Math 343 - Final Project

Word Frequency Counting Optimization in Java

Preston Duffield
duffiep@wwu.edu
Western Washington University
June 8, 2023

Contents

1 Introduction

The purpose of this experiment is to test the performance of a word counting program in Java[1]. The word counting program takes as input a buffer size, algorithm type, and input file.

We are interested in the time efficiency of the operation, which will be measured as the response variable. This variable, the time taken to complete the algorithm in milliseconds, will be recorded for each run of the experiment.

1.1 Design

The experiment is a 2^k factorial design, where $k = 3$. Each experiment will be run with a replication of $n = 1000$, meaning each factor level combination will be ran 1000 times. This is a balanced design, as each factor-level combination has an equal number of observations.

1.2 Factors

1. **Factor A: Buffer Size,** this is the amount of data the program will read from the file at once. The levels for this factor are 16 (low) bytes and 4096 (high) bytes.
2. **Factor B: Algorithm,** this is the specific approach used to perform the word frequency count. We have two levels for this factor, which are the Sorting (low) approach and the Hash Map (high) approach.
3. **Factor C: Input File,** this factor corresponds to the Input File that the program will process. We have 2 levels for this factor, which are Bible.txt (4.4 MB - low), and pride_and_prejudice.txt (757 KB - high)

Given that we have 3 factors each with 2 levels, we have a total of 8 treatment combinations. We plan to collect a sample size of 1000 for every treatment combination, resulting in a total of 8000 runs for the experiment.

1.3 Response

The response of the experiment is the time in seconds that the program took to execute. Seconds were recorded to 4 significant digits. The java code for the calculation is:

```
totalTimeInSeconds = (endTime - startTime) / 1000.0;
```

Where `startTime` was set before the algorithm was run, and `endTime` was set directly after.

1.4 Procedure

Listing 1: A Sample test.txt file.

```
This is a test for the ability of the word frequency count program
This is a test
```

[1]See Java Code in the Appendix.

Listing 2: A Sample invocation of the WordFrequencyCounter.java program

```
> java WordFrequencyCounter.java 16 sorting test.txt false
This: 2
a: 2
ability: 1
count: 1
for: 1
frequency: 1
is: 2
of: 1
program: 1
test: 2
the: 2
word: 1
Total time: 0.0070 seconds.
```

Listings 1 and 2 show a sample text file and a sample invocation of the program. From these listing we can see that the the program ran with a buffer size of 16, using the ”sorting” algorithm, on test.txt. The final argument ”false”, indicates whether or not the program should run in silent mode and not print the results. Finally, we see that the program took `0.0070` seconds to run in total.

The WordFrequencyCounter.java program was run multiple times with the help of a Python program[2]. The program run_java_experiments.py takes as input the number of replicants n. It then runs the program n times for each treatment level combination as defined by the `combinations` array.

Listing 3: A Sample invocation of the run_java_experiments.py program

```
> python3 run_java_experiments.py 1000
Running 104/1000 replicate for combination [-1, 1, 1]:  76% | 6104/8000 [1:53:37<31:00,  1.02it/s]
```

The program took 3318.06 seconds or about 55 minutes to complete. It was run on a MacBook Pro, which has a 2.2 GHz 6-Core Intel Core i7 Processor.

The output of this program is a CSV file containing the treatment level combination and response in seconds that each invocation of the WordFrequencyCounter.java incurred. The program was run on a single computer in a single program call. Collecting the data programatically in this way ensures the validity and reduces the variability of each experiement.

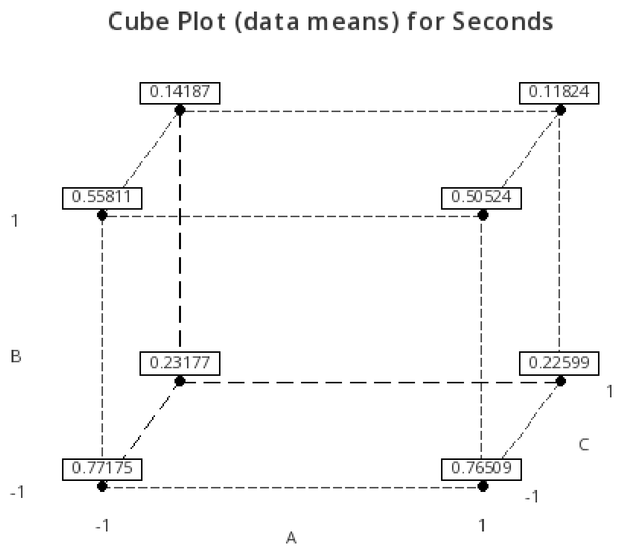


Figure 1: Cube Plot from Minitab.

