UNIVERSITY OF TARTU
Faculty of Science and Technology
Institute of Computer Science
Computer Science Curriculum

Mihkel Uutar

Beyond Worst-Case Complexity of the Simplex Method

Bachelor’s Thesis (9 ECTS)

Supervisor: Kallol Roy, PhD

Tartu 2024

Beyond Worst-Case Complexity of the Simplex Method

Abstract:

This thesis investigates the performance of the Simplex Method beyond its theoretical worst-case by empirically assessing the effect of input distributions on efficiency.The research consists of three experiments, which focus on different input distributions and an optimized pivoting rule, the Zero-Exploiting Simplex Method (ZESM). The results demonstrate that input distributions significantly affect performance with structured sparse requiring fewer operations compared to dense matrices. Implementing ZESM reduced the number of operations required by over 50% across various inputs. The research aims to provide a foundation for future research to optimize the Simplex Method based on input characteristics.

Keywords:

Simplex Method, Computational Complexity, Worst-Case Complexity, Input Distribution, Empirical Research, Algorithm Optimization

CERCS:

P170 Computer science, numerical analysis, systems, control

Simplexi meetodi halvima juhu keerukusest edasi

Lühikokkuvõte:

See lõputöö uurib Simplexi meetodit teoreetilisest halvima juhu keerukusest kaugemale, hinnates empiiriliselt sisendite jaotuste mõju sooritatud operatsioonide arvule. Praktiline analüüs koosneb kolmest katsest, mis keskenduvad erinevatele sisendite jaotustele ja optimeeritud pööramisreeglile (ZESM). Tulemused näitavad, et sisendite jaotused mõjutavad oluliselt vajaminevate operatsioonide hulka, kusjuures struktureeritud hõredad maatriksid nõuavad vähem operatsioone võrreldes tihedate maatriksitega. ZESM-i rakendamine vähendas erinevatel sisenditel nõutavate operatsioonide arvu üle 50%. Töö on aluseks edasisteks uuringuteks Simplexi meetodi optimeerimiseks lähtuvalt sisendi omadustest.

Võtmesõnad:

Simplexi meetod, Keerukus, Halvima juhu keerukus, Sisendite jaotus, Empiiriline uurimus, Algoritmide optimeerimine

CERCS: P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

Contents

1 Introduction

The Simplex Method is a linear programming algorithm used to solve optimization problems. It was named as one of the 10 most important algorithms by the journal of *Computing in Science & Engineering* [1]. Despite its theoretical polynomial worst-case complexity, its practical performance is often near linear. This thesis looks beyond worst-case complexity by proposing three research questions:

> **RQ1:** How consistent is the performance of Simplex Method across different types of input data distributions?
> **RQ2:** What are the impacts of simple optimization techniques on the computational expensiveness of the Simplex Method?
> **RQ3:** How does input preprocessing affect the performance of the Simplex Method?

This thesis investigates the effect of input distributions on the performance of the Simplex Method. Three experiments are designed which use the number of operations required to reach an optimal solution as a metric. As the research on the Simplex Method is mostly concerned with theoretical research on pivot rules, the findings serve as a foundational block for further research for optimizing the Simplex Method for specific data distributions.

The thesis organization is as follows: (i) Background section introduces the necessary mathematical preliminaries about linear programming, Simplex Method and worst-case complexity analysis to understand the following contents; (ii) Methodology section describes the tools and experiments used to evaluate Simplex Methods' performance across different data distributions; (iii) Results section presents the findings from three practical experiments, evaluates the impact of data distribution and links the empirical results with theoretical understanding. The discussion section underlines most important findings, discusses potential optimizations and further areas of research.

OpenAI's ChatGPT-4[1] has been used to improve the readibility and correct minor spelling errors of this thesis. Artifical intelligence-based tools have not been used to create any meaningful written content for this thesis. In the Visual Studio Code[2] Integrated Development Environment (IDE) GitHub's Github Copilot[3] was used for code recommendations and completion. All recommendations were carefully evaluated and tested before implementation.

[1]https://openai.com/index/gpt-4/
[2]https://code.visualstudio.com/
[3]https://github.com/features/copilot

2 Background

This section gives a necessary overview of linear programming, worst-case complexity analysis and development of the Simplex Method. The information in this section serves to support the practical research of this thesis and helps understand the research questions and experiment design.

2.1 Linear Programming

Linear Programming is a mathematical method used to find an optimal solution to an optimization problem by linear constraint. A large number of complex economics, engineering or resource management problems, such as minimizing the cost of operations or maximizing profit can be described as a set of linear constraints.

A linear program (LP) in standard form is defined as an optimization problem over $\mathbf{x} \in \mathbb{R}^m$:

$$\text{maximize} \quad \mathbf{c}^T\mathbf{x} \tag{1}$$

$$\text{subject to} \quad A^T\mathbf{x} = \mathbf{b} \tag{2}$$

$$\mathbf{x} \geq 0 \tag{3}$$

where:

- $\mathbf{c}^T\mathbf{x}$ is the objective function, whose value the algorithm aims to maximize
- $A^T\mathbf{x} = \mathbf{b}$ represents the constraints constraints under which the optimizations needs to be performed
- and $\mathbf{x} \geq 0$ is the non-negativity constraint for the inputs.

To provide a visual example, a linear program with 3 constraints is visualized as a polyhedron using a Python package GILP [2], developed by Robbins et al. Figure 1 displays the visualization of maximum value and Figure 2 visualizes the constraints of a linear problem.

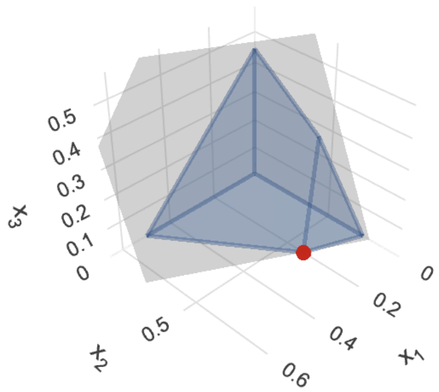

Figure 1. Visualized Maximum Value of LP

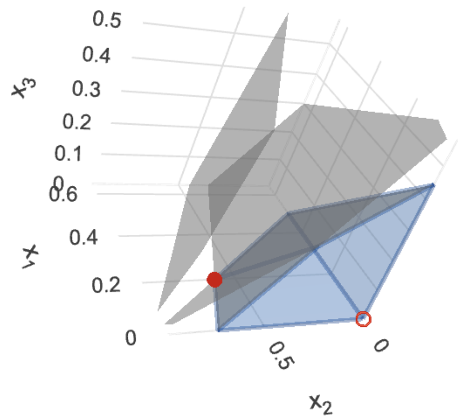

Figure 2. Visualized Constraints of LP

The light blue polyhedron formed by the linear constraints in Figures 1 and 2 represents the feasible region of the LP and the red dot represents the optimum value for that LP.

$$\text{maximize } Z = 7x_1 + 9x_2 + 6x_3 \tag{4}$$

$$\begin{aligned} \text{subject to } & 6x_1 + 7x_2 + 2x_3 \geq 7 \\ & 8x_1 + 4x_2 + 9x_3 \geq 5 \\ & 7x_1 + 5x_2 + 4x_3 \geq 4 \\ & x_1, x_2, x_3 \geq 0 \end{aligned}$$

2.2 Beyond Worst-Case Complexity

Worst-case complexity provides an upper bound on the resources required by an algorithm for any input size n, irrespective of the input's distribution. Worst-case complexity is expressed using Big-O notation, which categorizes algorithms according to the rate at which their runtime increases as does the input size [3]. Common complexity classes are Constant Time $(O(1))$, Linear $(O(n))$, Polynomial $(O(n^k))$ where k is a constant exponent and Exponential $(O(2^n))$.

In 1970, Klee and Minty constructed an example of a linear program of n inequality constraints and n variables, which required $2^n - 1$ iterations to be solved using Dantzigs' pivoting rule, proving that Simplex Method has exponential complexity in the worst-case [4]. In 1982, Borgwardt showed that a version of the Simplex Method, called *Schatteneckenalgorithmus* exists, which had a polynomial upper bound for a problem with m inequality constraints and n variables [5]. However, he assumed that the feasible regions are sampled from an independent and identical distribution in a high-dimensional space. In the same year, Smale published an article in which he showed that for a fixed number of constraints the number of steps required to solve a problem by a variant of the Simplex Method grows slower than the squaret root of the number of variables [6]. This indicates, that the computation growth time is less than exponential.

Spielman et al. introduced the concept of smoothed analysis to better understand algorithms that have poor theoretical worst-case performance, but good average-case performance on random input distributions [7]. The 2003 paper by Spielman et al. shows that the Simplex Method has polynomial *smoothed* complexity by introducing small randomness into the worst-case analysis, proving Simplex Method has polynomial complexity in almost all instances. The polynomial theoretical performance is contrasted by the performance of the most efficient implementations of the Simplex Method. In an article, Karp states that linear programs can be solved "with a number of pivoting steps that is roughly linear" [8].

