Computational Complexity Analysis of Linear Optimization Algorithms

by

Sophia Voulgaropoulou

B.S., University of Macedonia (2010)
M.S., Aristotle University of Thessaloniki (2012)

Submitted to the School of Information Sciences, Department of Applied Informatics
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

UNIVERSITY OF MACEDONIA

May 2022

Author . . . .
School of Information Sciences, Department of Applied Informatics
May 19, 2022

Certified by . . . .
Nikolaos Samaras
Professor
Thesis Supervisor

Accepted by . . . .
Professor Alexandros Chatzigeorgiou
Chairman

Computational Complexity Analysis of Linear Optimization Algorithms

by

Sophia Voulgaropoulou

Submitted to the School of Information Sciences, Department of Applied Informatics
on May 19, 2022, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

Computational complexity and performance analysis in Linear Optimization algorithms have always been topics of particular interest among the Operational Research community. In the current thesis, we are presenting all aspects and analysis results of our study on the performance of the Exterior Point Simplex algorithm, the Interior Point Method, and the Primal and Dual Simplex algorithms. Our objective is to generate valid and accurate prediction models for the computational performance of these algorithms. Our analysis is separated in three main parts, as described below.

First, we investigate the computational behavior of the Exterior Point Simplex algorithm (EPSA). Up until now, a significant difference has been observed between the theoretical worst case complexity and practical performance of simplex-type algorithms. To appropriately examine the latter, computational tests have been carried out on randomly generated sparse linear problems and on a small set of benchmark problems. Specifically, 6780 linear problems have been randomly generated, in order to formulate a respectable amount of experiments. This first part of our study consists of the measurement of the number of iterations that EPSA needs for the solution of the above mentioned problems and benchmark dataset. Our purpose is to form representative regression models, which would be significant for the evaluation of the algorithm's efficiency and could act as predictive models for the algorithm's performance. From each linear problem, we have taken several characteristics into account, such as the number of constraints and variables, the sparsity and bit length, and the condition of the constraint matrix. It is remarkable that the formulated model for the randomly generated problems reveals a linear relation between the number of EPSA iterations and the above mentioned characteristics.

Next, we extend our analysis, being concerned about the ability to choose the most efficient algorithm, in terms of execution time, for a given set of linear programming problems. Algorithm selection has been a significant, but at the same time, challenging process in all linear programming solvers. For the purpose of this part of our study, we utilize CPLEX Optimizer, which supports Primal and Dual variants of the Simplex algorithm and the Interior Point Method (IPM). We examine a performance prediction model using artificial neural networks for the

CPLEX's Interior Point Method on a set of 295 benchmark linear programming problems (etlib, ennington, Mészáros, Mittelmann) and measure the execution time needed for their solution. Specific characteristics of the linear programming problems are examined, such as the number of constraints and variables, the nonzero elements of the constraint matrix and the right-hand side, and the rank of the constraint matrix of the linear programming problems. Our purpose is to identify a model, which could be used for prediction of the algorithm's efficiency on linear programming problems of similar structure. This model can be used prior to the execution of the interior point method in order to estimate its execution time. Experimental results show a good fit of our model both on the training and test set, with the coefficient of determination value at 78% and 72%, respectively.

The current study is concluded by examining a prediction model using artificial neural networks for the performance of CPLEX's Primal and Dual Simplex algorithms on the same dataset and with the same variables as in IPM. The extracted results prove that a regression model cannot predict accurately the execution time of CPLEX's Primal and Dual Simplex algorithms. To overcome this issue, we treat the problem as a classification problem. Instead of estimating the execution time, our models estimate the class under which the execution time will fall. Experimental results show a good performance of the models both for Primal and Dual Simple algorithms, with an accuracy score of 0.83 and 0.84, respectively.

Thesis Supervisor: Nikolaos Samaras
Title: Professor

Research is the very soul of progress.
Whatever you do, try your best where there was almost nothing.
This, alone, can be everything for someone else to progress further.

To loving memory of my grandparents, Christos and Sophia
To Pantelis, Cleopatra and Christos
To Pavlos and my daughter

Publications

1. Voulgaropoulou, S.; Samaras, N.; Sifaleras, A. Computational complexity of the exterior point simplex algorithm. Oper. Res. Springer 2019, 19, 297–316.

2. Voulgaropoulou, S.; Samaras, N.; Ploskas, N., Predicting the execution time of the interior point method for solving linear programming problems using artificial neural networks. In Learning and Intelligent Optimization (LION 13); Lecture Notes in Computer Science; Springer: Cham, Switzerland, 2019; pp. 319–324.

3. Voulgaropoulou, S.; Samaras, N.; Ploskas, N. Predicting the Execution Time of the Primal and Dual Simplex Algorithms Using Artificial Neural Networks. Mathematics 2022, 10, 1038. https://doi.org/10.3390/math10071038

Acknowledgments

Thinking back to the moment the exciting journey for PhD studies was about to begin for me, I admit I could only imagine half of the experiences I was about to live. I knew this would not be easy, expecting all those moments of pressure, dead-ends and even, self-doubt. However, it was all those other moments, being the actual reward for the effort. Moments of great happiness and gratitude; when a goal was achieved, a finite result was met. Moments of gaining knowledge; even if this happened the hard way. Moments that opened my mind because I was surrounded by well-educated people, people who knew how to share and how to walk the extra mile, pushing and teaching me to do the same. Now, I am taking the opportunity to express my deepest appreciation to these people.

The person I could not thank more from the bottom of my heart, is my supervisor, Prof. Nikolaos Samaras. Nikolaos, after listening carefully and understanding the given circumstances, supported my decision to persue my dream. He knew from the beginning that, working as a full-time employee in an international firm, PhD studies would be rather demanding and challenging for me, both mentally and physically. His question seven years ago was very simple; ”Do you really want to do this? If not, it will be just a waste of time for both of us.“. I admit that back then, I kept thinking about this question and what he really meant. Now, I know that I would have never been able to complete my PhD studies if I was not willing enough to keep on with the effort, under any circumstances. For the productive discussions and his valuable advice; for the inspiration and great support from day one till now; for his determination and incredible expertise; for teaching me how to overcome dead-ends and explore alternative solutions; for this one-in-a-lifetime opportunity, I will always feel grateful, honored and blessed for having Nikolaos as my teacher, realizing that he already stands as one of the most important persons in my life.

Moreover, I would like to extend my sincere and grateful thanks to Prof. Angelo Sifaleras for his great assistance, whenever this was needed. His willingness to help me, by providing thorough explanation and sharing his feedback, was crucial for the progress of my analysis, especially during the first period of my PhD studies. I could not be more fortunate to have the honor to work and discuss with him. His always useful input and corrections helped me realize even more, that attention to detail always makes a difference.

Over the last three years of my studies, I also had the privilege of working with Prof. Nikolaos Ploskas and at this point, any attempt to express my sincere gratitude and appreciation for his support and advice would not be sufficient. Nikolaos has always been willing to listen to my concerns and ideas, introduce other alternatives, cross check and elaborate further on our results. I deeply value his feedback and most of all, his remarkable persistence, which was shared with me as a unique lesson from this brilliant mind.

Special thanks are extended to Prof. Ioannis Refanidis, Prof. Alexandros Chatzigeorgiou, Prof. Dimitrios Hristu - Varsakelis and Prof. Lefteris Angelis, as well. Through our conversations even before the beginning and in the course of my PhD studies, each one played a very significant role, without realizing it, for my concentration on my goal and this dissertation.

Last but not least, I am thankful and feel blessed for my family, for standing by my side throughout this journey. It has been my grandfather, Christos, sharing his love for lifelong learning since the very first moment I can recall and for making me set high standards for myself. It has been my grandmother, Sophia, who was always proud of me, even if she could not clearly understand my studies. It is my father, Pantelis; his love and pride of me have always been the most powerful engine to my personal development. It is my mother, Cleopatra; for her internal strength, for the trust she placed in me since I was a little girl and her unconditional love and support. It is my brother, Christos, who always stood bulletproof by my side to go through any negative life events in my life and during my studies. And last, but not least, it is my husband, my partner, my best friend; Pavlos. Without his support, love and encouragement, nothing would have been possible.

Contents

1 Introduction 15

1.1 Operational Research and Linear Programming 15

1.2 Contributions of this thesis 17

1.3 Overview of this thesis 18

2 Linear Programming 19

2.1 Concept, History & Applications 19

2.2 Linear Programming Algorithms 22

2.2.1 Primal and Dual Simplex Algorithms 24

2.2.2 Interior Point Method (IPM) 27

2.2.3 Exterior Point Simplex Algorithm (EPSA) 28

2.3 Complexity and Performance Analysis 30

3 Predictive Modelling 33

3.1 Regression Analysis 35

3.1.1 Regression Model and Evaluation Metrics 35

3.2 Artificial Neural Networks 40

3.2.1 Learning process 43

4 Dataset 45

4.1 Datasets for EPSA 46

4.1.1 Random Linear Problems 46

4.1.2 Linear Programming Benchmark Problems 48

4.2 Datasets for IPM, Primal, Dual algorithms 49

4.2.1 Linear Programming Benchmark Problems 49

4.3 Computing environment 70

5 Predictive Models 73

5.1 Exterior Point Simplex Algorithm (EPSA) 74

5.1.1 Predictive model for randomly generated LP problems 75

5.1.2 Validation of Predictive Model for randomly generated LP problems 79

5.1.3 Predictive model for benchmark LP problems 80

5.2 CPLEX - IPM, Primal and Dual Simplex Algorithms 82

5.2.1 Predictive model for Interior Point Method 83

5.2.2 Predictive models for Primal and Dual Simplex Algorithm . . 86

6 Conclusions 103

List of Figures

3-1 Structure of an artificial neuron 41

3-2 Structure of Artificial Neural Networks with 2 hidden layers 42

5-1 Normal Probability plot of iterations standardized residuals 77

5-2 Deviation Histogram - Validation 80

5-3 Outliers - Validation 81

5-4 Regression model for interior point method - Tuning the number of neurons (1 hidden layer) 86

5-5 Regression model for interior point method - Tuning the activation function 87

5-6 Regression model for primal method - Tuning the number of neurons (1 hidden layer) 90

5-7 Regression model for primal method - Tuning the activation function 91

5-8 Regression model for dual method - Tuning the number of neurons (1 hidden layer) 91

5-9 Regression model for dual method - Tuning the activation function . 92

5-10 Classification model for primal method - Tuning the number of hidden layers and neurons (*tanh* activation function, *lbfgs* solver) 98

5-11 Classification model for primal method - Tuning the activation function and solver (2 hidden layers, 100 neurons each) 99

5-12 Classification model for primal method - Testing different classification algorithms (1 hidden layer, *tanh* activation function, *lbfgs* solver) . . . 99

5-13 Classification model for dual method - Tuning the number of hidden layers and neurons (*relu* activation function, *lbfgs* solver) 100

5-14 Classification model for dual method - Tuning the activation function and solver (2 hidden layers, 100 neurons each) 101

5-15 Classification model for dual method - Testing different classification algorithms 101

List of Tables

2.1 Correspondence between LP problem attributes for Primal and Dual Simplex Algorithm . . . 25
2.2 Complexity of linear programming algorithms . . . 32

4.1 Value ranges of LP problems . . . 47
4.2 Characteristics of linear problems . . . 47
4.3 Characteristics of linear problems . . . 48
4.4 Characteristics of the MPS benchmark files . . . 48
4.4 Characteristics of the MPS benchmark files . . . 49
4.5 Netlib Optimal . . . 51
4.6 Netlib Kennington . . . 52
4.7 Mészáros Miscellaneous (a) . . . 53
4.8 Mészáros Miscellaneous (b) . . . 54
4.9 Mészáros Miscellaneous (c) . . . 55
4.10 Mészáros Miscellaneous (d) . . . 56
4.11 Mészáros Miscellaneous (e) . . . 57
4.12 Mészáros Miscellaneous (f) . . . 58
4.13 Mészáros Problematic . . . 59
4.14 Mészáros Stochastic LP (a) . . . 60
4.15 Mészáros Stochastic LP (b) . . . 61
4.16 Mittelman (a) . . . 62
4.17 Mittelman (b) . . . 63
4.18 Execution time for IPM, Primal and Dual (in seconds) . . . 64
4.18 Execution time for IPM, Primal and Dual (in seconds) . . . 65
4.18 Execution time for IPM, Primal and Dual (in seconds) . . . 66
4.18 Execution time for IPM, Primal and Dual (in seconds) . . . 67
4.18 Execution time for IPM, Primal and Dual (in seconds) . . . 68
4.18 Execution time for IPM, Primal and Dual (in seconds) . . . 69
4.19 Lower and Upper values in examined LP characteristics . . . 70
4.20 Computing and model creation environment for EPSA . . . 72
4.21 Computing and model creation environment for IPM, Primal and Dual Simplex . . . 72

5.1 Statistical values of regression model . . . 76
5.2 Statistical values of exponential Regression model . . . 78
5.3 Statistical values of logarithmic Regression model . . . 78

5.4 Validation Dataset 79
5.5 Deviation from Observed values in the resulted regression model . . . 80
5.6 Statistical values of Regression model - Benchmark 82
5.7 Model parameters for the MLP method 85
5.8 MLPRegressor model for the IPM execution time 85
5.9 MLP model parameters used for primal and dual simplex algorithms . 88
5.10 MLPRegressor model for the execution time of the primal and dual simplex algorithms 89
5.11 Other Regression models for the execution time of the primal simplex algorithm 93
5.12 Other Regression models for the execution time of the dual simplex algorithm 93
5.13 Classes of the primal and dual simplex algorithms execution time (in seconds) 96
5.14 Confusion matrix for the primal simplex algorithm execution time . . 97
5.15 Confusion matrix for the dual simplex algorithm execution time . . . 97
5.16 Classification reports for the primal and dual simplex algorithms execution time 97

Chapter 1

Introduction

1.1 Operational Research and Linear Programming

Operational Research (OR) is a scientific area where advanced analytical processes and methods are applied in order to support and enhance problem-solving and decision-making procedures. Although, OR is often considered to be part of the general field of Applied Mathematics, its concept and applications go far beyond mathematical activities and purposes. Specific mathematical approaches that formulated the first aspects of OR date back to the 17th century, however, its modern version was established after World War I and especially, during military actions in World War II. Soon after the end of World War II, scientists began applying OR methods to sectors, other than the military one, such as logistics, infrastructure, etc. At that time and after the development of the Simplex Algorithm (SA) for Linear Programming (LP) problems in 1947, OR became inextricably linked to the area of LP and deployed every piece of improvement coming from the development of computers to evolve to a widely applicable science for solving problems with tremendous amount of variables and constraints. A plethora of real-world problems can, nowadays, be formulated as LP problems, while LP algorithms have been used in various fields, such as logistics, transportation, decision making, data mining and more. The purpose has always been to reach an optimal solution for complex problems and this is achieved by engaging other mathematical methods, such as statistical analysis, mathematical modeling and optimization and other.

Operational Research can be used to determine the maximum profit or performance or the minimum costs or risk in numerous real-world problems and practical applications. It is often overlapping with other fields, such as operations management, organization science, business analytics and even psychology, especially when it comes to areas, where the human factor is also present, such as in customer services, supply chain management, project management, revenue management, etc. Other applications of OR can be found in modern transportation (e.g. scheduling of airline and train routes), assignment and allocation of resources (e.g. assigning certain employees to specific projects, etc.). Operational Research can be considered as a powerful tool for decision makers in various sections aiming to make better decisions, as it supports strategic and operational decisions in problems which often involve a considerable amount of risk or uncertainty. Depending on the section where OR is applied, the benefits of its use can vary from performance improvements, such as cost and risk reduction or revenue increase, to raising customer satisfaction levels or even saving human lives. Eventually, OR can lead to the development of more productive and efficient systems, taking into consideration all existing alternative options and making precise predictions and risk estimations. In the effort towards building such systems, there are always specific contributing factors, such as optimization effort (i.e. narrowing down any potential alternatives and achieving the best results under the current circumstances), simulation processes (i.e. engaging model building processes for testing and validation of the solutions found), statistics and probability analysis (i.e. application of appropriate algorithms to reveal useful insights, reach accurate prediction results and validate potential solutions).

As stated above, numerous practical problems in OR can be expressed as LP problems. Linear Programming addresses the optimization problem of a linear objective function, which is subject to linear equality and inequality constraints on the decision variables. Specific LP cases, such as network flow problems and multicommodity flow problems have triggered extensive research on several algorithms for their solution. Other LP algorithms work by solving LP problems as sub-problems. Historically, LP has had a significant impact on other key concepts of optimization theory, such as duality, decomposition, etc. On its turn, LP was heavily used in the early formation of microeconomics. Currently, despite the fact that the issues are ever-changing, it is vastly used in corporate management, since the majority of companies need to maximize profits and minimize costs with limited resources.

In order to effectively represent a real-world problem through a set of linear functions and constraints in LP, mathematical formulations of the problem are used. These formulations model the problem in such a way, so that it becomes suitable to be addressed and solved by LP algorithms. However, real-world problems are always more complex than their respective mathematical formulations and models. In our effort to depict real-world more accurately, the formulations and LP algorithms we use may inevitably become more complex. We should always be reminded that LP algorithms should only be as complex as needed, so that they solve real-world problems reasonably well. Any added complexity should definitely be followed by significant gains in our ability to represent and solve a problem and in the quality of the solutions we obtain. Questions, such as “What are the benefits of LP algorithms for each problem we are attempting to solve?” or “Will the use of a respective LP algorithm outweigh any additional costs?” are timeless and have resulted to the deployment of several performance analysis and measurement techniques for LP algorithms. The performance of LP algorithms is inextricably linked with the concept of complexity analysis, since complexity-related questions about LP have been raised since 1950, before the field of computational complexity started to develop in the decade of 1970 [74]. Although the theoretical complexity of some of the most widely-used LP algorithms has been proven over the years, there are still open topics for discussion and further research, when it comes to their computation complexity.