The data collected can be summarized in a cube plot. The cube plot shows the means of each treatment level combination. The Cube Plot in Figure 1 shows the means of (A) Buffer Size, (B) Algorithm, and (C) Input File, at each of the corresponding high and low levels.

[2]See Python Code in the Appendix.

2 Analysis of Data

Statisical analysis was performed with Minitab. The objective of the analysis was to determine what factors were significant in affecting the runtime in seconds of the computer program. The analysis will be performed in 5 steps:

1. Estimate factor effects
2. Form initial model
3. Perform statistical testing
4. Analyze residuals
5. Interpret results

2.1 Main and Interaction Effects

By using the standard order, and the means from the cube plot in Figure 1, we can estimate the main and interaction effects using contrast coefficients. For example, The estimated main effect of A is:

$$\begin{aligned}\text{Est. Main Effect of A} &= \frac{1}{4}\sum_{i=1}^{8} c_i \bar{y}_i \\ &= \frac{1}{4}\left(-\bar{y}_{(1)} + \bar{y}_a - \bar{y}_b + \bar{y}_{ab} - \bar{y}_c + \bar{y}_{ac} - \bar{y}_{bc} + \bar{y}_{abc}\right) \\ &= \frac{1}{4}(-0.77175 + 0.76509 - 0.55811 + 0.50524 - 0.23177 + 0.22599 - 0.14187 + 0.11824) \\ &= -0.022235\end{aligned}$$

Continuing using the contrast constants for a 2^3 factorial design yields the following table.

Effect	Estimated Main/Interaction Effect
I	0.829515
A	−0.022235
B	−0.167785
C	−0.016014
AB	0.470580
AC	0.007529
BC	0.068960
ABC	0.007090

2.2 Regression Model

Regression models for 2^3 factorial designs can be described as follows.

$$E(y) = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 + \hat{\beta}_{12} x_1 x_2 + \hat{\beta}_3 x_3 + \hat{\beta}_{13} x_1 x_3 + \hat{\beta}_{23} x_2 x_3 + \hat{\beta}_{123} x_1 x_2 x_3$$

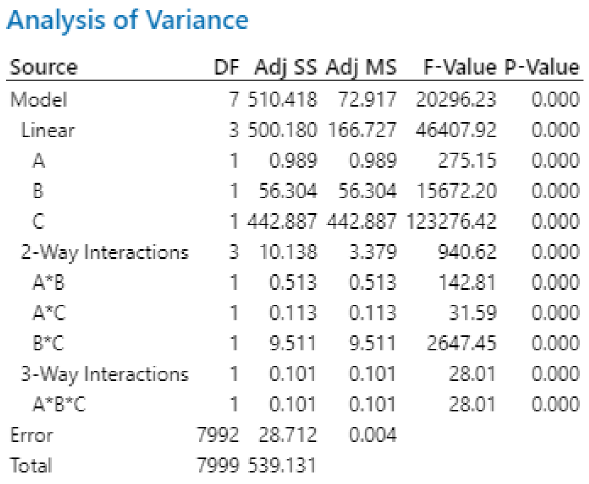
Where the coded variables x_1, x_2, and x_3 represent A, B, and C, respectively. The $x_1 x_2$ term is the AB interaction, and so on for the interaction terms AB through ABC. We can utilize the main and interaction effects found in section 2.1 to estimate the β parameters.

$$\begin{aligned} \hat{\beta}_1 &= \frac{\text{Main Effect of A}}{2} \\ &= \frac{-0.022235}{2} \\ &= -0.0111175 \end{aligned}$$

Continuing following this logic produces the regression model:

$$E(y) = 0.41475 + -0.01111x_1 + -0.08389x_2 + -0.008x_1x_2 + 0.23529x_3 + 0.00376x_1x_3 + 0.03448x_2x_3 + 0.00354x_1x_2x_3$$

2.3 Anova Analysis of Significant Effects



Analysis of Variance

Source	DF	Adj SS	Adj MS	F-Value	P-Value
Model	7	510.418	72.917	20296.23	0.000
Linear	3	500.180	166.727	46407.92	0.000
A	1	0.989	0.989	275.15	0.000
B	1	56.304	56.304	15672.20	0.000
C	1	442.887	442.887	123276.42	0.000
2-Way Interactions	3	10.138	3.379	940.62	0.000
A*B	1	0.513	0.513	142.81	0.000
A*C	1	0.113	0.113	31.59	0.000
B*C	1	9.511	9.511	2647.45	0.000
3-Way Interactions	1	0.101	0.101	28.01	0.000
A*B*C	1	0.101	0.101	28.01	0.000
Error	7992	28.712	0.004		
Total	7999	539.131			

Figure 2: ANOVA table from Minitab.

