筑波大学
University of Tsukuba

Experiment Design in Computer Science

Report 2: Comparative Analysis of Differential Evolution Algorithms Under Varied Configuration Settings

Mamanchuk Mykola, SID.202420671

July 16, 2024

1 Abstract

Upon studying the differential evolution (DE) algorithms using statistical techniques, this report evaluates the influence of parameter tuning on algorithm performance. The primary objective was to assess different DE configurations by manipulating parameters such as the number of benchmarks, repetition counts, Differential Weight (F), and Crossover Probability (Cr). It was found that all three selection policies exhibit statistically significant differences when specific parameter sets are applied and a substantial data volume is collected, enabling precise analysis. The findings suggest that parameter tuning plays a crucial role in the performance of DE algorithms, with some observations raising questions for further analysis. This report aims to present these findings systematically, providing a basis for more detailed investigations into DE algorithm optimization.

2 Introduction

2.1 Background

DE is a robust, population-based optimization algorithm commonly used to solve complex multi-dimensional functions that are otherwise challenging for traditional optimization methods. It relies on mechanisms of mutation, crossover, and selection to evolve solutions

toward an optimum. The effectiveness and efficiency of DE can significantly vary based on its parameter settings: these parameters control the perturbation of vectors and the recombination process, respectively, playing crucial roles in the convergence behavior of the algorithm.

This report explores DE within the framework of the CEC 2014 optimization benchmark set, which provides a diverse set of problems to test the adaptability and efficiency of evolutionary algorithms.

2.2 Objectives

The primary objective is to systematically analyze how different configurations of the DE algorithm influence its performance. This involves a comparative analysis of three distinct selection policies:

- **Select Best:** Focuses on selecting the best solution vectors for mutation and crossover, promoting rapid convergence.
- **Select Random:** Employs a random selection strategy, enhancing diversity in the population but potentially slowing convergence.
- **Select Random-to-Best:** A hybrid approach that mixes the best vector with random selections to balance exploration and exploitation.

Further, the report discuses:

1. Evaluate fine-tuning of parameters F and Cr on each selection policy to discern optimal settings for different types of optimization problems.
2. Determine the statistical significance of observed differences in performance by employing robust statistical techniques.
3. Estimate the required sample size and calculate power to ensure that the experimental design can reliably detect a practical difference, if one exists, between the configurations.

3 Methodology

3.1 Experimental Setup

The experimental setup is outlined as follows:

Software:

- Primary platform: Python v3.12.2
- Utilized libraries include pygmo, pandas, seaborn, matplotlib, scipy, statsmodels, itertools, numpy, scikit_posthocs, and other standard packages.
- Development environment: Visual Studio Code v1.91.0
- Operating System: MacOS 13.6.7 22G720 x86_64

Hardware:

- Processor: Intel i7-7920HQ 8 @ 3.10GHz
- Memory: DRAM LPDDR3 2133MHz 16GB

Utilized Scripts:

- `base_script.py` [2]: Provided as part of the assignment. Implements essential steps for data collection. This script was modified for semi-automatic output of data in a convenient way to create many distinct collections of individual experiments. It also provides information on individual algorithm's runtime.
- `all_stats.py` [2]: Main supplement, which implements statistical calculations, testing, validation, plotting, and file management essentials in a convenient way.
- `sample_size.py` [2]: Supplementary script used to calculate hypothetical sample size based on conditions stated in the task by simulating the Kruskal-Wallis non-parametric method. Also available in Appendix A.
- `record_performance_and_runtime.py` [2]: Supplementary script that takes out best cases for performance for Selection Policies based on F and CR and outputs into a file; outputs runtime of algorithms based on F and CR into a separate file. Availavle in Appendix B.

3.2 Data Collection

Data was collected by running `base_script.py` multiple times with various parameters. Following analysis, once hypotheses of individual tests are rejected or accepted, as assumptions are validated, parameters are tuned, and sample size is recalculated, data collection proceeds based on conditional needs of parameter settings for further analysis.

3.3 Statistical Techniques

This analysis leverages extensive use of Null Hypothesis Significance Testing (NHST) along with various other statistical tools to accurately understand, plot, and make informed decisions. More details will be covered in the subsequent section.

4 Data Analysis

4.1 Statistical Tests

4.1.1 Descriptive Statistics

For each combination of selection policy and problem, the mean, median, standard deviation, and variance of the best performance metrics were calculated. These descriptive statistics provide a summary of the central tendency and dispersion of the results, which are critical for understanding the overall performance of each algorithm configuration. Below is a sample table with the best results for one data collection:

Selection Policy	Problem	Mean	Median	Std Dev	Variance
0	0	2.519e+08	1.966e+06	8.052e+08	6.484e+17
0	1	6.103e+09	678.709	1.815e+10	3.296e+20
0	2	2.325e+06	3379.756	1.485e+07	2.205e+14
0	3	2525.907	424.238	6628.276	4.393e+07
0	4	520.3022	520.1575	0.3618	0.1309
0	5	608.0978	604.3698	8.902	79.251
0	6	761.3331	700.0996	184.716	34120
0	7	860.8563	808.7943	111.646	12465
1	0	2.537e+08	2.726e+06	8.191e+08	6.709e+17
1	1	5.890e+09	10032.26	1.762e+10	3.104e+20
...	...	...	...	...	...

Table 1: Essential statistics for the dataset 20240716062920-8-194-0.7-0.5, where Problems=8, Iterations=194, F (Differential Weight)=0.7, CR (Crossover Probability)=0.5. Runtime: 270.34 seconds, individually iterated: alg1=6.25s, alg2=5.91s, alg3=6.85s.

Detailed analysis and discussion of these results follow in the subsequent sections.

4.2 Shapiro-Wilk Test for Normality

The Shapiro-Wilk test was conducted to assess the normality of the distribution of the dataset. The hypotheses tested were:

- **Null Hypothesis** (H_0): The data is drawn from a normal distribution.
- **Alternative Hypothesis** (H_1): The data is not drawn from a normal distribution.