2.3 Simplex Method

The Simplex Method was first worked on by George B. Dantzig in the year 1947 and was first intended to be used for "planning of large-scale enterprises" [9]. The Simplex Method is not an algorithm in itself, but a class of different algorithms for solving LPs by moving from vertex to vertex along the edges of a feasible region until an optimal solution is found. The methods differ based on the way of choosing the next vertex, often referred to as the "pivot rule" [10]. This thesis focuses on the pivoting rule introduced by George B. Dantzig for its simplicity and good performance on a variety of inputs.

A pseudocode example of the Simplex Method for maximization is presented in Algorithm 1. This is the base for the Python implementation in the design of experiments used in the paper. The pivoting rule used in the pseudocode is based on the method developed by Dantzig in 1951 [11]. The rule chooses the edge which minimizes the quotients the most. Dantzig originally proposed the pivoting rules for minimization, so instead of choosing the largest quotient the rule in this implementation of the Simplex Method chooses the smallest instead.

The pseudocode and Python implementation use a structure named "Simplex Tableau" which is a tabular represantation organizing all coefficients of the objective function and the variables of the constraints into a matrix format. Each row in the tableau is constructed from one inequality constraint and the constraint bound is added as the last element. The objective function is added as the last row with its coefficients multiplied by -1 since the coefficients are moved from the right side of the equation to the left side.

An example of a Simplex Tableau is provided in Table 1. In the table x_1, x_2, x_3 represent the variables, s_1, s_2, s_3 represent the slack variables, Z represents the column for the objective function value and c represents the inequality constraint bounds.

Table 1. Example of a Simplex Tableau

	x_1	x_2	x_3	s_1	s_2	s_3	Z	c
Inequality Constraint (1)	6	3	4	1	0	0	0	1
Inequality Constraint (2)	2	7	8	0	1	0	0	7
Inequality Constraint (3)	7	5	4	0	0	1	0	4
Objective Function	-2	-8	-5	0	0	0	1	0

In 2006, Kelner and Spielman presented the first version of the Simplex Method which had polynomial-time complexity [12]. This was achieved by projecting the constraints of the linear problem into a lower-dimensional space (the so-called *shadow-vertex* pivoting) to simplify the original linear program.

Algorithm 1 Simplex Method for Maximization

Input: 3-tuple A, b, c; where A represents objective function coefficients, b is a matrix of inequality constraints and c contains constraint bounds

Result: Simplex Tableau, where the maximum value for the objective function is in the bottom right corner

```
Initialize simplex_tableau from A, b, and c
Add slack variables to simplex_tableau to convert inequality constraints and bounds
  into equalities.
while a negative coefficient exists in the objective row of simplex_tableau do
    Identify the pivot column with the most negative coefficient in the objective row of
      simplex_tableau
    foreach row in simplex_tableau do
        if element in the pivot column is positive then
            Calculate quotient by dividing the right-most element in the row by the
              element in the pivot column
        else
            Set quotient for the row as infinity.
    Select the pivot row as the one with the smallest quotient
    Set the value for pivot element as 1 and other values in the column to 0.
```

2.4 Empirical Experiments on the Performance of the Simplex Method

In his 1987 paper, "Origins of the Simplex Method" [9], Dantzig predicted that new methods will become more effective than Simplex, not because of any theoretical reasons regarding polynomial time, but because they can more effectively exploit the sparsity and structures of practical problems. Most of the literature on the topic is focused on implementation of pivoting rules to traverse the vertices more efficiently. This section contains results from empirical research about Simplex Methods' performance using

Dantzigs pivoting rule as well as experiments on the use of different pivot rules.

In 1962, Kuhn and Quandt investigate on practical performance of the Simplex Method. They experimented with matrices of 5×5, 10×10, 15×15, 20×20 and 25×25 with their elements distributed randomly in the range from 1 to 1,000. The results were promising, as the average number of iterations over 100 problems for matrixes with the size 25×25 was 18 to 19 [13].

Zadeh, stated in his 1980 report that the Simplex Method solves linear problems with n inequalities in n to $3n$ pivots [14]. In the same paper he suggests a pivoting rule named "least entered" to supress the algorithm from entering any column twice before all of them have been visited. In the same year, Dantzig estimated the bound of number of operations as a multiple of the number of equations in a linear problem [15] and theoretically showed that the distributions have an impact on the bounds for number of iterations.

In 2021, Adham et al empirically measured the performance of Simplex Method by selecting the pivoting rule using machine learning [16]. They used a test-set of 7,729 linear problems, each ranging from 120 to 200 inequality constraints and 50 to 100 variables. The results showed that Dantzigs' pivoting rule required two times more iterations on average when compared to the "best in theory" rule. This suggests, that a large effort in optimizing the algorithm should come from finding a more efficient pivoting rule.

3 Methodology

This section describes the tools and frameworks used in this thesis, the practical experiments, and data generation methods. It discusses the rationale for experiment parameter selection and also, identifies some potential shortcomings of the methodology. Python[4] version 3.10.13 is used for the implementation of the experiments. Primary Python packages used are NumPy[5] (version 1.26.4) for numerical operations and Matplotlib[6] (version 3.8.3) for visualizing the results and input distributions. Additional standard libraries used are `json`, `random`, `os` and `copy`. GILP (Geometric Implementation of Linear Programs) [2], is a Python package used to visualize geometry of linear problems and the Simplex Method. GILP is used for both visualizing and verifying the correctness of the self-implemented Simplex Method by cross-verifying the equality of solutions of GILP and the self-implemented Simplex Method. All of the experiments were run inside a Jupyter Notebook[7] both on a local computer and by using Google Cloud Computing Services[8] for larger inputs. A link to the code repository is provided in Appendix I.

Three practical experiments are designed to evaluate the Simplex Methods performance across various input distributions. Each of the experiment involves different input sizes described by $n_i \times n_v$ where n_i and n_v are the number of inequalities and variables in an input. For the experiments n_i and n_v are kept as equal to ensure uniformity in the complexity increases. However, it has to be noted that this is not alwasy the case for real-life problems, which often have different numbers of inequalities and variables. The experiments aimed to quantify the Simplex Methods performance by measuring the number of operations needed to solve a specific problem. The results are aggregated

[4]https://www.python.org/
[5]https://numpy.org/
[6]https://matplotlib.org/
[7]https://jupyter.org/
[8]https://cloud.google.com/

over multiple runs to obtain an average result and reduce the effect of outliers. In a number of referenced articles, the number of pivot-rule interactions is used as a metric for performance. This thesis uses the number of operations as it allows to be more precise for the operations required to implement the pivoting rule. Detailed descriptions of the experiment design and data distributions are available in Section 3.1 and Section 3.2.

3.1 Design of Experiments

All experiments in this thesis are based on a self-implemented Simplex Method for maximization which follows the pseudocode provided in Algorithm 1. The second experiment uses a variation of the method by using a optimized pivoting mechanism that skips zero values for better efficiency on sparse matrices. All implementations track the number of operations, which are counted in five categories: comparisons, assignments, arithmetic operations, accesses and function calls. Performance in Python can be measured by tracking time between operations or by using a more detailed profiling tool like cProfile[9] which returns how long and how often certain parts of the algorithm were run. A custom solution was preferred for this thesis, since it allows for more control and predictability on counting the operations. Using the number of operations as a metric is more beneficial than time spent, as it does not take computational resources into account.

The setup for the experiments is contained in the Jupyter Notebook named *thesis_experiments.ipynb* which has the necessary experiment code, Simplex Method implementation in *simplex.py* and *simplex_with_counts.py*, the latter being used for the experiments. The file *simplex_with_counts.py* has the same implementation of the Simplex Method, but with added operation counting variables, discussed in more detailed in the section about Design of Experiments. The codebase has a directory named *Utilities*

[9] https://docs.python.org/3/library/profile.html

which contains all functions needed to generate the inputs. The contents of the codebase can be accessed via the link provided in Appendix 5 with further details available in the *README* file.

Each experiment is configurable and allows the input sizes, types and number of iterations on each input size to be specified. The experiments repeat counting of operations on each size and type of input 2000 times to ensure statistical accuracy. Different default iteration numbers were tested in batches against a benchmark by averaging operation counts over 100,000 iterations. The results of the experiments on choosing the number of iterations are presented in the Appendix, Table 2. Each experiment also allows to specify the range for values of variables and coefficients. The default range of inequalities and variables used for all experiments is between 1 to 1,000,000.

3.2 Data Generation for the Experiments

Inputs used for the experiments are generated algorithmically, with a Python function implemented for each input type. The input generation functions use randomness, but employ some boundaries to maintain a structure needed for a specific type of input. The inputs are returned as 3-tuples A, b, c, where A represents objective function coefficients, b is a matrix of inequalities, and c contains inequality values.