1.2 Contributions of this thesis

This dissertation aims to contribute to the field of computational complexity and performance analysis of Linear Programming (LP) Optimization algorithms. More specifically, the current thesis:

- Focuses on the computational performance of four of the most widely applied LP Optimization algorithms; Exterior Point Simplex Algorithm (EPSA), Interior Point Method (IPM), Primal and Dual Simplex algorithms. Initially, our focus was concentrated on worst and average case complexity analyses and how the respective results can be interpreted more efficiently for the application of several algorithms. In that frame, we studied the work of Ho and Sundarraj [53], who analyzed the time of the revised simplex method, by using an economic order quantity (EOQ) formula and tried to form a timing model to reduce the cost of routine applications and to predict the performance of new variants of the algorithm. Upon completing our study on this work, our interest towards analyzing the computational behavior of LP Optimization algorithms became even stronger. Until then, it was clear that the theoretical behavior of an algorithm is associated with its complexity. One could even suggest that the analysis of an algorithm's complexity involves the worst case complexity at all times, even if the actual worst case may never occur in practice. In such scenarios, any timing model we may form for a particular algorithm will eventually be a non-representative one, consequently leading to false conclusions about the practical performance of the algorithm. By *timing model*, we are referring to mathematical model for estimation of the time or the number of iterations needed by an algorithm for the execution of specific problems with a given set of variables and constraints.

- Introduces new models, which can, not only, be further utilized for prediction of the computational performance of the examined algorithms on new problems, but also stand as the initiation point for further research on performance analysis of other LP algorithms.

- Provides significant insight to modeling methods, used for prediction of the performance and computational behavior of LP algorithms for solving new problems. Apart from regression analysis, artificial neural networks are also utilized, exploring and revealing strong potential in the performance analysis field for LP optimization algorithms. Even when it is not possible to generate a valid regression model for prediction of continuous values for a certain variable, classification methods can play a significant role in creating a model for prediction of a certain range or particular class of values.

In classical complexity analysis, a theoretical study of algorithms is conducted, taking into consideration the problem dimensions. Worst and average case analyses may describe execution time as a function of problem dimension parameters only. Aspects, such as computer specifications, programming style, programming language and operating system are not taken into consideration. Experimental analysis evaluates the real running time, the number of iterations performed and the solution quality on the selected dataset. Consequently, the analysis conducted for the purposes of this study does not aim to act as substitute for classical complexity analysis, but rather suggests that these types of analysis are complementary.

1.3 Overview of this thesis

At this point, we are briefly describing the document structure of this thesis and providing an overview of the content in each chapter. In Chapter 2, we are describing the main concepts, history and applications of Linear Programming, while we are also providing a presentation of the LP Optimization algorithms which are examined in this study (i.e. Exterior Point Simplex algorithm, Interior Point Method, Primal and Dual Simplex algorithms). This chapter also includes a brief presentation of complexity and performance analysis processes for LP Optimization algorithms. Chapter 3 consists of a thorough description of the predictive modelling methods that have been applied for the purposes of this study. Specific details are included for regression analysis technique and regression evaluation metrics, while elaborate input is provided on artificial neural networks and their learning process. Chapter 4 consists of a detailed description of the datasets and computing environments used in our study. The datasets are presented, based on how they were utilized in our study; random linear problems and LP benchmark problems for EPSA and LP benchmark problems for IPM, Primal and Dual Simplex algorithms. Chapter 5 presents our analysis and the respective results from the predictive models we have generated. The models are provided separately for each algorithm and are accompanied by their corresponding validation results. Finally, chapter 6 concludes the current thesis and compiles our findings for each examined algorithm. This chapter also includes further recommendations and ideas for future work or additional research.

Chapter 2

Linear Programming

2.1 Concept, History & Applications

In a fast-evolving era, especially during and after World War II, the majority of problems that had to be dealt with, were optimization problems. As such, each of these problems would require a solution which by (a) satisfying specific constraints and (b) taking into account, particular, clearly-defined criteria, would be the best possible one out of all solutions that indeed satisfy the specified constraints. Interestingly enough, the majority of the arisen optimization problems could be represented by using linear functions; not only for the constraints but also for the optimization criterion. This gradually resulted in the foundation of Linear Programming as a broader class of optimization problems which can be solved with the use of linear functions. Therefore, Linear Programming (LP) or Linear Optimization, can be considered as a method, which is applied for achieving the optimal outcome in a mathematical model where the requirements are defined as linear relationships.

The historical background of LP dates back in 1827, when Fourier published a method for solving a system of linear inequalities [46] while in 1939, the Soviet economist Leonid Kantorovich gave a linear programming formulation for a problem equivalent to the general linear programming problem and proposed a new method for solving it [61]. Although Kantorovich developed this innovative method during World War II to plan expenditures and returns to reduce costs in the army section and increase losses for the enemy, his study was underestimated by the USSR. Some years later, during 1946–1947, George B. Dantzig developed a general linear programming formulation for planning problems in the US Air Force. Dantzig managed to efficiently tackle the linear programming problem for the first time, by inventing the Simplex Algorithm in 1947 [36], [37], [35]. At that time, Dantzig met with John von Neumann to share his findings about Simplex Algorithm and it was only then that Neumann realized the conjecture of the theory of duality, since the problem he had been working in game theory was actually equivalent. This was a significant moment, becoming the realization point of the connection between Game Theory and Linear Programming [116], [117]. In 1979, Leonid Khachiyan proved theoretically that the linear programming problem was solvable in polynomial time with the Ellipsoid algorithm [64], however, in 1984 a new theoretical and practical breakthrough was made by Narendra Karmarkar, who introduced a new Interior Point Method for solving linear programming problems [62]. Practically, in any linear programming problem, we have a number of variables, which will be assigned with real values so that (a) specific linear equations and/or inequalities are satisfied and (b) the linear objective function is maximized or minimized (depending on the nature of the problem). Approaching LP from a mathematical perspective, we could safely consider that it aims to optimize a linear objective function, which is subject to linear equality and inequality constraints. The respective feasible region is a convex polytope; a set which is represented by the intersection of finitely many half spaces, each of which is defined by a linear inequality. The objective function is a real-valued affine (linear) function, which is defined on this specific polyhedron. A linear programming algorithm detects a point in the polytope where the objective function has the optimal value (if such a point exists at all).

$$\begin{aligned} Maximize \quad & c^T x \\ subject\ to \quad & Ax \leq b \\ and \quad & x \geq 0 \end{aligned}$$

In a linear problem of the above form, x stands for the vector of variables which is to be determined, c and b are vectors of known coefficients, A is a known matrix of coefficients, and $(.)^T$ is the matrix transpose. The objective function is the expression that needs to be maximized or minimized; in this case, this is $c^T x$. The constraints defining the convex polytope on which the objective function needs to be optimized are the inequalities $Ax \leq b$ and $x \geq 0$. In LP problems, two vectors are comparable only when they have the same dimensions, so if every entry in the first is less than or equal to the respective entry in the second, then the first vector is less than or equal to the second vector.

A linear problem can be expressed in either standard or canonical form. The maximization of a linear function is subject to constraints expressed as linear inequalities in standard form, while they are expressed as linear equalities in canonical (slack) form. The three main parts of standard form, in which all linear problems can be expressed, are described below. Other forms can always be transformed to equivalent problems in standard form. The three parts of the standard form are described below:

1. Linear function to be maximized
   Example: $f(x_1, x_2) = c_1 x_1 + c_2 x_2$

2. Problem constraints in the following form:
   Example:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 &\leq b_1 \\ a_{21}x_1 + a_{22}x_2 &\leq b_2 \\ a_{31}x_1 + a_{32}x_2 &\leq b_3 \end{aligned}$$

3. Non-negative variables
   Example: $x_1 \geq 0, x_2 \geq 0$

A quick example of a simple LP problem is described further below. Let us assume that Alpha Estate winery wants to increase production of the 2 most well-known labels: Axia Red and Alpha Estate Red (S.M.X.). How many bottles of each one should it produce to maximize its profits? Currently, Alpha Estate winery produces x_1 bottles of Axia Red per day with profit of 18 Euros each, while for Alpha Estate Red (S.M.X), the winery makes x_2 bottles per day with a profit of 25 Euros each. Although x_1 and x_2 are unknown values, obviously they need to be greater than zero (x_1, $x_2 \geq 0$). Moreover, we need to keep in mind that the daily demand for Axia Red is 100 bottles, while for Alpha Estate Red (S.M.X.) is 80 bottles. Last, but not least, Alpha Estate winery can make up to 150 bottles per day, so taking into consideration all these constraints, what are the optimal levels of production in this case? The linear problem representing this case is the following:

$$\begin{array}{rll} Objective function & max & 18x_1 + 25x_2 \\ Constraints & & x_1 \leq 100 \\ & & x_2 \leq 80 \\ & & x_1 + x_2 \leq 150 \\ & & x_1, x_2 \geq 0 \end{array}$$

A linear equation in x_1 and x_2 defines a line in the 2-dimensional space, while a linear inequality represents a half-space, i.e. the region on one side of the line. The feasible solutions of this particular linear problem, i.e. the points (x_1, x_2) which satisfy all constraints above, is the intersection of 5 half-spaces, as these are defined by the inequalities above and is a convex polygon. As a general rule, the optimal solution of a linear problem is achieved at a vertex of the feasible region. This is not the case if the problem is infeasible or unbounded. In an infeasible linear problem, the constraints are so tight that it is impossible to satisfy them all (for instance, x $\geq$ 0 and $x \leq$ 1). In an unbounded problem, the constraints are so loose, that the feasible region is actually unbounded and most probably, we will achieve arbitrarily high objective values (for instance, minimize $x_1 + x_2$ with x_1, $x_2 \geq 0$). The current optimization problem, such as all similar ones, can be solved by the Simplex Method, which would start at a vertex (i.e. (0, 0) in our case) and would repeatedly search for an adjacent vertex (neighbor) that would produce a better objective value; the adjacent vertex is connected through an edge of the feasible region. While Simplex performs this "hill-climbing" on the vertices of the polygon, it walks from neighbor to neighbor and gradually increases profit as it progresses. As soon as the algorithm reaches a vertex which has no better neighbor, Simplex stops and considers the current vertex as the optimal one. One may express doubts about this "local" point being optimal "globally". However, examining the geometry of the problem, we can think of a profit line passing through the optimal vertex. Since all neighbors are below this line (otherwise the vertex would have a neighbor with better objective value), this means that the rest of the feasible region is below this line as well, confirming the current vertex as the optimal solution indeed.

Having briefly described the function of the officially first of the most fundamental tools of LP, Simplex Method, it is now time to proceed to the following section, by presenting the algorithms which were examined during research for this thesis; Primal and Dual Simplex Algorithms, Interior Point Method (IPM) and Exterior Point Simplex Algorithm (EPSA). A more detailed description for each algorithm is provided in respective studies, referred in the following subsections. In Chapter 5, the algorithms are presented in a chronological sequence based on when they were examined during research for this thesis. Our interest in these algorithms derives from the fact that Linear Programming is nowadays applied in various fields, such as mathematics, economics and other industries; transportation, business planning, resource allocation, energy and manufacturing being only some of the most well-known ones. It stands as a key tool for Operational Research and is a significant contributor in modeling several types of problems of planning, routing, scheduling and assignment. However, the power of linear problems could not be revealed and would be of no use at all, if we did not have a way to solve them.
That is, Linear Programming Algorithms.

2.2 Linear Programming Algorithms

Prior to the presentation of the examined LP algorithms which follows in this section, it is important to describe the linear programming problem we are concerned with (LP.1 in the standard form):

$$\begin{aligned} \min\ & c^T x \\ \text{s.t.}\ & Ax = b \\ & x \geq 0 \end{aligned} \tag{LP.1}$$

Here, $A \in \mathbb{R}^{mxn}$, $b \in \mathbb{R}^m$, $c, x \in \mathbb{R}^n$, T denotes transposition and rank$(A) = m$, $1 \leq m < n$. The dual problem of (LP.1) is presented below and will be explained further in section 2.2.1.

$$\begin{aligned} &\max b^T w \\ &\text{s.t. } A^T w + s = c \\ &\quad s \geq 0 \end{aligned} \tag{DP.1}$$

where $w \in \mathbb{R}^m$ and $s \in \mathbb{R}^n$. Partitioning the matrix A of (LP.1) as $A = (B,N)$ and with a corresponding partitioning and ordering of $x^T = [x_B \ x_N]$ and $c^T = [c_B \ c_N]$ (LP.1) is written as:

$$\begin{aligned} &\min c_B^T x_B + c_N^T x_N \\ &\text{s.t. } A_B x_B + A_N x_N = b \\ &\quad x_B, x_N \geq 0 \end{aligned} \tag{LP.2}$$

Here A_B is an $m \times m$ non-singular submatrix of A, called basic matrix (or basis), whereas A_N is an $m \times (n\text{-}m)$ submatrix of A, called non-basic matrix. The columns of A which belong to B are called basic and the remaining ones, non-basic. The solution $x_B = B^{-1}b$, $x_N = 0$ is called a basic solution. This solution is feasible iff $x_B \geq 0$. Otherwise, it is called infeasible. The solution of dual problem which corresponds to the basis B, is given by $(s_N)^T = (c_N)^T - w^T A_N$, $(s_B)^T = 0$, where $w^T = (c_B)^T A_B^{-1}$ are the Simplex multipliers and s_N are the dual slack variables. This solution is dual feasible iff $s_N \geq 0$. The ith row of the coefficient matrix A is denoted by $A_{i.}$ and the jth column by $A_{.j}$. The basis inverse $(A_B)^{-1}$ is maintained in some factorized form. At every iteration, its factors have to be updated. There are many techniques for updating the invertible representation at the basis matrix. The simplest updating scheme is the Product Form of the Inverse (PFI). The current basis inverse $(A_{\overline{B}})^{-1}$ can be updated from the previous inverse $(A_B)^{-1}$, using the relation $(A_{\overline{B}})^{-1} = (A_B E)^{-1} = E^{-1}(A_B)^{-1}$, where E^{-1} is the inverse of the so-called eta-matrix. The matrix E^{-1} is computed using the following relation

$$E^{-1} = \begin{bmatrix} 1 & & -\frac{h_{1r}}{h_{rr}} & & \\ & \ddots & \vdots & & \\ & & \frac{1}{h_{rr}} & & \\ & & \vdots & \ddots & \\ & & -\frac{h_{mr}}{h_{rr}} & & 1 \end{bmatrix} \tag{2.1}$$

where h_{rr} is the pivot element and $h_j = (A_B)^{-1}A_{.j}$.

2.2.1 Primal and Dual Simplex Algorithms

In section 2.1, we provided an example of how Simplex would perform for solving a linear problem, given a specific objective function and a set of variables and constraints. At a very high level, Simplex works on a linear objective function and a set of linear inequalities as the ones presented in (LP.1), trying to find the optimal feasible point. On each iteration, the algorithm checks whether the current vertex is optimal and if so, it stops - otherwise it defines where to move next. Although it is easily understood how Simplex moves and progresses in order to identify the optimal solution, it is also very important to define the vertex from which Simplex should begin, i.e. initiation or **starting vertex**. As explained, we may have to transform the coordinate system of the problem, so that the algorithm moves to a vertex from where it can start. Apart from the starting vertex, there is the **degeneracy** problem as well. In case the neighbors of the current vertex are all identical and have no better objective value, Simplex will eventually return a sub-optimal degenerated vertex as result. To overcome this problem, we may want to modify Simplex so that it detects degeneracy and keeps on moving from vertex to vertex, despite the fact that we lack in any improvement in the objective function. However, such a modification would result in Simplex looping forever! This effect can be avoided by perturbations, small changes in the structural stability of the problem. By structural stability, we mean that the qualitative behavior of the trajectories in a dynamical system is not affected by small, almost tiny perturbations and remains stable. Last, but not least, for unbounded linear problems where the objective function may become arbitrarily large (if we are referring to a maximization problem) or small (in case of a minimization problem), Simplex searches for a vertex in the neighborhood of the current vertex and, eventually, identifies that, when it replaces a particular inequality with another one, this results to an eventually undetermined system of equations with infinite number of solutions. In this case, Simplex stops and reports the **unboundedness** of the problem. A more comprehensive explanation of the topics described above can be found in several books and lecture notes for Linear Programming, including (but certainly not limited to) [95], [27] and [114].

We will now consider a simple, non-degenerate linear problem, intending to find the lowest point in a convex polyhedron (geometrically this is the intersection of e.g. d halfspaces). We actually want the lowest vertex in the intersection of these halfspaces, so let's examine the fundamental steps of Primal Simplex Algorithm (PSA) that will be followed in this case:

```
    Step 1 (Feasibility check)
    if $\cap H = \varnothing$ , return INFEASIBLE
    $x \leftarrow$ any feasible vertex
Step 2 (Unboundedness check)
    while $x$ is not locally optimal
    pivot down and maintain feasibility
    if every feasible neighbor of $x$ is higher than $x$, return UNBOUNDED
    else $x \leftarrow$ any feasible neighbor of $x$ that is lower than $x$
    return $x$
```

As for Dual Simplex Algorithm, the respective description is provided below:

```
Step 1 (Unboundedness check)
    if there is no locally optimal vertex, return UNBOUNDED
    $x \leftarrow$ any locally optimal vertex
Step 2 (Feasibility check)
    while $x$ is not feasible
    pivot up and maintain local optimality
    if every locally optimal neighbor of $x$ is lower than $x$, return INFEASIBLE
    else $x \leftarrow$ any locally optimal neighbor of $x$ that is higher than $x$
    return $x$
```

Trying to understand the linear algebra behind the 2 algorithms, we can see that there is actually no real difference between Primal and Dual Simplex implementation, other than their subsequent geometrical interpretation. Dual Simplex is very important alternative for solving LPs, being effective on a plethora of problems, particularly of integer linear programming. A more detailed overview of the correspondence between attributes of LP problems, when solved by Primal and Dual Simplex algorithms, is presented in the following table 2.1.