The following table presents the Shapiro-Wilk test results for the dataset identified as ‘20240716062920-8-194-0.7-0.5‘, encompassing various selection policies across different problems:

Given the extremely small p-values, the null hypothesis (H_0) is rejected for each group, suggesting that the data does not follow a normal distribution. This outcome necessitates the use of non-parametric tests for further analysis. In data collections with a lower sample size, the p-values are relatively higher yet remain low enough to suggest deviations from normality. These results indicate that, although rare, there are instances where the data may appear to follow a normal distribution, as illustrated by subsequent QQ-plots.

Problem	Selection Policy	SW Statistic	p-value
0.0	0	0.3525	1.02×10^{-63}
0.0	1	0.3496	8.38×10^{-64}
0.0	2	0.3507	9.01×10^{-64}
1.0	0	0.3677	1.19×10^{-62}
1.0	1	0.3658	2.48×10^{-63}
1.0	2	0.3679	1.11×10^{-62}
2.0	0	0.1441	5.08×10^{-69}
2.0	1	0.1157	1.18×10^{-69}
2.0	2	0.1322	2.73×10^{-69}
3.0	0	0.3571	1.38×10^{-63}
3.0	1	0.3544	1.15×10^{-63}
3.0	2	0.3541	1.13×10^{-63}
4.0	0	0.5964	3.14×10^{-55}

Table 2: Shapiro-Wilk test results for normality check.

Intermediate progression showcase On this plot, we can clearly see a monotone decreasing values, suggesting that the algorithms perform as expected with all problems.

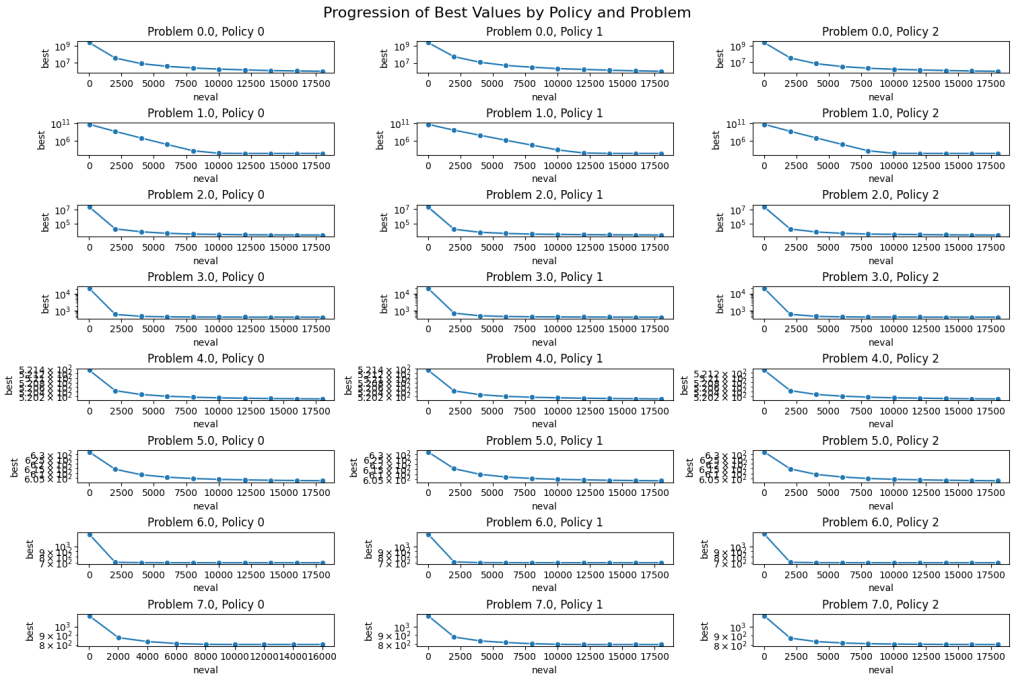

Figure 1: Progression results for 20240716062920-8-194-0.7-0.5

4.2.1 Quantile-Quantile Plots Showcase

Quantile-Quantile (QQ) plots provide a graphical method to assess whether the data follows a specified distribution—in this case, the normal distribution. Below are the QQ plots for two specific cases from the dataset '20240716062920-8-194-0.7-0.5':

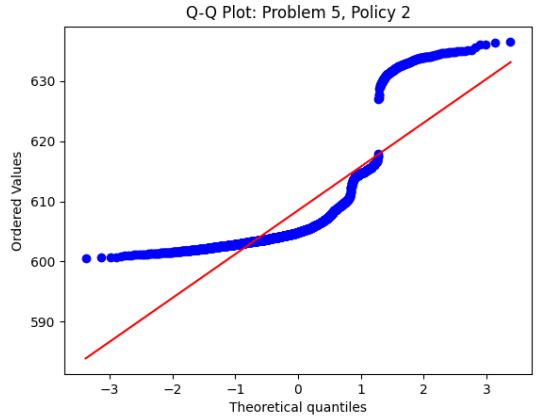

Figure 2: (a) Algorithm 2, Problem 5

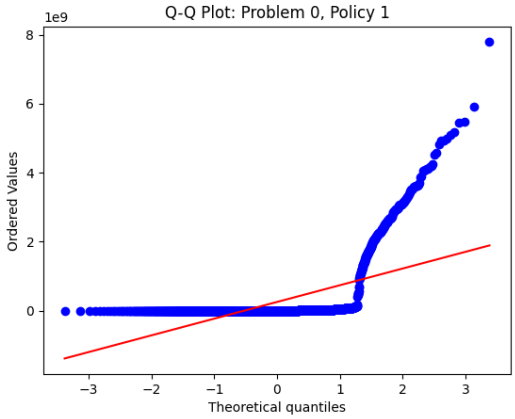

Figure 3: (b) Algorithm 1, Problem 0

Figure 4: QQ Plots for 20240716062920-8-194-0.7-0.5 showing deviations from normality. (a) Shows less significant deviation from the normality line for Algorithm 2, Problem 5, suggesting closer adherence to a normal distribution. (b) Shows significant deviation for Algorithm 1, Problem 0, indicating a departure from normal distribution.

These plots illustrate how the data distribution compares against a theoretical normal distribution. For Algorithm 2, Problem 5, the data shows a less significant deviation from the normality line, suggesting a closer adherence to normal distribution characteristics. In contrast, Algorithm 1, Problem 0 displays a significant deviation, clearly indicating that the data does not follow a normal distribution. These observations are consistent across different datasets, reinforcing the rejection of the hypothesis that the data follows a normal distribution.

