1 worsened the solution.

Experiment 2: Starting with a crossover probability of 0.5 improved results, likely due to the rapid increase in the crossover probability given the large growth factor.

Experiment 3: Lowering the growth factor to 0.01 worsened the results, reinforcing the correlation between a higher crossover probability and better outcomes in this GA implementation.

De Jong 1 Function:

Experiment 1: Starting the GA with a low crossover probability of 10% and a growth factor of 0.01 resulted in poor outcomes.

Experiment 2: Increasing the growth factor to 0.1 yielded better but still suboptimal results compared to the generic GA.

Function	Exp	Mean	Median	Avg Time	Min	Max	Std Dev	IQR
Schwefel (Adaptive 50 % CX)	1	-12495.43095	-12496.80812	1.78779	-12538.18714	-12453.31583	23.86059	30.98445
Schwefel (Fixed 50% CX)	2	-12332.42983	-12341.11693	1.44129	-12437.07573	-12240.70097	58.43293	94.98618
Schwefel (100% CX)	3	-12508.60033	-12517.87753	1.75102	-12536.30451	-12455.90464	23.08266	36.11989
Rastrigin (Adaptive CX 0.1)	1	140.65749	138.49685	1.33260	86.77414	198.62654	29.69915	36.93150
Rastrigin (Adaptive CX 0.5)	2	1.10827	0.00000	1.72889	0.00000	4.92342	1.94229	1.23582
Michalewicz (Adaptive CX 0.1)	1	-28.18094	-28.16936	2.14868	-28.82766	-27.66339	0.35241	0.51472
Michalewicz (Adaptive CX 0.5, 1)	2	-28.82990	-28.81648	2.23449	-29.25293	-28.48459	0.22959	0.36069
Michalewicz (Adaptive CX 0.5, 2)	3	-28.63688	-28.61086	2.27633	-29.01730	-28.37158	0.16620	0.13170
De Jong 1 (Adaptive CX 1)	1	15.97454	15.16012	0.64209	2.08457	29.84416	8.25801	8.83273
De Jong 1 (Adaptive CX 2)	2	7.65781	6.79020	0.82798	0.98230	17.20072	5.06341	5.99444

Table 7: Summary of Adaptive Crossover Experiments

4.9 Adaptive Mutation Probability and Hypermutation

Since only the adaptive mutation and hypermutation exhibited enhancements in the GA, we'll combine these elements to assess their combined effects on each function.

Schwefel's Function:

Experiment 1: Initially, we used the parameters for adaptive mutation ($\frac{6}{L}$ initial mutation rate) and added a condition that increased to 20% the mutation probability if 30 generations passed with no improvement. The experiment extended to 300 generations, but no improvement was observed over the standard adaptive mutation.

Several similar experiments were conducted, but none yielded significant results. Consistency in reaching the global minima was challenging or unattainable.

Rastrigin's Function:

Experiment 1: Hypermutation was implemented with a 20% probability when 20 generations went without improvement. The initial mutation rate was set to $\frac{25}{L}$, and the decay factor was set to 0.01. This achieved results ten times better than the basic GA, within the same time frame, in the 100-dimensional space.

Michalewicz's Function:

Experiment 1: This initial experiment used a $\frac{1}{L}$ initial mutation, 0.05 decay factor, and a 5% mutation gene reset phase after 20 generations with no improvement.

Experiment 2: Recognizing the limited relevance of mutation, an attempt was made to optimize the basic Genetic Algorithm (GA) by reducing ϵ from 10^{-4} to 10^{-5}. The initial mutation rate was also increased to $\frac{2}{L}$.

Experiment 3: Further optimization involved increasing the number of generations to 400, setting the activation condition to 50 generations without improvement, and adjusting the hypermutation probability to 10%.

De Jong 1 Function:

Experiment 1: The initial mutation probability was set at $\frac{1}{L}$, with a decay factor of 0.01. After 30 generations with no improvement, a 20% mutation stage was performed. This used 100 generations.

Experiment 2: The first experiment was modified to run for 500 generations.

Experiment 3: Removing hypermutation and adaptive mutation seemed to somewhat improve the mean.

In conclusion, none of the modified GA configurations succeeded in improving the mean performance on the De Jong 1 function.

Function	**Exp**	**Mean**	**Median**	**Avg Time**	**Min**	**Max**	**Std Dev**	**IQR**
Schwefel	1	-12540.34064	-12542.21039	4.41433	-12553.78502	-12521.76724	10.35596	19.09421
Rastrigin (100D)	1	4.80379	1.84176	18.29784	0.00930	16.28519	6.16379	7.91721
Michalewicz (1/L)	1	-28.92027	-29.01519	2.25134	-29.22617	-28.59155	0.22181	0.35995
Michalewicz (2/L)	2	-29.08123	-29.13658	3.03415	-29.22232	-28.84375	0.12976	0.22184
Michalewicz (400G)	3	-29.21710	-29.14603	4.10773	-29.37664	-29.05235	0.13193	0.23007
De Jong 1 (Adaptive)	1	1.91437	1.81424	0.35572	1.27614	3.52222	0.63647	0.80925
De Jong 1 (Adaptive, 500G)	2	1.06416	0.87372	1.72354	0.42540	2.09819	0.53275	0.69067
De Jong 1 (Simple, 500G)	3	0.76825	0.72496	1.76149	0.20627	1.63598	0.46189	0.53541