Table 2.1: Correspondence between LP problem attributes for Primal and Dual Simplex Algorithm

Primal			**Dual**	
minimum		↔	maximum	
constraint	$=$	↔	variable	free
constraint	$\geq$	↔	variable	≥ 0
constraint	$\leq$	↔	variable	≤ 0
variable	free	↔	constraint	$= 0$
variable	≥ 0	↔	constraint	$\leq$
variable	≤ 0	↔	constraint	$\geq$

A formal description of the Revised Primal Simplex algorithm and the Revised Dual Simplex algorithm is given briefly below, while more information and details on definitions, optimality and feasibility conditions and examples can be found in [95], [27], [114], [41] and [87].

Revised Primal Simplex Algorithm

Step 0 (Initialization)

Start with a feasible partition *(B,N)*. Compute $(A_B)^{-1}$ and vectors x_B, w and s_N.

Step 1 (Test of optimality)

if $s_N \geq 0$, STOP. Problem (LP.2) is optimal.

else choose the index l of the entering variable using a pivoting rule. Variable x_l enters the basis.

Step 2 (Pivoting)

Compute the pivot column $h_l = (A_B)^{-1} A_l$.

$h_l \leq 0$, STOP. Problem (LP.2) is unbounded.

else choose the leaving variable $x_{B[r]} = x_k$ using the relation:

$$x_{B[r]} = \frac{x_{B[r]}}{h_{il}} = min \left\{ \frac{x_{B[i]}}{h_{il}} : h_{il} < 0 \right\}$$

Step 3 (Update)

Swap indices k and l. Update the new basis inverse $(A_{\overline{B}})^{-1}$ using a basis update scheme. Update vectors x_B , w, and s_N .Go to Step 1.

Revised Dual Simplex Algorithm

Step 0 (Initialization)

Start with a feasible partition *(B,N)*. Compute $(A_B)^{-1}$ and vectors x_B, w and s_N.

Step 1 (Test of optimality)

if $x_B \geq 0$, STOP. The primal problem (LP.2) is optimal.

else choose the leaving variable k, so that $x_{B[r]} = x_k = min \left\{ x_{B[i]} : x_{B[i]} < 0 \right\}$. Variable x_k leaves the basis.

Step 2 (Pivoting)

Compute the vector $H_{rN} = (A_B)^{-1}{}_{r.} A_N$.

if $H_{rN} \geq 0$, STOP. The primal problem (LP.2) is infeasible.

else choose the entering variable $x_{N[t]} = x_l$ using the following minimum ratio test:

$$x_l = x_{N[t]} = \frac{-s_{N[t]}}{H_{rN}} = min \left\{ \frac{-s_{N[i]}}{H_{iN}} : H_{iN} < 0 \right\}$$

Step 3 (Update)

Swap indices k and l. Update the new basis inverse $(A_{\overline{B}})^{-1}$ using a basis update scheme. Update vectors x_B , w, and s_N .Go to Step 1.

2.2.2 Interior Point Method (IPM)

As already discussed, IPM was a major breakthrough in Linear Programming, with its concept being quite different than Simplex or Ellipsoid algorithms. IPM “cuts” path in the interior of the polyhedron, instead of moving from vertex to vertex on the boundary of the feasible region. Thus, although nobody could be absolute in a decision whether Simplex or IPM performs better in solving particular problems, one thing is for certain; the most positive outcome of this “competition“ between the 2 algorithms, is that it became the lightning start of very fast and efficient coding for Linear Programming, as it put remarkable pressure on developers of existing commercial Simplex applications. Most of the times, the large size of problems tended to be in favor of interior point methods, but it has been quite difficult to predict the winner on a particular class of problems. For instance, the sequential nature of Simplex may make parallelisation difficult [50], though it performs better in a hyper-sparse linear problem [51]. On the contrary, interior point methods can significantly speed up in massive parallelisation, by utilizing block-matrix structures in linear algebra operations [49]. A quick look at the most widely-known implementation of Karmarkar’s Interior Point Method would explain that it is based on a predictor-corrector technique as this was suggested by Mehrotra, back in 1992 [75]. This technique implies that in each iteration of the IPM, it is necessary to calculate the Cholesky decomposition (factorization) of a large matrix to find the search direction [31]. From a computational point of view, this factorization step is the most expensive one of the algorithm. For this reason, Mehrotra’s predictor–corrector method (MPC) uses the same Cholesky decomposition (without recalculating it) to find two different directions: a predictor and a corrector. The basic concept is to begin by calculating an optimized search direction based on the predictor. Then, the size of the step that is taken towards this direction will be used to evaluate the centrality correction that is needed and the corrector will be computed. The complete search direction results from the sum of the predictor’s and the corrector’s direction. Mehrotra’s predictor-corrector method is widely known in practice, although there is no theoretical complexity linked to it [88]. During corrector step, it uses the same Cholesky decomposition found during the predictor step and in this way, it is only marginally more expensive than a standard interior point algorithm. On the other hand, this additional increase per iteration is usually balanced by a reduction in the number of iterations, needed to achieve the optimal solution. Further research has led to the development of efficient IPMs which outperform PSA on large-scale problems, practically as well. All IPMs maintain the approach of reaching the optimal solution through a sequence of points inside the feasible region. Highly influential and contributing studies have been introduced by Gondzio and Wright, [48] and [121]. The implementation of IPM which is examined in this thesis is OB1, developed by Lustig, Marsten and Shanno in 1994 [38], implementing a primal-dual predictor-corrector interior point code.

The majority of primal-dual IPMs needs a strictly feasible interior point as a starting point, which, for some LPs, is difficult to find. MPC is an infeasible primal-dual IPM and it just requires that $(x^0,\ s^0) > 0$ for the starting point. At each iteration of the algorithm, a point $(x,\ w,\ s)$ is calculated. This point is permitted to be infeasible with $(x,\ s) > 0$. A formal description of MPC method is briefly presented below, while more details about all calculations and mathematical equations can be found in [87].

Mehrotra's predictor-corrector method

Step 0 (Initialization)

Presolve the LP problem. Scale the LP problem.

Find an initial interior point $(x^0,\ w^0,\ s^0)$.

Step 1 (Test of optimality)

Calculate the primal (r_p), dual (r_d) and complementarity (r_c) residuals.

Calculate the duality measure (μ).

if $max(\mu, \|r_p\|, \|r_d\|) \leq tol$, STOP. The problem LP.1 is optimal.

Step 2 (Predictor)

Solve the system 2.2 for $(\varDelta x^p,\ \varDelta w^p,\ \varDelta s^p)$.

$$\begin{bmatrix} 0 & A^T & I \\ A & 0 & 0 \\ S & 0 & X \end{bmatrix} \begin{bmatrix} \varDelta x^p \\ \varDelta w^p \\ \varDelta s^p \end{bmatrix} = \begin{bmatrix} A^T w + s - c \\ Ax - b \\ Xs \end{bmatrix} = \begin{bmatrix} r^d \\ r^p \\ r^c \end{bmatrix} \tag{2.2}$$

Calculate the largest possible step lengths ${\alpha_p}^p$ and ${\alpha_d}^p$.

Step 3 (Centering Parameter)

Compute the centering parameter σ.

Step 4 (Corrector)

Solve the system 2.3 for $(\varDelta x,\ \varDelta w,\ \varDelta s)$.

$$\begin{bmatrix} 0 & A^T & I \\ A & 0 & 0 \\ S & 0 & X \end{bmatrix} \begin{bmatrix} \varDelta x \\ \varDelta w \\ \varDelta s \end{bmatrix} = \begin{bmatrix} A^T w + s - c \\ Ax - b \\ Xs - \sigma\mu e \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \varDelta X^p \varDelta s^p \end{bmatrix} \tag{2.3}$$

Calculate the primal and dual step lengths α_p and α_d.

Step 5 (Update)

Update the solution $(x,\ w,\ s)$.Go to Step 1.

2.2.3 Exterior Point Simplex Algorithm (EPSA)

The Exterior Point Simplex Algorithm (EPSA) was first introduced by Paparrizos in 1991 [77]. All simplex-type algorithms can be interpreted with the process of walking on simplex-type paths, which eventually lead to the optimal solution. The basic concept of EPSA lays on the improvement it attempts to introduce, by avoiding the boundary of the polyhedron of the feasible region and thus, constructing two paths to reach the optimal solution. One path is exterior to the feasible region while the other is interior. In 1993, Paparrizos extended EPSA concept to the general Linear

Problem [78]. EPSA generates solutions that are not feasible. In every iteration, EPSA generates two paths to the optimal solution, with one path being infeasible (exterior) and the other one being feasible. Using this movement, EPSA does not need to proceed by visiting one edge after another along the polyhedron $P = \{x \mid Ax \leq b, x \geq 0\}$. A more formal description of the EPSA implementation is given below.

Step 0 (Initialization)

Start with a feasible partition *(B,N)*. Compute $(A_B)^{-1}$ and vectors x_B, w and s_N. Find the sets $P = \{j \in N : s_j < 0\}$ and $Q = \{j \in N : s_j \geq 0\}$. Compute s_0 using the relation

$$s_0 = \sum_{j \in P} s_j$$

Also, compute the vector direction d_B from

$$d_B = -\sum_{j \in P} h_j$$

with $h_j = (A_B)^{-1} A_{.j}$.

Step 1 (Test of termination)

i. (Optimality test): If $P = \varnothing$, STOP. Problem (LP.1) is optimal.

ii. (Choice of leaving variable): If $d_B \geq 0$, STOP. If $s_0 = 0$, problem (LP.1) is optimal. If $s_0 < 0$, problem (LP.1) is unbounded. Otherwise, choose the leaving variable $x_k = x_{B[r]}$ using the minimum ratio test:

$$\alpha = \frac{x_{B[r]}}{-d_{B[r]}} = min\left\{\frac{x_{B[i]}}{-d_{B[i]}} : d_{B[i]} < 0\right\}$$

If $\alpha = \min\{\varnothing\} = +\infty$, the problem (LP.1) is unbounded.

Step 2 (Pivoting)

Compute the row vectors $H_{rP} = (A_B)^{-1}_{r.} A_P$ and $H_{rQ} = (A_B)^{-1}_{r.} A_Q$, where $(A_B)^{-1}_{r.}$ denotes the rth row of the basis inverse $(A_B)^{-1}$. Compute the ratios ϑ_1 and ϑ_2, using the relations:

$$\vartheta_1 = \frac{-s_p}{H_{rp}} = min\left\{\frac{-s_j}{H_{rj}} : H_{rj} > 0 \wedge j \in P\right\}$$

$$\vartheta_2 = \frac{-s_q}{H_{rq}} = min\left\{\frac{-s_j}{H_{rj}} : H_{rj} < 0 \wedge j \in Q\right\}$$

Determine the indexes t_1 and t_2 so that $P(t_1) = p$ and $Q(t_2) = q$. If $\vartheta_1 \leq \vartheta_2$, set $l = p$. Otherwise, set $l = q$. The non-basic variable x_l enters the basis.

Step 3 (Update)

Set $B[r] = l$. If $\vartheta_1 \leq \vartheta_2$, set $P \leftarrow P \setminus \{l\}$ and $Q \leftarrow Q \cup \{k\}$. Otherwise, set $Q(t_2) = k$. Using the new partition *(B,N)*, where $N =$ *(P,Q)*, update the basis inverse

$(A_{\overline{B}})^{-1} = E^{-1}(A_B)^{-1}$ and the vectors x_B, w and s_N. Also, update $d_{\overline{B}}$ by $d_{\overline{B}} = E^{-1}d_B$, where E^{-1} is computed by 2.1 and compute s_0. Go to Step 1.

In order to solve general LP problems, we used the Two Phases method. This method for EPSA was initially presented in [112]. The problem of Phase I is constructed by the following procedure. First an artificial variable $x_{n+1} \geq 0$ is added to the problem (LP.3). The coefficients of x_{n+1} are given by the relation $g = -A_B e$, where $e \in \mathbb{R}^m$ is a column vector of ones. The artificial problem which is solved in Phase I has the form

$$\begin{aligned} \min\ & x_{n+1} \\ \text{s.t.}\ & Ax + gx_{n+1} = b \\ & x, x_{n+1} \geq 0 \end{aligned} \tag{LP.3}$$

The first iteration in Phase I inserts the artificial variable x_{n+1} into the basis. The leaving variable is selected by

$$x_k = x_{B[r]} = min\{x_{B[i]} : i = 1, 2, ..., m\}$$

Now, the new partition is $B[r] = n + 1$ and $N[n + 1] = k$. Obviously, the corresponding basic solution is feasible, since $x_{B[r]} = -b_r > 0$, $x_{B[i]} = b_i - b_r \geq 0$, $i \neq r$ and $x_j = 0, j \in N$. In Phase II, the original problem (LP.1) is solved.

In order to solve general LP problems, we used EPSA in both Phases. Specifically, EPSA is applied to the problem (LP.3) of Phase I. EPSA exits Phase I if (i) the artificial variable x_{n+1} leaves the basis and at the same time, direction d_B crosses the feasible region, or (ii) direction d_B does not cross the feasible region after x_{n+1} leaves the basis. In this case, EPSA must reach (LP.3) to optimality in order to obtain a feasible solution for the problem (LP.1). Using the following relation, EPSA checks if the current direction d_B crosses the feasible region.

$$\beta = max\left\{\frac{x_{B[i]}}{-d_{B[i]}} : x_{B[i]} < 0\} \leq \alpha = min\{\frac{x_{B[i]}}{-d_{B[i]}} : d_{B[i]} < 0\right\}$$

where 1 $\leq i \leq m$.

2.3 Complexity and Performance Analysis

The community of OR analysts and researchers is interested not only in the goals which are to be achieved with LP algorithms, but also in the efficiency of these algorithms, i.e. how many resources they would require. By “resource”, we refer to any kind of original sources (in terms of time, hardware, software, human resources, etc.) that an algorithm would need to perform as expected and find the optimal solution for a specific LP problem. At this point, the concept of complexity comes into play. Complexity analysis is a central area of research in theoretical computer science. There are three different approaches to analyzing algorithms; best case, worst case and average case. These approaches can explain an LP algorithm’s resource usage at least, at most and on average, respectively and they represent the minimum, maximum and average number of steps that the algorithm needs, in order to process the input data of x elements. In real life LP problems, we mostly examine the worst-case scenario as the respective execution time is important, so that we can guarantee that the algorithm will, at least, finish on time or finish after a certain number of iterations. It is easy to imagine what would happen, if the worst-case analysis prevailed when decisions about algorithm efficiency had to be made. PSA would be one of the first "victims" since its theoretical complexity has proved to be exponential (despite the fact that it performs very well in practice, especially in problems of small or medium size). If it was only for theoretical performance, Simplex would have never made it so far.

PSA starts with a feasible basis of the polyhedron $P = x|Ax \leq b, x \geq 0$ where $A \in \mathbb{R}^{m\times n}$, $x \in \mathbb{R}^n$ and $b \in \mathbb{R}^m$ and by using pivot operations it preserves feasibility of the basis and monotonicity of the objective value. The pivot operation is defined by the pivoting rule that will be applied and this is of vital significance for the performance of PSA. Efficient pivoting rules will result to fast convergence to the optimal solution, while poor pivoting rules may lead to worse execution times or even no solution for the examined LP problem. According to Maros et.al., the pivoting rule is one of the main factors that will eventually determine the number of iterations that PSA needs [71]. To understand the significance of the pivoting rules for the behavior of PSA, we should refer back to the study of Klee and Minty [65], who, in 1972, proved that Dantzig's largest coefficient pivoting rule [35] performs exponentially on some specially constructed problems. This finding classified PSA as an exponential-time algorithm, rather than a polynomial-time one. There are numerous linear problems which cause Simplex to perform an exponential number of iterations, going from one vertex of the feasible region to a better one and then to a better one, and so on and so forth, for an exponential number of times. Nevertheless, the computational improvement of simplex-type algorithms still remains a topic of great interest. The average case behavior as polynomial was proven by Borgwardt in 1982 ([22] and [23]) using a probabilistic model. A thorough presentation of the plethora of pivoting rules for simplex-type algorithms can be found in [110] by Terlaky and Zhang, while a computational study on eight different pivoting rules for the Revised Simplex Algorithm has been conducted by Ploskas and Samaras in [86], extending Thomadakis' work on the topic [111]. The following table (2.2) presents the computational complexities of the Simplex, Ellipsoid and Interior Point algorithms, respectively. In this case, let n be the number of variables in an LP problem and L, the number of bits necessary to represent the input data.

Remarkably contributing studies about the theoretical behavior and performance of EPSA, among other LP algorithms have been conducted in [82], [80], [96], [97] and [81].