5 Visualizing using Box Plots and Discussing Tuning of F and CR

This section discusses the impact of parameter tuning on the performance of Differential Evolution (DE) algorithms, particularly focusing on the Differential Weight (F) and Crossover Probability (CR). By adjusting these parameters, we can observe varying behaviors in the optimization process which are captured through the distribution of final best values as illustrated in the provided box plots.

5.1 Influence of Parameters on Algorithm Performance

5.1.1 High Crossover Probability (CR=1)

Figure 5 shows the box plot for DE algorithms with a high crossover probability (CR=1) and a differential weight (F=0.5). Here, each algorithm displays distinct convergence behaviors across different benchmark problems. This setting appears to promote rapid convergence for Algorithm 1 ("Select Best"), as it leverages the best solution vectors for mutation and crossover. Conversely, Algorithm 2 ("Select Random") and Algorithm 3 ("Select Random-to-Best") exhibit more diversity in their search but at a cost of potentially slower convergence rates. This variety in algorithmic behavior underscores the sensitivity of DE outcomes to crossover settings, with high CR fostering more aggressive exploration strategies.

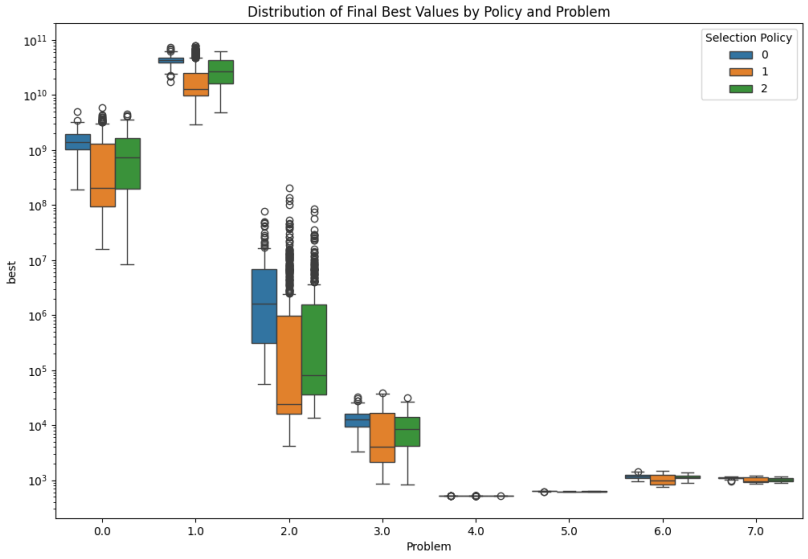

Figure 5: Box plot showing the performance distribution of DE algorithms with high crossover probability (CR=1) across various problems.

5.1.2 Moderate Parameter Settings

In contrast, Figure 6 with a moderate setting of F=0.7 and CR=0.5 illustrates less pronounced differences among the algorithms, particularly between Algorithms 1 and 3. This

suggests that under more balanced parameter settings, the inherent strategies of these algorithms towards finding the global optimum converge, resulting in similar performance profiles across different problems. This effect emphasizes the nuanced impact that fine-tuning of parameters can have, potentially mitigating extreme behaviors observed in more aggressive or conservative settings.

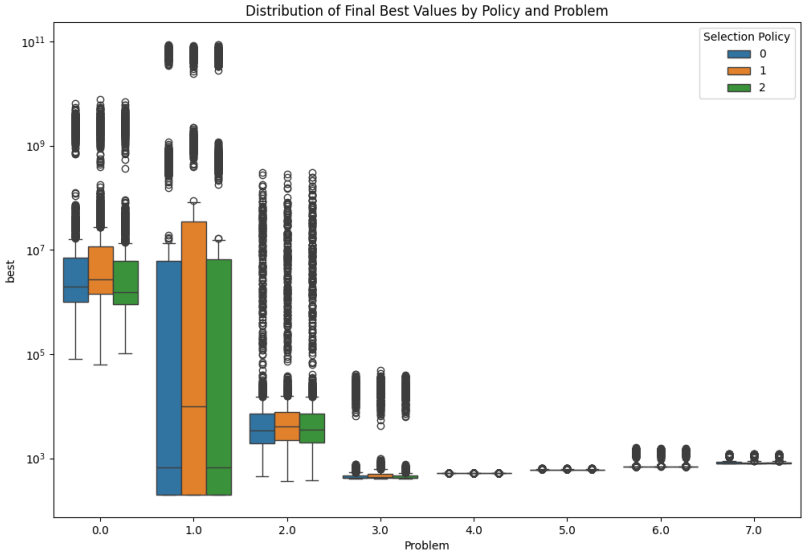

Figure 6: Comparison of DE algorithm performances with moderate settings of F and CR, highlighting minimal performance variance.

5.2 Discussion on Parameters F and CR

The analysis of parameter tuning reveals that the various selection policy configurations can indeed drastically change the DE algorithm behaviour under different problem constraints. The tuning of F and CR parameters affects the speed and quality of convergence as we can see from outliers and longer histogram box overall.

The algorithms' ability to escape local optima and explore the solution space, advertently has to do with crossover jumps. Once CR is significantly decreased, as we can see from the performance of the first policy, Select Best is absolutely doomed: it struggles to leave local optimums, hence the worst case result everywhere. Even with a high mutation constant F, which might help find a suitable candidate for a further solution (not sure if applies for this particular case, but it seems legit), wouldn't help in this case.

These findings illustrate the complexity of algorithmic behavior in meta-heuristic optimization and the importance of parameter selection in achieving desired optimization outcomes for particular selection policies and corresponding benchmarks - problems.

On the other hand, as I noted in 5.1.2, algorithms also tend to perform equally well once conditions are met. A sub-trivial inference, since benchmarks of similar performance on Figure 6 are Best and Best-to-Random. Subsequently, a second policy of random selection might perform significantly better with a high F due to selection mechanics - and conversely looses in overall convergence speed due to obvious reasons.