For each main and interaction effect we will test the following hypothesis:
H_0: True Effect = 0.
H_a: True Effect $\neq$ 0.

Using the p-values from the ANOVA table, we can conclude that the interaction effects AB, AC, BC, and ABC, as well as the main effects A, B, and C, are significant at $\alpha = 0.05$. Since each Factor is significant, we will not remove any factors from the model as it is already refined.

2.4 Interaction Analysis

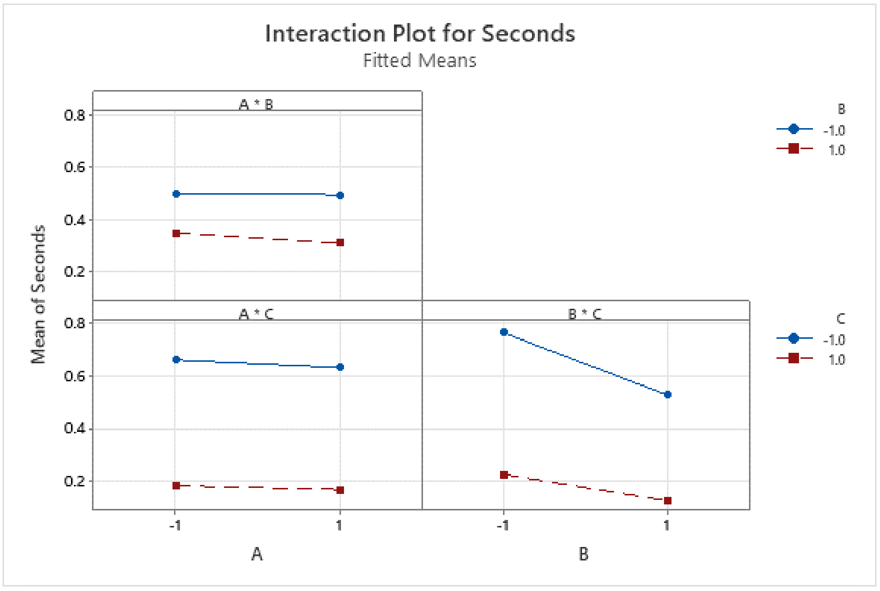


Figure 3: Interaction Plot from Minitab.

As we can see in Figure 3, the interaction plot would indicate that there is not 2-way interaction between the terms. This is due to the large sample size of $n = 1000$. We can observe that there does exist slight non-parallelity between the lines in the plot. This slight non-parallelity is indicative of interaction between the terms.

2.5 Main Effects Analysis

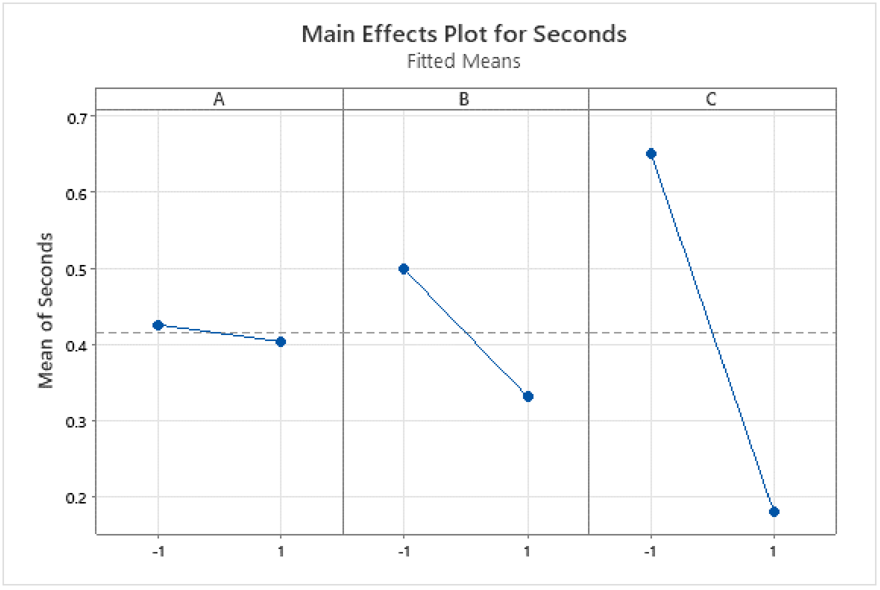


Figure 4: Main Effects Plot from Minitab.