Table 2.2: Complexity of linear programming algorithms

	Complexity	
Algorithm	Worst case	Average case
Simplex	$O(2^n)$	$O(n^2)$
Ellipsoid	$O(n^4 L)$	$O(n^4 L)$
Interior point	$O(n^{3.5} L)$	$O(n^{3.5} L)$

Apart from the theoretical performance of LP algorithms, the computational behavior has also been the major focus of numerous studies so far. Spielman and Teng focused on smoothed analysis, starting in [103] by studying the termination phase for LP algorithms, providing an introduction to smoothed analysis and a tutorial on proof techniques that have been used in smoothed analysis. Moving forward in [104], Spielman and Teng continued working on smoothed analysis of algorithms, which continuously interpolates between the worst-case and average-case analyses of algorithms. They measure the maximum over inputs of the expected performance of an algorithm under small random perturbation of that input. The performance is measured in terms of both the input size and the magnitude of the perturbations and show that the simplex algorithm has smoothed complexity polynomial in the input size and the standard deviation of Gaussian perturbations. In 2009, they extended their study in [105], explaining that several algorithms and heuristics work well on real data, despite having poor complexity under the standard worst-case measure. They present smoothed analysis as a step towards a theory that explains the behavior of algorithms in practice. It is based on the assumption that inputs to algorithms are subject to random perturbation and modification in their formation. A concrete example of such a smoothed analysis is a proof that the simplex algorithm for linear programming usually runs in polynomial time, when its input is subject to modeling or measurement noise. Other interesting studies can be found in [76], [102], [96] and [101], using techniques that explore the practical performance of algorithms.

Chapter 3

Predictive Modelling

Predictive Modelling is a mathematical process, which aims to create and validate a model for forecasting future results, based on already known results and measurements. Predictive Modelling may also be referred to as Predictive Analytics, especially when it comes to commercial deployment, with the main question that drives all efforts in this field, being "what if, taking into consideration and analyzing the history background and known past behavior, we could forecast and make provision for the behavior in the future?". Statistics and Statistical Analysis, along with Machine Learning techniques, are the core elements for Predictive Modelling, serving among others, a common purpose, i.e. data reduction and interpretation. Fisher stated in his classic paper in 1922 [45] that "*the object of statistical methods is the reduction of data*". Indeed, a volume of data which may be impossible to be processed by any human mind, should be replaced by quantities which will adequately represent the whole or at least, contain a sufficient amount of information, capable of representing the original data.

Applications of Predictive Modelling exist in our everyday life in such a wide range that may almost be "too common" to notice. From customer relationship management and e-commerce to social networks and health care system ([15], [69]), Predictive Modelling has brought a remarkable added value with a plethora of tools and techniques that are utilized, based on the nature of the examined data and the model that is to be formed. Companies in the retail market, banks, social networks as well as telecommunications companies maintain a vast amount of information about how people live their everyday lives, since they do keep records of where we live, what we buy and how we spend our money, what we like and dislike, how often we tend to visit particular sites and what we post online. Back in 2000, perhaps this feedback of data was provided to companies through our electronic footprints or online purchases, however, nowadays, there is an "ocean" of data coming from numerous directions. This information "flood" is a powerful tool at the hands of companies and quite often governments, not only to understand behavior and tendency of people towards products, services and topics of public interest, but also to "predict" future behavior and reactions. This tool will enable maximization of the value in the relationship that has been established with people, which for a commercial organization may result to a further profit maximization and for a government to people's additional support and even wider public acceptance. Of course, as more and more information is held by organizations at a worldwide level, the concept of privacy, data security and ownership, anonymity and decision making has become rather concerning. By decision making, it is meant what kind of decisions will be taken and will these be taken by humans or by an automated process, based on the Predictive Modelling results? Such a discussion about ethics in Predictive Modelling is vital, especially when people's data are involved; insightful details about this topic can be found in Steven Finlay's book, i.e. "Predictive Analytics, Data Mining and Big Data" ([43]). A quite representative example of enhancing Predictive Modelling in business processes on a corporate level is how companies worldwide use customers' data to proceed with the, so-called, "upsales" step. That is, examination and perception of the customer's needs, requests and actions over time so that the service or product provider can already forecast the next inquiry and make a particular suggestion or offer, even before it is asked to do so. Such an action would result to an even higher level of the customer's engagement and commitment to the provider who successfully utilizes Predictive Modelling techniques. However, before we initiate a predictive model creation process, we should be able to respond affirmatively to the following three questions:

1. Will we be able to perform our current process, to which the model is related to, more efficiently?

2. Will we reach a better decision making process?

3. Will we be able to do something new that we have not been able to do so far?

As said, organizations gather data from various sources, consequently the data is, most of the times, unstructured or quite complex for the human brain to analyze, especially when results are requested back in a rather short period of time. Therefore, predictive modelling tools, deployed by computer software programs, are used so that the historical data can be analyzed and some patterns can be identified. Based on this analysis, the model will most probably stand as an assessment of the expected behavior which is likely to occur in the future. A predictive model can understand how different fragments of data are linked to each other and can be interpreted, based on the technique that has been followed. In Linear Programming, there are a few studies that have utilized Predictive Modelling techniques, such as the one from Rao and Rawlings in [91], where they explore the practicality of model predictive control, which is partially limited by the ability to solve optimization problems in real time. To our knowledge, Predictive Modelling has not been extensively utilized in analyzing the practical performance of LP algorithms; adding value to the novelty of the current thesis, as this has been described already in 1.Further below, we are describing two of the most commonly used techniques in Predictive Modelling, which have been applied during the study for the current thesis as well; Regression Analysis and Artificial Neural Networks.

3.1 Regression Analysis

Regression Analysis (RA) is a statistical method, which examines the relationship between 2 or more variables, called the **dependent** variable and the **independent** variable(s); the latter may be one or more. The variables may be referred to as "output" and "input" variables, respectively, however, we will use the terms "dependent" and "independent" for the rest of this section. Regression Analysis can be distinguished to simple regression and multiple regression, based on the number of independent variables that are engaged during the analysis; it can be utilized not only to evaluate the strength of the existing relationship between variables, but also to predict the future relationship between them. Applications of Regression Analysis in modern business and economics are quite prominent, since the method is vastly implemented in Predictive Modelling and Decision Making systems. Regression Analysis offers the capability of forecasting potential opportunities and identifying risks, while it can also be used to optimize business process on a corporate level. It supports the reduction of huge amounts of data to interpretable and actionable information, which subsequently supports a faster, smarter and more accurate decision making process.

3.1.1 Regression Model and Evaluation Metrics

A simple regression model is the linear regression model, which implies that there is a linear relationship between the dependent and independent variable. This linear relationship is represented by a line, i.e. the Regression Line, which is found to be closer to the data points than other lines, according to a specific mathematical criterion, and can be calculated with the Least Squares method [26], [106]. The distinctive feature of the Least Squares Regression Line is the vertical distance between the data points and the regression line, which is the smallest possible. The Least Squares method, and thus the regression line, are named as such because the best line of fit is the one that minimizes the sum of squares of the errors (i.e. variance). This may be difficult to visualize, however, the main purpose is to find the equation that fits the data points as closely as possible. A simple linear regression model is represented by equation Eq.1 below:

$$Y_i = (b_0 + b_1 X_i) + \varepsilon_i \tag{Eq.1}$$

where Y_i is the dependent variable, b_0 represents the intercept with the vertical axis, b_1 is the slope of the regression line and X_i is the independent variable. The value of ε_i represents the amount of residual. Generally, the residual value is calculated as the difference between the observed value and the estimated value of the regression model. Small residuals correspond to a good fit of the regression model, while the opposite implies that the model does not fit well to the examined data. The entities b_0 and b_1 are characterized as "Regression coefficients" and are necessary for the Least Squares method, since we need to identify their values and thus, the regression line, so that the following quantity (Eq.2) is minimized.

$$\sum_i \varepsilon_i^2 = \sum_i (Y_i - b_0 - b_1 X_i)^2 \tag{Eq.2}$$

The following amounts of Total Sum of Squares, Residual Sum of Squares and Model Sum of Squares (Eq.3, Eq.4 and Eq.5, respectively) contribute to the evaluation of good fit of the regression line to the examined data. Sum of squares (SS) indicates the deviation from the mean and is calculated as the sum of the squares of the differences from the mean [67].

$$SS_T = \sum_i (Y_i - \overline{Y})^2 \tag{Eq.3}$$

$$SS_R = \sum_i (Y_i - b_0 - b_1 X_i)^2 \tag{Eq.4}$$

$$SS_M = SS_T - SS_R \tag{Eq.5}$$

A rather useful and simple interpretation of the Sums of Squares would be that SS_T and SS_R represent the deviation of the examined data from the "worst model" (mean value) and the "best model" (line), respectively, while SS_M denotes the difference between the "worst model" and the "best model". The bigger the value of SS_M, the more important the contribution of the model to the prediction of the independent variable Y. The smaller the value of SS_M, the lower the contribution of the model to the improvement of the "worst prediction" of the mean value. The quality of the model fitting can be calculated as the percentage of the improvement in prediction (Eq.6), which is introduced by the model. This implies the percentage of the independent variable's volatility, which is explained by the model and is named "coefficient of determination", corresponding to the square of Pearson's coefficient [83], [107].

$$R^2 = \frac{SS_M}{SS_T} = \frac{SS_T - SS_R}{SS_T} = 1 - \frac{SS_R}{SS_T} \tag{Eq.6}$$

R-squared (R-Sq) or R^2 is a metric that defines the good fit of a statistical model to the examined data, therefore the bigger its value is, the better fitting the model has. Although the significance of this coefficient is crucial for all regression models, we should always take into consideration its two main drawbacks. It has been reported that the R-squared value increases, every time a new parameter is added in the examined model. This is one of the reasons why the R-squared value alone cannot guarantee the good fit of a model. Moreover, the metric may be affected by random noise of the dataset, especially in cases of large number of parameters and higher order polynomials in the examined model. This problem is known as "over-fitting" and produces misleadingly high values for R-Sq, making the model unsuitable to be used for predictions [90]. An additional measure for evaluation of the regression model is the F-test, which is calculated by the Mean Sums of Squares as shown in equations Eq.7, Eq.8 and Eq.9 below. Mean squares (MS) amount is calculated by dividing the respective sum of squares by the degrees of freedom. This metric is an estimate of the population variance. In a regression model, the mean squares are used to determine whether the parameters of the model are significant [67].

$$MS_M = \frac{SS_M}{DegreesOfFreedom} = \frac{SS_M}{NumberOfVariables} \tag{Eq.7}$$

$$MS_R = \frac{SS_R}{DegreesOfFreedom} = \frac{SS_R}{n - NumberOfRegressionCoefficients} \tag{Eq.8}$$

$$F = \frac{MS_M}{MS_R} \tag{Eq.9}$$

Degrees of freedom is the number of values in the final calculation of a statistic that are free to vary. The concept of this metric was introduced by Student in 1908 [108], while the specific naming belongs to Fisher, who used it some years later in 1922 ([44]). The significance of F and the corresponding P values of a regression model derives from the fact that, although R^2 provides an estimate of the strength of the relationship between a regression model and the dependent variable, it does not provide any formal hypothesis test for this relationship. The F-test is the metric which determines if this relationship is statistically significant or not. Therefore, if the P value for the overall F-test is less than the applied significance level, then the specific regression model has statistically significant predictive capability [39]. In case the statistical significance is < 0.001, we can safely conclude that the model highly contributes to the prediction of the independent variable.

Moreover, the statistical significance of the regression coefficients is crucial for the validity of the regression model and the evaluation of its quality. More specifically, the value of b_0 defines the change upon the dependent variable if the respective independent variable changes by one unit. To examine the statistical significance of b_1 we apply a t-test with significance < 0.05 [108]. The standard error of the coefficient (*SE Coef*) is the standard deviation of the estimate of a coefficient in a regression model. It measures the precision of the model's estimation about the coefficient's unknown value. *SE Coef* value is always positive and the smaller it is, the more precise is the estimate. The division of the coefficient by the respective standard error results to a specific t-value (T). This value is also known as t-statistic and measures the likelihood of the actual value of the parameter being not zero. The larger the absolute value of T, the less possible that the real value of the parameter is zero. If the probability value (P) related to the t-statistic is less than the determined level of significance, we conclude that the coefficient is significantly different from zero ([39]).

An introduction to multiple regression would be the extension of the linear model with more than one, independent variables (Eq.10). In case of 2 independent variables, the regression line's equation extends to a plane, while in case of more than 2 independent variables, to a hyperplane.

$$Y_i = (b_0 + b_1 X_i + ... + b_k X_k) + \varepsilon_i \tag{Eq.10}$$

In multiple regression, the amounts of SS_T, SS_R and SS_M are calculated in a more complicated way but their meaning and significance remain the same. The fact that multiple independent variables are involved in the regression makes it imperative to calculate a coefficient of multiple correlation that reveals the strength of the relationship of the dependent variable with all independent ones. The value of R^2 is calculated similarly to the simple linear regression, as the volatility percentage of the independent variable, which is actually explained by the model.

One fundamental issue that needs to be clarified before initiating the creation of a multiple regression model is how the independent variables will be selected. Taking into consideration, that during examination of a specific dataset, we can use particular attributes and features as independent variables, it is clear that the latter are usually correlated to each other. However, there are several methodologies for the selection of the most appropriate variables for the regression model, such as Forced Entry, when all variables enter the model simultaneously, Stepwise, where the order of variables is defined by mathematical criteria, Forward, Backward, etc [29]. In general, the researcher should have a good understanding of the dataset that needs to be examined, so that the most appropriate methodology can be selected. Another matter that concerns researchers is the model's accuracy, since it is crucial that the model can achieve a good fit to the data and its behavior is not affected by a few extreme instances. Such instances, called "outliers", are cases which differ significantly from the rest of the dataset. They can stand as a "diagnostics" measure of the model's fitting, since they may have a great impact on the regression coefficients' values. Outliers can be detected by their large residual values. For better comprehension and comparison of residuals, these can be standardized by dividing their value by their standard deviation. Standardized residuals with an absolute value > 3 may be concerning, while in case over 1% or 5% of the standardized residuals is > 2.5 or > 2, respectively, then this is a indication of poor fitting. Other measures of checking for outliers are the Adjusted Predicted Value, which is calculated for each case separately, by removing it from the sample and estimating it with the new regression model that is formed; Cook's distance, which is a measure of overall impact of a data point on the model (e.g. data points with a value > 1 may be concerning); etc [33], [34]. Elaborate explanation of additional metrics which are examined for the selection of best-fitting regression models is provided below:

1. Adjusted R-squared (*R-Sq(adj)*): adjusted coefficient of determination. This metric proves to be useful during the comparison of models with different number of predictors (i.e. independent variables), since it is adjusted according to the number of predictors in a model. In more detail, its value increases only if a new predictor improves the model more than it was

expected by chance and decreases when a predictor improves the model less than expected by chance. Interestingly enough, its value turns out to always be lower than the R-squared value [90].

2. Predicted R-squared (*R-Sq(pred)*): the predicted R-squared explains the predictability of a regression model, i.e. predicting responses for new observations. A regression model that seems to fit the original data, may not be capable of providing valid predictions for new observations. Similarly to adjusted R-squared, predicted R-squared is always lower than R-squared and there are times, when even a negative value has been observed. Perhaps the most important benefit of this metric, is that it can "prevent" researchers from using models which over-fit. Since it is rather impossible to predict random noise, the value of predicted R-squared would drop, in case of an over-fitted model. Kutner et al. explain that if the predicted R-squared value in a regression model is much lower than its regular R-squared value, this may indicate that the model is most probably over-fitted and cannot be used for predictions [67].

3. Standard error of regression (S): it measures the units of the "response" (dependent variable) and represents the standard distance between data values and the estimated regression line. The lower the value of S, the better the predictability of the model. When comparing different models, the model with the lowest S value reflects the best fit [54].

In Chapter 5, where the predictive models of this study are presented, respective probability plots are included after the regression model equation and statistical details. The P–P plot is a normal probability–probability (P–P) plot based on the standardized residuals. In this study, the X axis represents the observed cumulative probability (observed cum prob), which is based on the percentiles in the frequency distribution of the residuals. The Y axis, which represents the expected cumulative probability, is based on the standardized residual (Z-score) and on the computation of the cumulative density from the normal distribution. If the residuals are normally distributed, then the values should fall exactly on the diagonal line. A systematic deviation from the diagonal line may indicate a positive skewness of the distribution. This means that the right side tail of the curve, if this was depicted in a histogram, is longer than the left side tail and the mean is greater than the mode. Skewness is actually the asymmetry of a distribution and can be quantified to measure the extent to which this distribution is distorted and how much it differs from a normal distribution.

Having described the basic concept of Regression Analysis and after presenting some fundamental metrics that enable the evaluation of regression models, we are now concluding this section by summing up with the following three properties; autocorrelation, heteroscedasticity and multicollinearity. These properties are not expected to occur in a good-fitting model.

1. **Autocorrelation** is identified when the residuals of a regression model are not independent from each other. Autocorrelation can be detected not only from graphs as explained above, but also from other statistical measures, such Durbin-Watson metric [40]. Autocorrelation can be eliminated, by performing appropriate transformations in variables of the model.

2. **Heteroscedasticity** occurs when the variances of the residuals in a regression model are not equal. Similarly to autocorrelation, this issue can be identified through graphical representation of the residuals and can be overcome with transformations of data.

3. **Multicollinearity** arises in multiple regression analysis, where two or more independent variables are highly correlated to each other. The problem can be detected through statistical measures and the most efficient solution is usually to remove the affected variables from the regression model.