[this report can be ended here]

6 Statistical Difference Assessment

Due to the data having lack of normal distribution means as discussed earlier, it is necessary to proceed with a non-parametric test.

6.1 Approach to Handling Outliers

Outliers in optimization experiments often provide valuable insights into the behavior of algorithms under extreme conditions. Here, outliers are particularly evident due to the slow convergence rate. The regular appearance of these outliers across different runs and parameter settings suggests a pattern tied to the inherent stochastic nature of the DE algorithms, rather than random anomalies or data errors.

6.1.1 Rationale for Preserving Outliers

The decision to retain outliers in the analysis was based on several key observations:

- **Consistency Across Parameter Settings:** Outliers consistently appear under similar conditions, hinting at their dependency on the F, CR and individual problems rather than being mere statistical flukes.
- **Information Content:** Outliers here represent extreme cases where the algorithm either performed exceptionally well or poorly, providing a fuller understanding in its capabilities and limitations.
- **Robustness of Non-Parametric Tests:** Given the non-parametric nature of the tests employed in this study, including the Kruskal-Wallis and Dunn's post-hoc tests, the analysis remains robust against the influence of extreme values. These tests do not assume a normal distribution making rank-evaluation ideal for this context.

6.1.2 Implications of Outliers

Retaining these values allows for a comprehensive evaluation of the DE algorithms, ensuring that the analysis captures the full range of algorithmic behaviors across all benchmarks. This approach is particularly justified in a controlled computational experiment where data generation and collection are well-defined and virtually free from external noise and errors. Figures above (5, 6) illustrate these points vividly, showcasing the breadth of data and highlighting the potential for extreme outcomes inherent in evolutionary computation. Even though the extreme values do not map for these 2 cases, they have a pattern tendency, which was confirmed by many test data collections. Verification can be done following the reference to archve [2].

6.2 Sample Size Calculation

6.2.1 Statistical Power and Sample Size Estimation

The power of a statistical test is the probability that it will reject a false null hypothesis. In this experiment, where the null hypothesis posits no difference in performance among the three selection policies, achieving substantially enough high power is essential to confidently assert differences or even distinguish them.

6.2.2 Methodology

The Kruskal-Wallis test, a non-parametric method for testing equality across multiple groups, was chosen due to its suitability for the non-normal distribution observed in the DE performance data. To estimate the required sample size to achieve a pre-specified power level, a simulation approach was used, iteratively increasing the sample size until the desired power was reached.

Simulation Parameters

- **Number of Groups:** 3, corresponding to the three selection policies.
- **Effect Size:** 0.05, this hypothetical value represents the minimal meaningful difference we aim to detect.
- **Significance Level (α):** 0.05, the probability of rejecting the null hypothesis when it is true (type I error).
- **Desired Power:** 80%, the probability of correctly rejecting the null hypothesis when it is false (type II error complement).
- **Iterations:** 1000, to ensure the stability of the power estimation.

Note: Output of such an estimation was 1940. Full implementation can be reviewed in Appendix A.

6.3 Kruskal-Wallis test

- **Null Hypothesis** (H_0): There is no statistical difference in the data.
- **Alternative Hypothesis** (H_1): There is some statistical difference.

Results after execution for data collection "20240716062920-8-194-0.7-0.5":

- Kruskal-Wallis H-test test-statistic: 55.85858748842646.
- **P-value:**: 7.420989949905072e-13
- (H_0) rejected - There is a statistically significant difference between the groups.

6.4 Visualization of Overall Performance

Here is showcased a plot for average performance of three Search Policies.

As we can see, this result is not particulary easy to understand without a more detailed analysis. It is important to note, however, that there are visible some patterns in extreme values locations. Since there is a statistical difference indicated by Kruskal-Wallis test, we should proceed with a more deterministic analysis.

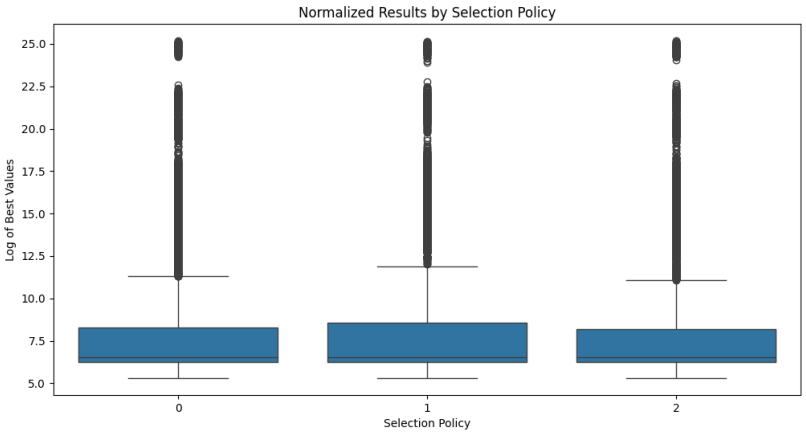

Figure 7: Comparison of DE algorithm performances, log scaled. Difficult to distinguish.

6.5 Post Hoc Dunn Test Analysis

Here is examined the statistical significance between selection policies across different parameter settings using the Dunn's post-hoc test following a Kruskal-Wallis test. The results help identify whether any significant differences exist between the three selection policies under varying conditions of Crossover Probability (CR) and Differential Weight (F).

6.5.1 Analysis Overview

The Dunn's post-hoc test is conducted to compare each pair of selection policies under specific experimental settings. The results are presented in heatmaps, which visually depict the p-values obtained, aiding in the interpretation of the statistical differences between policies.

6.5.2 Heatmap Interpretation

The heatmaps below show the p-values from Dunn's post-hoc test across selection policies for two distinct sets of parameters. These visualizations help ascertain the presence or absence of statistically significant differences between the policies.

In-Between Problem Comparison: The analysis includes a detailed comparison across different benchmarks, which due to the complexity and volume of data, is not included. Detailed results are stored in the archive [2]. Sample picture can be viewed in Appendix C. Overall, Dunn's post-hoc test consistently identifies significant statistical differences among the policies, confirming the effectiveness of the parameter adjustments.

7 Independence Verification via Design Assessment

Independence of observations is crucial for the validity of many statistical tests.