The main effects plot shows each of the terms having a negative effect on the response. With A having the smallest effect, B having more of an effect, and C having the largest negative effect.

3 Residual Analysis

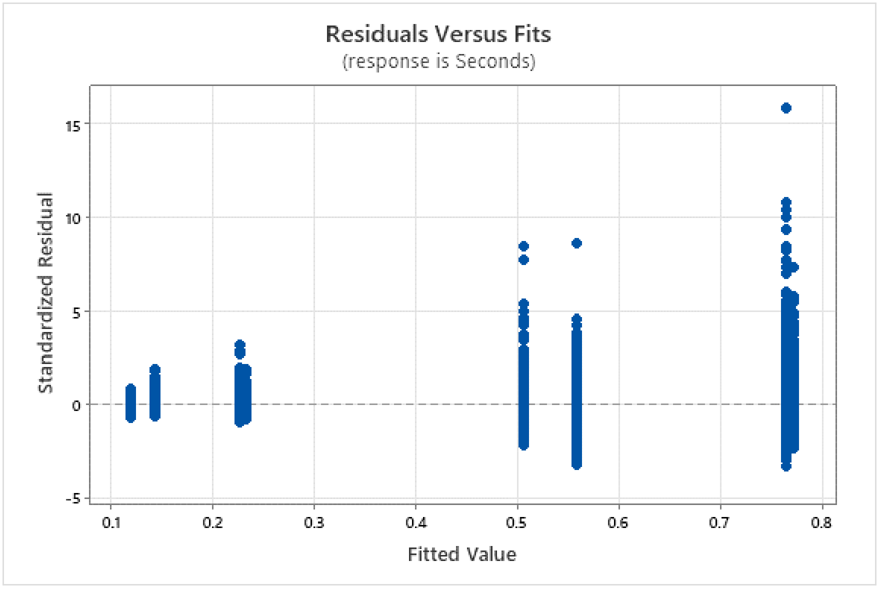


Figure 5: Residual Plot from Minitab.

The residuals show heteroskedasticity, or non-constant variance. This indicates that the model assumtions are not satisfied in the current model.

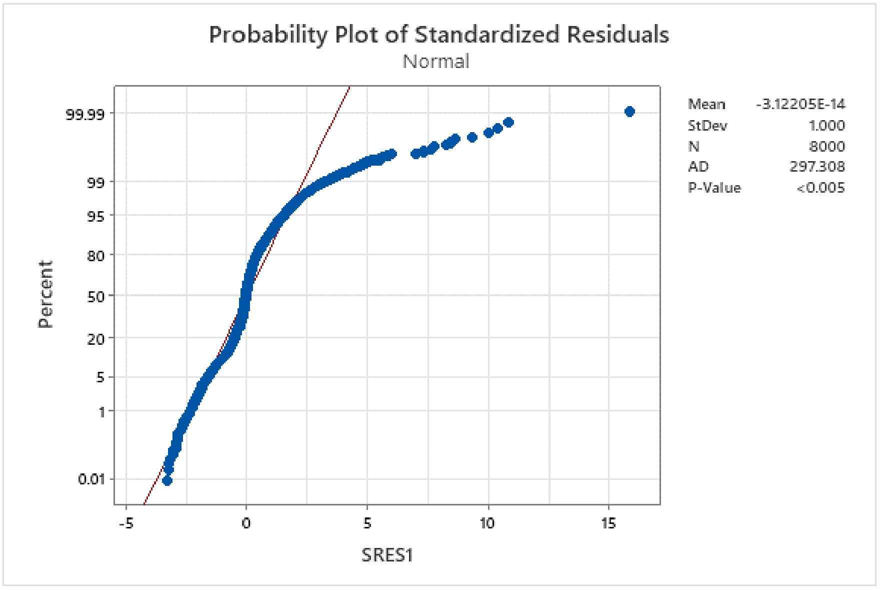


Figure 6: Normal Probability Plot from Minitab.