Table 8: Summary of Experiments Combining Adaptive Mutation and Hypermutation

5 Comparing the Results

We'll compare the standard Genetic Algorithm (GA) with its modified counterpart, denoted as AGA in the table, alongside the Hill Climbing (HC) method (Best Improvement Version) and Simulated Annealing (SA). Our examination will focus on the 30-dimensional space, where the most noticeable differences become apparent.

After examination, it's clear that GA outperforms HC and SA, especially in complex functions where SA is typically known for faster time complexity. AGA also shows superior performance across most functions, but due to results closely approaching the global minima in both cases, differences are not that notable.

Function	**Mean**	**Avg Time (s)**	**Min Value**	**Max Value**	**StdDev**	**IQR**
De Jong 1 HC	0.00000	0.32021	0.00000	0.00000	0.00000	0.00000
De Jong 1 SA	0.00042	2.94586	0.00023	0.00080	0.00019	0.00020
De Jong 1 GA	1.65646	0.40939	0.78183	3.87855	0.71848	0.96648
De Jong 1 AGA	0.72895	0.87433	0.29261	1.70257	0.44983	0.54178
Schwefel HC	-10599.32729	8.84485	-11105.20120	-9972.67254	212.11733	320.32227
Schwefel SA	-12516.92261	2.95995	-12568.64841	-12296.91321	105.20059	129.70564
Schwefel GA	-12491.13136	1.76852	-12532.39661	-12404.42769	33.42110	30.03230
Schwefel AGA	-12549.49157	1.73418	-12565.77601	-12515.09874	10.73967	14.85258
Rastrigin HC	28.03236	389.93805	20.33712	32.23608	3.21356	3.95454
Rastrigin SA	15.54186	2.75145	6.28744	26.49976	4.07673	15.01699
Rastrigin GA	0.00000	1.89477	0.00000	0.00000	0.00000	0.00000
Rastrigin AGA	0.00000	1.86797	0.00000	0.00000	0.00000	0.00000
Michalewicz HC	-25.44161	3.29517	-26.22963	-24.79306	0.37175	0.68674
Michalewicz SA	-26.01103	1.63445	-27.45690	-24.42465	0.65915	0.25970
Michalewicz GA	-28.72712	1.40788	-29.14783	-28.29714	0.21092	0.32845
Michalewicz AGA	-29.21710	4.10773	-29.37664	-29.05235	0.13193	0.23007

Table 9: Combined Results for 30-dimensional space: HC (Best Improvement), SA, and GA

6 Conclusion

This study clearly demonstrated the significant advantages of genetic algorithms over the other two heuristic methods, namely Hill Climbers and Simulated Annealing. The differences in both mean values obtained and time complexity were so substantial that a direct comparison is unfair. There is no practical reason to choose Hill Climbers or Simulated Annealing over genetic algorithms, except, of course, when dealing with very simple and convex functions like De Jong 1. In such cases, Hill Climbers is the preferable choice. Simulated Annealing did not excel in any aspect; although there might be specific problems or domains where Simulated Annealing could outperform genetic algorithms. This study did not identify any such instances.

References

[1] Eugen Croitoru, Teaching: Genetic Algorithms https://profs.info.uaic.ro/~eugennc/teaching/ga/

[2] Albert Alexandru, Testing Standard Benchmark Functions. https://github.com/Alex90210/Testing-Standard-Benchmark-Functions-with-Hill-Climbing-and-Simulated-Annealing

[3] Wikipedia, Simulated annealing. https://en.wikipedia.org/wiki/Simulated_annealing

[4] Wikipedia, Hill climbers. https://en.wikipedia.org/wiki/Hill_climbing

[5] Wikipedia, Genetic Algorithms. https://en.wikipedia.org/wiki/Genetic_algorithm

[6] Wikipedia, Adaptive Mutation. https://en.wikipedia.org/wiki/Adaptive_mutation

[7] Wikipedia, Crossover. https://en.wikipedia.org/wiki/Crossover_(genetic_algorithm)

[8] Wikipedia, Crossover. https://en.wikipedia.org/wiki/Crossover_(genetic_algorithm)

[9] Wikipedia, Selection. https://en.wikipedia.org/wiki/Fitness_proportionate_selection

[10] Wikipedia, Grey Codes. https://en.wikipedia.org/wiki/Gray_code

[11] Robert Miles, Hill Climbing Algorithm and Artificial Intelligence - Computerphile. https://www.youtube.com/watch?v=oSdPmxRCWws&ab_channel=Computerphile

[12] Function photos. https://al-roomi.org/benchmarks/unconstrained/n-dimensions/174-generalized-rastrigin-s-function