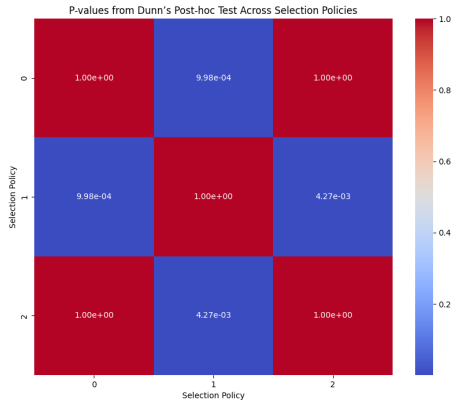

Figure 8: Post Hoc Dunn Test Heatmap for 20240716042411-8-194-1-0.2. With a significantly low CR of 0.2, the test fails to distinguish a statistical difference between selection policies 1 and 3, indicating similar performances.

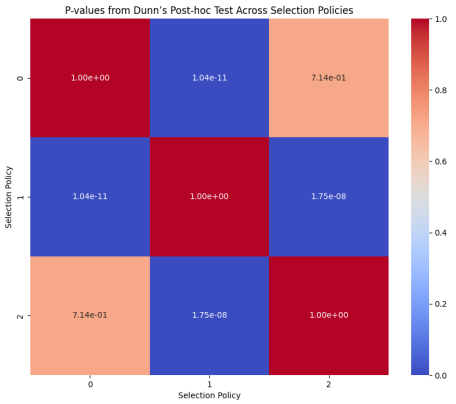

Figure 9: Post Hoc Dunn Test Heatmap for 20240716062920-8-194-0.7-0.5. This scenario with a CR of 0.5 shows some statistical differences detected by the test, illustrating the subtle impacts of parameter tuning on algorithm performance.

Figure 10: Comparative heatmaps demonstrating the effects of parameter tuning on the statistical differences between selection policies as identified by Dunn's post-hoc test.

7.1 Verifying Independence Across Algorithms and Benchmarks

- **Across Algorithms:** Each algorithm operates under its own initial conditions and is expected to use a unique seed for random processes. This setup ensures that the runs of different algorithms are independent, as **they do not share nor influence each other's execution paths or outcomes**.
- **Across Benchmarks:** The benchmarks are designed to test various optimization scenarios, ideally capturing diverse problem landscapes. If each benchmark function targets different aspects of optimization challenges, their results can be considered as independent.

7.2 Statistical Assessment of Independence Using Chi-Square

To further assess independence formally, a **Chi-square test was performed** on the distribution of results categorized by Selection Policy and Problem. The extremely low Chi-square test p-value of 3.47×10^{-9} suggests rejection of the null hypothesis (H_0) of independence. This result might initially seem to indicate potential dependencies within the data.

However, this outcome primarily reflects the convergent nature of the algorithms rather than a flaw in experimental design. Each algorithm's performance progressively improves or stabilizes, which could be misinterpreted as dependency in statistical tests. Importantly, the setup ensures that each algorithm and benchmark operates independently by design, with separate independently defined parameters, ensuring that the results from one do not influence another.

8 Additional Tasks

8.1 Tweaking of F and CR for Optimal Performance

To identify the optimal settings for the F and CR that minimize the optimum of each selection policy in the DE algorithm, the following table summarizes the best performances achieved under different configurations:

Selection Policy	Best Score	F	CR
0	183.3173193083033	0.7	0.9
1	200.00000000109992	0.3	0.5
2	200.0000000112696	0.5	0.5

Table 3: Optimal configurations for each selection policy based on the minimum best score achieved.

Statistical Analysis of Performance Differences

The analysis involved 35 different pairs of F and CR while featuring 40000 examples in each file for all results. Hence, more than 1.400.000 individual examples were gained to get the aforementioned result. The gained pairs are distinct, and inticate the settings which lead Selection Policies to the least optimum anywhere considered. It seems that there is indeed a limit of what algorithms can achieve since there was no considerable improvement after reaching 200+.

Following pairs were used:

CR \ F		0.05	0.1	0.3	0.5	0.7	0.9	1.0
CR	**1.0**	X	X	X	X	X	X	X
	0.9	X	X	X	X	X	X	X
	0.5	X	X	X	X	X	X	X
	0.1	X	X	X	X	X	X	X
	0.0	X	X	X	X	X	X	X

Table 4: Matrix of F and CR values used to evaluate algorithm performance. Each cell marked 'X' represents a tested combination of F and CR.

8.1.1 Performance over F and CR Showcase

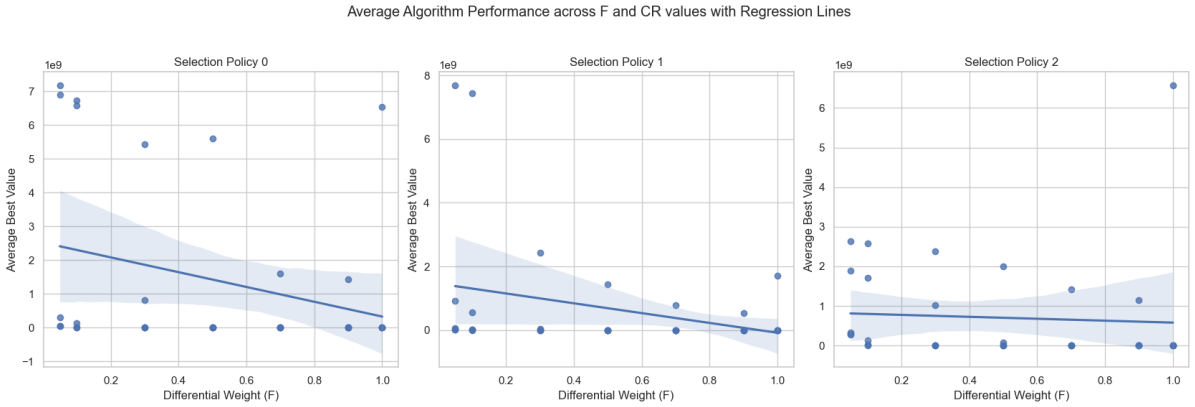

Figure 11: Performance of algorithms with different values of F. Plot utilizes 40000 examples per dot.