Different input types are chosen, to feature both structures found in real-life problems (Top-Zero, Gaussian) and irregular distributions (Geometric, Linear). These types include:

- **Random** - generates inputs randomly in a set range
- **Symmetric** - generates input that follows a pattern where every second inequality reverses the sign of a chosen variable, maintaining the same absolute value from the previous inequality

- **Geometric varied** - generates input such that the variables for each inequality are defined by geometric progression with added variance in a set range
- **Linear varied** - variables for each inequality are defined by linear progression with added variance
- **Prime numbers** - generates input such that each variable and inequality value was chosen randomly from all prime numbers in a range
- **Pseudoprimes** - generates input such that the variables are chosen from "pseudoprimes" in a set range. Pseudoprimes are defined as integers where the difference between two adjacent numbers increases as the numbers do. This helps simulate characteristics of prime numbers without needing to use exceedingly large primes.
- **Gaussian distribution** - generates input such that each variable and inequality was calculated based on a preset mean value for 97% of the values to be within three standard deviations from the mean.
- **Sparse distributions** - generated using the same function as random inputs, but a number of variables based on the sparsity parameter is set to zero for each input
- **Top-Zero distributions** - generated using random input generation, with all values above the top diagonal of the inequality matrix set as zero.

Examples of different input configurations are visualized in Figure 3, showing linear problems with the size $n_i \times n_v = 32 \times 32$. Each row in a matrix represents an inequality constraint with each pixel in the row representing a variable. The values of variables are shown as colors with darker colors for larger variable values and white pixels indicating that the value of a variable is 0 (variable is absent from the inequality).

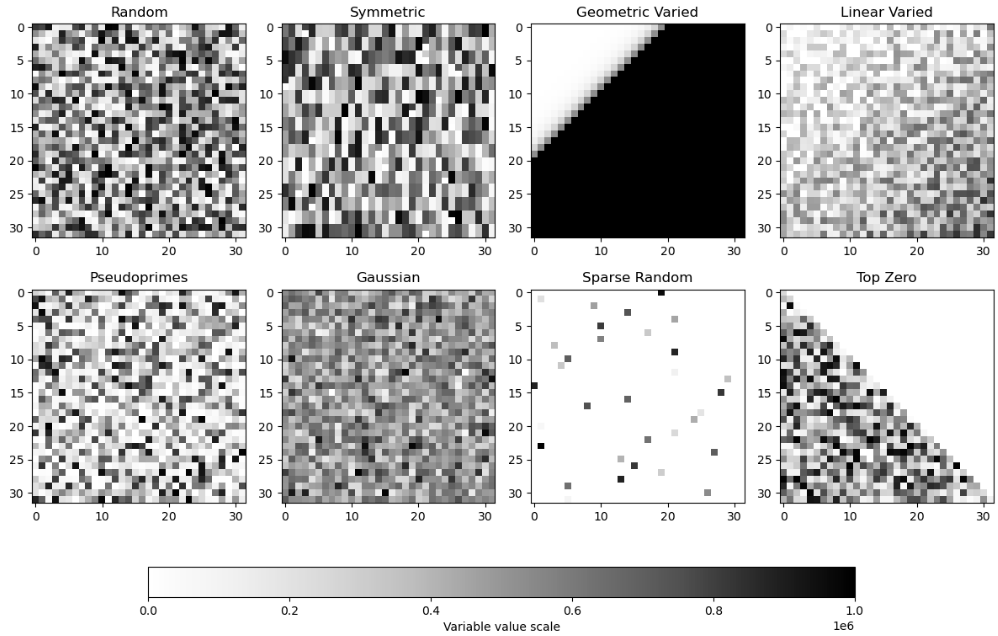

Figure 3. Visualization of Data Distributions

3.3 Experiment 1 - Effects of Data Distribution

Experiment 1 quantifies the effects of data distributions on unoptimized Simplex Methods performance, which addresses Research Question 1 (**RQ1**): "How consistent is the performance of Simplex Method across different types of input data distributions?" This experiment utilizes all of the mentioned input types and creates a benchmark for comparison.

Inputs are generated to cover small to medium problem sizes with the number of inequalities and variables increasing incrementally from 10 to 30. Larger inputs up to $n_i \times n_v = 1000 \times 1000$ are also tested with a smaller number of iterations per input to ensure the results scale. The results are then plotted to a line graph for comparison and saved to a text file for more detailed analysis. Results for both experiment runs are covered in the Results Section.

3.4 Experiment 2 - Effects of Optimizing the Simplex Method

Experiment 2 quantifies the impacts on relative performance using a simple optimization method. The Simplex Methods pivoting mechanism is altered to bypass zero elements to reduce number of operations required on sparse matrices. The experiment aims to answer Research Question 2 (**RQ2**): “What are the impacts of simple optimization techniques on the computational expensiveness of the Simplex Method?”. Three different inputs are utilized: Random, Sparse Random with 50% sparsity and Top-Zero. The input choice was made to include a structured input (Top-Zero), a sparse input with similar sparsity as Top-Zero (Sparse Random with 50% sparsity) and a non-sparse input (Random).

The efficiency of Zero-Exploiting Simplex Method (ZESM) is compared with the standard Simplex Method (SM) across the inputs to measure the reduction in the number of operations to reach a solution. The solutions of ZESM and SM are then compared to ensure validity and an error is raised if they deviate more than 0.001. The error trigger was chosen to account for minor rounding differences in the pivoting rules between two methods. During the running of the experiments, if the error exceeds the threshold a warning is printed to console and the iteration is excluded from the calculation of average operations. Additionally, after each experiment the number of errors is manually assessed to ensure accuracy of the methods. This experiment covers input sizes from 10 to 30 inequalities and variables over 2000 iterations, while larger inputs of 100 to 1000 inequalities and variables are evaluated over 30 iterations.

3.5 Experiment 3 - Effects of Input Preprocessing

Experiment 3 explores the effect of input preprocessing on the performance of the Zero-Exploiting Simplex Method and addresses Research Question 3 (**RQ3**): "How does input preprocessing affect the performance of the Simplex Method?". The experiment uses a generally well-performing Top-Zero input and shuffles its columns and rows to break the structure while maintaining the correctness of inequalities.

Four variations of the input are compared: Baseline Top-Zero, Rows (Order of inequalities) Shuffled, Columns (order of variables) Shuffled and both Rows and Columns Shuffled. The experiment was run on inputs with 10 to 30 inequalities and variables and after each run the values returned are compared. If the solutions differed by more than 0.001, an error was raised to ensure validity of the results and to make sure shuffling did not change the original problem.

Figure 4 is a visualization of $n_i \times n_v = 64 \times 64$ Top-Zero inputs, that have been shuffled as they would be in the third experiment. A visualization of a Sparse Random input with 50% sparsity is presented for comparison.

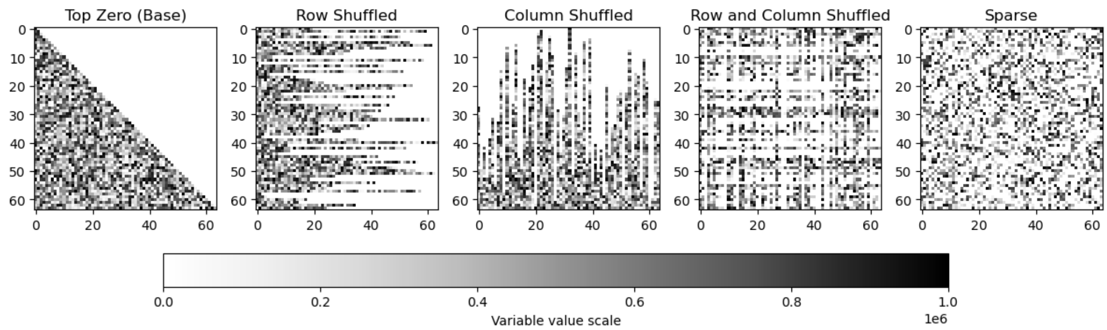

Figure 4. Visualization of Top-Zero Inputs After Shuffling

4 Results and Discussion

This section presents the results of three experiments which are designed to quantify the computational effectiveness of the Simplex Method. The findings discussed will also address the Research Questions outlined in the introduction of this thesis.

4.1 Experiment 1

The first experiment explored the impact of different data distributions on the Simplex Method's operation count, with results visualized in Figure 5. The experiment uses incrementally increasing input sizes ranging from 10 to 30 inequalities and variables. The experiment was repeated for input sizes ranging from 100 to 1000 inequalities to see how the results scale. The recorded results establish a benchmark for comparing the Simplex Method's performance across various input types.