3.2 Artificial Neural Networks

Artificial Neural Networks (ANNs) concept is very well known to Artificial Intelligence (AI) researchers, however it may seem a bit complicated to perceive at a glance. Artificial Neural Networks stand as one of the main tools in the field of Machine Learning and as the word "neural" in their naming suggests, they are systems designed to replicate the learning process of the human brain. An ANN consists of a set of connected nodes called "artificial neurons", which actually represent the neurons of a biological brain. The response time of human biological neurons can be estimated in milliseconds; still, the human brain can make difficult and complex decisions, incredibly fast. The computational capabilities of the human brain and the information it can store and maintain are organized similarly to a Parallel Distributed Processing (PDP) system [93], [94]. Similarly to a biological neural network which is continuously learning and gaining knowledge and understanding, based on experience, ANNs are supported by mathematical algorithms that work together, calculate input data and produce an output. The outputs support the ANN to learn and improve its accuracy. The connections between neurons in ANNs (often called "edges") transmit signals to other neurons, similarly to the synapses in a biological brain. A neuron will process a signal it receives as input and then, transmit it as output to the neurons it is connected to. In ANNs, this signal is a real number while the output is calculated by a non-linear function (i.e. activation function) based on the sum of the neuron's inputs. Neurons and edges are characterized by a weight that fluctuates as the learning process progresses and its value increases or decreases depending on the strength of the signal that is transmitted through the respective connection. A basic representation of a neuron is shown in the following figure 3-1.

A fundamental aspect of the ANNs' structure is that the neurons are usually set up in layers to serve different transformations per layer on their inputs. A signal may "travel" from the first (input layer) to the last (output layer) after traversing

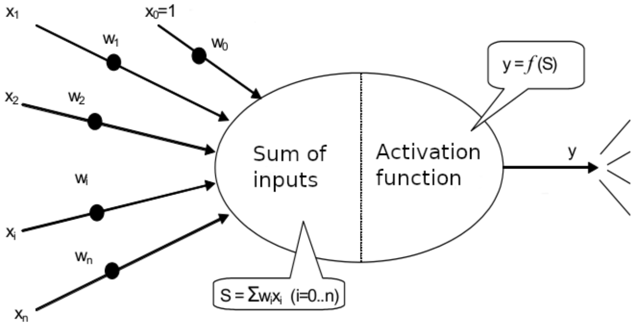

Figure 3-1: Structure of an artificial neuron

the intermediate (or "hidden", as they are also called) layers multiple times. The concept of an ANN consists of three main steps, i.e. a) for each neuron in a layer, the input is multiplied to the respective weights, b) for each layer, all products of input × weights of neurons are summed together and c) the activation function is applied on the result to compute the new output. It is important to clarify that non-linearity of the activation function is crucial so that the ANN can model complex, non-linear problems. There are several types of activation functions, such as:

- Sigmoid, which produces an S-shaped curve ($\frac{1}{1+exp(x)}$). It is not linear but still it cannot usually detect slight changes within inputs; as a result, variations in inputs yield similar results.

- Hyperbolic Tangent (*Tanh*), which is superior than Sigmoid ($\frac{1-(exp(-2(x))}{1+exp(-2x)}$). Nevertheless, it cannot detect relationships better and is generally slower at converging.

- Rectified Linear Units (*ReLu*), which converges faster (when compared to the two previous functions), optimizes and produces the objective value quicker. It is the most popular activation function used within the hidden layers.

- Softmax, which is used in output layer, mainly because it can reduce dimensions and can represent categorical distribution.

The structure of an ANN where neurons are organized in multiple layers is shown in figure 3-2.

Neurons within layers may be fully or partially connected, while one layer connects only to the neurons of the exact preceding and exact following layers. Two layers are considered to be fully connected, when every neuron in one layer connects to every neuron in the next layer. The layer that receives external data is the input layer, while the layer that produces the ultimate result is the output layer. In between, we may have zero or more hidden layers. Single layer and unlayered networks are also commonly used. Between two layers, multiple connection patterns can be identified. Moreover, there may be cases when a group of neurons in one layer connects to a single neuron in the next layer, resulting in reduction of the number of neurons in the latter.

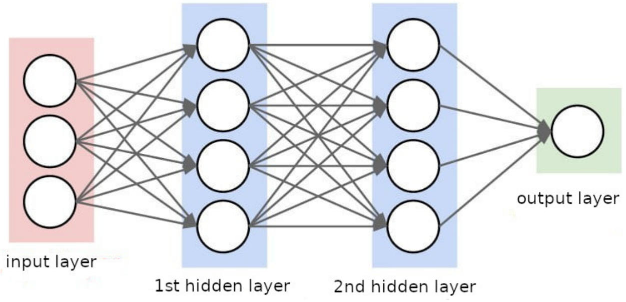

Figure 3-2: Structure of Artificial Neural Networks with 2 hidden layers

ANNs with only such kind of connection form directed acyclic graphs and are known as *feedforward* networks. This is the most basic type of neural network in which information travels in only one direction (i.e. from input to output). Additionally, there are networks which allow connections between neurons in the same or previous layers and are known as *feedback* or *recurrent* networks. They are more widely used, due to the fact that data can flow in multiple directions, and since they have greater learning abilities, they are deployed for more complex tasks, e.g. even language recognition. Before discussing about the applications of ANNs and how their learning process is formulated, we are summarizing upon their main components, which could be presented as follows:

- Neurons
  Neurons receive an input signal, combine the input with their internal state and an optional threshold using an activation function, and produce an output signal. The activation function is significant for the development of any ANN as it provides a smooth, differentiable transition while input signal values change, i.e. a small change in input produces a small change in output.

- Connections and weights
  Connections provide the output of a neuron as input to adjacent neurons. Each connection is assigned a weight that represents its relative importance, while neurons can have multiple input and output connections.

- Activation function
  The activation function is responsible for the computation of the input to a neuron, taking into consideration the outputs of the preceding neurons and their connections as a weighted sum.

As explained already, the initial purpose of ANNs was to achieve problem solving in a way that would be the same as the one of human brain. Over time, research focus was shifted to more specific tasks, deviated from biology and spread to a variety of tasks, including but not limited to pattern recognition (speech, sound, image, etc.), machine translation [17], social network filtering and medical diagnosis [24]. The "door" to this new area of Artificial Intelligence was first opened by McCulloch and Pitts, who, in 1943, created a computational model for neural networks [73]. However, the major breakthrough happened two decades later when backpropagation came into play, allowing networks to adjust their hidden layers when the outcome was not what had been expected. One more significant step for the progress of ANNs was the introduction of “deep learning” concept, which implied that in case of different layers in a multilayer network, the layers may extract different features (due to different transformations performed in each layer, as stated already) until the network can recognize what it is searching for ([119], [98], [99]).

3.2.1 Learning process

ANNs are designed to recognize and identify patterns in data, while the tasks they can perform include (but are not limited to) classification (i.e. classifying datasets into predefined classes), clustering (i.e. classifying data into different undefined categories), and prediction (i.e. using past events to forecast future ones). Artificial Neural Networks are widely utilized for the solution of problems which have non-predictable behavior and may be not clearly perceived. Applications of ANNs through classification are common in fields of medicine, defense, agriculture and business economy; pattern recognition is particularly useful in banking, information technology and telecommunications, while prediction methods are vastly used in business, social networking and online, personalized advertising. The **learning process** of ANNs has a clear resemblance to the learning process of biological brains that learn from experience and require data. Similarly to a gradually increased efficiency when a human performs the same process multiple times, an ANN becomes even more accurate as the amount of data it takes as input increases. Before setting up the training process of an ANN, the dataset is typically split into two sets; a) the training set, which helps the network establish weights between its nodes and b) the test set, which examines if the network can successfully convert the input signal into the desired output. The learning process of ANNs has proved to be quite a challenge on a technical level. Reason is that the amount of time that is necessary to train networks, subsequently requires remarkable amount of resources and computing power. There are three widely known methods for training an ANN, listed below, which represent the process of adjusting the weights of the network so that, given a particular vector as input, the network can produce a vector as output:

- Supervised Learning
- Unsupervised Learning
- Reinforcement Learning

In practice, most ANN applications use Supervised Learning, which is supported by numerous algorithms. In this case, the input and respective output data is provided to the network for training, in order to get a desired output for a specified input. One of the most characteristic examples of a Supervised ANN is the spam filters supplied by our mail service providers. On a training level, the input to the ANN is actually a set of words in the body of the e-mail, which are considered to be content of spam emails, while the output is marking email as either spam or not spam. The Unsupervised ANNs are more complex, since they attempt to understand the data provided as input, on their own. An example could be housing lists of Airbnb online platform which get grouped together in neighborhoods, so that the users can navigate more easily. Reinforcement learning (RL) involves software agents which are designed to take actions towards solving a problem with the purpose of maximizing a cumulative reward. In contrast to the other methods of learning, reinforcement learning does not need specifically labelled input and output values but it works through exploration of the possible solutions and exploitation of the knowledge the network has gained so far [60].

Despite the actual learning process, it is true that the most difficult task is the **interpretation** of the ANNs results. They are often characterized as "black boxes" in which the user feeds in data and receives answers. Indeed, although the users may be able to fine-tune the answers, they cannot have clear access to the exact decision making process. This topic is an active "work in progress" for a plethora of researchers and its impact is expected to become even more profound as ANNs continue getting a bigger role in our everyday lives. To be able to interpret the results generated by an ANN in a meaningful and practical way, the researcher should have a clear understanding of the dataset that is available and the tools (i.e. algorithms) which can be used for training of the ANN. It is a common practice testing multiple learning algorithms and experimentally determine which one works best on the problem at hand. Another approach is trying to fine tune the performance of a learning algorithm, although this may prove to be a very time-consuming process. Therefore, taking into consideration a limitation in available resources, it may often prove "wiser", although is still questionable by many researchers, to spend some more time collecting a sufficient amount of training data than attempting to adjust the learning algorithm itself. Commonly used algorithms in Supervised Learning is the Delta rule, Back Propagation, Competitive Learning and Random Learning. While building an ANN, a researcher takes into consideration several metrics and aspects which facilitate the evaluation of the generated model's accuracy and efficiency. For instance, the model may show an **overfitting** or an **underfitting** behavior instead of a good fit on the examined dataset. Overfitting occurs when the generated model has the ability to predict the data it was trained on very well, but it cannot accurately predict new data and thus, cannot be generalized. Underfitting occurs when a model cannot predict well not only new data, but even the data it was trained on. While overfitting is often an indication of an extremely complicated model, which even "captures" the noise of our dataset, it can be prevented by fitting multiple models and using validation or cross-validation to compare their predictive accuracy values on test data. Similarly, underfitting is a strong indication of an excessively simple model. In both cases, the accuracy of the model can be increased by adjusting the number of layers in the ANN, the number of neurons per layer, etc. Books and studies have been published about ANN training, performance evaluation and characteristics, including a more thorough analysis on the topics described above; some indicative examples which proved very useful for the current study are [59], [100], [52], [115] and [13].

Chapter 4

Dataset

One of the primary steps in any kind of scientific work, which is of vital importance for every researcher, is to examine, understand and be able to interpret the dataset which is subject to the respective analysis. Data can provide significant knowledge and insight to any topic under examination, as long as the researchers can explore and understand the available datasets. Otherwise, data will only stand as measured values and observations instead of getting transformed to valuable information. A dataset may consist of attributes of different nature; variables may be quantitative or qualitative, categorical or nominal, and may be analyzed and examined in different ways, depending on the problem that needs to be solved and the tools, methods and approaches that are most suitable every time. Thus, before diving into modelling, a researcher takes a close and careful look at the available data. Further elaboration on the examined dataset may reveal noise that should be excluded or minimized, outliers which may have an impact on the analysis and patterns or correlations between the examined variables that need to be taken into consideration.

In our study, we are dealing with numerical values which may need to be transformed before being analyzed. This chapter includes a thorough description of the datasets, which are used for the purpose of our study and are presented based on the respective algorithms that are studied. Our datasets consist of linear problems which a) have been created through a random linear problem generator and b) belong to widely used benchmark problem libraries. Benchmark problems are publicly available LP problems, which are, most of the times, hard to solve. Some of the most widely used benchmark libraries, which have also been utilized in our study, are listed below:

- Netlib LP problems [1]
  Kennington LP problems [2]
  Netlib LP problems (infeasible) [3]
- Mészáros LP problems
  Miscellaneous [4]
  Problematic [5]
  Stochlp [6]
- Mittelman LP problems [7]

Netlib test set for Linear Programming is actually a collection of real-life LP problems from a variety of sources. In "Linear Programming Using MATLAB®" [87], Ploskas and Samaras explain that, over the years, Netlib has become a standard set of problems for testing and comparing algorithms and software in LP field. However, as stated also by Ordóñez and Freund [76], almost 71% of Netlib LP problems are ill-conditioned and may introduce obstacles in calculation of numerical values. Moreover, a remarkable number of Mészáros and Mittelman LP problems are very degenerate and hard to solve. These conditions are a powerful motivation for the use of the described libraries in order to examine EPSA, IPM, Primal and Dual Simplex algorithms and generate trustworthy models for their performance.

Prior to a more thorough presentation of our datasets, we should clarify that the files generated by the random LP problem generator are in MAT format (as produced by MATLAB), while the LP problems of the aforementioned libraries are in MPS format (widely accepted format for defining LP problems and considered as input for numerous of LP solvers). Closing this chapter, we are going into further details about the computing environment that was used in order to solve the aforementioned problems with the examined algorithms.

4.1 Datasets for EPSA

4.1.1 Random Linear Problems

Exterior Point Simplex Algorithm was examined upon 6,780 sparse linear problems which were randomly generated. Sparse linear problems exist in a plethora of applications, thus our goal has been to examine how EPSA performs when it is applied for their solution. The expected number of feasible vertices of a random linear problem is less or equal to 2^m, where m is the dimension of the constraint set [21]. The randomly generated sparse linear problems are subject to inequality constraints and they are optimal, meaning that the algorithm reaches an optimal solution after a specific number of iterations. These problems have been created using a generator, that was specifically designed to generate random optimal LP problem instances [79]. The planes of the constraints are tangent on a sphere, so that its center is feasible. Also, these problems have a closed feasible region that is a closed polyhedron. We used an LP problem generator that takes as input the number of constraints and decision variables (m and n, respectively), the density of the nonzero entries of matrix A ($0<$ *density* ≤ 1) and the seed number for the random number generator. The ranges of the values used in matrices A, b, and c, in order to create the linear problems of our study are shown in Table 4.1 below. Based on existing bibliography material and previous work, the lower and upper bounds of values in matrices A, b and c have been set to 10, 10, -300 and 400, 100, 900, respectively. The entries of A, b and c were randomly generated with Matlab *sprand* function.

Table 4.1: Value ranges of LP problems

A	[10 400]
b	[10 100]
c	[-300 900]

As stated earlier, one of the values that the random generator takes as input is that of density. The density has been formed by using Matlab *rand* function with an upper limit of 30%, due to the fact that we are especially interested in examining the algorithm's behavior when dealing with sparse LP problems. The LP problem characteristics examined for EPSA are shown in the list below.

- m: number of linear constraints
- n: number of variables
- *sparsity*: problem's sparsity
- *nnz*: number of nonzero elements in matrix A
- L: data length (bit length)
- *cond(A)*: condition of matrix A

Apart from the first two characteristics (which are well-known LP attributes), the rest have a significant contribution to the algorithm's performance and overall efficiency as well. The respective values of the characteristics above were randomly formed, except for the benchmark LP problems which are described further below. Some descriptive statistical information about the above mentioned characteristics is presented in Table 4.2.

Table 4.2: Characteristics of linear problems

	m	n	*sparsity* (in %)	*nnz*	L	*cond(A)*
min. value	1,122	1,088	71.7%	127,970	3,478,842	24.9788
max. value	12,470	12,301	97.47%	27,051,872	311,324,768	573,956,480

The quantities of m and n were formed by using Matlab *rand* function, within the bounds shown in Table 4.2 above. The number of nonzero elements indicates the elements within matrix A which are not zero. Data length L is the number of bits which are required in order to represent integer data of A, b and c.

Regarding *cond(A)*, its number is close to 1 when the data within matrix A is of good condition. The condition number can be used to predict how ill-conditioning affects the computed solution of a LP. The above mentioned values are uniformly distributed, according to the *sprand* and *rand* functions of Matlab. A more detailed presentation of the measured values for the randomly generated LP problems (6,780 in total) could not be supported in printed form, however it is available in electronic format (i.e. Microsoft Excel file, $EPSASparseALL.xlsx$).

4.1.2 Linear Programming Benchmark Problems

Apart from the previously described randomly generated LP problems, 60 benchmark problems were examined as well (Table 4.4), coming from the Netlib library [8] and Mészáros miscellaneous LP collection (Linear Programming Test Problems [4]). Some descriptive statistical information about the characteristics of the benchmark dataset is provided in Table 4.3.

