Optimizing Synthetic Data Using Differential Evolution Algorithm

Project Seminar Report

March, 2024

Submitted By:

Mahek Pankhaniya(221100672)

Kenil Patel(222202241)

Kishan Sheladiya(222202236)

Meet Pala(222202052)

Guided By:

Dr. Radomir Pestow

Department of Mathematics,

Universität Koblenz

university of koblenz

Mathematics / Natural Sciences

Declaration

We hereby declare that our project report entitled "Optimizing Synthetic Data Using the Differential Evolution Algorithm" is our original work submitted in partial fulfillment of the "Mathematical Modeling, Simulation, and Optimization". All sources of information, including but not limited to literature, data, and ideas, have been appropriately cited and referenced in accordance with academic standards and guidelines. This report represents the findings and conclusions based on our research and analysis in the field of optimizing synthetic data using the differential evolution algorithm.

Mahek Pankhaniya (221100672)

Kenil Patel (222202241)

Kishan Sheladiya (222202236)

Meet Pala (222202052)

Date: 6 March, 2024

Acknowledgment

We extend our heartfelt gratitude to our guide Dr. Radomir Pestow, for his invaluable mentorship, support, and expert guidance throughout this project. His extensive expertise and constructive feedback played a pivotal role in shaping the trajectory of our research and fine-tuning our methodological approaches. Lastly, we appreciate the Department of Mathematics, Universität Koblenz for providing us the opportunity to be a part of such research project.

Abstract

This report documents the optimization of synthetic data through the utilization of the Differential Evolution (DE) algorithm, with the objective of mirroring the demographic structure and characteristics of real-world data and analyzing the impact of various parameters on the DE algorithm. A systematic methodology was employed throughout the research, includes synthetic data generation, optimization using the DE algorithm, fitness calculation and comparison with empirical data. Emphasis was placed on the visualization of results, offering insightful comparisons between empirical and synthetic data sets across various household configurations. It is revealed in the study that synthetic data closely resembling actual household size distributions is efficiently generated by the DE algorithm. The effort culminates with the establishment of a set of optimized parameters that significantly improve the fitness of synthetic data while ensuring computational efficiency.

Keywords: Differential evolution algorithm, Synthetic data generation, Optimization

Contents

List of Figures

List of Tables

List of Abbreviations

COVID-19	Coronavirus Disease - 2019
DEA	Differential Evolution Algorithm
DE	Differential Evolution

Chapter 1

Introduction

The generation of synthetic data has become increasingly important in today's world, as it serves as a vital tool for modeling and understanding complex phenomena such as epidemics. Synthetic data allows researchers and policymakers to simulate real-world scenarios like the spread of COVID-19, providing valuable insights without relying on incomplete real-world data.

While the generation of synthetic data is crucial, at the same time it must reflect the structure and characteristics of empirical data. It requires optimization. The Differential Evolution (DE) algorithm, a stochastic approach introduced by Storn and Price (1997) [1], excels in the optimization of global minimization problems, showcasing remarkable adaptability and efficiency as evidenced by subsequent advancements. [2]

This report presents a thorough study of the generation of synthetic data using the Differential Evolution (DE) algorithm, aiming to closely replicate the structures and characteristics observed in real-world data. At the core of our methodology was the generation of synthetic datasets, optimization using DE, the calculation of fitness, fine-tuning of parameters and analysis. Our process included iterative adjustments and evaluations, using both quantitative measures and visualizations to compare synthetic data against actual empirical datasets.

Lastly, a refined set of parameters was obtained that greatly enhanced the quality of the generated synthetic data. The results demonstrate the Differential Evolution (DE) algorithm as an efficient tool for synthetic data generation.

Chapter 2

Literature Review

The generation of synthetic data that mimics real datasets has gained attention due to its importance in various areas. Liu et al. (2023) introduce a privacy-preserving genetic algorithm that generates synthetic data by evolving populations of datasets through crossover and mutation operations. This approach is particularly beneficial in contexts where traditional differential privacy mechanisms fall short, offering a new way to balance privacy and utility in synthetic datasets [3].

The Differential Evolution (DE) algorithm, introduced by Storn and Price in 1997, has been extensively recognized for its simplicity and effectiveness in solving global optimization problems [1]. This evolutionary algorithm manipulates a population of candidate solutions towards an optimum by applying mutation, crossover, and selection processes. The mutation process involves creating a mutant solution by adding randomness to the original candidate solution, creating diversity. Subsequently, the crossover process combines elements of the existing candidate solution, governed by a crossover rate, to form a crossover solution. Finally, selection of the best solutions is done based on fitness. These steps collectively facilitate balanced exploration and exploitation of the search space, driving the algorithm towards convergence on optimal or near-optimal solutions.

DEA has significantly evolved through continuous advancements to improve its optimization capabilities in complex scenarios. Beginning with Wong and Dong's (2005) exploration of its adaptability [4], followed by the introduction of adaptive parameters by Qin, Huang, and Suganthan (2009) in the Self-Adaptive Differential Evolution (SADE) algorithm [5], and further enhancements for constrained and global optimization challenges by Gao and Liu (2011) [6] and Mohamed, Sabry, and Khorshid (2012) [2], DE has demonstrated remarkable flexibility and effectiveness.

Recent developments, such as Deng et al.'s (2021) novel mutation strategies [7], Samal et al.'s (2022) Modified Differential Evolution (MDE) algorithm [8], and the introduction of fitness-based crossover rates by Cheng et al. (2023) [9], have pushed DE's performance further, showcasing its ongoing refinement and adaptability in addressing complex optimization problems.

Our project leverages differential evolution algorithm's ability to solve complex optimization problems efficiently. By using DE's strengths, we aim to generate synthetic data that closely matches empirical data patterns and distributions.

Objectives

- To generate synthetic data that accurately reflects the characteristics and structures of empirical data using the Differential Evolution algorithm.
- To analyze the impact of various parameters on computational time and the fitness.

Chapter 3

Methodology

This chapter outlines the computational approach adopted to generate synthetic household data with the aim of closely replicating the structure and characteristics of real data. The methodology leverages strategies to generate synthetic data and implement DEA optimization technique to ensure the synthetic data's accuracy and utility. In addition, box plots were created to gain insights into the behavior of various parameters. Separate box plots were created by keeping one parameter variable within a specified range while keeping others at their original levels. The entire process was executed using Python programming.

3.1 Real Data

The empirical foundation of our synthetic data generation approach is a dataset sourced from the official statistics portal of Rheinland-Pfalz (Statistisches Landesamt Rheinland-Pfalz)[10], which provides a detailed census of households and family structures across the state. This dataset is publicly available and can be accessed at Statistisches Landesamt Rheinland-Pfalz, ensuring transparency and reproducibility in our research.