Furthermore, the normal probability plot indicates that the data does not come from a normal distribution. Using the P-value from the Anderson-Darling normality test we can conclude that there is enough statistical evidence to support that the data are not drawn from a normal distribution.

This together with the residuals show that the model assumtions are not satisfied in the current model.

4 Transformed Model

The remedy to this problem with the model assumptions is to perform a transformation on the response. After testing multiple transformations, a logarithmic transformation was selected. The transformed response y' is:

$$y' = ln(\text{Seconds})$$

4.1 Factor Effects

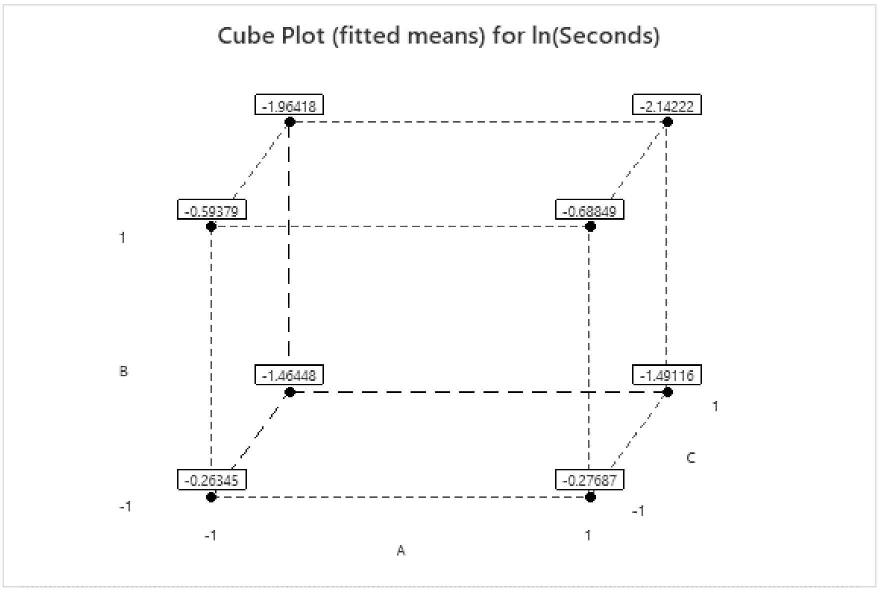


Figure 7: Cube Plot from Minitab.

The Cube plot in figure 7 shows the new means for the transformed y' for each Factor (A) Buffer Size, (B) Algorithm, and (C) Input File, at each of the corresponding high and low levels.

We can then estimate the Main and interaction Effects using contrast vectors. This results in the following table.

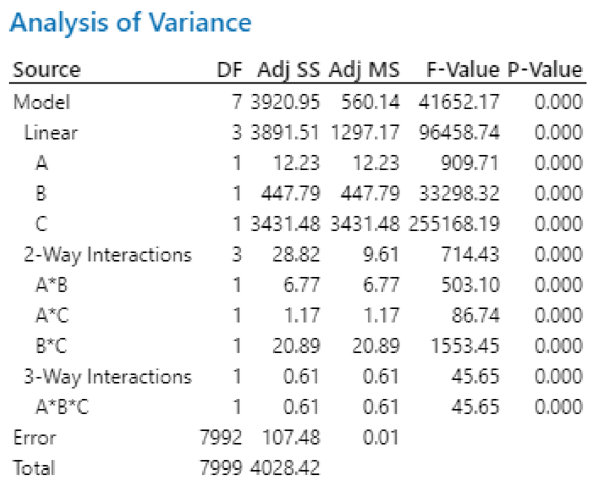
Effect	Estimated Main/Interaction Effect
I	-2.22116
A	-0.07821
B	-0.47318
C	1.30986
AB	-0.05815
AC	-0.02415
BC	-0.10219
ABC	-0.01751

4.2 Regression Model

Using the new Main and Interaction Effects, the new model is:

$$E(y') = -1.11058 - 0.039105x_1 - 0.23659x_2 - 0.02907x_1x_2 + 0.65493x_3 - 0.01207x_1x_3 - 0.05109x_2x_3 - 0.00875x_1x_2x_3$$