Table 4.3: Characteristics of linear problems

	m	*n*	*density* (in %)	*nnz*	*L*	*cond(A)*
min. value	7	17	0.05%	41	411	3.1292
max. value	14,310	12,465	34.45%	152,800	178,605,200	2.1027×10^{35}

Table 4.4: Characteristics of the MPS benchmark files

Problem	**m**	**n**	**density**	**nnz**	**L**	**cond(A)**	**niter**
adlittle	56	97	0.0705	383	7,056	9.3747E+02	158
afiro	27	32	0.0961	83	1,095	9.5044E+16	22
agg	488	163	0.0303	2,410	88,832	4.9751E+16	135
agg2	516	302	0.0275	4,284	168,155	1.0680E+25	224
agg3	516	302	0.0276	4,300	168,171	7.7242E+32	229
aircraft	3,754	7,517	0.0007	20,267	28,330,724	2.0809E+04	3,504
beaconfd	173	262	0.0745	3,375	52,616	1.4610E+04	306
blend	74	83	0.0799	491	7,167	4.9210E+17	89
brandy	220	249	0.0392	2,148	59,001	INF	400
cari	400	1,200	0.3183	152,800	636,613	3.1292E+00	1,658
cr42	905	1,513	0.0048	6,614	1,399,865	6.8812E+02	1,842
degen2	444	534	0.0168	3,978	243,841	8.7744E+16	1,278
e226	223	282	0.0410	2,578	69,812	2.1027E+35	645
farm	7	17	0.3445	41	411	4.9293E+02	2
fffff800	524	854	0.0139	6,227	464,488	5.3703E+21	756
israel	174	142	0.0918	2,269	33,749	3.7416E+16	331
jendrec1	2,109	4,228	0.0100	89,608	9,287,885	1.3477E+03	6,858
lotfi	153	308	0.0229	1,078	50,578	4.1496E+07	373
nemscem	651	1,712	0.0034	3,840	1,125,700	4.4628E+01	514
nsic1	451	463	0.0137	2,853	245,545	1.9515E+22	404
nsic2	465	463	0.0140	3,015	247,175	2.9912E+18	522
nsir1	4,407	5,717	0.0055	138,955	27,224,988	4.5551E+20	4,078

Table 4.4: Characteristics of the MPS benchmark files

Problem	**m**	**n**	**density**	**nnz**	**L**	**cond(A)**	**niter**
nsir2	4,453	5,717	0.0059	150,599	27,126,852	3.1966E+20	7,886
p0201	133	334	0.0463	2,056	52,294	2.8742E+02	423
p0291	252	543	0.0167	2,283	143,616	1.4445E+02	102
p0040	23	63	0.0918	133	2,567	6.0275E+03	38
p2756	755	3,511	0.0037	9,692	2,718,092	1.4161E+04	1,211
problem	12	46	0.1558	86	1,232	1.0082E+01	14
rosen2	1,032	2,048	0.0220	46,504	2,251,347	9.5635E+01	3,885
rosen7	264	512	0.0575	7,770	159,181	7.3814E+01	691
rosen8	520	1,024	0.0292	15,538	580,394	1.2165E+02	1,570
rosen10	2,056	4,096	0.0074	62,136	8,613,209	2.3561E+02	3,923
sc50a	50	48	0.0542	130	2,746	4.7169E+01	40
sc50b	50	48	0.0492	118	2,694	7.4264E+01	44
sc105	105	103	0.0259	280	11,518	1.1079E+02	101
sc205	205	203	0.0132	551	42,971	6.3909E+02	249
scagr7	129	140	0.0233	420	20,072	1.0272E+04	130
scagr25	471	500	0.0066	1,554	242,644	2.7013E+09	641
scfxm1	330	457	0.0172	2,589	157,088	1.8342E+18	629
scfxm2	660	914	0.0086	5,183	615,824	1.7107E+18	1,412
scfxm3	990	1,371	0.0057	7,777	1,376,179	1.2020E+18	2,234
scrs8	490	1,169	0.0056	3,182	584,955	1.5015E+17	1,165
scsd1	77	760	0.0408	2,388	63,338	2.1212E+01	837
scsd6	147	1,350	0.0217	4,316	207,014	8.8483E+01	1,848
scsd8	397	2,750	0.0079	8,584	1,108,874	9.9320E+02	4,416
sctap1	300	480	0.0118	1,692	150,945	7.3158E+16	675
sctap2	1,090	1,880	0.0033	6,714	2,076,689	2.8883E+17	2,146
sctap3	1,480	2,480	0.0024	8,874	3,706,126	6.3612E+17	2,327
share1b	117	225	0.0437	1,151	32,368	1.3814E+05	356
share2b	96	79	0.0915	694	10,378	1.5057E+18	258
ship04l	402	2,118	0.0074	6,332	879,180	INF	782
ship04s	402	1,458	0.0074	4,352	605,233	INF	434
ship08l	778	4,283	0.0038	12,802	3,389,137	INF	1,643
ship08s	778	2,387	0.0038	7,114	1,889,137	INF	678
ship12l	1,151	5,247	0.0026	16,170	6,309,794	INF	978
ship12s	1,151	2,763	0.0026	8,178	3,212,891	INF	509
stocfor1	117	111	0.0344	447	14,627	6.3562E+05	85
stocfor2	2,157	2,031	0.0019	8,343	4,411,089	1.0662E+06	1,072
sws	14,310	12,465	0.0005	93,015	178,605,200	3.3301E+18	2,066
zed	116	43	0.1137	567	7,494	5.6907E+05	135

4.2 Datasets for IPM, Primal, Dual algorithms

4.2.1 Linear Programming Benchmark Problems

For the purpose of our computational study on IPM, Primal and Dual algorithms, 295 benchmark linear programming problems were used from the Netlib (25), Kennington (13), Mészáros (217), and Mittelmann (40) libraries. The problems were solved with CPLEX's 12.6.1 [9] primal and dual simplex algorithms and the respective execution time, needed for their solution, was recorded for each problem (in seconds) (Table

4.18). The LP characteristics which were examined in regards to these algorithms and set as input in the respective models are the following:

- m: the number of constraints
- n: the number of variables
- *nnzA*: the number of nonzero elements of the constraint matrix
- *nnzb*: the number of nonzero elements of the right-hand side vector
- *rankA*: the rank of the constraint matrix

The execution time was set as the output of the examined models. Apart from the above characteristics, we also took into consideration the number of variables in the problems after adding slack variables, the density of the problem, the data length (bit length), required in order to represent integer data, as well as the norm of the constraint matrix. However, these characteristics showed no statistically significant contribution to the creation of our models; this will be further documented in the following section.

The values of the described attributes are presented in tables 4.5 to 4.17 below, while the time is presented in Table 4.18. The respective measurements were taken with no limitations regarding decimal places, however the values are rounded down to five decimal places only for printing purposes in the following tables.

Table 4.5: Netlib Optimal

Problem	Filesize	m	n	m-equal	m-inequal	n-slack	densA	nnzA	densc	nnzc	densb	nnzb	normA	rankA
25fv47	373834	821	1571	516	305	1876	0.00806	10400	0.46276	727	0.34957	287	436.91582	818
2q06c	1194691	2171	5167	1507	664	5831	0.00289	32417	0.63035	3257	0.40258	874	3568.32614	2170
80bau3b	1141232	2262	9799	0	2262	12061	0.00095	21002	0.82263	8061	1.00000	2262	567.22442	2237
bnl2	565836	2324	3489	1327	997	4486	0.00173	13999	0.60906	2125	0.35370	822	211.69630	2247
d6cube	1406453	415	6184	415	0	6184	0.01469	37704	1	6184	0.01446	6	703.40803	404
degen3	871149	1503	1818	717	786	2604	0.00902	24646	0.87129	1584	0.39521	594	54.63300	1503
dfl001	1465563	6071	12230	6071	0	12230	0.00048	35632	0.51030	6241	0.27393	1663	15.91692	6071
fit2d	4780670	25	10500	1	24	10524	0.49150	129018	0.85714	9000	0.56000	14	17513.23301	25
fit2p	2308581	3000	13525	3000	0	13525	0.00124	50284	0.77634	10500	0.50000	1500	9377.30554	3000
greenbea	1050444	2392	5405	2199	193	5598	0.00239	30877	0.11508	622	0.03595	86	106.22474	2389
grow22	303275	440	946	440	0	946	0.01983	8252	0.06977	66	0.00000	0	2.51231	440
maros-r7	4892894	3136	9408	3136	0	9408	0.00491	144848	0.66667	6272	0.99872	3132	3.40322	3136
nesm	639486	662	2923	568	94	3017	0.00687	13288	0.23948	700	0.81873	542	62.00721	608
perold	222857	625	1376	495	130	1506	0.00700	6018	0.00581	8	0.39680	248	52658.04988	625
pilot4	187371	410	1000	287	123	1123	0.01254	5141	0.00400	4	0.46341	190	59355.37334	410
pilot87	2451962	2030	4883	233	1797	6680	0.00738	73152	0.13352	652	0.13941	283	1002.55009	2012
pilotnov	464041	975	2172	701	274	2446	0.00617	13057	0.03315	72	0.40000	390	11504487.26651	969
qap08	274914	912	1632	912	0	1632	0.00490	7296	0.61765	1008	0.01754	16	6.98743	912
qap12	1446230	3192	8856	3192	0	8856	0.00136	38304	0.67073	5940	0.00752	24	8.72518	3192
qap15	3603050	6330	22275	6330	0	22275	0.00067	94950	0.70707	15750	0.00474	30	9.82907	6330
scfxm3	275152	990	1371	561	429	1800	0.00573	7777	0.05033	69	0.36162	358	1014.13925	978
stocfor3	3110673	16675	15695	8829	7846	23541	0.00025	64875	0.58165	9129	0.05925	988	1696.20261	15695
truss	1312537	1000	8806	1000	0	8806	0.00316	27836	1	8806	0.08800	88	5.59268	1000
wood1p	2217625	244	2594	243	1	2595	0.11094	70215	0.00039	1	0.00820	2	22731.15684	244
woodw	1219749	1098	8405	1085	13	8418	0.00406	37474	0.00048	4	0.03097	34	19572.71261	1098

Table 4.6: Netlib Kennington

Problem	Filesize	m	n	m-equal	m-inequal	n-slack	densA	nnzA	densc	nnzc	densb	nnzb	normA	rankA
cre-a	674161	3516	4067	335	3181	7248	0.00105	14987	1	4067	0.08788	309	193.01243	3255
osa-07	5486127	1118	23949	0	1118	25067	0.00537	143694	1	23949	1	1118	680.32409	1118
pds-06	3193314	9881	28655	9185	696	29351	0.00022	62524	0.68906	19745	0.07266	718	9.85141	9861
pds-10	5449008	16558	48763	15389	1169	49932	0.00013	106436	0.68960	33627	0.07193	1191	9.85247	16528
pds-20	11804810	33874	105728	31427	2447	108175	0.00006	230200	0.69946	73953	0.07289	2469	9.85258	33726
cre-b	10666973	9648	72447	4958	4690	77137	0.00037	256095	1	72447	0.03234	312	193.14671	7240
cre-c	600769	3068	3678	335	2733	6411	0.00117	13244	1	3678	0.10365	318	180.98677	2864
cre-d	10142757	8926	69980	4958	3968	73948	0.00039	242646	1	69980	0.03596	321	181.31645	6476
ken-13	8457756	28632	42659	28632	0	42659	0.00008	97246	0.99834	42588	0.30941	8859	13.23603	28632
ken-18	30590519	105127	154699	105127	0	154699	0.00002	358171	0.99902	154548	0.22334	23479	18.10183	105127
osa-14	12013582	2337	52460	0	2337	54797	0.00257	314760	1	52460	1	2337	994.48027	2337
osa-30	22901956	4350	100024	0	4350	104374	0.00138	600138	0.99998	100022	1	4350	1365.89370	4350
osa-60	53349731	10280	232966	0	10280	243246	0.00058	1397793	1.00000	232965	1	10280	2138.50159	10280

Table 4.7: Mészáros Miscellaneous (a)

Problem	Filesize	m	n	m-equal	m-inequal	n-slack	densA	nnzA	densc	nnzc	densb	nnzb	normA	rankA
aa03	2901853	825	8627	825	0	8627	0.00995	70806	1	8627	1	825	39.34052	822
aa5	2407623	801	8308	801	0	8308	0.00991	65953	1	8308	1	801	38.25759	800
car4	3245616	16384	33052	16384	0	33052	0.00012	63724	0.99141	32768	0.03625	594	1.68805	16384
dbic1	39753058	43200	183235	118	43082	226317	0.00013	1038761	0.97299	178285	0.99975	43189	1113689.81681	43192
delf001	730419	3098	5462	1990	1108	6570	0.00078	13214	0.53607	2928	0.53486	1657	3934.77202	3002
delf005	728521	3103	5464	1992	1111	6575	0.00080	13494	0.53587	2928	0.56171	1743	3932.67507	3011
delf008	740381	3148	5472	1998	1150	6622	0.00080	13821	0.53509	2928	0.57147	1799	3930.92791	3055
delf013	742272	3116	5472	1997	1119	6591	0.00081	13809	0.53509	2928	0.59275	1847	3930.05740	3023
delf015	740414	3161	5471	2003	1158	6629	0.00080	13793	0.53519	2928	0.58146	1838	3926.41085	3068
delf025	756189	3197	5464	1997	1200	6664	0.00083	14447	0.53587	2928	0.54145	1731	3913.64032	3103
delf028	757477	3177	5452	1977	1200	6652	0.00083	14402	0.53705	2928	0.58672	1864	3912.23991	3084
df2177	1678184	630	9728	0	630	10358	0.00354	21706	1	9728	1	630	9.71900	630
large000	921456	4239	6833	2499	1740	8573	0.00057	16573	0.53564	3660	0.66832	2833	4066.32821	3819
large005	959644	4237	6837	2498	1739	8576	0.00061	17575	0.53532	3660	0.70734	2997	4056.72967	3820
large008	971242	4248	6837	2498	1750	8587	0.00062	17898	0.53532	3660	0.71681	3045	4055.21723	3831
large013	974656	4248	6838	2501	1747	8585	0.00062	17941	0.53524	3660	0.73046	3103	4054.56063	3830
large019	968573	4300	6836	2512	1788	8624	0.00061	17786	0.53540	3660	0.72605	3122	4047.22086	3880
large021	982655	4311	6838	2516	1795	8633	0.00062	18157	0.53524	3660	0.73927	3187	4041.86746	3893
large034	999380	4294	6831	2499	1795	8626	0.00064	18855	0.53579	3660	0.70843	3042	4035.41393	3876
lpl2	1391994	3294	10755	3168	126	10881	0.00091	32106	1	10755	0.03916	129	500.10316	3294
model3	853556	1609	3840	871	738	4578	0.00376	23236	0.38750	1488	0.11125	179	11299.58551	1607
model5	2917267	1888	11360	1446	442	11802	0.00417	89483	0.06620	752	0.05085	96	503.09562	1744
nemsemm1	35856356	3945	71413	6	3939	75352	0.00373	1050047	0.99175	70824	0.53587	2114	14596.75613	3893
nemspmm2	2656303	2301	8413	1980	321	8734	0.00351	67904	0.22988	1934	0.03738	86	9389.45856	2276
orna2	264565	882	882	882	0	882	0.00400	3108	1	882	1	882	21835.95516	882
orna7	264565	882	882	882	0	882	0.00400	3108	1	882	0.92857	819	21778.07490	882
pldd000b	484503	3069	3267	1287	1782	5049	0.00090	8980	0.54545	1782	0.43304	1329	142.52148	2475
pldd005b	484502	3069	3267	1287	1782	5049	0.00090	8985	0.54545	1782	0.43174	1325	142.52148	2475
stat96v4	18151975	3173	62212	2309	864	63076	0.00248	490472	0.00002	1	0.29593	939	19.59346	3173

Table 4.8: Mészáros Miscellaneous (b)

Problem	Filesize	m	n	m-equal	m-inequal	n-slack	densA	nnzA	densc	nnzc	densb	nnzb	normA	rankA
aa6	1917498	646	7292	646	0	7292	0.01098	51728	1	7292	1	646	37.61396	646
delf026	747262	3190	5462	1990	1200	6662	0.00082	14220	0.53607	2928	0.52821	1685	3912.93259	3097
large020	981147	4315	6837	2515	1800	8637	0.00061	18136	0.53532	3660	0.73418	3168	4044.03397	3897
model4	1491814	1337	4549	924	413	4962	0.00745	45340	0.15718	715	0.14660	196	4369.51485	1331
aa01	2993221	823	8904	823	0	8904	0.00996	72965	1	8904	1	823	38.72970	823
aa3	2570614	825	8627	825	0	8627	0.00995	70806	1	8627	1	825	39.34052	822
aa4	1916041	426	7195	426	0	7195	0.01700	52121	1	7195	1	426	41.54544	426
air02	2442271	50	6774	50	0	6774	0.18174	61555	1	6774	1	50	131.39007	50
air04	2993219	823	8904	823	0	8904	0.00996	72965	1	8904	1	823	38.72970	823
air05	2191266	426	7195	426	0	7195	0.01700	52121	1	7195	1	426	41.54544	426
air06	2901851	825	8627	825	0	8627	0.00995	70806	1	8627	1	825	39.34052	822
aircraft	944809	3754	7517	3754	0	7517	0.00072	20267	0.50113	3767	1	3754	3202.65559	3754
bas1lp	18778892	5411	4461	47	5364	9825	0.02413	582411	0.00403	18	1	5411	1489.67814	4444
baxter	3904608	27441	15128	11836	15605	30733	0.00023	95971	0.91565	13852	0.03316	910	600530.08820	15110
cari	4854951	400	1200	400	0	1200	0.31833	152800	0.66667	800	1	400	4.42541	400
ch	1445988	3700	5062	471	3229	8291	0.00111	20873	0.76788	3887	0.24811	918	757.70698	3682
co5	3059662	5774	7993	1442	4332	12325	0.00116	53661	0.63531	5078	0.35573	2054	3957.18172	5710
co9	5798412	10789	14851	2716	8073	22924	0.00063	101578	0.64507	9580	0.32663	3524	3957.55777	10685
complex	1542082	1023	1408	1023	0	1408	0.03226	46463	0.22727	320	0	0	113.57925	1023
cq5	2761262	5048	7530	830	4218	11748	0.00125	47353	0.60106	4526	0.42334	2137	2239.52798	4950
cq9	5189525	9278	13778	1522	7756	21534	0.00070	88897	0.60865	8386	0.39028	3621	2239.52798	9096
crew1	1954946	135	6469	135	0	6469	0.05376	46950	1	6469	1	135	70.19399	135
dano3mip	2648780	3202	13873	1224	1978	15851	0.00179	79655	0.00007	1	0.50906	1630	1817.34751	3066
dbir1	34320496	18804	27355	384	18420	45775	0.00206	1058605	0.33668	9210	0.51021	9594	44318956.33528	15507
dbir2	36876944	18906	27355	384	18522	45877	0.00220	1139637	0.33668	9210	0.51142	9669	4445409.21712	15579
delf000	694789	3128	5464	1993	1135	6599	0.00074	12606	0.53587	2928	0.51886	1623	3935.29082	3032
delf002	716720	3135	5460	1990	1145	6605	0.00078	13287	0.53626	2928	0.52504	1646	3934.38118	3039
delf003	718189	3065	5460	1991	1074	6534	0.00079	13269	0.53626	2928	0.53475	1639	3934.06560	2972
delf004	725574	3142	5464	1992	1150	6614	0.00079	13546	0.53587	2928	0.52037	1635	3933.68458	3050
delf006	732276	3147	5469	1997	1150	6619	0.00079	13604	0.53538	2928	0.56180	1768	3931.68427	3054