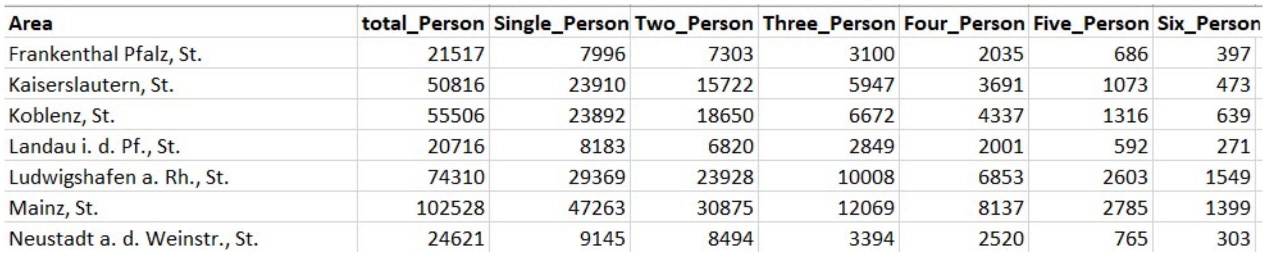
The dataset contains the household configurations across various areas, classified by the number of residents per household. The excerpt provided in Figure 3.1 reflects a structured tabulation of areas along with corresponding counts for total persons and households categorized by size — from single-person to six-person units as shown.

Key metrics within the dataset include:

- Area: Name of the areas across Germany
- Total_Person: The total number of individuals across all households in the area.
- Single_Person: The number of households with a single occupant.
- Two_Person: The number of households with two occupants.
- Three_Person: The number of households with three occupants.

- Four_Person: The number of households with three occupants.
- Five_Person: The number of households with five occupants.
- Six_Person: The number of households with six and more occupants.

This real dataset serves dual purpose in this study. First, it provides a realistic baseline from which synthetic household data can be generated. Second, it acts as a comparative benchmark for the optimization and validation of the synthetic data, ensuring that the final synthetic output accurately reflects the household size distribution.



Area	total_Person	Single_Person	Two_Person	Three_Person	Four_Person	Five_Person	Six_Person
Frankenthal Pfalz, St.	21517	7996	7303	3100	2035	686	397
Kaiserslautern, St.	50816	23910	15722	5947	3691	1073	473
Koblenz, St.	55506	23892	18650	6672	4337	1316	639
Landau i. d. Pf., St.	20716	8183	6820	2849	2001	592	271
Ludwigshafen a. Rh., St.	74310	29369	23928	10008	6853	2603	1549
Mainz, St.	102528	47263	30875	12069	8137	2785	1399
Neustadt a. d. Weinstr., St.	24621	9145	8494	3394	2520	765	303

Figure 3.1: Distribution of the number of people per household in the areas of Rhineland-Pfalz, Germany. Source: Statistisches Landesamt Rheinland-Pfalz

In this project, real household data of "Mainz St." was taken as the benchmark for the comparison of synthetic data. The target proportions were calculated by dividing each data point by the total number of households and stored as *'target_proportion'*. Later it was used for determining the fitness of the synthetic data.

3.2 Synthetic Data Generation

The process of generating synthetic data begins with defining the parameters, includes the number of persons and the number of settings. The Python function *'generate_synthetic_data'* was designed to produce synthetic data for a specified number of persons and settings. The function utilizes the "random.choice" method from the Python "random" module to randomly assign a household number to each person from a range of 1 to the total number of settings. The resulting synthetic data is a list, where each element represents the house number assigned to a corresponding person. The pseudo-code is as follows:

Algorithm 1 Generate Synthetic Data

```
1: Input: $num\_persons$ (integer), $num\_settings$ (integer)
2: function GENERATE_SYNTHETIC_DATA($num\_persons$, $num\_settings$)
3:     Initialize an empty list $'synthetic\_data'$
4:     for $i$ in range(1 to $'num\_persons'$) do
5:         Generate a random number in the range from 1 to $'num\_settings' + 1$
6:         Append the random number to $'synthetic\_data'$
7:     end for
8:     return $'synthetic\_data'$
9: end function
```

The generated synthetic data is subsequently used as the input for the Differential Evolution (DE) algorithm, which optimizes the distribution to closely match that of the real data. The optimization process is discussed in Section 3.4 in detail.

3.3 Fitness Calculation

The optimization process is guided by a fitness function, "calculate_fitness". It assesses the deviation of the synthetic data from target proportions derived from real data. This function computes the distribution of household characteristics within the synthetic dataset and evaluates its fitness by comparing it to the desired target proportions.

The fitness calculation follows a series of steps: The initial step involves counting the number of persons in each household, resulting in an array length corresponding to the number of settings. Subsequently, a new array is created by categorizing households based on the number of persons living. The histogram method is used for computational efficiency in the counting process. The next step calculates the sum of this array and normalizes each data point by dividing it by the total sum. In the final step, fitness calculation applies the infinity norm method, involving subtraction, absolute value, and maximum operations between the proportions of synthetic and real data points. To handle potential negative values post-subtraction, mode is applied. The highest value from the resulting array is considered as fitness.

Algorithm 2 Fitness Calculation

```
1: Input: solution (a list), target_proportions (a list)
2: function CALCULATA_FITNESS(solution, target_proportions)
3:     Initialize an empty list 'counts'
4:     for i in range(1 to 'num_settings' + 1) do
5:         Append the count of i in solution to 'counts'
6:     end for
7:     Calculate 'distribution' by creating a histogram of 'counts' with bins ranging from 1 to 7
8:     Append to 'distribution' the sum of counts that are greater or equal to 6
9:     Calculate 'z' as the sum of 'distribution'
10:    Initialize an empty list 'proportions'
11:    for each value 'val' in 'distribution' do
12:        Append 'val' divided by 'z' to 'proportions'
13:    end for
14:    return the maximum absolute difference between 'proportions' and 'target_proportions'
15: end function
```

3.4 Optimization using Differential Evolution Algorithm

The optimization of synthetic household data utilizes the Differential Evolution (DE) algorithm, a robust method known for its effectiveness in solving complex optimization problems[11]. This section outlines the comprehensive approach taken, from the initialization of parameters through to the evaluation of the DE algorithm's performance. DE algorithm consists mainly of four processes: Initialization, Mutation, Crossover, and Selection, which are discussed in later sections. The overall algorithm is shown in Algorithm 3.