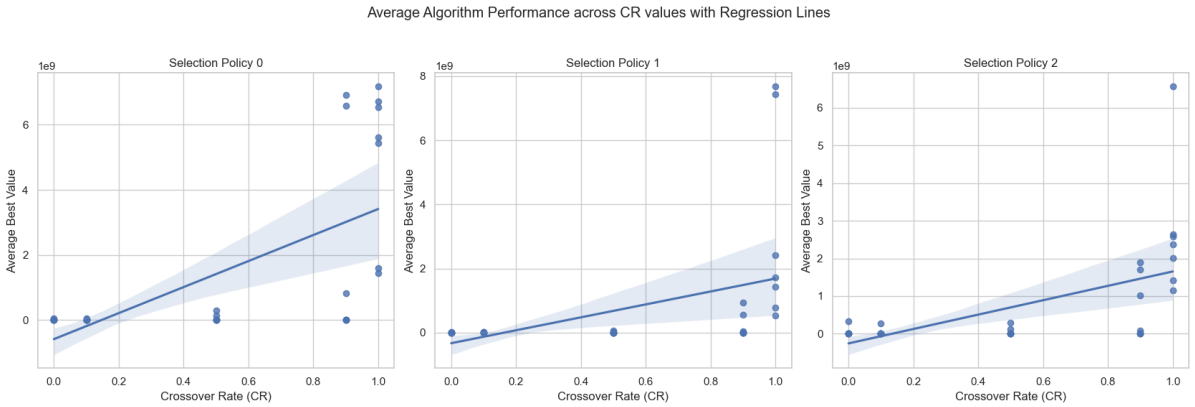

Figure 12: Performance of algorithms with different values of CR. Plot utilizes 40000 examples per dot.

As we can clearly see, performance for higher F is typically increased since the algorithms tend to find optimums better. This however is almost not as evident with the second slection policy which might do with a hybrid evaluation approach.

For the average best result with CR, a lesser CR seems to provide a statistically better value, which actually impressed me quite much. (maybe I did a mistake in calculations. I am not sure)

Cohen's d Values:

- Between Policy 0 and 1: 0.065 (small effect size, indicating a minimal practical difference)
- Between Policy 0 and 2: -0.156 (small to medium negative effect size, suggesting that Policy 2 might perform slightly worse than Policy 0)
- Between Policy 1 and 2: -0.218 (medium negative effect size, indicating a more noticeable underperformance of Policy 2 compared to Policy 1)

These statistical measures assist in understanding the statistical significance.

8.2 Runtime Analysis

This subsection provides a detailed analysis of the runtime performance of the differential evolution algorithms across various settings of the Differential Weight (F) and Crossover Probability (CR). This analysis is crucial for understanding the computational efficiency of each algorithm configuration under different parameterizations.

8.2.1 Runtime Variation with Differential Weight and Crossover Probability

The following figures illustrate how the runtime of algorithms varies with changes in the Differential Weight (F) and Crossover Probability (CR). These plots provide a visual representation of the runtime dynamics that are further quantified in the subsequent table.

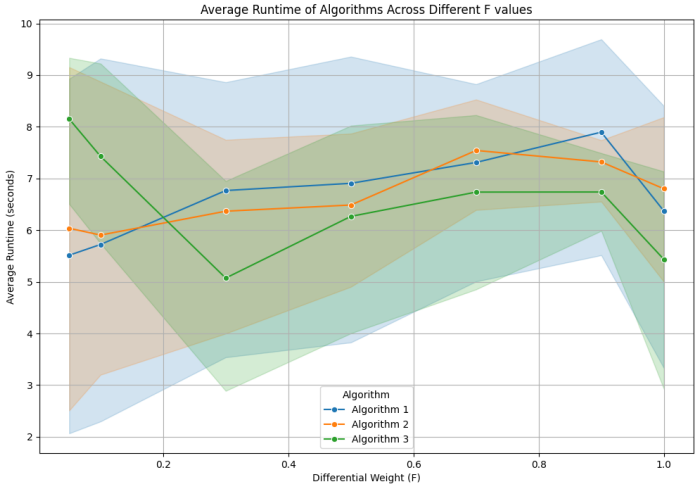

Figure 13: Variation of algorithm runtimes with different values of F.

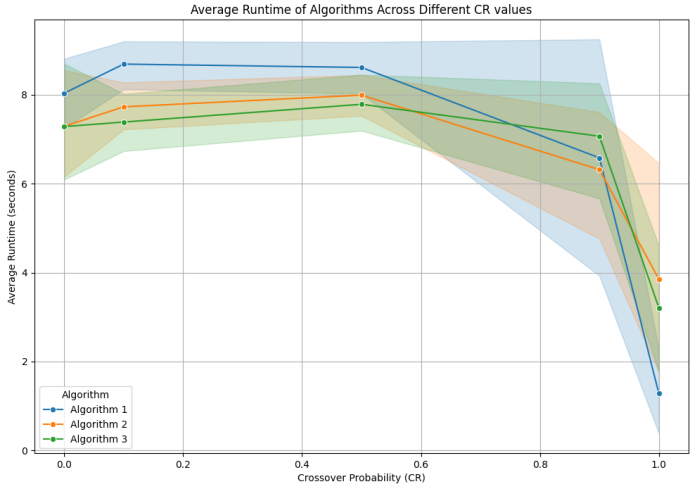

Figure 14: Variation of algorithm runtimes with different values of CR.

8.2.2 Tabulated Runtime Statistics

The table below provides a detailed breakdown of the average runtimes for each algorithm across various settings of F and CR. Each entry shows the overall runtime, along with individual runtimes for three algorithms, enabling a comparative analysis of performance efficiency.