Table 4.9: Mészáros Miscellaneous (c)

Problem	Filesize	m	n	m-equal	m-inequal	n-slack	densA	nnzA	densc	nnzc	densb	nnzb	normA	rankA
delf007	738407	3137	5471	1997	1140	6611	0.00080	13758	0.53519	2928	0.57475	1803	3931.66255	3044
delf010	740983	3147	5472	1997	1150	6622	0.00080	13802	0.53509	2928	0.58468	1840	3930.81277	3054
delf011	740337	3134	5471	1996	1138	6609	0.00080	13777	0.53519	2928	0.58679	1839	3930.92463	3041
delf012	740471	3151	5471	1996	1155	6626	0.00080	13793	0.53519	2928	0.58394	1840	3930.35428	3058
delf014	742745	3170	5472	2000	1170	6642	0.00080	13866	0.53509	2928	0.58297	1848	3928.60579	3077
delf017	738358	3176	5471	2004	1172	6643	0.00079	13732	0.53519	2928	0.58281	1851	3925.13941	3083
delf018	739270	3196	5471	2007	1189	6660	0.00079	13774	0.53519	2928	0.58010	1854	3924.21095	3103
delf019	739291	3185	5471	2008	1177	6648	0.00079	13762	0.53519	2928	0.58399	1860	3923.20041	3089
delf020	749863	3213	5472	2013	1200	6672	0.00080	14070	0.53509	2928	0.58886	1892	3919.26949	3120
delf021	749479	3208	5471	2013	1195	6666	0.00080	14068	0.53519	2928	0.58603	1880	3917.07370	3114
delf022	747683	3214	5472	2014	1200	6672	0.00080	14060	0.53509	2928	0.57032	1833	3916.01770	3120
delf023	748896	3214	5472	2014	1200	6672	0.00080	14098	0.53509	2928	0.56938	1830	3915.08974	3120
delf024	761097	3207	5466	2007	1200	6666	0.00082	14456	0.53568	2928	0.58216	1867	3914.33598	3114
delf027	745859	3187	5457	1987	1200	6657	0.00082	14200	0.53656	2928	0.52557	1675	3912.91010	3094
delf029	756480	3179	5454	1979	1200	6654	0.00083	14402	0.53685	2928	0.57565	1830	3912.25602	3086
delf030	752601	3199	5469	1999	1200	6669	0.00082	14262	0.53538	2928	0.56330	1802	3911.84911	3106
delf031	748658	3176	5455	1976	1200	6655	0.00082	14205	0.53676	2928	0.55856	1774	3911.96163	3083
delf032	751666	3196	5467	1996	1200	6667	0.00082	14251	0.53558	2928	0.56008	1790	3911.95865	3103
delf033	748540	3173	5456	1978	1195	6651	0.00082	14205	0.53666	2928	0.55563	1763	3912.17178	3080
delf034	748522	3175	5455	1980	1195	6650	0.00082	14208	0.53676	2928	0.55654	1767	3912.07832	3082
delf035	752973	3193	5468	1998	1195	6663	0.00082	14284	0.53548	2928	0.56060	1790	3911.89551	3100
delf036	748765	3170	5459	1975	1195	6654	0.00082	14202	0.53636	2928	0.55804	1769	3911.26581	3077
dsbmip	382268	1182	1886	443	739	2625	0.00330	7366	0.56628	1068	0.57022	674	35999.91296	1045
e18	5249013	24617	14231	246	24371	38602	0.00038	132095	0.85665	12191	1	24617	2040.01703	14230
ex3sta1	2647307	17443	8156	8083	9360	17516	0.00042	59419	0.00012	1	0.53660	9360	251.62400	8156
ge	1914517	10099	11098	4828	5271	16369	0.00035	39554	0.36700	4073	0.24111	2435	13744.99634	9085
jendrec1	3056622	2109	4228	2109	0	4228	0.01005	89608	1	4228	1	2109	3430.87674	2109
kl02	9479891	71	36699	71	0	36699	0.08157	212536	1	36699	1	71	157.77296	71
large001	947908	4162	6834	2500	1662	8496	0.00061	17225	0.53556	3660	0.70855	2949	4057.76599	3742
large002	983774	4249	6835	2504	1745	8580	0.00063	18330	0.53548	3660	0.71146	3023	4056.89412	3831

Table 4.10: Mészáros Miscellaneous (d)

Problem	Filesize	m	n	m-equal	m-inequal	n-slack	densA	nnzA	densc	nnzc	densb	nnzb	normA	rankA
large003	976694	4200	6835	2499	1701	8536	0.00063	18016	0.53548	3660	0.73214	3075	4056.79312	3784
large004	964140	4250	6836	2500	1750	8586	0.00061	17739	0.53540	3660	0.70329	2989	4057.30475	3833
large006	970641	4249	6837	2499	1750	8587	0.00062	17887	0.53532	3660	0.71382	3033	4056.76852	3832
large007	970005	4236	6836	2497	1739	8575	0.00062	17856	0.53540	3660	0.71837	3043	4055.95368	3819
large009	970902	4237	6837	2498	1739	8576	0.00062	17878	0.53532	3660	0.71867	3045	4054.92719	3820
large010	972085	4247	6837	2497	1750	8587	0.00062	17887	0.53532	3660	0.72616	3084	4054.60028	3830
large011	972225	4236	6837	2497	1739	8576	0.00062	17878	0.53532	3660	0.72875	3087	4054.69155	3819
large012	973178	4253	6838	2498	1755	8593	0.00062	17919	0.53524	3660	0.72537	3085	4054.71946	3836
large014	975841	4271	6838	2501	1770	8608	0.00062	17979	0.53524	3660	0.73098	3122	4052.77150	3854
large015	975963	4265	6838	2505	1760	8598	0.00062	17957	0.53524	3660	0.73740	3145	4051.34834	3848
large016	978127	4287	6838	2507	1780	8618	0.00062	18029	0.53524	3660	0.73688	3159	4050.15161	3870
large017	977204	4277	6837	2506	1771	8608	0.00061	17983	0.53532	3660	0.74258	3176	4049.83160	3857
large018	968638	4297	6837	2509	1788	8625	0.00061	17791	0.53532	3660	0.72492	3115	4048.82134	3877
large022	978873	4312	6834	2512	1800	8634	0.00061	18104	0.53556	3660	0.72820	3140	4041.07432	3894
large023	976892	4302	6835	2513	1789	8624	0.00062	18123	0.53548	3660	0.70595	3037	4040.21044	3884
large024	993888	4292	6831	2492	1800	8631	0.00063	18599	0.53579	3660	0.72367	3106	4039.32589	3874
large025	993822	4297	6832	2497	1800	8632	0.00064	18743	0.53571	3660	0.69211	2974	4038.53169	3879
large026	989233	4284	6824	2484	1800	8624	0.00064	18631	0.53634	3660	0.69071	2959	4038.01290	3866
large027	985984	4275	6821	2475	1800	8621	0.00064	18562	0.53658	3660	0.68819	2942	4037.50241	3857
large029	1003062	4301	6832	2501	1800	8632	0.00064	18952	0.53571	3660	0.71007	3054	4036.27987	3883
large030	998896	4285	6823	2485	1800	8623	0.00064	18843	0.53642	3660	0.71342	3057	4035.95762	3868
large031	999689	4294	6826	2494	1800	8626	0.00064	18867	0.53619	3660	0.71099	3053	4035.96850	3877
large032	998918	4292	6827	2492	1800	8627	0.00064	18850	0.53611	3660	0.70969	3046	4035.75107	3875
large035	989381	4293	6829	2498	1795	8624	0.00064	18881	0.53595	3660	0.62823	2697	4035.22091	3875
lp22	2646138	2958	13434	0	2958	16392	0.00165	65560	1	13434	1	2958	25.76285	2944
lpl1	20236309	39951	125000	39951	0	125000	0.00008	381259	0.64694	80868	0.02966	1185	546.25429	39946
stat96v5	12574881	2307	75779	2307	0	75779	0.00134	233921	0.97293	73728	0.00043	1	19.52339	2305
sws	3731014	14310	12465	5040	9270	21735	0.00052	93015	1	12465	0.38050	5445	36.25902	10980
t0331-4l	17007686	664	46915	664	0	46915	0.01384	430982	1	46915	1	664	125.84267	664
ulevimin	5497735	6590	44605	4258	2332	46937	0.00055	162206	0.00002	1	0.32595	2148	57278239.69242	6575

Table 4.11: Mészáros Miscellaneous (e)

Problem	Filesize	m	n	m-equal	m-inequal	n-slack	densA	nnzA	densc	nnzc	densb	nnzb	normA	rankA
us04	11415230	163	28016	163	0	28016	0.06516	297538	1	28016	1	163	237.50302	162
world	8745289	34506	32734	93	34413	67147	0.00015	164470	0.67071	21955	0.70585	24356	8520.10338	28783
pldd007b	484426	3069	3267	1287	1782	5049	0.00090	8987	0.54545	1782	0.43043	1321	142.52148	2475
rat1	3129695	3136	9408	3136	0	9408	0.00299	88267	0.66667	6272	0.99872	3132	1.72928	3136
rat5	4677257	3136	9408	3136	0	9408	0.00466	137413	0.66667	6272	1	3136	1.73106	3136
rat7a	8801563	3136	9408	3136	0	9408	0.00911	268908	0.66667	6272	0.99872	3132	1.79719	3136
rlfddd	10053178	4050	57471	0	4050	61521	0.00112	260577	0.86073	49467	1	4050	45.66143	4050
rlfdual	11887960	8052	66918	0	8052	74970	0.00051	273979	0.82059	54912	0.00596	48	33.47216	8052
rlfprim	12025777	58866	8052	4202	54664	62716	0.00056	265927	0.00596	48	0.79605	46860	33.45724	8048
rosen10	2183354	2056	4096	0	2056	6152	0.00738	62136	0.94043	3852	1	2056	342.30063	2056
rosen2	1577736	1032	2048	0	1032	3080	0.02200	46504	0.94141	1928	1	1032	246.95259	1032
route	8425864	20894	23923	1798	19096	43019	0.00038	187686	0.93379	22339	0.92342	19294	13043.02331	20894
seymourl	1365291	4944	1372	0	4944	6316	0.00495	33549	1	1372	1	4944	30.42055	1302
slptsk	2395694	2861	3347	2861	0	3347	0.00757	72465	0.17837	597	0.48445	1386	449.71552	2861
south31	5466170	18425	35421	17525	900	36321	0.00017	111498	1	35421	0.99973	18420	12894.28638	17832
stat96v1	32070409	5995	197472	5995	0	197472	0.00050	588798	0.97132	191808	0.00017	1	25.99911	5995
pldd001b	484528	3069	3267	1287	1782	5049	0.00090	8981	0.54545	1782	0.43304	1329	142.52148	2475
pldd002b	484427	3069	3267	1287	1782	5049	0.00090	8982	0.54545	1782	0.43174	1325	142.52148	2475
pldd003b	484452	3069	3267	1287	1782	5049	0.00090	8983	0.54545	1782	0.43174	1325	142.52148	2475
pldd004b	484477	3069	3267	1287	1782	5049	0.00090	8984	0.54545	1782	0.43174	1325	142.52148	2475
p010	4793732	10090	19000	10000	90	19090	0.00062	117910	1	19000	0.70268	7090	17.41287	10081
p05	2398942	5090	9500	5000	90	9590	0.00122	58955	1	9500	0.70530	3590	12.69845	5081
pcb1000	874928	1565	2428	1173	392	2820	0.00528	20071	1	2428	1	1565	62.81253	1565
pcb3000	2443398	3960	6810	3038	922	7732	0.00210	56557	1	6810	1	3960	91.88371	3960
pf2177	1192831	9728	900	450	9278	10178	0.00248	21706	1	900	1	9728	9.71900	630
orna3	264565	882	882	882	0	882	0.00400	3108	1	882	0.98866	872	21789.18265	882
orna4	264565	882	882	882	0	882	0.00400	3108	1	882	1	882	21717.31529	882
nemswrld	6285127	7138	27174	5762	1376	28550	0.00098	190907	0.08747	2377	0.09428	673	653.95033	6584
nl	3529351	7039	9718	1432	5607	15325	0.00061	41428	0.80490	7822	0.26012	1831	287.36265	7031
nsct1	21799256	22901	14981	421	22480	37461	0.00191	656259	0.75028	11240	0.50919	11661	33388862.85700	14978

Table 4.12: Mészáros Miscellaneous (f)

Problem	Filesize	m	n	m-equal	m-inequal	n-slack	densA	nnzA	densc	nnzc	densb	nnzb	normA	rankA
nsct2	22397904	23003	14981	421	22582	37563	0.00196	675156	0.75028	11240	0.51054	11744	8930036.67552	14980
nsir1	4603254	4407	5717	113	4294	10011	0.00552	138955	0.37555	2147	0.51282	2260	19673736.48629	4404
nsir2	4971978	4453	5717	113	4340	10057	0.00592	150599	0.37555	2147	0.51763	2305	10078622.39012	4446
nug08	278721	912	1632	912	0	1632	0.00490	7296	0.61765	1008	0.01754	16	6.98743	912
nug12	1468221	3192	8856	3192	0	8856	0.00136	38304	0.67073	5940	0.00752	24	8.72518	3192
nug15	3660057	6330	22275	6330	0	22275	0.00067	94950	0.70707	15750	0.00474	30	9.82907	6330
nw14	37477009	73	123409	73	0	123409	0.10045	904910	1	123409	1	73	506.27865	73
nemsemm2	7153903	6943	42133	198	6745	48878	0.00060	175267	0.89066	37526	0.34510	2396	14010.54400	6922
nemspmm1	1988193	2372	8622	2091	281	8903	0.00272	55586	0.27546	2375	0.06577	156	9383.77441	2342
model6	949586	2096	5001	1808	288	5289	0.00261	27340	0.17257	863	0.04676	98	1111.15669	2086
model7	1807426	3358	8007	1783	1575	9582	0.00184	49452	0.40827	3269	0.10870	365	11352.75069	3358
model8	1243375	2896	6464	2896	0	6464	0.00135	25277	0.00248	16	0.23550	682	5.08705	2896
model9	1964968	2879	10257	2197	682	10939	0.00187	55274	0.47655	4888	0.09587	276	1000.00239	2771
lpl3	4348549	10828	33538	10680	148	33686	0.00028	100377	1	33538	0.02327	252	500.27288	10828
mod2	8815002	34774	31728	93	34681	66409	0.00015	165129	0.68917	21866	0.71007	24692	8520.02210	28716
model10	4972699	4400	15447	3028	1372	16819	0.00219	149000	0.19583	3025	0.14705	647	37694.17194	4376
model11	2397385	7056	18288	7056	0	18288	0.00043	55859	0.52843	9664	0.33645	2374	200.21311	7056

Table 4.13: Mészáros Problematic

Problem	Filesize	m	n	m-equal	m-inequal	n-slack	densA	nnzA	densc	nnzc	densb	nnzb	normA	rankA
gen1	2074721	769	2560	768	1	2561	0.03204	63085	0.60000	1536	0.67360	518	39.19175	769
de080285	238250	936	1488	516	420	1908	0.00335	4662	0.57258	852	0.70620	661	1945.54371	840
gen	2074721	769	2560	768	1	2561	0.03204	63085	0.60000	1536	0.67360	518	39.19175	769
gen2	2715557	1121	3264	1121	0	3264	0.02237	81855	0.68627	2240	0.99376	1114	47.32863	1121
gen4	3559818	1537	4297	1536	1	4298	0.01622	107102	0.71492	3072	0.95966	1475	55.42562	1537
l30	2122152	2701	15380	1800	901	16281	0.00123	51169	0.88563	13621	0.33358	901	5.00001	2701

Table 4.14: Mészáros Stochastic LP (a)