4.3 Anova Analysis of Significant Effects



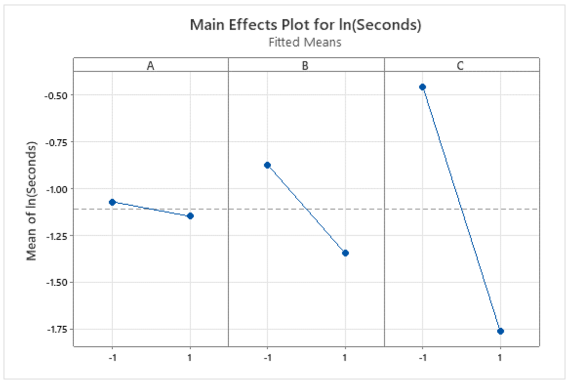
Analysis of Variance

Source	DF	Adj SS	Adj MS	F-Value	P-Value
Model	7	3920.95	560.14	41652.17	0.000
Linear	3	3891.51	1297.17	96458.74	0.000
A	1	12.23	12.23	909.71	0.000
B	1	447.79	447.79	33298.32	0.000
C	1	3431.48	3431.48	255168.19	0.000
2-Way Interactions	3	28.82	9.61	714.43	0.000
A*B	1	6.77	6.77	503.10	0.000
A*C	1	1.17	1.17	86.74	0.000
B*C	1	20.89	20.89	1553.45	0.000
3-Way Interactions	1	0.61	0.61	45.65	0.000
A*B*C	1	0.61	0.61	45.65	0.000
Error	7992	107.48	0.01		
Total	7999	4028.42			

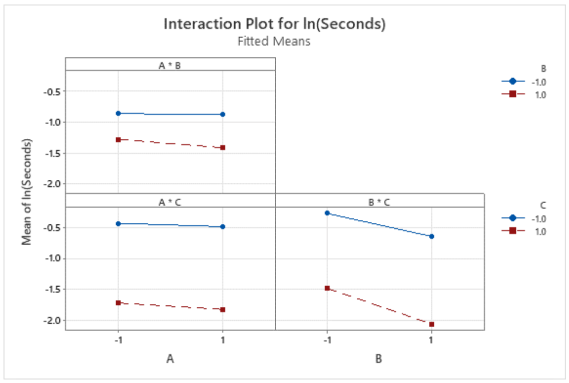
Figure 8: ANOVA table from Minitab.

Since each P-value $< \alpha$, we can conclude that the interaction effects AB, AC, BC, and ABC, as well as the main effects A, B, and C, are significant at $\alpha = 0.05$. Since each Factor is significant, we will not remove any factors from the model as it is already refined.

4.4 Interaction and Main Effect Analysis



(a) Main Effects Plot from Minitab.



(b) Interaction Plot from Minitab.

We can observe that there does exist slight non-parallelity between the lines in the plot. This slight non-parallelity is indicative of interaction between the terms.

The main effects plot shows each of the terms having a negative effect on the response. With A having the smallest effect, B having more of an effect, and C having the largest negative effect.

4.5 Residual Analysis

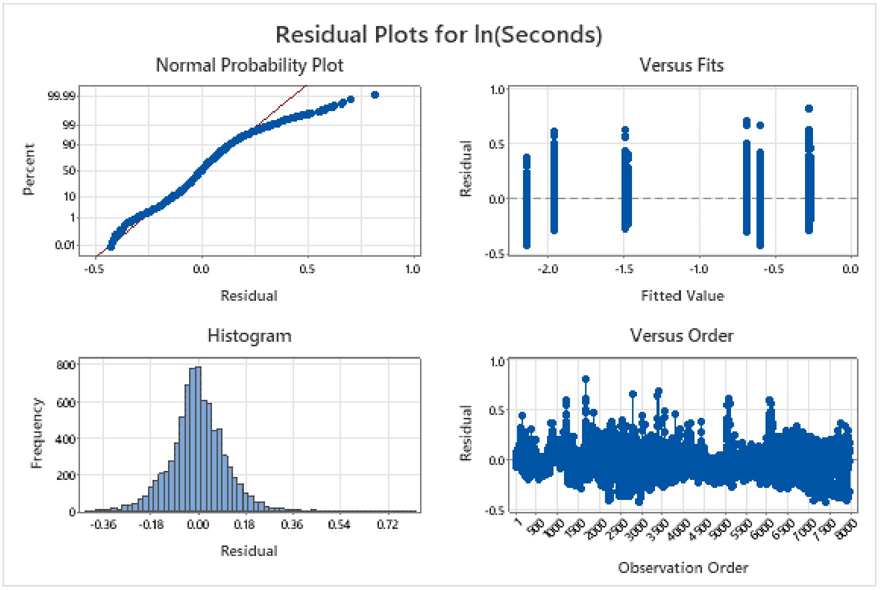


Figure 10: Four in One Plot from Minitab.