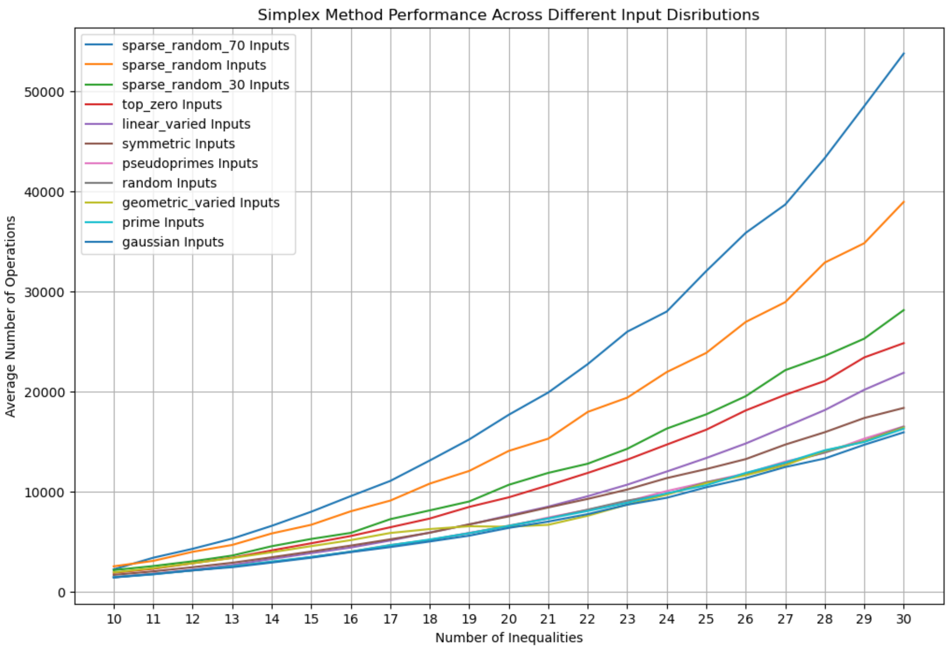

Figure 5. Visualization of Average Number of Operations over 10 to 30 Inequalities and Variables: Results from Experiment 1

Figure 5 shows a line graph visualizing the average number of operations required as the number of inequalities and variables increase. The experiment used 9 different input types with three levels of sparsity which are color-coded and shown in the legend. All variables and coefficients were within 1 to 1,000,000 for this experiment. Detailed data from the experiment is provided in the Appendix, in Table 3.

The results show that the inputs with varying levels of sparsity required the most operations to be solved. The number of operations for Sparse Random inputs with 30%, 50% and 70% sparsity and Sparse Random was between 2,135 to 28,126; 2,512 to 38,941; and 2,210 to 53,767 operations, respectively. However, Top-Zero inputs, which are also sparse consistently required fewer operations and indicated some efficiency in structured sparse distributions. The average number of operations needed for Top-Zero input was between 1,871 to 24,817.

Non-Sparse inputs such as Random, Geometric Varied, Prime numbers, Gaussian and Pseudoprimes performed relatively similarly, with Gaussian distribution seeming to perform slightly better with an average of 15,908 operations for a $n_i \times n_v = 30 \times 30$ input. Symmetrical inputs needed on average from 1,662 to 18,346 operations and Linear Varied inputs from 1,434 to 21,859 operations.

An interesting trend can be noticed with Geometric Varied inputs. At smaller problem sizes it tends to perform slightly worse than Random or Gaussian inputs, while the performance is similar for larger inputs. There is a notable drop in the number of operations required to solve a Geometric Varied input, presumably due to the simplification of calculations as the values converged to the maximum allowed value for inequality variables. This can be attributed to the generation function of the Geometric Varied input, which uses a geometric progression to set a value for variables. However, as the problem size increased, the average number also rose reflecting the increasing size of the matrix.

The results confirm that the Simplex Method's performance fluctuates significantly on input distributions. The research findings indicate a strong need for an algorithm that exploits sparsity. The sparse inputs require more operations than non-sparse. The benefits of optimizing the Simplex Method for sparse matrices are discussed under the results for Experiment 2. The experiment was repeated on inputs from 100 to 1000 inequalities and variables to see if the same patterns for an average number of operations hold across larger inputs. The results are presented as a line graph in Figure 6 and as a table for detailed operation counts across input sizes in the Appendix, Table 4.

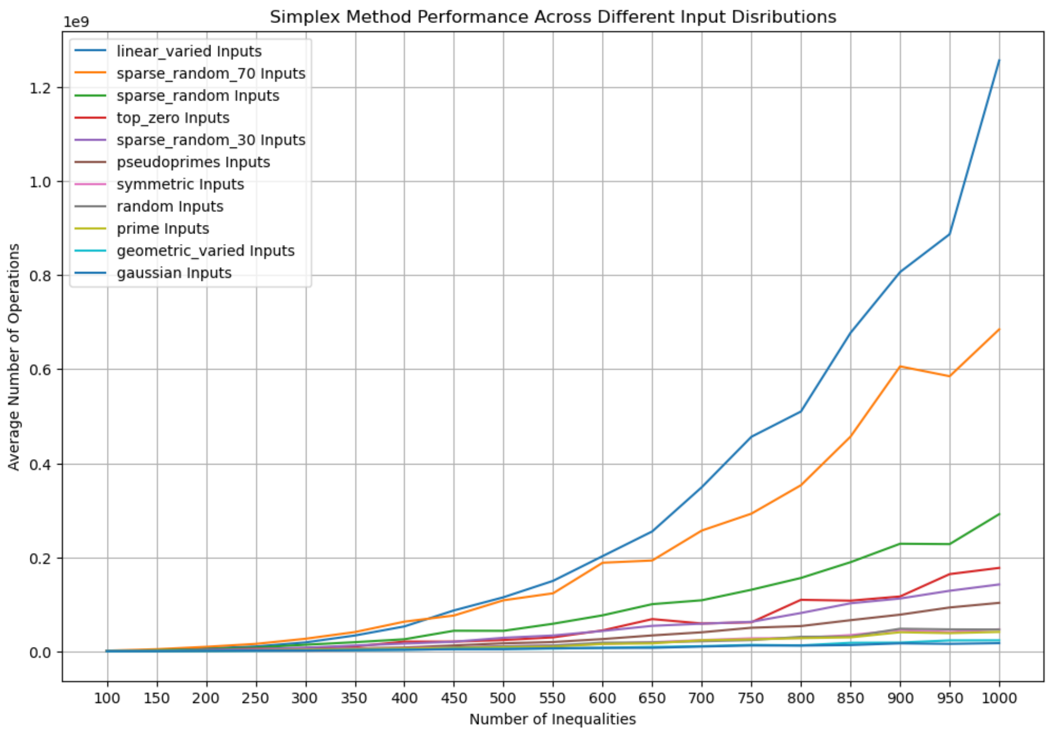

Figure 6. Visualization of Average Number of Operations over 100 to 1000 Inequalities and Variables: Results from Experiment 1

Interestingly, based on the larger experiment, Linear inputs seemed to perform worse than with smaller inputs. The average number of operations for solving a Linear Varied input for 100 and 1000 inequalities and variables was 797,499 and 1,256,456,290, respectively. The sparse inputs with 70%, 50% and 30% of sparsity gave comparable results to the smaller experiment, with a larger rate of sparsity requiring on average more operations to solve. Top-Zero inputs have similar average number of operations as Sparse Random 30%, which was the case for the experiment with smaller inputs as well.

From the non-sparse inputs, Pseudoprime inputs performed slightly worse than others, which was not the case for smaller inputs, where the performance was similar for all sparse input types. For inputs ranging from 100 to 1000 inequalities and variables, the Pseudoprime inputs required an average of 258,619 to 103,103,613 operations to be solved. For 100 to 1000 inequalities and variables, the Symmetric, Random, Prime number, and Gaussian input needed an average of 290,400 to 46,953,618; 276,150 to 46,418,955; 243,296 to 41,285,171; and 218,089 to 17,434,350 operations, respectively. The relatively higher count for Pseudoprime inputs suggests, the Simplex Method performs better on uniformly distributed values. This is further confirmed by the lower number of operations required for Gaussian inputs, which are uniform around average values.

In summary, the first experiment addressed **RQ1** by empirically showing that the unoptimized Simplex Method is more effective on non-sparse inputs across various sizes, from $n_i \times n_v = 10 \times 10$ to $n_i \times n_v = 1000 \times 1000$. The larger inputs highlighted performance disparities between input types and provide some unexpected results. For instance, the Simplex Method performed better on uniformly distributed Gaussian inputs and worse on Pseudoprime inputs. The results confirm that input distributions have a large impact on the performance of the Simplex Method which can be measured on different input sizes.

4.2 Experiment 2

The second experiment evaluated the performance benefits of using an optimized Simplex Method on sparse matrices with the results visualized in Figure 7. The experiment includes Random, Sparse Random, and Top-Zero inputs and incrementally increasing input sizes ranging from 10 to 30 inequalities and variables. The average operation counts are plotted for unoptimized Simplex Method (SM) and Zero-Exploiting Simplex-Method (ZESM) for comparison. The experiment was repeated on inputs ranging from 100 to 1000 inequalities and variables to verify that the results scale. Average operation counts for ZESM are marked with the `_exploit` suffix in the legend. Notably, only three cases of differing output values were observed out of nearly 50,000 solves. All of these differences were less than 0.001 and can be attributed to rounding inaccuracies. Data recorded during the experiment can be accessed in the Appendix, in Table 5, Table 6, Table 7 and Table 8.