Algorithm 3 DE Algorithm Function

```
1: Input: $num\_candidate\_solutions$, $Z$, $num\_mutation\_solutions$, $num\_place\_mutate$, $iterations$
2: function DEA FUNCTION($num\_candidate\_solutions$, $Z$, $num\_mutation\_solutions$, $num\_place\_mutate$, $iterations$)
3:     Set $cs$ equal to $num\_candidate\_solutions$
4:     for $i$ in range($iterations$) do
5:         Initialize an empty list $'new\_candidates'$
6:         for $solution$ in range(length of $cs$) do
7:             Append the solution from $cs$ to $'new\_candidates'$
8:         end for
9:         Generate a list of random indices $'random\_candidate\_solution\_crossover'$ from the range of the length of $cs$ with $2 \cdot Z$ elements
10:        for $index$ in range(0 to the length of $'random\_candidate\_solution\_crossover' - 1$, stepping by 2) do
11:            Set $parent1$ and $parent2$ to the solutions at the current and next index in $'random\_candidate\_solution\_crossover'$ from $cs$
12:            Generate two children by crossing over $parent1$ and $parent2$
13:            Append the children to $'new\_candidates'$
14:        end for
15:        Generate a list of random indices $'random\_candidate\_solution\_mutation'$ from the range of the length of $cs$ with $'num\_mutation\_solutions'$ elements
16:        for $index$ in $'random\_candidate\_solution\_mutation'$ do
17:            Append the mutated solution at the current index to $'new\_candidates'$
18:        end for
19:        Calculate the fitness scores for all solutions in $'new\_candidates'$
20:        Get the indices of the solutions with the smallest fitness scores, up to the $'num\_candidate\_solutions'$ and store in $'min\_fitness\_indices'$
21:        Set $cs$ to the solutions at the indices in $'min\_fitness\_indices'$ from $'new\_candidates'$
22:    end for
23:    return the low fitness solution from $cs$
24: end function
```

3.4.1 Initialization

The DE algorithm begins with the initialization of key parameters that define the boundaries and objective of the optimization process. Key parameters are:

Parameter	Interpretation	Initial Value
num_persons	Total number of Persons	1000
num_settings	Total number of Households	550
iterations	Number of Iterations	30
num_candidate_solutions	Total number of Candidate Solutions	30
num_mutation_solutions	Total number of Mutation Solutions	15
Z	Total number of pair of Offspring Solutions	7
num_place_mutate	Total number of Place Mutate	200

Table 3.1: Parameters and their initial values

Total Number of Persons('num_persons'): It was set to 1000 & distributed across a variety of household configurations. This parameter sets the scale of our synthetic data environment.

Total number of Households ('num_settings'): In order to represent the variety of real-world living situations, 550 households were taken into account. This range allows for a comprehensive exploration of possible household compositions.

Total number of Candidate Solutions ('num_candidate_solutions'): Algorithm operates on a set of 30 initial candidate solutions. These candidate solutions are the data which were generated using $'generate_synthetic_data'$ function as discussed in Section 3.2

Total number of pair of Offspring Solutions ('Z'): In each iteration, 7 pairs of solutions are chosen to produce offspring through the DE algorithm's crossover mechanism.

Total number of Mutation Solutions ('num_mutation_solutions'): 15 solutions were selected to undergo mutation process. These mutations are important for exploring new areas of the solution space.

Number of Iterations ('iterations'): Iterations were kept at 30. That allows algorithm for better exploration.

Total number of Place Mutate ('num_place_mutate'): To add randomness in the exploration, 200 places within each solution taken for mutation.

3.4.2 Mutation

In the mutation process of a DE algorithm, A safe copy of the original solution vector is created to preserve its integrity. Then, a specified number of unique indices is randomly chosen from the range corresponding to the *'candidate_solution'*'s length. These indices indicate which elements in the *'candidate_solution'* will be altered during the mutation process. Each of the unique indices are processed as *'index'*. After that, A random integer between 1 and *'num_settings'* is generated. This random number represents a new value for the mutated gene. The mutation is applied by replacing the current value at the selected index with the newly generated random household number. In the end mutated solution is returned.

Algorithm 4 Mutation

```
1: Input: candidate_solution (a list), num_place_mutate (an integer), num_settings (an integer)
2: function MUTATION_FUNCTION(candidate_solution, num_place_mutate, num_settings)
3:     Create a deep copy 'candidate_solution_copy' of the solution 'candidate_solution'
4:     Generate an empty list 'unique_indices'
5:     for i in range(num_place_mutate) do
6:         Append a random index to 'unique_indices' from the range of indices of 'candidate_solution_copy'
7:     end for
8:     for i in range(length of 'num_place_mutate') do
9:         Retrieve the 'random_index' from 'unique_indices' at position i
10:        Generate a random integer 'random_house_number' between 1 and 'num_settings' (inclusive)
11:        Set the value at 'random_index' in 'candidate_solution_copy' to 'random_house_number'
12:    end for
13:    return the mutated solution 'candidate_solution_copy'
14: end function
```

3.4.3 Crossover

In the crossover process of the DE algorithm, the *crossover'* function facilitates the creation of two offspring solutions (*'child1'* & *'child2'*) by merging segments of two parent solutions (*'parent1'* & *'parent2'*). The process begins with the random selection of a crossover point within the length of the parent solutions. Subsequently, the first offspring is formed by combining the prefix of one parent with the suffix of the other, while the second offspring results from the complementary combination – taking the prefix from the second parent and the suffix from the first. This dynamic combination, defined by the randomly chosen crossover point, introduces genetic diversity to the population of candidate solutions. The function returns these two offspring solutions, representing potential candidates for subsequent iterations of the DE algorithm.

Algorithm 5 Crossover

```
1: function CROSSOVER(parent1, parent2)
2:     point is a random integer between 1 and the length of (parent1) -1
3:     'child1' is the combination of the prefix of parent1 from the beginning to point and the suffix of parent2 from point to the end
4:     'child2' combination of the prefix of parent2 from the beginning to point and the suffix of parent1 from point to the end
5:     return child1, child2
6: end function
```

3.4.4 Selection

In the selection process of the DE algorithm, the objective is to identify optimal fitness candidates from a pool of solutions. This was achieved by calculating the fitness of each solution within the $'new\_candidates'$ set. The fitness of each solution is determined by comparing it to a target solution using the $'calculate\_fitness'$ method, and the resulting fitness scores are stored in the $'fitness\_scores'$ list. Using NumPy's *argsort* function, the $'fitness\_scores'$ are arranged in ascending order. The algorithm then selects the first $'num\_candidate\_solution'$ indices, representing the solutions with the lowest fitness scores. The $'candidate\_solutions'$ list is updated by choosing solutions from $'new\_candidates'$ based on the indices stored in $'min\_fitness\_indices'$. Consequently, $'candidate\_solutions'$ now comprises the best-performing solutions among the original candidate solutions, mutated solutions, and offspring solutions.

3.5 Visualization

This section focuses on how the visualization is done to check the performance of optimization process and to see the behaviour of various parameters on the DE algorithm. Box plots and line graphs are employed to gain insights into computational efficiency and solution quality, using Python's *matplotlib* library.

Boxplots were utilized as they illustrate the distribution of computational times and fitness scores. They provide a visual summary of the central tendency, variability, and potential outliers in the data.

By connecting the means of the box distribution, line graphs offer a clear visualization of trends, directly correlating changes in the parameters with shifts in average computational time and fitness score.