F	CR	Overall (s)	Algorithm 1 (s)	Algorithm 2 (s)	Algorithm 3 (s)
0.05	0.0	333.57	9.0	9.83	9.56
0.05	0.1	276.14	8.5	8.47	8.93
0.05	0.5	263.9	9.15	8.82	7.82
0.05	0.9	155.08	0.79	2.55	9.31
0.05	1.0	102.23	0.13	0.51	5.13
0.1	0.0	368.17	9.73	9.75	10.34
0.1	0.1	282.12	9.66	6.7	6.05
...	...	...	...	...	...
1.0	0.0	253.21	7.88	6.91	6.75
1.0	0.1	289.0	9.41	9.02	7.74
1.0	0.5	274.96	7.43	6.96	6.69
1.0	0.9	193.59	6.68	7.4	5.54
1.0	1.0	38.45	0.45	3.71	0.45

Table 5: Average runtime statistics for different algorithms under varying settings of F and CR. More data included in average_runtimes.csv [2]

This comprehensive analysis shows that the runtime is influenced by the selection of F and CR parameters, suggesting that optimization of these parameters is essential for improving the efficiency of the algorithms.

9 Conclusions

Parameter Tuning and Algorithm Performance: The F and CR parameters significantly influence the performance of DE algorithms. Fine-tuning these parameters can lead to optimal performance, as evidenced by the varying results across different configurations.

Statistical Significance: The use of robust statistical methods, such as the Shapiro-Wilk test, Kruskal-Wallis test, and Dunn's post-hoc test, confirmed the statistical significance of differences in performance between selection policies. These differences underline the importance of careful parameter selection in optimizing DE algorithms.

Sample Size and Power Analysis: Simulations to estimate the required sample size for achieving desired statistical power underscored the importance of sufficiently large datasets. This ensures reliable detection of practical differences between configurations.

Outliers and Data Distribution: Outliers were consistently observed and retained for analysis due to their regular appearance across different parameter settings, suggesting they are inherent to the algorithm's stochastic nature rather than anomalies.

Runtime Analysis: Detailed runtime analysis revealed how the computational efficiency of DE algorithms varies with different F and CR settings. This analysis aids in understanding the trade-offs between runtime and algorithmic performance.

Optimal Configurations: The study identified optimal F and CR settings for each selection policy, providing actionable insights for practical applications of DE algorithms.

References

1. **Mamanchuk N., University of Tsukuba**, Github, July 16, 2024. Available online: `https://github.com/RIFLE`

2. **Mamanchuk N., University of Tsukuba**, Archive with Results for EDCS, Report 2, 2024. Available online: `https://drive.google.com/drive/folders/1IDyFj6jiNZ1P3mERxWX4vkladGiPFlor?usp=drive_link` [Uploaded: 2024-07-16]

Appendix A. Listing sample_size.py

```python
import numpy as np
import scipy.stats as stats

def simulate_kruskal(n, num_groups, effect_size, iterations):
    power_count = 0
    for _ in range(iterations):
        # Create data for each group with some effect
        groups = [np.random.normal(loc=i*effect_size, scale=1, size=n) for i in range(num_groups)]
        _, p_value = stats.kruskal(*groups)
        if p_value < 0.05:
            power_count += 1
    return power_count / iterations

# Set parameters
num_groups = 3  # As per three selection policies
desired_power = 0.8
alpha = 0.05
effect_size = 0.05  # Hypothetical effect size, adjust based on expected differences
iterations = 1000  # More iterations, more accurate power estimation

# Estimate required sample size
sample_size = 10  # Start with a low number
while True:
    power = simulate_kruskal(sample_size, num_groups, effect_size, iterations)
    print(f"Sample size: {sample_size}, Estimated Power: {power}")
    if power >= desired_power:
        break
    sample_size += 5

print(f"Required sample size to achieve {desired_power*100}% power is approximately {sample_size}")
```