The residuals no longer show heteroskedasticity, or non-constant variance. This indicates that the model assumtions are satisfied in the transformed model.

The normal probability plot is a better fit than the untransofrmed model, as it appears to follow a straight line. However, using the P-value from the Anderson-Darling normality test we can conclude that there is enough statistical evidence to support that the data are not drawn from a normal distribution.

To counter this non-normality we will use hypothesis tests and confidence intervals that are robust to non-normality in the following analysis. Since the residuals appear to be randomly scattered, we can proceed as though the residuals are normally distributed.

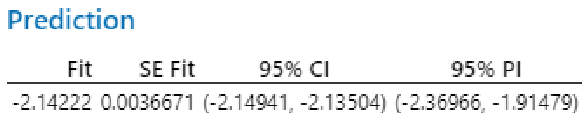
5 Interpretation of the Results

5.1 Optimal Factor-Level Combination

Using the statistical analysis we can find the factor-level combination that gives the optimal mean response. Note that the optimal mean response would be the lowest value in seconds as we want our program to run fast.

To do this we can refer back to Figure 1, in Section 1, the Cube Plot. The Cube Plot shows that the mean response is lowest at the factors A^+, B^+, and C^+. Equivalently, the program is the fastest when Buffer Size is 4096, Algorithm is Hash Map, and Input File is pride_and_prejudice.txt.

5.2 Confidence Interval for the True Mean Response



Prediction

Fit	SE Fit	95% CI	95% PI
-2.14222	0.0036671	(-2.14941, -2.13504)	(-2.36966, -1.91479)

Figure 11: Prediction for y' from Minitab.

Figure 11 shows the prediction and 95% confidence interval for the transformed model at A^+, B^+, and C^+. In order to interpret these results we must use the untransform the predicted values using the inverse of the natural log, e^x.

Thus the untransformed confidence interval is $(0.11655, 0.11823)$, that is, we are 95% confident that the true mean response when Buffer Size is 4096, Algorithm is Hash Map, and Input File is pride_and_prejudice.txt, is between 0.11655, and 0.11823 seconds.

6 Conclusion

What I learned is that even though the interaction plots are nearly parallel, since n was so large, even slight non-parallelity indicated interaction. This was also a surprising result, as initially, when I did a hypothesis test on the interaction effects, I predicted the interaction plots to intersect, or at least be more non-parallel than they were.

One surprising result was that all the interaction terms were significant. When I initially ran the experiment I anticipated that the Main effects would be significant, but the interaction effects would not. This was not the case and was an interesting result.

Another surprising result is how difficult it was to try to get the residuals normally distributed. I performed a Box-Cox transformation in order to find the best transformation was the natural log, however even after the transformation the data was not normally distributed.

One modifications I might make to the expereiment in a future trial would be to find a way to implement threading. In my opinion this would be a more interesting factor that file size, as large files take longer to process is not a particularly interesting conclusion. Writing the code for the threading became quite complicated, and since it was not required in the scope of the project, I decided to scrap the idea. The input text file was a simple factor I could include to keep the experiment a 2^3 design.