Chapter 4

Results and Analysis

In this chapter, the results obtained by performing DE algorithm to optimize synthetic data are analyzed. To understand the behaviour of various parameters on algorithm's computational time and solutions' fitness quality, results were obtained by varying one parameter at a time, while other parameters were kept at values given in Table 3.1. The results are complemented by visualizations that provide a richer understanding of the data. Each parameter's analysis is visualized through box-plot graphs. These visual tools not only depict the raw performance but also highlight trends and anomalies.

4.1 Variable Parameter: Number of Candidate Solutions

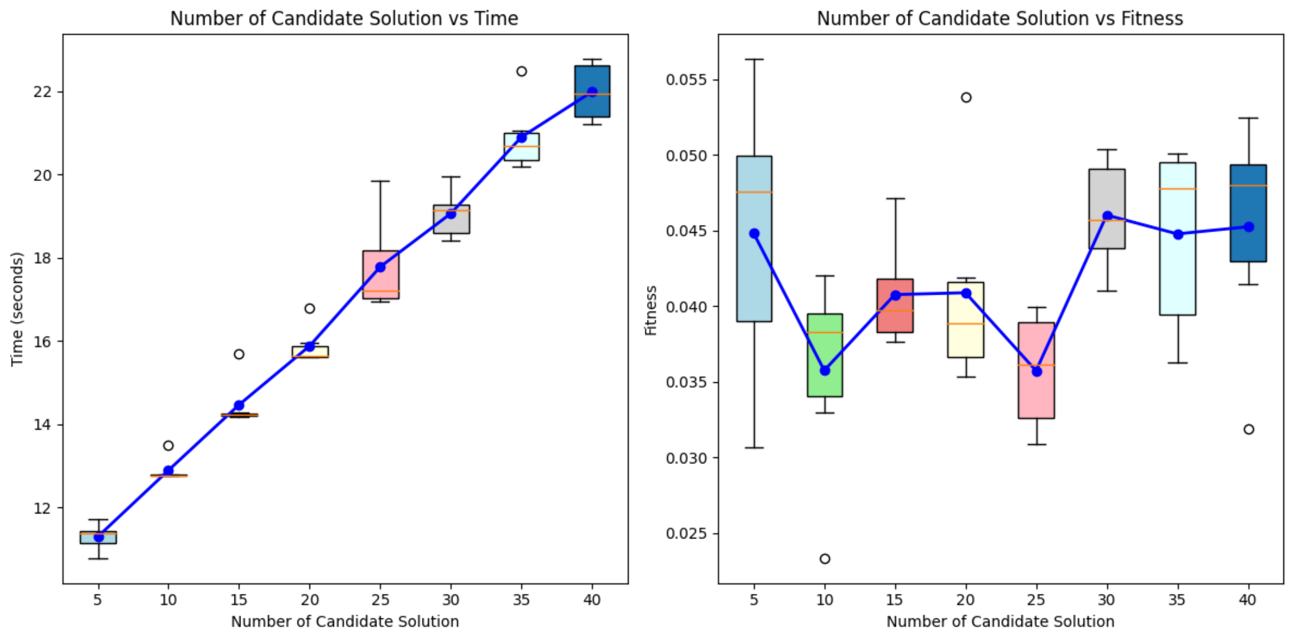


Figure 4.1: Behaviour of Number of Candidate Solutions on (a) Computational Time & (b) Fitness

It is observed from Figure 4.1 (a) that, while increasing number of candidate solutions, the computational time required to perform DE algorithm also increases proportionally. The linear trend suggests that there is a direct correlation between the size of the candidate solutions and the algorithm's time complexity. This could be because, although producing more candidate solutions could allow for a more complete exploration of the solution space, doing so also necessitates more time and computing power.

As shown in Figure 4.1 (b), the number of candidate solutions and fitness does not follow a consistent trend, while the achieved mean fitness values are varying in a particular range. Here, lack of a clear increasing or decreasing trend in fitness suggests a non-linear and complicated relationship. This implies that increasing the number of candidate solutions does not always improve the algorithm's solution quality. This complexity could be due to the algorithm's stochastic characteristic and suggests that good parameter adjustment, such as selecting an adequate number of candidate solutions, is essential.

4.2 Variable Parameter: Number of Mutation Solutions

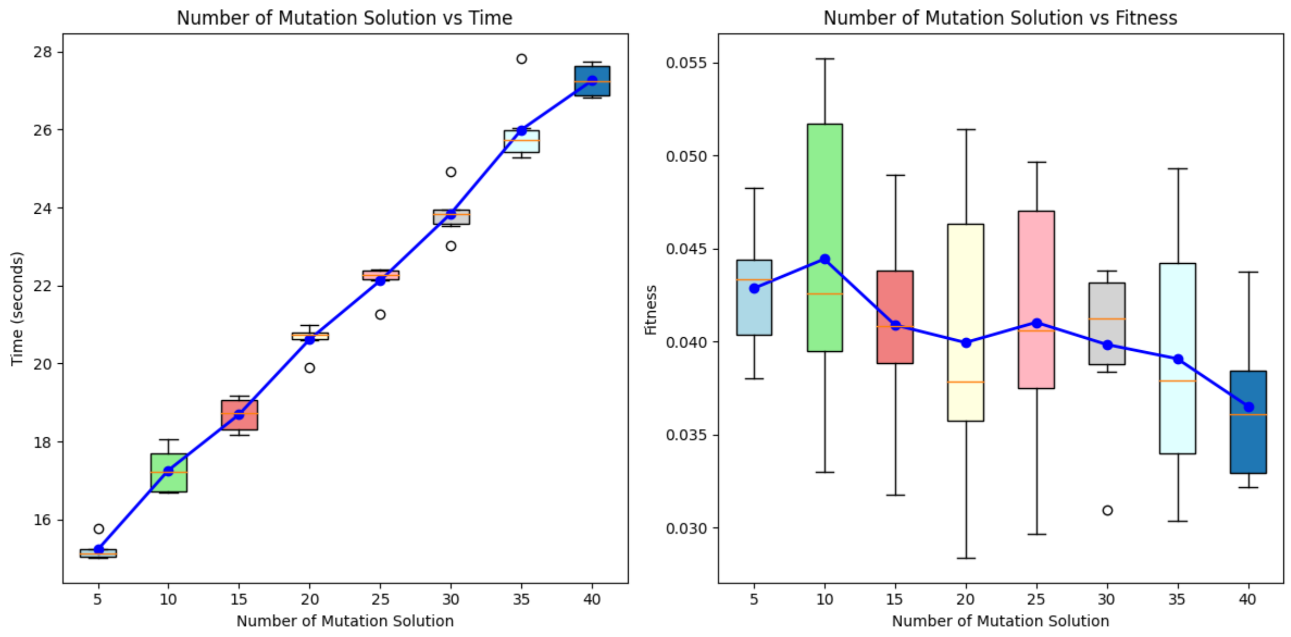


Figure 4.2: Behaviour of Number of Mutation Solutions on (a) Computational Time & (b) Fitness

As reflected in Figure 4.2 (a), the time it takes for the algorithm to finish its execution rises proportionately with the number of mutation solutions. This pattern implies that the number of mutation candidate taken for the mutation process from candidate solutions has a considerable impact on the computational cost (time and power) of the DE algorithm. Here, outliers suggest that in certain configurations, time might vary more or less than the overall trend predicts. This could be due to the random selection of candidates for mutation solutions and the varying computational load of evaluating each mutated solution.