For Random inputs, which are not inherently sparse, ZESM showed to be more efficient. At a size of $n_i \times n_v = 10 \times 10$, ZESM required 784 operations on average, compared to 1,418 operations needed for SM, meaning a 44.71% decrease in the number of operations needed. The decrease in number of operations is more pronounced for larger inputs. At $n_i \times n_v = 30 \times 30$, ZESM required an average of 7,735 operations and SM an average of 16,492 operations, marking a 53.1% decrease. While Random inputs are not sparse, some elements are set as zeroes during the pivoting phase of the algorithm; for the following steps of the algorithm, this could be beneficial, as not all elements need to be evaluated.

Sparse Random inputs also showed similar tendencies, as ZESM showed to be more efficient in solving linear problems than SM. This decrease in the number of operations was consistent over various problem sizes. For size $n_i \times n_v = 10 \times 10$, SM needed 2,526

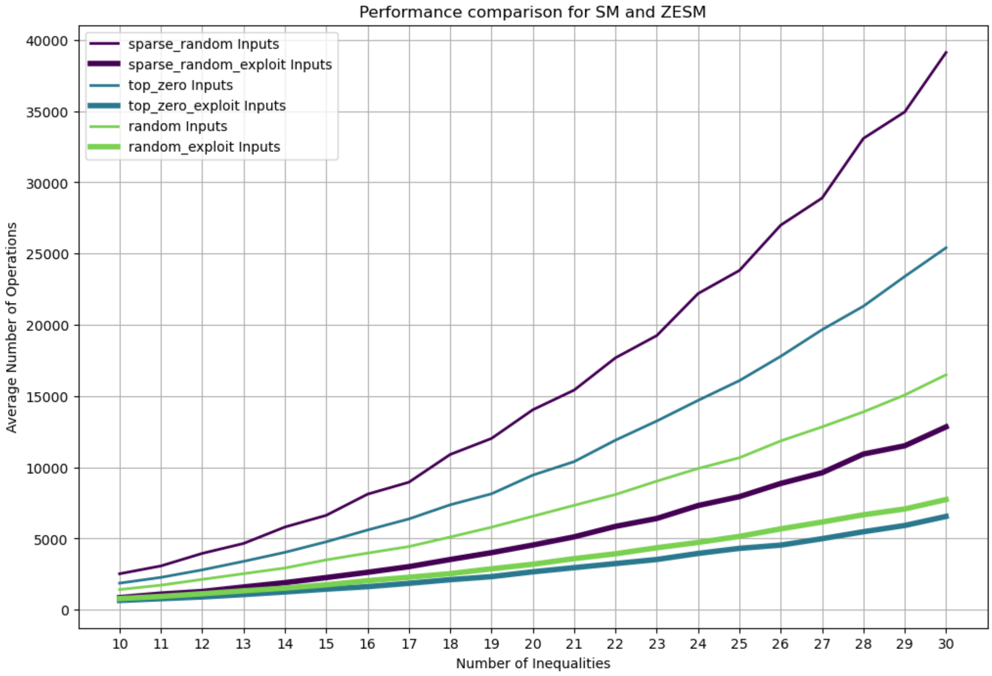

Figure 7. Visualization of Average Number of Operations over 10 to 30 Inequalities and Variables: Results from Experiment 2

operations, compared to 848 for ZESM, marking a 66.43% decrease in the number of operations. For $n_i \times n_v = 30 \times 30$ size inputs, SM needed 39,112 operations, compared to 12,833 for ZESM, marking a 67.19% decrease. The average reduction in the number of operations for Sparse Random inputs was 66.77% across inputs from 10 to 30 inequalities and variables. The consistent performance benefit across input sizes can most likely be attributed to the method's handling of sparsity, as sparsity rate for every input was consistently 50% and the primary variable for increasing operations count was the matrix size.

Top-Zero inputs displayed the largest decrease in the number of operations needed when using ZESM over SM. The relative decrease in number of operations ranged from 65.13% for inputs of 10 inequalities and variables to 74.22% for 30 inequalities and variables. The average decrease in the number of operations was 71.39%. For size $n_i \times n_v = 10 \times 10$, SM needed 1,867 operations, compared to 651 for ZESM. For larger, $n_i \times n_v = 30 \times 30$ inputs, SM needed 25,406 operations, compared to 6,549 for ZESM. The higher performance benefit is because of the structure of Top-Zero algorithm, as the Simplex Method does not need to traverse the full matrix.

This experiment was repeated on inputs with 100 to 1000 inequalities and variables for 30 iterations to see how larger inputs affect the performance of ZESM compared to SM. The results are displayed in Figure 8.

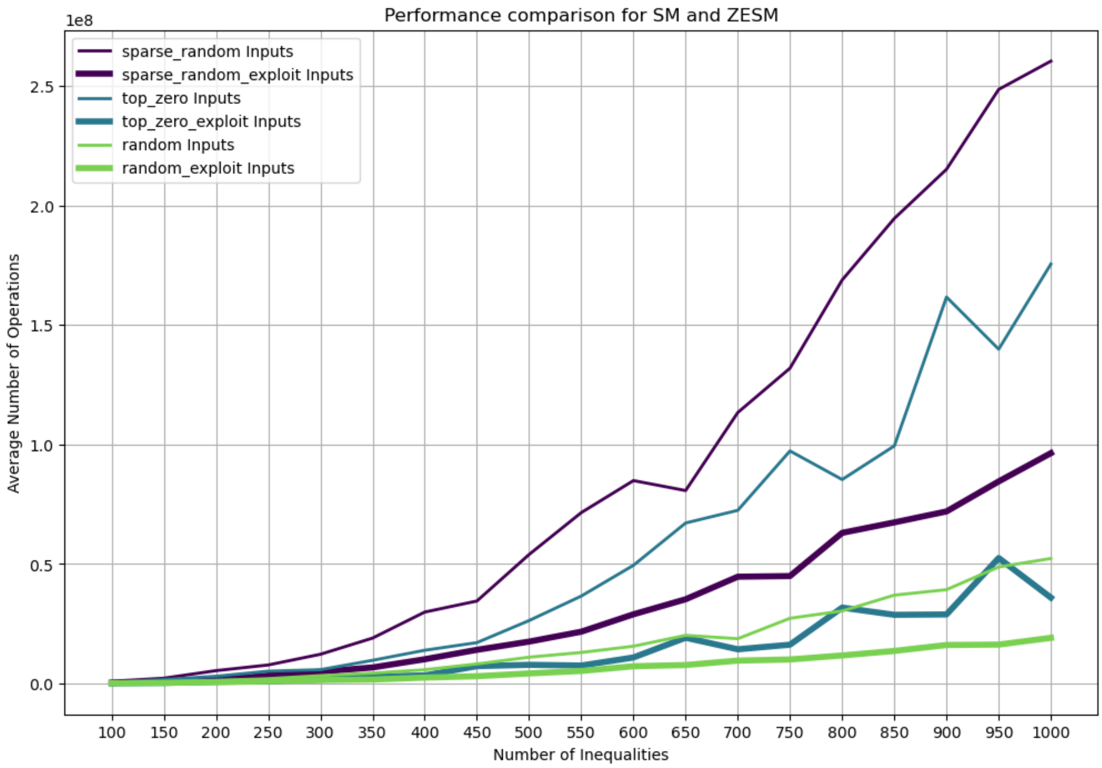

Figure 8. Visualization of Average Number of Operations over 100 to 1000 Inequalities and Variables: Results from Experiment 2

Figure 8 displays a pattern similar to the one observed for smaller inputs. For Random inputs, the average decrease in the number of operations was 58.67%, a little more than observed with 30 inequalities and variables. For Sparse Random inputs the average decrease was similar to one observed before, 64,8%. Number of operations decreased by 70.12% on average for Top-Zero inputs, however, it's worth noting that due to the smaller number of iterations, the accuracy of the averages could be improved.

The graphs are not completely similar as a divergence can be seen for ZESM on Top-Zero and Random inputs. For 10 to 30 inequalities and variables, Top-Zero input required fewer operations when using ZESM, presumably thanks to the optimized methods ability to exploit sparse inputs. This Was not the case for 100 to 1000 inequalities and variables, as the Top-Zero input frequently required more operations than Random input when solved with ZESM. The results suggest that the experiment could benefit from being run again with larger number of iterations for increased accuracy. However, if the results were to persist on a larger number of iterations it would mean that the optimization is less effective on larger inputs.

In conclusion, Experiment 2 has demonstrated performance improvements achieved by using an optimized version of the Simplex Method. This method works particualrily well for sparse matrices but has benefits for non-sparse ones as well. The experiment answers **RQ2** by stating that the number of operations of the Simplex Method is reduced by over 50% when using ZESM over SM. Due to the nature of the experiments, results for smaller inputs of 10 to 30 inequalities and variables were different from the ones found for 100 to 1000 inequalities. This can be improved by running the experiments again for a larger number of iterations and comparing the results again.

