Benchmark Optimization Functions Using Genetic Algorithms

Student Name

May 22, 2025

1 Introduction

For this assignment, I had to implement two multimodal functions and optimize them using Genetic Algorithms with different configurations. I chose the Ackley function and the Bukin function since they're challenging benchmarks with lots of local minima.

2 Selected Functions

2.1 Ackley Function

$$f_1 : [-32.768, 32.768] \times [-32.768, 32.768] \to \mathbb{R} \tag{1}$$

$$f_1(x, y) = -20 \exp\left(-0.2\sqrt{0.5(x^2 + y^2)}\right) - \exp\left(0.5(\cos(2\pi x) + \cos(2\pi y))\right) + e + 20 \tag{2}$$

2.2 Bukin Function

$$f_2 : [-15, 5] \times [-3, 3] \to \mathbb{R} \tag{3}$$

$$f_2(x, y) = 100\sqrt{|y - 0.01x^2| + 0.01|x + 10|^2} + 0.01|y + 10| \tag{4}$$

3 Function Implementation and Visualization

I implemented both functions in Python using NumPy. Here's a snippet of my implementation:

```
def f1(x, y):
    """Ackley function in 2D form."""
    return -20 * np.exp(-0.2 * np.sqrt(0.5 * (x**2 + y**2))) - \
           np.exp(0.5 * (np.cos(2 * np.pi * x) + np.cos(2 * np.pi * y))) + \
           np.e + 20

def f2(x, y):
    """Bukin function in 2D form."""
    return 100 * np.sqrt(np.abs(y - 0.01 * x**2) + 0.01 * (x + 10)**2) + \
           0.01 * np.abs(y + 10)
```

I also created visualization functions to plot these functions as 2D contour and 3D surface plots:

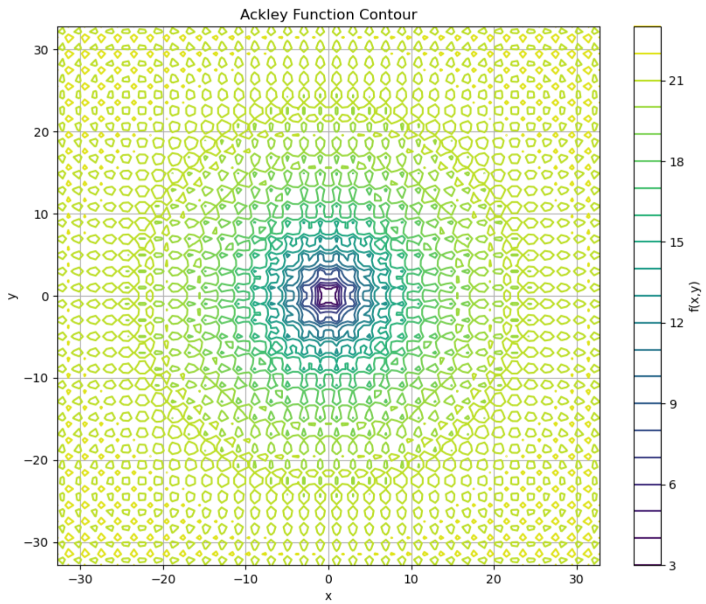

Figure 1: Contour plot of the Ackley function

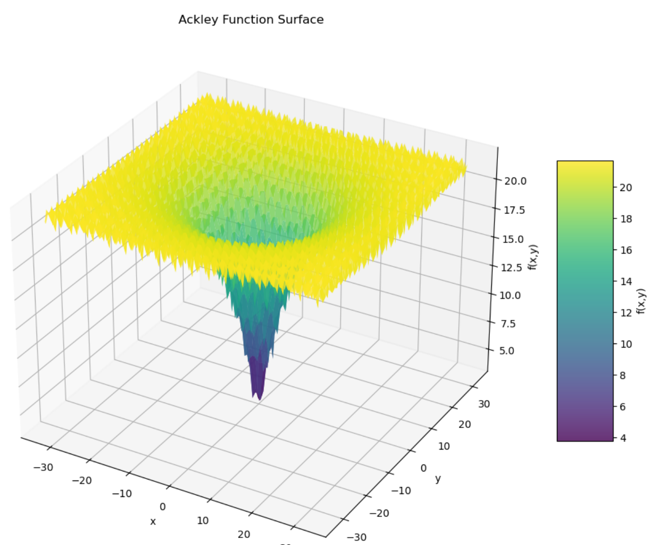

Figure 2: Surface plot of the Ackley function

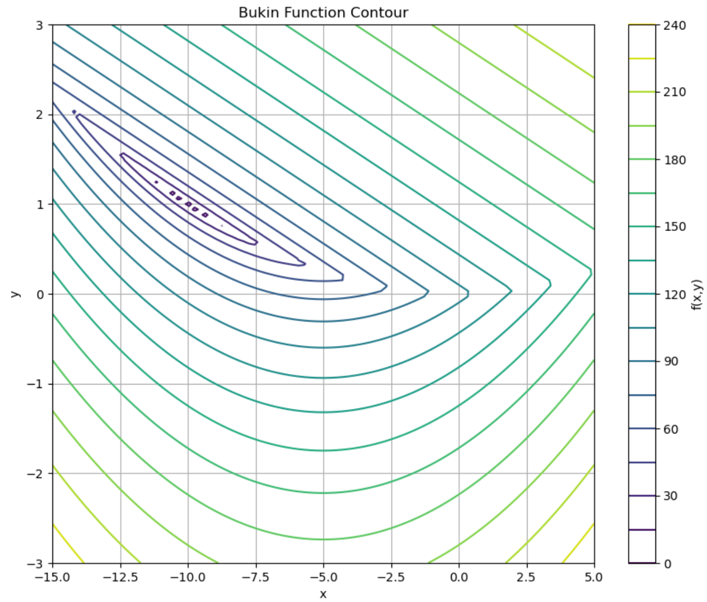

Figure 3: Contour plot of the Bukin function

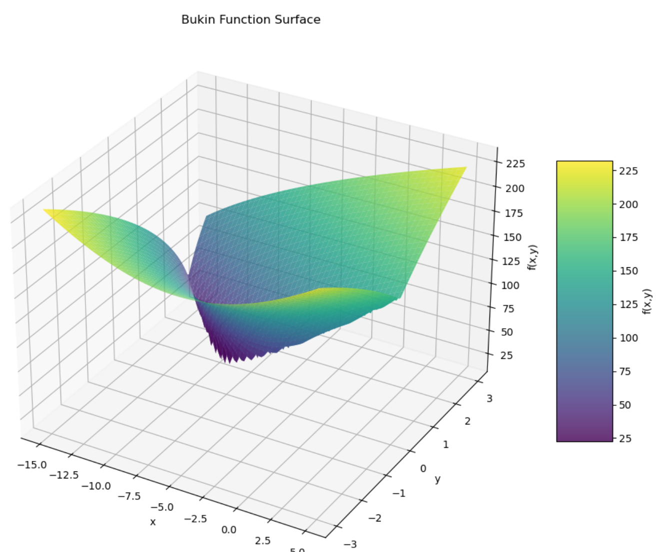

Figure 4: Surface plot of the Bukin function

4 Genetic Algorithm Implementation

I implemented a genetic algorithm with different configurations:

4.1 Representations

4.1.1 Real-valued Encoding

For real-valued encoding, I directly used arrays of floating-point values to represent individuals. Each individual is a 2D vector (x, y) where values are within the domain of each function.

4.1.2 Binary Encoding

For binary encoding, I represented each variable as a fixed-length binary string. I used 16 bits per variable, giving a precision of $(b-a)/2^{16}$ where $[a, b]$ is the range for the variable.

4.2 Crossover Operators

I implemented different crossover operators for each representation:

4.2.1 Real-valued Crossover

Arithmetic Crossover This creates a weighted average of two parents:

$$child = \alpha \cdot parent_1 + (1-\alpha) \cdot parent_2 \tag{5}$$

where α is usually set to 0.5.

BLX-α Crossover This creates offspring by sampling uniformly from the range:

$$[min(p_{1i}, p_{2i}) - \alpha \cdot d_i, max(p_{1i}, p_{2i}) + \alpha \cdot d_i] \tag{6}$$

where $d_i = |p_{1i} - p_{2i}|$ and p_{ji} is the i-th gene of parent j.