Figure 4.2 (b) illustrates that the fitness decreases with increase in the number of mutation solutions, but dose not exhibits a linear trend. It is observed that the optimal fitness can be archived at higher number of mutation solutions. The fluctuation in fitness outcomes, as seen by the spread of box plots in Figure 4.2 (b) and the existence of outliers, emphasizes the DE algorithm's stochastic characteristic. The analysis suggest that, if the number of mutation solutions selected is greater than half of the number of candidate solutions, then the lower fitness can be archived with fewer iterations.

4.3 Variable Parameter: Number of Pair of Offspring Solutions

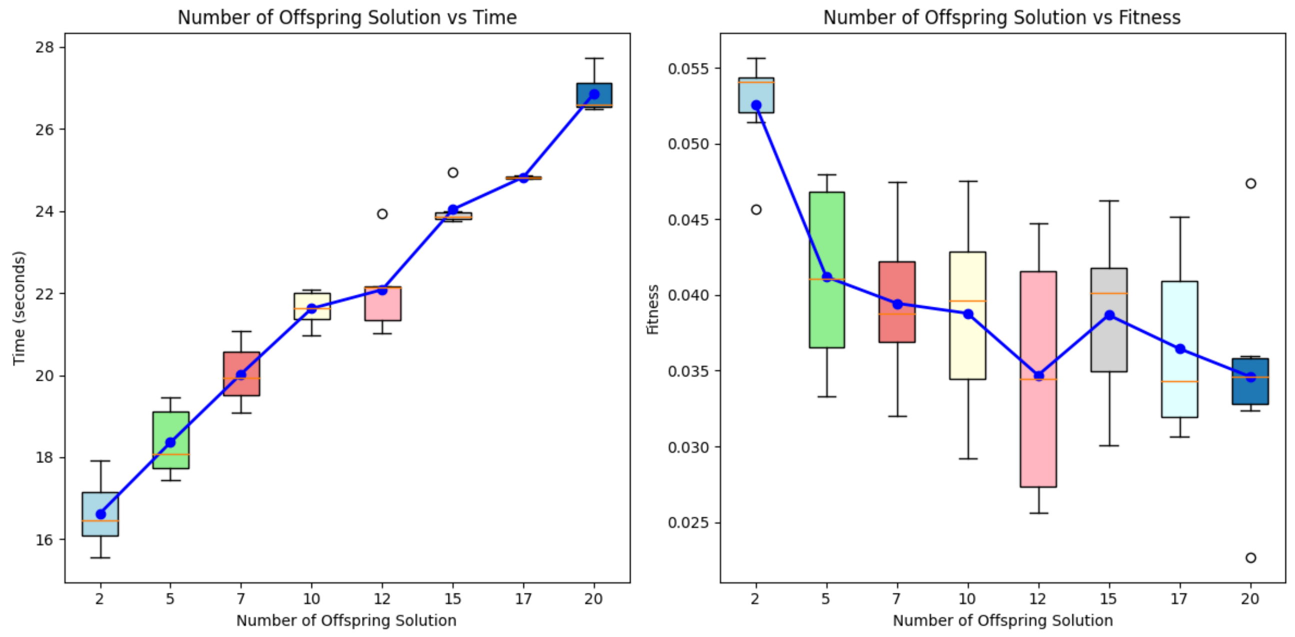


Figure 4.3: Behaviour of Number of Pair of Offspring Solutions on (a) Computational Time & (b) Fitness

As seen in the Figure 4.3 (a), there is an increasing approximate-linear trend, indicating that as the number of pair of offspring solutions rises, the time required to complete the algorithmic process also rises. This is because, more number of pair of offspring solutions requires additional computations for their evolution and so for the iteration. The presence of outliers is due to certain configurations of offspring solutions can lead to computation times that deviate from the average.

The graph shown in Figure 4.3 (b) illustrates that having more pairs of offspring solutions can reduce fitness, but there is no consistent pattern in the box and whiskers range. The different fitness outcomes emphasize the algorithm's unpredictability. Crucially, when total offspring solutions exceeds half of the candidate solutions, it is possible to achieve lower fitness in fewer iterations.

4.4 Variable Parameter: Number of Iterations

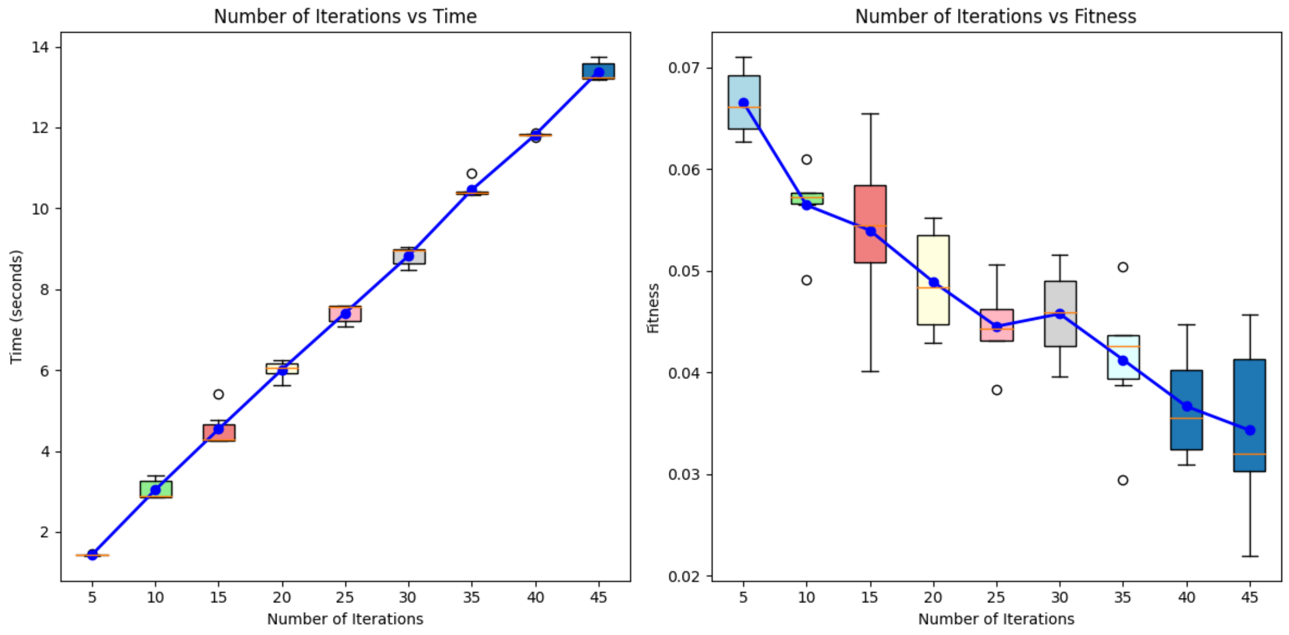
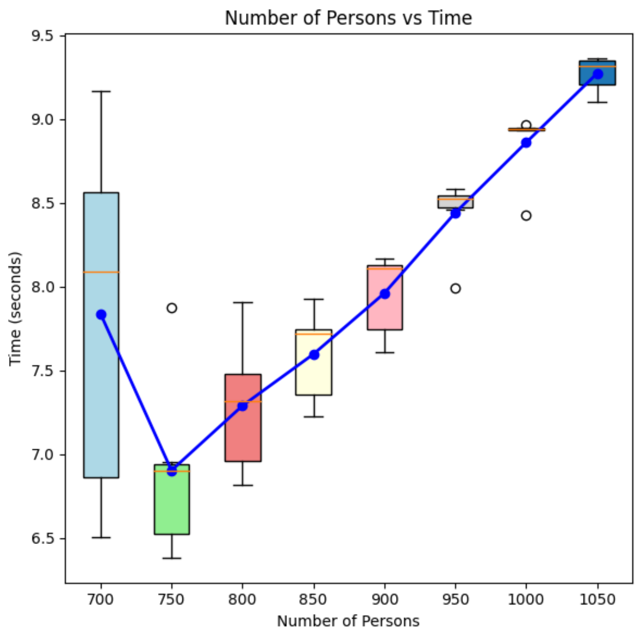