Showcase of implementation. sample_size is 1940

Appendix B. Listing record_performance_and_runtime.py

```
import os
import pandas as pd

def analyze_performance(directory, file_name):
    file_path = os.path.join('results', directory, 'best', file_name)
    data = pd.read_csv(file_path)
    runtime_info = data.iloc[-1, 0]  # Extract runtime information
    data = data.iloc[:-1]  # Remove the last info row
    min_best = data.groupby('Selection Policy')['best'].min().reset_index()
    return min_best, runtime_info

def extract_runtime_info(runtime_str):
    parts = runtime_str.split(',')
    runtime = {
        'overall': float(parts[3].split(':')[1].strip().split(' ')[0]),
        'alg1': float(parts[4].split('=')[1].strip('s')),
        'alg2': float(parts[5].split('=')[1].strip('s')),
        'alg3': float(parts[6].split('=')[1].strip('s'))
    }
    return runtime

# List of all directories and files
configurations = [
    {"dir": "20240716145045-8-194-0.05-1", "file": "resultbest_20240716145045.csv"},
    {"dir": "20240716143339-8-194-0.1-1", "file": "resultbest_20240716143339.csv"},
    {"dir": "20240716143827-8-194-0.3-1", "file": "resultbest_20240716143827.csv"},
    {"dir": "20240716144045-8-194-0.5-1", "file": "resultbest_20240716144045.csv"},
    {"dir": "20240716144207-8-194-0.7-1", "file": "resultbest_20240716144207.csv"},
    {"dir": "20240716144534-8-194-0.9-1", "file": "resultbest_20240716144534.csv"},
    {"dir": "20240716144847-8-194-1-1", "file": "resultbest_20240716144847.csv"},
    {"dir": "20240716145310-8-194-0.05-0.9", "file": "resultbest_20240716145310.csv"},
    {"dir": "20240716145612-8-194-0.1-0.9", "file": "resultbest_20240716145612.csv"},
    {"dir": "20240716145906-8-194-0.3-0.9", "file": "resultbest_20240716145906.csv"},
    {"dir": "20240716150244-8-194-0.5-0.9", "file": "resultbest_20240716150244.csv"},
    {"dir": "20240716150658-8-194-0.7-0.9", "file": "resultbest_20240716150658.csv"},
    {"dir": "20240716151149-8-194-0.9-0.9", "file": "resultbest_20240716151149.csv"},
    {"dir": "20240716151700-8-194-1-0.9", "file": "resultbest_20240716151700.csv"},
    {"dir": "20240716155602-8-194-0.05-0.5", "file": "resultbest_20240716155602.csv"},
    {"dir": "20240716155052-8-194-0.1-0.5", "file": "resultbest_20240716155052.csv"},
```

```
    {"dir": "20240716154521-8-194-0.3-0.5", "file": "resultbest_20240716154521.csv"},
    {"dir": "20240716154006-8-194-0.5-0.5", "file": "resultbest_20240716154006.csv"},
    {"dir": "20240716153246-8-194-0.7-0.5", "file": "resultbest_20240716153246.csv"},
    {"dir": "20240716152736-8-194-0.9-0.5", "file": "resultbest_20240716152736.csv"},
    {"dir": "20240716152245-8-194-1-0.5", "file": "resultbest_20240716152245.csv"},
    {"dir": "20240716160103-8-194-1-0.1", "file": "resultbest_20240716160103.csv"},
    {"dir": "20240716160604-8-194-0.9-0.1", "file": "resultbest_20240716160604.csv"},
    {"dir": "20240716161151-8-194-0.7-0.1", "file": "resultbest_20240716161151.csv"},
    {"dir": "20240716161808-8-194-0.5-0.1", "file": "resultbest_20240716161808.csv"},
    {"dir": "20240716162356-8-194-0.3-0.1", "file": "resultbest_20240716162356.csv"},
    {"dir": "20240716162912-8-194-0.1-0.1", "file": "resultbest_20240716162912.csv"},
    {"dir": "20240716163616-8-194-0.05-0.1", "file": "resultbest_20240716163616.csv"},
    {"dir": "20240716164204-8-194-1-0", "file": "resultbest_20240716164204.csv"},
    {"dir": "20240716164826-8-194-0.9-0", "file": "resultbest_20240716164826.csv"},
    {"dir": "20240716165344-8-194-0.7-0", "file": "resultbest_20240716165344.csv"},
    {"dir": "20240716165822-8-194-0.5-0", "file": "resultbest_20240716165822.csv"},
    {"dir": "20240716170310-8-194-0.3-0", "file": "resultbest_20240716170310.csv"},
    {"dir": "20240716170844-8-194-0.1-0", "file": "resultbest_20240716170844.csv"},
    {"dir": "20240716171551-8-194-0.05-0", "file": "resultbest_20240716171551.csv"},
]

best_performances = {}
runtime_data = []

for config in configurations:
    best_data, runtime_info = analyze_performance(config['dir'], config['file'])
    runtime = extract_runtime_info(runtime_info)
    runtime['F'] = config['dir'].split('-')[3]  # Extract F value
    runtime['CR'] = config['dir'].split('-')[4]  # Extract CR value
    runtime_data.append(runtime)

    for index, row in best_data.iterrows():
        policy = int(row['Selection Policy'])
        best_score = row['best']
        if policy not in best_performances or best_score < best_performances[policy]['best_score']:
            best_performances[policy] = {
                'best_score': best_score,
```

```
                'F': config['dir'].split('-')[3],
                'CR': config['dir'].split('-')[4]
            }

# Create dataframes from the performance and runtime data
performance_df = pd.DataFrame.from_dict(best_performances, orient='
    index')
runtime_df = pd.DataFrame(runtime_data)

# Save the performance data to a CSV file
performance_df.to_csv('performance_stats.csv', index_label='Selection
    Policy')
print("Performance statistics saved to 'performance_stats.csv'")

# Save the runtime data to another CSV file
runtime_df.to_csv('runtime_stats.csv', index=False)
print("Runtime statistics saved to 'runtime_stats.csv'")
```

Showcase of implementation. Files are recorded using all_stats.py.

Appendix C. Interbenchmark Host Hoc Dunn Performance

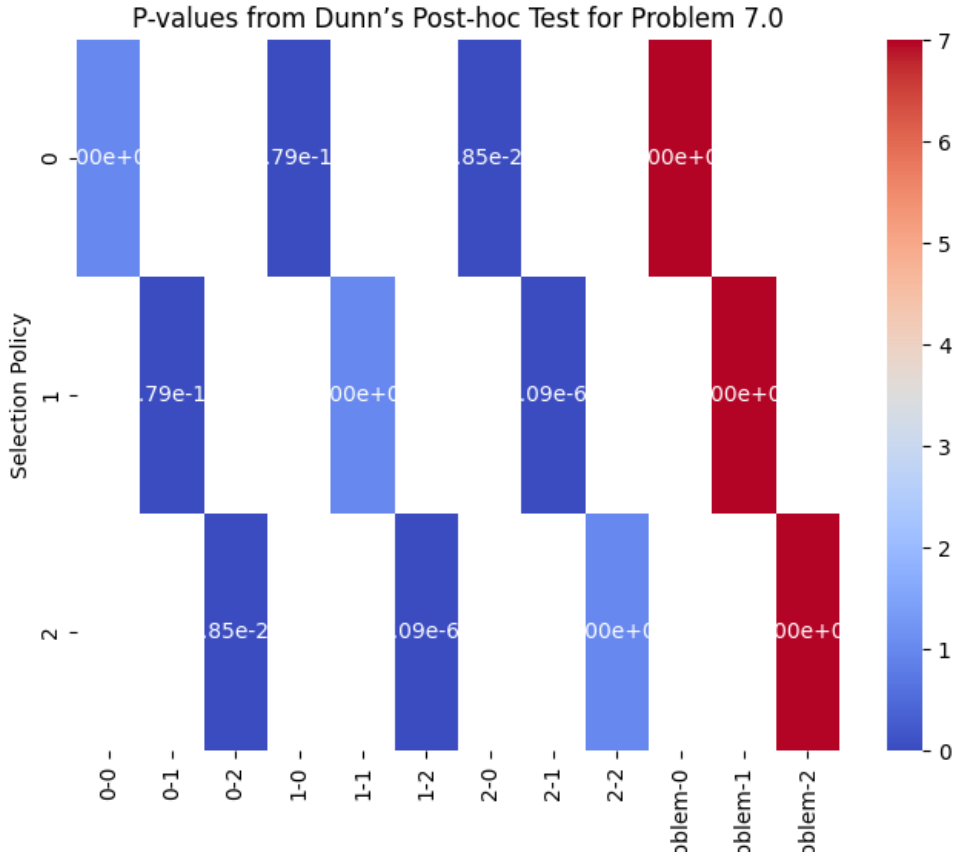

Figure 15: Results showcase for PH analysis for 20240716062920-8-194-0.7-0.5_P7.0_hosthoc_heatmap.png