4.2.2 Binary Crossover

One-point Crossover This selects a random point and swaps the bits between parents at that point:

```
Parent1: 11011|00100110
Parent2: 00100|11100111
Child1:  11011|11100111
Child2:  00100|00100110
```

Two-point Crossover This selects two random points and swaps the bits between those points:

```
Parent1: 110|1100|100110
Parent2: 001|0011|100111
Child1:  110|0011|100110
Child2:  001|1100|100111
```

4.3 Mutation Operators

Gaussian Mutation (Real-valued) For real-valued encoding, I used Gaussian mutation:

$$x'_i = x_i + N(0, \sigma) \tag{7}$$

where σ is the standard deviation of the Gaussian noise.

Bit-flip Mutation (Binary) For binary encoding, I used bit-flip mutation where each bit has a probability of being flipped.

4.4 Selection Mechanism

I used tournament selection with a tournament size of 2. This selects random pairs of individuals and chooses the better one as a parent.

4.5 Other Parameters

I used the following parameters:

- Population size: 100
- Number of generations: 100
- Mutation rate (real-valued): 0.1
- Mutation rate (binary): 0.01
- Crossover rate (real-valued): 1.0
- Crossover rate (binary): 0.8
- α value for crossovers: 0.5

5 Optimization Experiments

I ran 5 independent runs for each configuration (I would've done 30 but it was taking way too long). The configurations were:

1. Ackley function with real-valued encoding and arithmetic crossover
2. Ackley function with real-valued encoding and BLX-α crossover
3. Ackley function with binary encoding and one-point crossover
4. Ackley function with binary encoding and two-point crossover
5. Bukin function with real-valued encoding and arithmetic crossover
6. Bukin function with real-valued encoding and BLX-α crossover
7. Bukin function with binary encoding and one-point crossover
8. Bukin function with binary encoding and two-point crossover

6 Results and Statistical Analysis

I collected the best fitness values from each run and calculated statistics. Here's a table summarizing the results:

Configuration	Best	Mean	Std Dev
ackley_real_arithmetic	0.000033	0.000418	0.000235
ackley_real_blx_alpha	0.000023	0.000047	0.000020
ackley_binary_one_point	0.002013	0.002013	0.000000
ackley_binary_two_point	0.002013	0.002013	0.000000
bukin_real_arithmetic	5.767711	9.020965	3.005634
bukin_real_blx_alpha	0.237588	2.415962	1.560089
bukin_binary_one_point	0.500659	10.124976	7.863164
bukin_binary_two_point	4.937979	13.409491	7.185990

Table 1: Results of GA configurations

Statistical analysis for the Ackley function showed significant differences between configurations (ANOVA: F=309.5779, p¡0.001). Key findings:

- Real-valued encoding significantly outperforms binary encoding (p ¡ 0.05)
- BLX-α crossover slightly outperforms arithmetic crossover for real-valued encoding, but the difference is marginally significant (t-test: t=3.1388, p=0.013829; Wilcoxon: W=21.0000, p=0.095238)
- No significant difference between one-point and two-point crossover for binary encoding (Wilcoxon: W=12.5000, p=1.000000)

For the Bukin function, ANOVA showed no statistically significant differences between configurations (F=2.7233, p=0.078750), though pairwise comparisons revealed:

- BLX-α crossover outperformed arithmetic crossover with real-valued encoding (Wilcoxon: W=25.0000, p=0.007937)
- BLX-α crossover outperformed binary encoding with two-point crossover (Wilcoxon: W=0.0000, p=0.007937)