Figure 4.4: Behaviour of Number of Iterations on (a) Computational Time & (b) Fitness

The graph 4.4 (a) shows a clear trend: computing time grows with increase in the number of iterations. More iterations leads to more calculations, therefore this is to be expected. This plot indicates that while the method grows linearly with the number of iterations, there is variability in the computing time at each iteration level, as evidenced by the dispersion of the box-plots.

The graph 4.4 (b) presents that the number of iterations and the fitness value have inverse relationship, it means with more number of iterations optimal solution can archived. The median fitness value, which indicates the divergence from target proportions, often decreases as iterations progress, suggesting an improvement in solution quality. The falling trend line mean fitness values highlight the algorithm's ability to improve solutions over iterations. Outliers indicate rare situations in which the DEA might not function as well even after many iterations.

4.5 Variable Parameter: Number of Persons



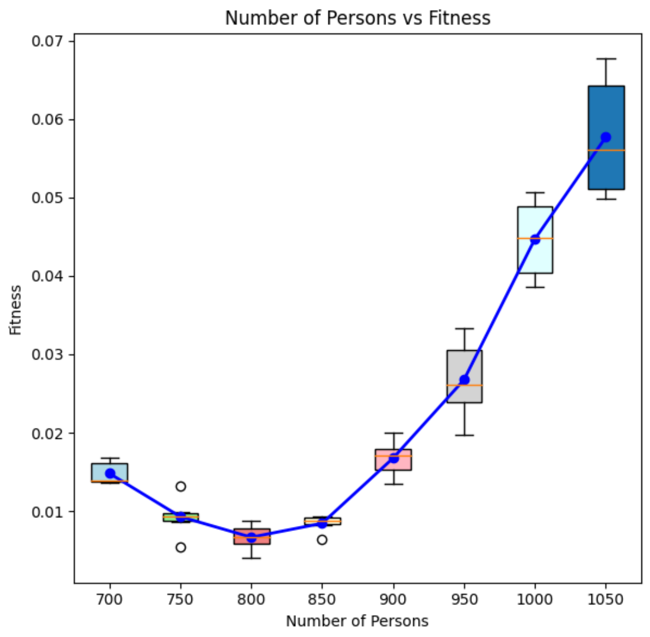


Figure 4.5: Behaviour of Number of Persons on (a) Computational Time & (b) Fitness

The graph 4.5 (a) depicts the computing time required vs the number of people. The linearly growing trend line indicates that the time complexity of the algorithm rises linearly with the number of people. The lowest point suggests that if the number of people considered for optimal solution is around 750, it will take less time to find the ideal answer. This relationship is critical for practitioners to consider when scaling the DEA for bigger populations in order to guarantee that computational resources are effectively allocated.

The number of persons vs. fitness is shown in graph 4.5 (b). There is a noticeable decrease in fitness as the number of persons hits specific thresholds, after which the improvement in fitness drops, implying a decreasing return on solution quality as the population size expands. The trend line through the mean fitness values of each group helps visualize this pattern. This graph is important for understanding how the DEA's solution quality changes with the number of persons as it leads to achieve optimal fitness.

4.6 Variable Parameter: Number of Settings

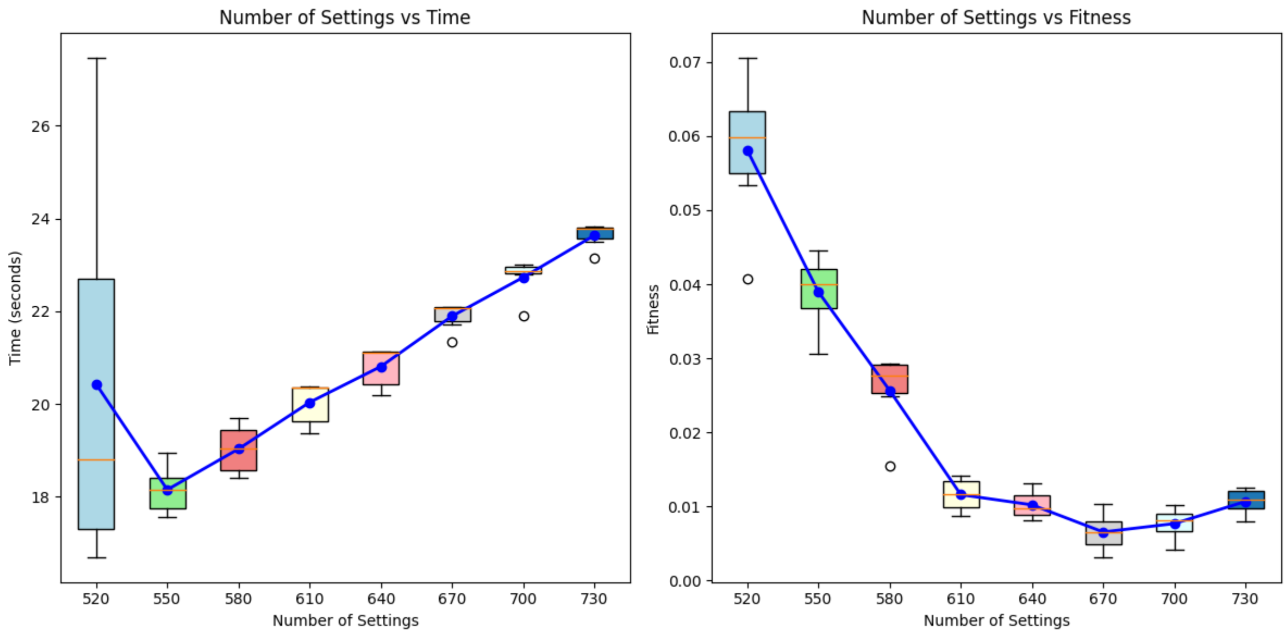


Figure 4.6: Behaviour of Number of Settings on (a) Computational Time & (b) Fitness