4.3 Experiment 3

The third experiment evaluated the effects of shuffling structured inputs on the efficiency of the Zero-Exploiting Simplex Method (ZESM). ZESM was chosen for its better performance on sparse inputs. The experiment compared four variations of Top-Zero inputs: an unshuffled baseline, Top-Zero with rows shuffled, Top-Zero with columns shuffled and Top-Zero with both rows and columns shuffled. The experiment is run on inputs from 10 to 30 inequalities and variables, with larger inputs from 100 to 1000 inequalities tested over 30 iterations to assess the scalability of the results. Figure 9 demonstrates the results from Experiment 3. Detailed data about the experiment iterations and average operation counts is presented in the Appendix, Table 9.

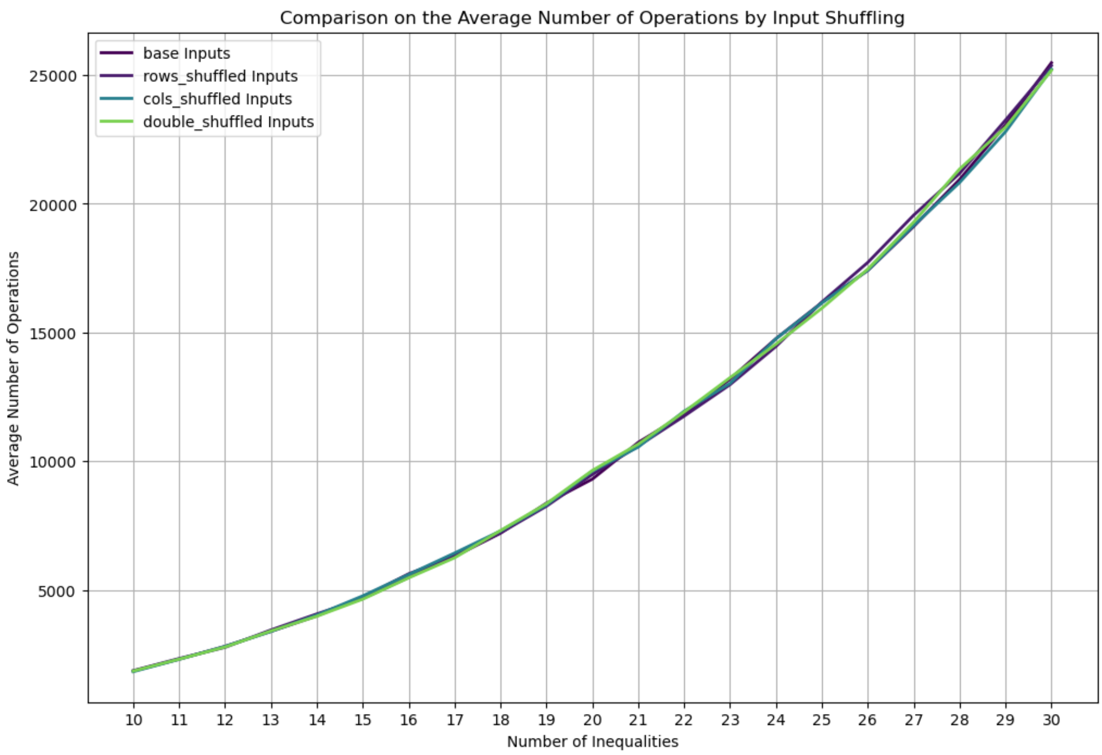

Figure 9. Visualization of Average Number of Operations over 10 to 30 Inequalities and Variables: Results from Experiment 3

For size $n_i \times n_v = 10 \times 10$, baseline Top-Zero input, Rows shuffled inputs, Columns shuffled input, and Rows and Columns shuffled input needed an average of 1,848; 1,824; 1,862; and 1,850 operations, respectively. For larger, size $n_i \times n_v = 30 \times 30$ the inputs needed 25,480; 25,228; 25,371; and 25,184 operations, respectively. The operation counts for all three versions of shuffling, as well as the baseline deviate about 1% from each other and visually form one line. The small deviations in the average number of operations that can be seen arise from experiment design. For this experiment, a new input is generated for every iteration and shuffled, which means that all types of shuffling input take a different Top-Zero input as a base.

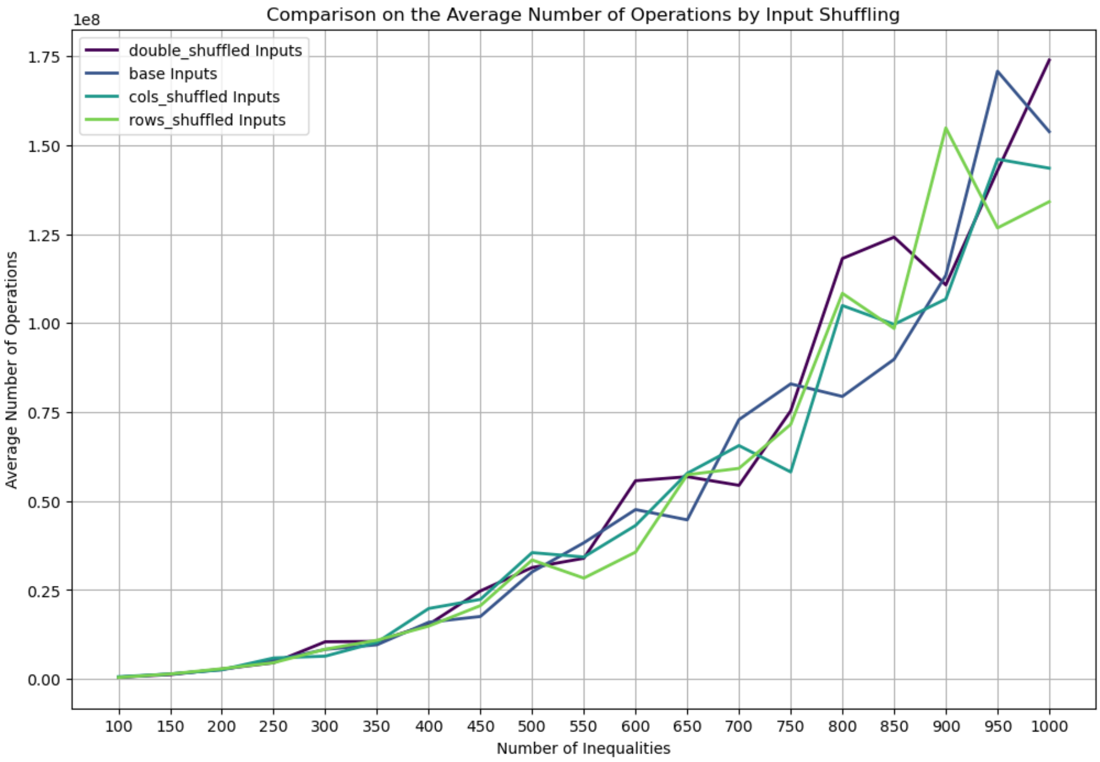

Figure 10. Visualization of Average Number of Operations over 100 to 1000 Inequalities and Variables: Results from Experiment 3

Larger inputs were tested for 30 iterations with the results visualized in Figure 10. Data about average operations based on the number of inequalities and variables for this run is presented in the Appendix, Table 10. The number of inequalities and variables ranged from 100 to 1000 and the experiment was repeated for 30 times for each size of input. The results are similar, though they do have more variance since the experiment was run only 30 times, instead of the usual 2000. For inputs with 100 inequalities and variables, the baseline Top-Zero required an average of 459,887 operations and Columns shuffled, Rows shuffled and both Rows and Columns shuffled require an average of 614,286; 430,305; and 486,156 operations. Larger inputs with 1000 inequalities and variables needed an average of 153,792,091; 143,524,583; 134,113,421; and 173,898,153 for baseline, Columns shuffled, Rows shuffled and both Rows and Columns shuffled, respectively.

While the results from Experiment 3 were somewhat disappointing they follow expectations. Since the implementation of the ZESM does not account for specific data structures except for sparsity it can not effectively exploit such structures. Additionally, any negligible performance benefits for this experiment do not take into account the operations needed for preprocessing. A simple example shows that if preprocessing requires iterating over all elements in the matrix, then the complexity of those operations alone is $O(n^2)$.

5 Conclusion

The primary objective of this thesis was to look beyond the traditional worst-case analysis of the Simplex Method and get a better understanding on how different input distributions affect the performance. The empirical research was based on three experiments which evaluated the performance of the method on different inputs and by using an optimized pivoting rule for the Zero-Expoliting Simplex Method (ZESM).

The experiments revealed that the distribution of input data significantly affects the number of operations required by the Simplex Method, with structured sparse matrices performing better for all tested input sizes. Additionally, the implementation of ZESM demonstrated a decrease in number of operations across all tested input distributions and worked particularly well on sparse distributions. The thesis addressed preprocessing of the inputs, but found no improvements by changing the order of inequalities and variables.

The findings in this thesis confirm the Simplex Method is directly linked to distribution of input data. For future research, it would be beneficial to replicate the experiments with larger input sizes over more iterations. Another promising field of research would involve optimizing the Simplex Method using a machine learning model as demonstrated by Adham et al. [16], with a focus on adapting the pivot rule selection to the input data distribution.