A Appendix

A.1 Java Code

Listing 4: Source Code for the WordFrequencyCounter.java file.

```java
import java.io.*;
import java.nio.file.*;
import java.util.*;

public class WordFrequencyCounter {
  private static long startTime;

  public static void main(String[] args) throws IOException {
    if (args.length != 4) {
      System.err.println(
          "Usage:␣WordFrequencyCounter␣<buffer␣size>␣<algorithm>␣<input␣file>␣<quiet␣flag>");
      System.exit(1);
    }

    int bufferSize = Integer.parseInt(args[0]);
    String algorithm = args[1];
    Path inputFilePath = Paths.get(args[2]);
    boolean isQuiet = Boolean.parseBoolean(args[3]);

    if (!Files.exists(inputFilePath)) {
      System.err.println("The␣input␣file␣does␣not␣exist.");
      System.exit(2);
    }

    startTime = System.currentTimeMillis();

    switch (algorithm.toLowerCase()) {
      case "hashmap":
        hashMapApproach(inputFilePath, bufferSize, isQuiet);
        break;
      case "sorting":
        sortingApproach(inputFilePath, bufferSize, isQuiet);
        break;
      default:
        System.err.println("Invalid␣algorithm␣type.␣It␣should␣be␣'hashmap'␣or␣'sorting'.");
        System.exit(3);
    }

    long endTime = System.currentTimeMillis();
    double totalTimeInSeconds = (endTime - startTime) / 1000.0;
    System.out.printf("Total␣time:␣%.4f␣seconds.%n", totalTimeInSeconds);
  }

  private static void
      hashMapApproach(Path filePath, int bufferSize, boolean isQuiet) throws IOException {
    try (BufferedReader reader = new BufferedReader(new FileReader(filePath.toFile()), bufferSize)) {
      HashMap<String, Integer> wordCount = new HashMap<>();
      String line;

      while ((line = reader.readLine()) != null) {
        String[] words = line.split("\\s+");
        for (String word : words) {
          wordCount.put(word, wordCount.getOrDefault(word, 0) + 1);
        }
      }

      if (!isQuiet) {
        for (Map.Entry<String, Integer> entry : wordCount.entrySet()) {
          System.out.println(entry.getKey() + ":␣" + entry.getValue());
        }
      }
    }
  }

  private static void
      sortingApproach(Path filePath, int bufferSize, boolean isQuiet) throws IOException {
    try (BufferedReader reader = new BufferedReader(new FileReader(filePath.toFile()), bufferSize)) {
      ArrayList<String> wordList = new ArrayList<>();
      String line;

      while ((line = reader.readLine()) != null) {
        String[] words = line.split("\\s+");
        wordList.addAll(Arrays.asList(words));
      }

      Collections.sort(wordList);

      if (!isQuiet) {
        int count = 1;
        for (int i = 1; i < wordList.size(); i++) {
          if (wordList.get(i).equals(wordList.get(i - 1))) {
            count++;
          } else {
            System.out.println(wordList.get(i - 1) + ": " + count);
            count = 1;
          }
        }

        // Print the last word in the list and its count
        System.out.println(wordList.get(wordList.size() - 1) + ": " + count);
      }
    }
  }
}
```

A.2 Python Code

Listing 5: Source Code for the run_java_experiments.py file

```
import subprocess
import csv
import sys
from tqdm import tqdm

def main(replicants):
    # Define the mapping of parameters
    parameters = {
        'Buffer␣Size': {-1: '16', 1: '4096'},
        'Algorithm␣Type': {-1: 'sorting', 1: 'hashmap'},
        'Input␣File': {-1: 'bible.txt', 1: 'pride_and_prejudice.txt'}
    }

    # Define the combinations of parameters to run
    combinations = [
        [-1, -1, -1],
        [1, -1, -1],
        [-1, 1, -1],
        [1, 1, -1],
        [-1, -1, 1],
        [1, -1, 1],
        [-1, 1, 1],
        [1, 1, 1]
    ]

    # Prepare the CSV file
    with open('results.csv', 'w', newline='') as csvfile:
        fieldnames = ['Buffer␣Size', 'Algorithm␣Type', 'Input␣File', 'Seconds']
        writer = csv.DictWriter(csvfile, fieldnames=fieldnames)

        writer.writeheader()

        total = len(combinations) * replicants
        pbar = tqdm(total=total, ncols=120)

        # For each combination of parameters...
        for combination in combinations:
            # Repeat the experiment the desired number of times
            for i in range(replicants):
                # Prepare the arguments for the Java program
                args = ['java', 'WordFrequencyCounter.java']
                args += [parameters[fieldnames[i]][combination[i]] for i in range(len(combination))]
                args.append('true')

                # Run the Java program and capture the output
                result = subprocess.run(args, capture_output=True, text=True)

                # Extract the time value from the output
                time = float(result.stdout.split()[-2])

                # Write the result to the CSV file
                writer.writerow({
                    'Buffer␣Size': combination[0],
                    'Algorithm␣Type': combination[1],
                    'Input␣File': combination[2],
                    'Seconds': time
                })

                pbar.set_description(
                  f"Running␣{i+1}/{replicants}␣replicants␣for␣combination␣{combination}")
                pbar.update()
        pbar.close()

if __name__ == "__main__":
    main(int(sys.argv[1]))
```