Figure 4.6 (a) indicates, as the number of settings increases, so does the algorithm's computing time. Median times follow a similar pattern, demonstrating that algorithmic time increases, which gives complexity. Outliers ('o' marks) indicate instances of significant divergence from the median, providing information about probable variances in computing performance. This information is critical for understanding DEA scalability and anticipating computing requirements for various issue sizes.

Figure 4.6 (b) demonstrates that as the number of settings increases, the fitness of the solutions improves upto a certain number of settings, after which improvement in fitness drops. The outliers show that, certain solutions stand out as much better or worse.

4.7 Variable Parameter: Number of Place Mutates

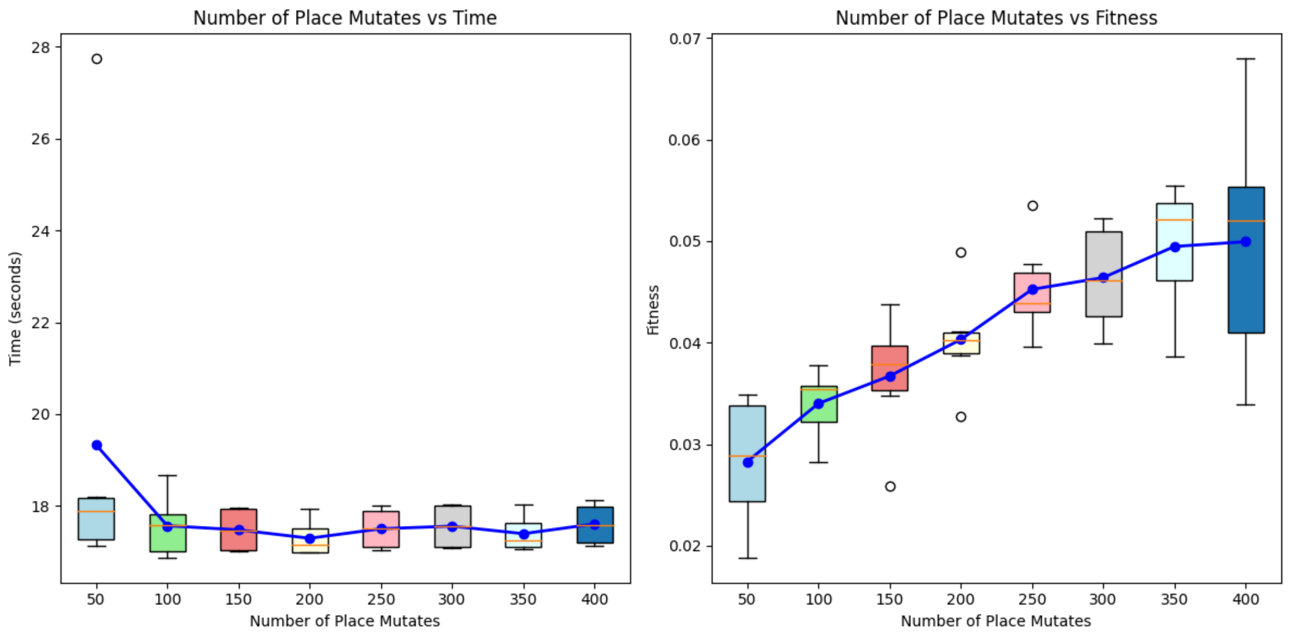


Figure 4.7: Behaviour of Number of Place Mutate on (a) Computational Time & (b) Fitness

Figure 4.7 (a) reveals that the time required to run with the different number of mutation places does not have any upward or downward trend, which means a number of places mutate does not have any effects. The median values of time across different mutation counts are highly steady, with only minor changes. This shows that the amount of place mutate has no affect on the algorithm's computational cost, allowing for scalability in the mutation operation without increasing computation time.

In Figure 4.7 (b), as the number of mutation places goes up, fitness values also increase, pointing to a decrease in solution quality. This trend suggest that more number of mutation places lead the algorithm away from getting optimal solutions. The variability and outliers with more number of mutation places suggest potential instability. The rise in fitness with more mutations implies a limit to their benefits, beyond which the algorithm's performance degrades.

Chapter 5

Discussion

5.1 Optimal Parameters

The study provided substantial insights into reaching optimal fitness levels within a computationally efficient framework, as seen in Figure 4.5 (b). The identified parameters for optimal outcomes are shown in Table 5.1.

Parameter	Interpretation	Optimal Value
num_persons	Total number of Persons	800
num_settings	Total number of Households	550
iterations	Number of Iterations	30
num_candidate_solutions	Total number of Candidate Solutions	30
num_mutation_solutions	Total number of Mutation Solutions	15
Z	Total number of pair of Offspring Solutions	7
num_place_mutate	Total number of Place Mutate	200

Table 5.1: Parameters and their optimal values

The experimental results show that using these specific parameter values results in a notable drop in fitness, with a minimum value ranging from 0.0035 to 0.01, which aligns with objective, as shown in Figure 4.5 (b). This suggests that using these particular parameters helps quickly reach the best fitness levels, highlighting their efficiency. The research implies that computational resources can be used effectively, as these identified parameters speed up the process of reaching the best solutions. In summary, this study establishes a practical and fast parameter set for achieving optimal results within a limited computational time.