In conclusion, this thesis serves as a baseline for further research on the effects of input distribution and has succesfully addressed the posed research questions. The framework for experiments that was developed for this thesis is available in Github and can be used for further analysis on the Simplex Method.

References

[1] J. Dongarra and F. Sullivan. Guest editors introduction to the top 10 algorithms. *Computing in Science & Engineering*, 2(01):22–23, jan 2000.

[2] Henry W. Robbins, Samuel C. Gutekunst, David B. Shmoys, and David P. Williamson. Gilp: An interactive tool for visualizing the simplex algorithm. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, SIGCSE 2023, pages 108–114. Association for Computing Machinery, 2023.

[3] Ian Chivers and Jane Sleightholme. *An Introduction to Algorithms and the Big O Notation*, pages 359–364. Springer International Publishing, Cham, 2015.

[4] Victor Klee and George J. Minty. How good is the simplex algorithm. Technical report, Sep 2020.

[5] Karl Heinz Borgwardt. The average number of pivot steps required by the simplex-method is polynomial. *Zeitschrift für Operations Research*, 26:157–177, 1982.

[6] Steve Smale. On the average number of steps of the simplex method of linear programming. *Math. Program.*, 27(3):241–262, oct 1983.

[7] Daniel A. Spielman and Shang-Hua Teng. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time, 2003.

[8] Richard M. Karp. George dantzig's impact on the theory of computation. *Discrete Optimization*, 5(2):174–185, 2008. In Memory of George B. Dantzig.

[9] George B. Dantzig. *Origins of the simplex method*, page 141–151. Association for Computing Machinery, New York, NY, USA, 1990.

[10] Daniel Dadush and Sophie Huiberts. *Smoothed Analysis of the Simplex Method*, page 309–333. Cambridge University Press, 2021.

[11] George B. Dantzig. Linear programming. *Applied Mathematics*, 15:18–21, 1951.

[12] Jonathan A. Kelner and Daniel A. Spielman. A randomized polynomial-time simplex algorithm for linear programming. In *Proceedings of the Thirty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '06, page 51–60, New York, NY, USA, 2006. Association for Computing Machinery.

[13] Harold W Kuhn and Richard E Quandt. *An experimental study of the simplex method*. 1962.

[14] Norman Zadeh. What is the worst case behavior of the simplex algorithm. 1980.

[15] George B. Dantzig. Expected number of steps of the simplex method for a linear program with a convexity constraint. 1980.

[16] Imran Adham, Jesus De Loera, and Zhenyang Zhang. (machine) learning to improve the empirical performance of discrete algorithms, 2021.

Appendix

I. Code Repository

The Python code and Jupyter Notebooks used to run the experiments, visualize data and generate inputs is available from GitHub: `https://github.com/mihkeluutar/simplex-practical-experiments`

II. Tables

Table 2. Deviation of Operation Counts for Random Input from a Benchmark of 100,000 Iterations

Inequalities/Iterations	100	300	2000	10000	20000	50000	100000
2	1.39%	0.04%	0.61%	0.04%	0.16%	0.08%	0.00%
3	2.39%	4.08%	0.51%	0.50%	0.23%	0.04%	0.00%
4	1.23%	0.30%	1.60%	0.19%	0.04%	0.13%	0.00%
5	5.25%	2.21%	1.02%	0.58%	0.45%	0.16%	0.00%
6	1.18%	1.19%	0.58%	0.21%	0.37%	0.31%	0.00%
7	0.19%	1.15%	0.45%	0.30%	0.11%	0.09%	0.00%
8	2.66%	4.77%	0.50%	0.45%	0.23%	0.04%	0.00%
9	3.74%	1.23%	0.31%	0.65%	0.66%	0.14%	0.00%
Average Deviation	2.26%	1.87%	0.70%	0.37%	0.28%	0.12%	0.00%

Table 3. Detailed Data Overview for Experiment 1 (10-30 Inequalities and Variables)

Ineq. No.	Symm.	Rn.	Geo. Var.	Lin. Var.	Prm.	Gaus.	Pseu.	Sprs. Rn. 30	Sprs. Rn. 70	Sprs. Rn. 50	TZ
10	1662	1407	1906	1434	1425	1400	1404	2135	2210	2512	1871
11	2009	1771	2322	1754	1722	1717	1787	2540	3370	3055	2266
12	2428	2096	2819	2179	2112	2113	2105	3014	4264	3965	2817
13	2867	2511	3339	2643	2501	2424	2497	3587	5280	4649	3365
14	3418	2945	3921	3249	2984	2889	2943	4521	6557	5795	4103
15	3991	3423	4527	3827	3446	3370	3447	5248	7968	6662	4816
16	4583	3979	5128	4388	3973	3931	3952	5853	9528	8010	5541
17	5210	4639	5842	5090	4618	4432	4512	7223	11044	9079	6414
18	5868	5146	6237	5886	5192	4991	5150	8102	13088	10777	7286
19	6711	5825	6519	6684	5823	5577	5854	8993	15195	12045	8451
20	7500	6578	6466	7594	6545	6351	6510	10657	17658	14048	9407
21	8395	7310	6662	8479	7286	6986	7391	11849	19883	15269	10603
22	9249	8179	7566	9518	8043	7727	8156	12768	22722	17944	11846
23	10187	9083	8682	10671	8872	8666	9002	14259	25962	19369	13171
24	11329	9711	9684	11985	9808	9362	10033	16276	27976	21927	14676
25	12233	10915	10821	13337	10616	10409	10924	17703	32023	23841	16165
26	13228	11680	11622	14788	11847	11313	11788	19518	35842	26939	18095
27	14678	12807	12618	16449	12882	12437	12979	22117	38670	28917	19655
28	15915	13893	14056	18127	14107	13290	13861	23539	43317	32871	21033
29	17337	15044	14999	20156	14935	14661	15272	25269	48512	34806	23392
30	18346	16468	16357	21859	16264	15908	16494	28126	53767	38941	24817

Table 4. Detailed Data Overview for Experiment 1 (100-1000 Inequalities and Variables)

Ineq. No.	Sym.	Rand.	Geo. V.	Lin. V.	Prm.	Gaus.	Psdp.	Sprs. Rn. 30	Sprs. Rn. 70	Sprs. Rn.	TZ
100	290400	276150	209350	797499	243296	218089	258619	562017	1425962	783198	399655
150	747742	735533	474134	2181640	596275	383704	696427	1505031	4497152	2281749	1218709
200	1427152	1189330	713694	6230704	1081234	1055273	1530915	3000993	9628817	4440673	2325787
250	2493459	2170028	1118816	10403715	2055492	1577020	2655186	5067378	15860996	8031939	5463892
300	3506675	3032054	1753518	18906321	3003015	1598487	3952222	7816762	26693090	14063754	6827507
350	5675311	4687840	2383615	33653544	4490353	2620593	6360053	12443000	40934161	19473615	10295267
400	6563719	5361029	3436757	52984942	5876472	3264835	8127160	16494336	63104299	25668384	20701354
450	8235791	8214098	4672219	86999381	7105899	4650454	12581538	20468887	76178731	43783038	20705380
500	10912200	10563709	5228469	115010466	8633407	4585006	17105261	28820863	108549739	43619151	23992410
550	12548847	12354286	5869384	149892410	9857659	6679966	19844226	33689102	123402737	58687090	29697862
600	15892407	18438090	7522141	202457470	15776665	7213605	26075179	43355098	188418231	76602682	44547219
650	19368957	19052250	9322721	254947271	16970588	7422133	33849744	54213080	193134260	100324448	68553673
700	23978405	21407449	10966489	349053982	23611092	10126881	40505393	58763416	256820464	108606018	59231413
750	27699754	23845974	13790126	456175040	25411595	12405076	50159744	62744211	292697417	130966001	61776416
800	28495301	30961174	12672651	509938009	28015775	12193111	53701493	81510314	353225213	156031828	109587328
850	34558601	31388937	18246215	677142459	29688140	13375581	66255653	102126892	456507178	189562457	107764534
900	45323099	48096134	18632417	806504260	40643586	16985905	77992828	112308684	605899542	228774212	116635575
950	41126732	46725971	23267248	886977765	38713172	15929922	93257390	128783170	584917502	228022463	164205113
1000	46953618	46418955	23423880	1256456290	41285171	17434350	103103613	142247793	684701759	291765282	177427721

Table 5. Detailed Data Overview for Experiment 2 (10-30 Inequalities and Variables)