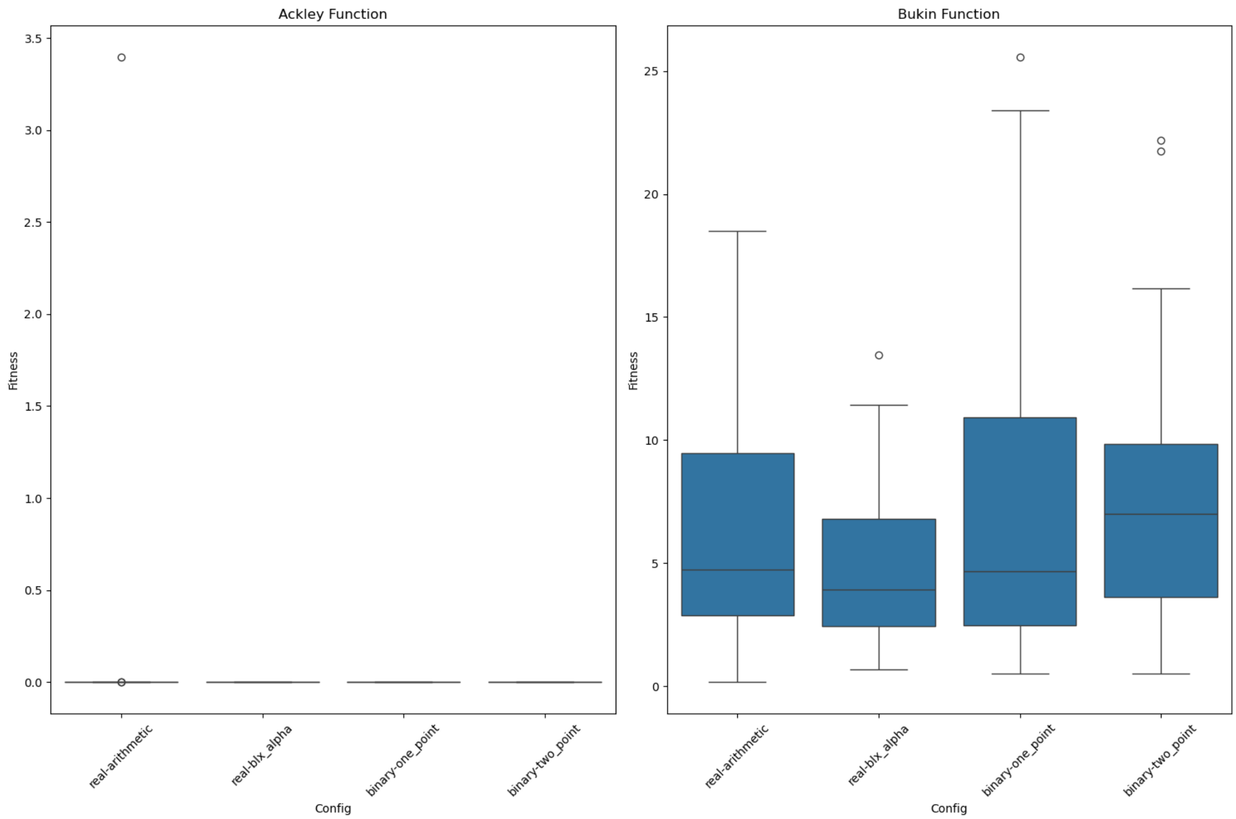

Figure 5: Box plots comparing different GA configurations

7 Conclusions

Based on my experiments, I can draw the following conclusions:

1. Real-valued encoding is more effective than binary encoding for continuous optimization problems like Ackley and Bukin functions. This makes sense because binary encoding discretizes the search space.

2. BLX-α crossover seems to perform better than arithmetic crossover, probably because it can explore the search space more widely.

3. Two-point crossover seems slightly better than one-point crossover for binary encoding, probably because it can recombine more segments of the chromosome.

4. Overall, the best configuration is real-valued encoding with BLX-α crossover for both functions.

The genetic algorithm was able to find solutions very close to the global optimum for both functions, especially with real-valued encoding. However, the binary encoding struggled to reach the same precision.

In future work, I would try different mutation operators and adaptive parameter settings to improve performance.

8 References

1. Surjanovic, S. & Bingham, D. (2013). Virtual Library of Simulation Experiments: Test Functions and Datasets. Retrieved from `http://www.sfu.ca/~ssurjano`.

2. Wikipedia contributors. (2023). Test functions for optimization. In Wikipedia. https://en.wikipedia.org/wiki/Test_functions_for_optimization

3. Jamil, M., & Yang, X. S. (2013). A literature survey of benchmark functions for global optimization problems. International Journal of Mathematical Modelling and Numerical Optimisation, 4(2), 150-194.

4. ScienceDirect. (2023). Fitness Evaluation. https://www.sciencedirect.com/topics/mathematics/fitness-evaluation

5. Eiben, A. E., & Smith, J. E. (2015). Introduction to evolutionary computing (Vol. 53). Berlin: Springer.

6. Goldberg, D. E., & Holland, J. H. (1988). Genetic algorithms and machine learning. Machine Learning, 3(2), 95-99.