5.2 Comparison of Synthetic Data with Empirical Data

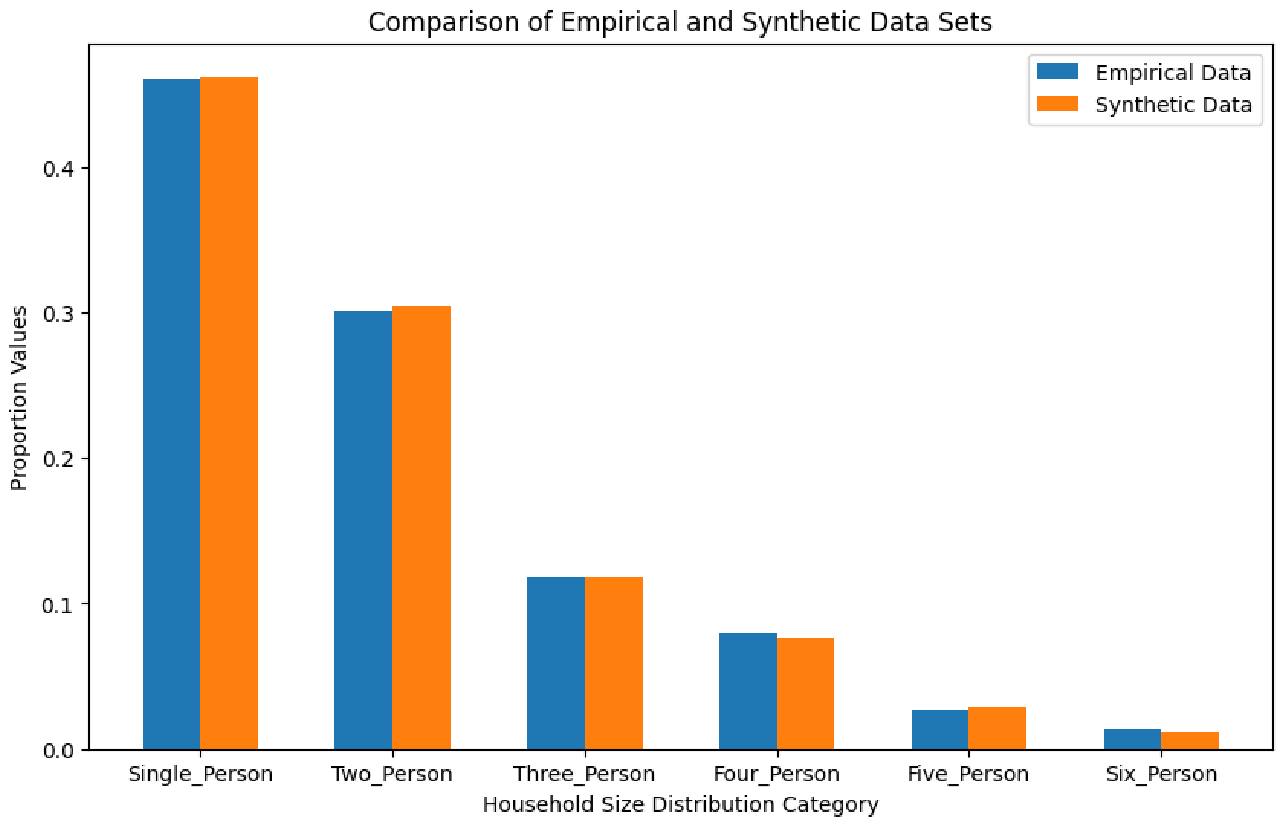


Figure 5.1: Comparision of Empirical and Synthetic Data

The bar chart (Figure 5.1) showcases an insightful comparison between the actual data collected and the synthetic data that has been finely optimized using DE algorithm as discussed in Chapter 3. The synthetic data which is generated using minimum fitness is used for comparison.

It is evident that the synthetic data generation approach is quite effective, as demonstrated by the closely matching proportions across all household categories, from single-person up to six-person households. The minimal discrepancies across these categories signify a high degree of accuracy in the synthetic data, reflecting a robust algorithmic performance which aligns with our objectives.

Chapter 6

Conclusion

In conclusion, our project has thoroughly investigated the optimization of synthetic data using the Differential Evolution (DE) Algorithm. A methodical approach was utilized that includes generating, evaluating, and optimizing synthetic datasets that accurately reflect empirical data. Moreover, the visualization techniques used provided critical insights of the effect of various parameters on the algorithm.

Key findings from our study reveal that specific parameter configurations significantly impact the DE algorithm's performance, influencing both the computational time and the fitness of the generated synthetic data. Moreover, certain parameters have a significant role —namely the number of persons, settings, and iterations— in steering the optimization process toward achieving the desired fitness levels.

On the contrary, other parameters, including the number of candidate solutions, mutation solutions, pairs of offspring solutions, and place mutates, lack a direct correlation with achieving optimal fitness values. In addition, it was observed that to achieve optimal fitness, the number of mutation solutions and the number of pairs of offspring solutions must be more than half of the number of candidate solutions.

Combining the specific findings related to the impact of various parameters on the DE algorithm's performance in mirroring empirical data, our report contributes significantly to the field of synthetic data generation. It expands on the current knowledge of generating synthetic data and provides assistance for future research. The ability to closely match the synthetic data with empirical data emphasizes our approach, highlighting its application in scenarios where real data may be incomplete or sensitive.

The insights and methodologies detailed in this report will serve as a valuable resource for researchers, supporting development in the field of synthetic data generation and optimization.

Bibliography

[1] Rainer Storn and Kenneth Price. Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11:341–359, 01 1997.

[2] Ali Wagdy, Hegazy Sabry, and Motaz Khorshid. An alternative differential evolution algorithm for global optimization. *Journal of Advanced Research*, 3, 04 2012.

[3] Terrance Liu, Jingwu Tang, Giuseppe Vietri, and Steven Wu. Generating private synthetic data with genetic algorithms. In *International Conference on Machine Learning*, pages 22009–22027. PMLR, 2023.

[4] Kit Wong and Z.Y. Dong. Differential evolution, an alternative approach to evolutionary algorithm. pages 73 – 83, 12 2005.

[5] Kai Qin, Vicky Huang, and Ponnuthurai Suganthan. Differential evolution algorithm with strategy adaptation for global numerical optimization. *Evolutionary Computation, IEEE Transactions on*, 13:398 – 417, 05 2009.

[6] Yuelin Gao and Junmei Liu. An improved differential evolution algorithm for solving constrained optimization problems. *Proceedings - 4th International Joint Conference on Computational Sciences and Optimization, CSO 2011*, 04 2011.

[7] Wu Deng, Shifan Shang, Xing Cai, Huimin Zhao, Yingjie Song, and Junjie Xu. An improved differential evolution algorithm and its application in optimization problem. *Soft Computing*, 25, 04 2021.

[8] Padarbinda Samal, Rakesh Swain, Chitralekha Jena, Pampa Sinha, Sanhita Mishra, and Sarat Swain. A modified differential evolution algorithm solving for engineering optimization problems. pages 1–5, 05 2022.

[9] Lianzheng Cheng, Jiaxi Zhou, Xing Hu, Ali Wagdy, and Yun Liu. Adaptive differential evolution with fitness-based crossover rate for global numerical optimization. *Complex Intelligent Systems*, 10, 07 2023.

[10] Statistisches Landesamt Rheinland-Pfalz. Haushalte und familien daten. https://www.statistik.rlp.de/de/gesellschaft-staat/haushalte-und-familien/basisdaten-land/, May 2011.

[11] Bilal Mirza, Millie Pant, Hira Zaheer, Laura Garcia-Hernandez, and Ajith Abraham. Differential evolution: A review of more than two decades of research . *Engineering Applications of Artificial Intelligence*, 90:103479, 04 2020.