Ineq. No.	Rand. (SM)	Sprs. Rn. 50 (SM)	TZ (SM)	Rand. (ZESM)	Sprs. Rn. 50 (ZESM)	TZ (ZESM)
10	1418	2526	1867	784	848	651
11	1724	3074	2271	935	1098	777
12	2131	3955	2799	1124	1280	899
13	2531	4652	3396	1329	1599	1065
14	2928	5805	4029	1519	1901	1248
15	3499	6626	4768	1752	2258	1445
16	3974	8111	5595	2038	2627	1620
17	4434	8949	6372	2278	3025	1855
18	5103	10896	7364	2532	3526	2110
19	5792	12023	8137	2874	4005	2325
20	6560	14035	9448	3192	4542	2661
21	7328	15416	10396	3589	5117	2954
22	8086	17680	11902	3927	5851	3240
23	9025	19243	13246	4351	6406	3527
24	9913	22192	14694	4725	7314	3952
25	10669	23813	16082	5149	7938	4308
26	11849	26994	17788	5677	8864	4536
27	12833	28893	19655	6156	9614	4988
28	13886	33080	21304	6666	10928	5475
29	15072	34939	23390	7072	11507	5914
30	16492	39112	25406	7735	12833	6549

Table 6. Detailed Data Overview for Experiment 2 with Comparison of SM and ZESM Operations (10-30 Inequalities and Variables)

Inequalities	Rand. (SM)	Rand. (ZESM)	Difference	Sprs. Rn. 50 (SM)	Sprs. Rn. 50 (ZESM)	Difference	TZ (SM)	TZ (ZESM)	Difference
10	1418	784	44.71%	2526	848	66.43%	1867	651	65.13%
11	1724	935	45.77%	3074	1098	64.28%	2271	777	65.79%
12	2131	1124	47.25%	3955	1280	67.64%	2799	899	67.88%
13	2531	1329	47.49%	4652	1599	65.63%	3396	1065	68.64%
14	2928	1519	48.12%	5805	1901	67.25%	4029	1248	69.02%
15	3499	1752	49.93%	6626	2258	65.92%	4768	1445	69.69%
16	3974	2038	48.72%	8111	2627	67.61%	5595	1620	71.05%
17	4434	2278	48.62%	8949	3025	66.20%	6372	1855	70.89%
18	5103	2532	50.38%	10896	3526	67.64%	7364	2110	71.35%
19	5792	2874	50.38%	12023	4005	66.69%	8137	2325	71.43%
20	6560	3192	51.34%	14035	4542	67.64%	9448	2661	71.84%
21	7328	3589	51.02%	15416	5117	66.81%	10396	2954	71.59%
22	8086	3927	51.43%	17680	5851	66.91%	11902	3240	72.78%
23	9025	4351	51.79%	19243	6406	66.71%	13246	3527	73.37%
24	9913	4725	52.34%	22192	7314	67.04%	14694	3952	73.10%
25	10669	5149	51.74%	23813	7938	66.67%	16082	4308	73.21%
26	11849	5677	52.09%	26994	8864	67.16%	17788	4536	74.50%
27	12833	6156	52.03%	28893	9614	66.73%	19655	4988	74.62%
28	13886	6666	51.99%	33080	10928	66.96%	21304	5475	74.30%
29	15072	7072	53.08%	34939	11507	67.07%	23390	5914	74.72%
30	16492	7735	53.10%	39112	12833	67.19%	25406	6549	74.22%
Average Decrease			50.16%			66.77%			71.39%

Table 7. Detailed Data Overview for Experiment 2 (100-1000 Inequalities and Variables)

Ineq. No.	Rand. (SM)	Sprs. Rn. 50 (SM)	TZ (SM)	Rand. (ZESM)	Sprs. Rn. 50 (ZESM)	TZ (ZESM)
100	241649	934916	525577	94525	273713	84619
150	630477	2389242	1399466	251179	721498	474460
200	1180689	4870902	1809123	615450	1603257	752773
250	2304795	9581708	5618830	764816	2869115	731714
300	2978775	13802253	5955864	1257035	5484892	2604679
350	3654229	19592099	8708679	1549785	6285210	2439206
400	6546555	25565352	11647342	2974516	9859235	5171042
450	7692575	39893618	25061929	3156117	12709200	6570404
500	10738019	43069547	21190982	5484581	17964934	8031968
550	14397047	67068717	30735600	6629831	22380394	8075707
600	15795871	69833559	35116908	6946050	27085842	13909589
650	21247068	97405599	75273254	8785450	30769995	17603139
700	27467438	105851468	68150621	9234609	39733160	16731224
750	25110464	139275309	69633883	9813435	52676510	20808645
800	26919855	147538452	70272801	10696337	58995311	23136002
850	38617394	173714225	149356241	13423127	67492790	53737069
900	46016493	203687050	116592187	16217255	70775865	29444503
950	52276979	226912510	158268434	18662719	76505265	39156890
1000	45402800	268717211	159139105	22422677	96842924	32314837

Table 8. Detailed Data Overview for Experiment 2 with Comparison of SM and ZESM Operations (100-1000 Inequalities and Variables)

Inequalities	Rand. (SM)	Rand. (ZESM)	Difference	Sprs. Rn. 50 (SM)	Sprs. Rn. 50 (ZESM)	Difference	TZ (SM)	TZ (ZESM)	Difference
100	241649	94525	60.88%	934916	273713	70.72%	525577	84619	83.90%
150	630477	251179	60.16%	2389242	721498	69.80%	1399466	474460	66.10%
200	1180689	615450	47.87%	4870902	1603257	67.09%	1809123	752773	58.39%
250	2304795	764816	66.82%	9581708	2869115	70.06%	5618830	731714	86.98%
300	2978775	1257035	57.80%	13802253	5484892	60.26%	5955864	2604679	56.27%
350	3654229	1549785	57.59%	19592099	6285210	67.92%	8708679	2439206	71.99%
400	6546555	2974516	54.56%	25565352	9859235	61.44%	11647342	5171042	55.60%
450	7692575	3156117	58.97%	39893618	12709200	68.14%	25061929	6570404	73.78%
500	10738019	5484581	48.92%	43069547	17964934	58.29%	21190982	8031968	62.10%
550	14397047	6629831	53.95%	67068717	22380394	66.63%	30735600	8075707	73.73%
600	15795871	6946050	56.03%	69833559	27085842	61.21%	35116908	13909589	60.39%
650	21247068	8785450	58.65%	97405599	30769995	68.41%	75273254	17603139	76.61%
700	27467438	9234609	66.38%	105851468	39733160	62.46%	68150621	16731224	75.45%
750	25110464	9813435	60.92%	139275309	52676510	62.18%	69633883	20808645	70.12%
800	26919855	10696337	60.27%	147538452	58995311	60.01%	70272801	23136002	67.08%
850	38617394	13423127	65.24%	173714225	67492790	61.15%	149356241	53737069	64.02%
900	46016493	16217255	64.76%	203687050	70775865	65.25%	116592187	29444503	74.75%
950	52276979	18662719	64.30%	226912510	76505265	66.28%	158268434	39156890	75.26%
1000	45402800	22422677	50.61%	268717211	96842924	63.96%	159139105	32314837	79.69%
Average			58.67%			64.80%			70.12%

Table 9. Detailed Data Overview for Experiment 3 (10-30 Inequalities and Variables)

Ineq. No.	Base (Random Top-Zero)	Columns Shuffled	Rows Shuffled	Rows and Columns Shuffled
10	1848	1824	1862	1850
11	2306	2303	2333	2310
12	2806	2804	2768	2780
13	3388	3384	3443	3405
14	3990	4024	4066	3976
15	4716	4767	4663	4641
16	5623	5598	5486	5468
17	6299	6431	6350	6247
18	7202	7291	7218	7305
19	8369	8301	8251	8349
20	9303	9615	9484	9640
21	10729	10549	10592	10663
22	11818	11943	11752	11912
23	13191	13024	12975	13246
24	14761	14757	14459	14547
25	16163	16149	16182	15944
26	17412	17417	17721	17466
27	19120	19156	19559	19289
28	20945	20825	21181	21351
29	23066	22802	23253	22976
30	25480	25228	25371	25184

Table 10. Detailed Data Overview for Experiment 3 (100-1000 Inequalities and Variables)

Ineq. No.	Random (Top-Zero)	Columns Shuffled	Rows Shuffled	Rows and Columsn Shuffled
100	459887	614286	430305	486156
150	1401913	1375082	1384802	1204093
200	2572183	2602477	2879204	2779725
250	5174076	5888264	4554305	4540822
300	8299678	6401351	8348153	10449781
350	9597255	10295222	10874461	10558575
400	15925176	19790902	14842745	15306826
450	17555009	22356922	20596933	24725172
500	30078074	35493528	33429103	31311304
550	38257520	34269490	28336345	33912930
600	47594256	43120187	35637427	55693855
650	44706366	57783755	57376843	56833635
700	72872157	65579706	59179092	54404624
750	82911144	58163581	71530886	75324347
800	79381799	104929765	108354634	118149486
850	89829217	99647532	98487750	124152716
900	113429540	106757080	154850401	110742994
950	170769721	146056492	126749611	142870869
1000	153792091	143524583	134113421	173898153

III. Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, **Mihkel Uutar**,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to

   reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

   Beyond Worst-Case Complexity of the Simplex Method,

   supervised by Kallol Roy.

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 4.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Mihkel Uutar
14/05/2024