Problem	Filesize	m	n	m-equal	m-inequal	n-slack	densA	nnzA	densc	nnzc	densb	nnzb	normA	rankA
deter0	587496	1923	5468	1923	0	5468	0.00106	11173	0.00329	18	0.00156	3	7.06732	1923
deter3	2340381	7647	21777	7647	0	21777	0.00027	44547	0.00331	72	0.00052	4	10.29728	7647
fxm3_6	1935133	6200	9492	3067	3133	12625	0.00093	54589	0.02665	253	0.45258	2806	1014.13925	6056
scagr7-2b-64	1587464	9743	10260	6156	3587	13847	0.00032	32298	0.95000	9747	0.36898	3595	152.68007	8723
scfxm1-2b-64	4030987	19036	28914	14903	4133	33047	0.00019	106919	0.02258	653	0.46543	8860	106.67100	19036
scrs8-2r-512	2417254	14364	19493	6166	8198	27691	0.00018	50241	0.49997	9746	0.52611	7557	46.60166	14363
scsd8-2r-432	8420788	8650	60550	8650	0	60550	0.00036	190210	1	60550	0.12936	1119	59.39625	8650
sctap1-2r-108	1697086	6510	10416	2604	3906	14322	0.00056	38124	0.75000	7812	1	6510	630.59235	6510
stormg2-8	1207831	4409	10193	3280	1129	11322	0.00061	27424	0.70803	7217	0.25607	1129	305.39590	4329
aircraft_stoch	952312	3754	7517	3754	0	7517	0.00072	20267	0.50113	3767	1	3754	3202.65559	3754
cep1	369145	1521	3248	0	1521	4769	0.00136	6712	0.99877	3244	0.43195	657	14.86603	1520
deter1	1691210	5527	15737	5527	0	15737	0.00037	32187	0.00330	52	0.00072	4	9.23743	5527
deter2	1871568	6095	17313	6095	0	17313	0.00034	35731	0.00462	80	0.00066	4	10.69981	6095
deter5	1561377	5103	14529	5103	0	14529	0.00040	29715	0.00330	48	0.00078	4	9.00753	5103
deter6	1301709	4255	12113	4255	0	12113	0.00048	24771	0.00330	40	0.00094	4	8.53603	4255
deter7	1950879	6375	18153	6375	0	18153	0.00032	37131	0.00331	60	0.00063	4	9.67449	6375
deter8	1171875	3831	10905	3831	0	10905	0.00053	22299	0.00330	36	0.00104	4	8.29848	3831
fxm2-16	1108915	3900	5602	2167	1733	7335	0.00143	31239	0.03088	173	0.36821	1436	1014.13925	3836
fxm3_16	13142203	41340	64162	19927	21413	85575	0.00014	370839	0.02514	1613	0.47015	19436	1014.13925	40316
fxm4_6	8470051	22400	30732	5947	16453	47185	0.00036	248989	0.03752	1153	0.27795	6226	1014.13925	21536
pgp2	958749	4034	9220	0	4034	13254	0.00050	18440	0.81171	7484	0.42885	1730	31.98000	4034
pltexpa3_16	6541669	28350	74172	28350	0	74172	0.00007	150801	0.41591	30849	0.18356	5204	2325.42984	28350
pltexpa4_6	6206733	26894	70364	26894	0	70364	0.00008	143059	0.41594	29267	0.18458	4964	2325.47608	26894
sc205-2r-800	1893977	17613	17614	4004	13609	31223	0.00015	48030	0.00006	1	0.16715	2944	40.10951	16814
scagr7-2r-108	668496	4119	4340	2604	1515	5855	0.00076	13542	0.95000	4123	0.36975	1523	99.19808	3691
scagr7-2r-216	1334424	8223	8660	5196	3027	11687	0.00038	27042	0.95000	8227	0.36909	3035	140.25123	7363
scagr7-2r-432	2666280	16431	17300	10380	6051	23351	0.00019	54042	0.95000	16435	0.36875	6059	198.31972	14707
scagr7-2r-64	398792	2447	2580	1548	899	3479	0.00128	8106	0.95000	2451	0.37066	907	76.38969	2195
scagr7-2r-864	5329992	32847	34580	20748	12099	46679	0.00010	108042	0.95000	32851	0.36859	12107	280.44842	29395

Table 4.15: Mészáros Stochastic LP (b)

Problem	Filesize	m	n	m-equal	m-inequal	n-slack	densA	nnzA	densc	nnzc	densb	nnzb	normA	rankA
scfxm1-2b-16	525163	2460	3714	1911	549	4263	0.00153	13959	0.02504	93	0.46016	1132	106.67101	2460
scfxm1-2r-128	4030987	19036	28914	14903	4133	33047	0.00019	106919	0.02258	653	0.46543	8860	106.67100	19036
scfxm1-2r-256	8037643	37980	57714	29751	8229	65943	0.00010	213159	0.02240	1293	0.46582	17692	106.67099	37980
scfxm1-2r-32	1025995	4828	7314	3767	1061	8375	0.00077	27239	0.02365	173	0.46313	2236	106.67101	4828
scfxm1-2r-64	2027659	9564	14514	7479	2085	16599	0.00039	53799	0.02294	333	0.46466	4444	106.67103	9564
scfxm1-2r-96	3029323	14300	21714	11191	3109	24823	0.00026	80359	0.02270	493	0.46517	6652	106.67102	14300
scsd8-2b-64	4996306	5130	35910	5130	0	35910	0.00061	112770	1	35910	0.15029	771	45.76740	5130
scsd8-2c-64	4996306	5130	35910	5130	0	35910	0.00061	112770	1	35910	0.15029	771	45.76740	5130
scsd8-2r-108	2110726	2170	15190	2170	0	15190	0.00145	47650	1	15190	0.13410	291	29.82361	2170
scsd8-2r-216	4213324	4330	30310	4330	0	30310	0.00073	95170	1	30310	0.12540	543	42.05848	4330
sctap1-2b-64	4019057	15390	24624	6156	9234	33858	0.00024	90220	0.75000	18468	1	15390	961.78431	15390
sctap1-2r-216	3389689	12990	20784	5196	7794	28578	0.00028	76140	0.75000	15588	1	12990	884.56760	12990
sctap1-2r-480	7524895	28830	46128	11532	17298	63426	0.00013	169068	0.75000	34596	1	28830	1312.82962	28830
stormg2_1000	146736708	528185	1259121	410000	118185	1377306	0.00001	3341696	0.70456	887121	0.22376	118185	3283.88301	526121
stormg2-125	18372380	66185	157496	51250	14935	172431	0.00004	418321	0.70475	110996	0.22566	14935	1162.54588	65871
stormg2-27	3995535	14441	34114	11070	3371	37485	0.00018	90903	0.70558	24070	0.23343	3371	544.63823	14323

Table 4.16: Mittelman (a)

Problem	Filesize	m	n	m-equal	m-inequal	n-slack	densA	nnzA	densc	nnzc	densb	nnzb	normA	rankA
fome11	2943178	12142	24460	12142	0	24460	0.00024	71264	0.51030	12482	0.27393	3326	15.91692	12142
long15	92020497	32769	753687	32769	0	753687	0.00006	1507374	0.99987	753587	0.49992	16382	128.00391	32769
neos3	55154867	512209	6624	1	512208	518832	0.00045	1542816	0.02174	144	1	512209	80.98793	6624
netlarge6	1786086218	8000	15000000	8000	0	15000000	0.00025	30000000	1	15000000	0.15000	1200	77.77794	8000
pds-50	30000053	83060	270095	77341	5719	275814	0.00003	585114	0.66290	179047	0.06912	5741	9.85342	82679
pds-90	51552753	142823	466671	134046	8777	475448	0.00002	1005359	0.61332	286220	0.06161	8799	9.86216	142416
rail516	11620711	516	47311	0	516	47827	0.01290	314896	1	47311	1	516	143.88804	502
watson_2	66239929	352013	671861	346650	5363	677224	0.00001	1841028	0.00400	2688	0.09905	34867	19.96628	352013
16_n14	30485248	16384	262144	16384	0	262144	0.00012	524288	0.95336	249917	0.00012	2	71.57240	16384
cont1	20478791	160792	40398	79998	80794	121192	0.00006	399990	0.00002	1	0.99010	159200	284.24285	40398
cont4	20453717	160792	40398	79998	80794	121192	0.00006	398398	0.00002	1	0.99010	159200	284.24285	40398
fome12	5886289	24284	48920	24284	0	48920	0.00012	142528	0.51030	24964	0.27393	6652	15.91692	24284
fome13	11772511	48568	97840	48568	0	97840	0.00006	285056	0.51030	49928	0.27393	13304	15.91692	48568
fome20	11804810	33874	105728	31427	2447	108175	0.00006	230200	0.69946	73953	0.07289	2469	9.85258	33726
fome21	23607452	67748	211456	62854	4894	216350	0.00003	460400	0.69946	147906	0.07289	4938	9.85258	67452
I_n13	85639935	8192	741455	8192	0	741455	0.00024	1482910	0.97319	721576	0.00024	2	207.32531	8192
L1_sixm250obs	182307993	986069	428032	986069	0	428032	0.00001	4280320	1	428032	0.00000	0	8.21402	428032
lo10	47688728	46341	406225	46341	0	406225	0.00004	812450	0.99978	406136	0.49999	23170	186.46716	46341
neos	55003077	479119	36786	0	479119	515905	0.00006	1047675	1	36786	1	479119	135.47739	36786
neos1	16482131	131581	1892	0	131581	133473	0.00188	468009	0.04493	85	0.98627	129775	60.50279	1892
neos2	19156423	132568	1560	0	132568	134128	0.00267	552519	0.04936	77	1.00000	132568	87.82013	1560
netlarge1	943830897	47700	8001358	47700	0	8001358	0.00004	16002716	1	8001358	0.95983	45784	43.80947	47700
netlarge2	139191119	40000	1160000	40000	0	1160000	0.00005	2320000	1	1160000	0.34033	13613	14.09134	40000
netlarge3	566518697	40000	4676000	40000	0	4676000	0.00005	9352000	1	4676000	0.35888	14355	23.72216	40000

Table 4.17: Mittelman (b)

Problem	Filesize	m	n	m-equal	m-inequal	n-slack	densA	nnzA	densc	nnzc	densb	nnzb	normA	rankA
ns1687037	46856900	50622	43749	12000	38622	82371	0.00064	1406739	0.54858	24000	0.88147	44622	80340635.68170	43749
ns1688926	55179838	32768	16587	8192	24576	41163	0.00315	1712128	0.50606	8394	1	32768	41022456.74456	16587
pds-100	55729538	156243	505360	147026	9217	514577	0.00001	1086785	0.60106	303754	0.05867	9167	10.11538	155764
pds-30	17280220	49944	154998	46453	3491	158489	0.00004	337144	0.68215	105732	0.07034	3513	9.85213	49698
pds-40	23696227	66844	212859	62172	4672	217531	0.00003	462128	0.67439	143550	0.07022	4694	9.85363	66499
pds-60	36539333	99431	329643	92653	6778	336421	0.00002	712779	0.65401	215589	0.06839	6800	9.85408	99030
pds-70	42336513	114944	382311	107250	7694	390005	0.00002	825771	0.64246	245619	0.06713	7716	9.85423	114539
pds-80	47151503	129181	426278	120879	8302	434580	0.00002	919524	0.62786	267641	0.06444	8324	9.85578	128774
rail2586	284515578	2586	920683	0	2586	923269	0.00336	8008776	1	920683	1	2586	496.00297	2570
rail4284	394339679	4284	1092610	0	4284	1096894	0.00241	11279748	1	1092610	1	4284	399.78028	4282
rail507	15125155	507	63009	0	507	63516	0.01281	409349	1	63009	1	507	149.32246	506
rail582	14655675	582	55515	0	582	56097	0.01243	401708	1	55515	1	582	185.90398	582
sgpf5y6	32658832	246077	308634	242171	3906	312540	0.00001	828070	0.24043	74205	0.03174	7810	8.46692	246077
square15	92000817	32762	753526	32762	0	753526	0.00006	1507052	0.99987	753425	0.50003	16382	128.00391	32762
watson_1	38726181	201155	383927	198090	3065	386992	0.00001	1052028	0.00400	1536	0.09906	19927	20.59371	201155
wide15	92020497	32769	753687	32769	0	753687	0.00006	1507374	0.99987	753587	0.49992	16382	128.00391	32769

Table 4.18: Execution time for IPM, Primal and Dual (in seconds)

Category	**Problem**	**Primal**	**Dual**	**IPM**
Netlib_Optimal	80bau3b	0.110	0.090	1.010
	bnl2	0.140	0.030	1.000
	d6cube	1.860	0.050	0.190
	degen3	0.330	0.090	0.560
	fit2p	1.620	0.840	0.980
	greenbea	0.230	0.200	1.010
	perold	0.090	0.050	1.010
	pilot4	0.030	0.010	1.010
	pilotnov	0.090	0.050	1.000
	scfxm3	0.020	0.010	0.030
	stocfor3	0.440	0.310	1.000
	wood1p	0.020	0.010	0.050
	25fv47	0.090	0.080	0.050
	2q06c	1.280	1.120	0.310
	dfl001	5.680	3.870	1.510
	fit2d	0.940	1.050	0.110
	grow22	0.050	0.080	0.030
	maros-r7	0.780	0.550	0.370
	pilot87	2.700	6.540	1.670
	qap08	0.980	1.170	0.080
	qap12	22.010	35.460	1.720
	qap15	236.530	487.190	7.290
	truss	1.310	2.710	0.080
	nesm	0.060	0.090	1.030
	woodw	0.010	0.030	0.050
Netlib_Kennington	cre-a	0.130	0.050	1.000
	cre-b	1.390	0.940	1.120
	cre-c	0.110	0.030	1.010
	cre-d	1.230	0.310	1.010
	ken-13	1.580	0.220	0.970
	ken-18	10.110	1.650	2.030
	osa-30	0.330	0.250	0.950
	osa-60	1.260	0.770	1.030
	pds-06	0.140	0.090	1.000
	pds-10	1.030	0.200	1.090
	pds-20	3.680	1.080	4.010
	osa-07	0.030	0.050	0.970
	osa-14	0.110	0.120	0.970
Meszaros_Misc	aa01	2.150	0.440	1.000
	aa03	2.530	0.220	1.000
	aa3	2.340	0.220	0.610
	aa4	1.110	0.130	0.270
	aa6	1.830	0.140	0.390
	air04	2.140	0.440	1.000
	air05	1.140	0.120	1.110
	air06	2.530	0.220	1.000
	baxter	0.810	0.340	3.880
	car4	0.170	0.110	0.140
	ch	1.010	0.140	1.000
	co5	1.550	1.000	1.110
	cq5	1.140	0.440	1.010

Table 4.18: Execution time for IPM, Primal and Dual (in seconds)

Category	Problem	Primal	Dual	IPM
	cq9	2.560	1.760	2.930
	dbir1	0.420	0.300	7.530
	dbir2	0.920	0.450	6.070
	delf000	0.060	0.030	1.010
	delf001	0.060	0.030	1.000
	delf002	0.060	0.020	1.010
	delf003	0.050	0.030	1.010
	delf004	0.080	0.050	1.010
	delf005	0.060	0.050	1.030
	delf006	0.080	0.050	1.010
	delf007	0.090	0.050	1.010
	delf008	0.110	0.060	1.010
	delf010	0.090	0.060	1.010
	delf011	0.080	0.050	1.010
	delf012	0.080	0.050	1.010
	delf013	0.080	0.050	1.030
	delf014	0.090	0.060	1.030
	delf015	0.080	0.050	1.010
	delf017	0.090	0.050	1.010
	delf018	0.080	0.030	1.010
	delf019	0.080	0.030	1.010
	delf020	0.090	0.050	1.030
	delf021	0.130	0.050	1.030
	delf022	0.090	0.060	1.010
	delf023	0.160	0.050	1.010
	delf024	0.110	0.050	1.010
	delf025	0.130	0.060	1.030
	delf026	0.110	0.060	1.010
	delf027	0.090	0.050	1.010
	delf028	0.090	0.050	1.010
	delf029	0.120	0.050	1.010
	delf030	0.140	0.050	1.010
	delf031	0.110	0.050	1.010
	delf032	0.140	0.050	1.030
	delf033	0.140	0.030	1.010
	delf034	0.140	0.050	1.030
	delf035	0.140	0.050	1.010
	delf036	0.130	0.050	1.010
	e18	2.180	1.000	79.870
	ge	0.980	0.270	1.090
	jendrec1	1.000	0.390	1.090
	large000	0.120	0.050	1.010
	large002	0.120	0.080	1.040
	large003	0.140	0.080	1.030
	large004	0.140	0.060	1.040
	large005	0.130	0.050	1.010
	large006	0.170	0.060	1.010
	large007	0.160	0.080	1.030
	large008	0.140	0.080	1.030
	large009	0.170	0.080	1.010
	large010	0.160	0.080	1.030

Table 4.18: Execution time for IPM, Primal and Dual (in seconds)

Category	Problem	Primal	Dual	IPM
	large011	0.140	0.080	1.010
	large012	0.170	0.060	1.030
	large013	0.190	0.080	1.030
	large014	0.140	0.060	1.030
	large015	0.140	0.050	1.040
	large016	0.160	0.060	1.030
	large017	0.110	0.050	1.030
	large018	0.130	0.050	1.030
	large019	0.160	0.050	1.030
	large020	0.190	0.060	1.040
	large021	0.160	0.080	1.040
	large022	0.190	0.080	1.030
	large023	0.220	0.080	1.030
	large024	0.170	0.130	1.050
	large025	0.200	0.090	1.060
	large026	0.200	0.120	1.040
	large027	0.160	0.110	1.030
	large029	0.140	0.110	1.050
	large030	0.140	0.110	1.030
	large031	0.160	0.130	1.030
	large032	0.280	0.140	1.030
	large034	0.160	0.120	1.030
	large035	0.190	0.120	1.030
	model11	0.390	0.360	1.030
	model9	1.060	1.000	1.030
	nemsemm1	0.420	0.300	1.050
	nemsemm2	0.170	0.120	1.000
	nl	2.620	1.000	1.870
	nsct1	0.220	0.130	7.400
	nsct2	0.560	0.250	7.210
	nsir1	0.090	0.020	0.280
	nsir2	0.280	0.090	0.980
	orna2	0.030	0.010	1.000
	orna3	0.050	0.020	1.000
	orna4	0.030	0.020	1.010
	orna7	0.050	0.020	1.000
	p010	0.420	0.190	0.250
	pcb1000	0.080	0.060	1.000
	pcb3000	0.310	0.230	1.010
	pf2177	0.130	0.110	0.120
	pldd000b	0.030	0.010	1.000
	pldd001b	0.050	0.010	1.000
	pldd002b	0.030	0.020	1.000
	pldd003b	0.030	0.010	1.000
	pldd004b	0.050	0.030	1.000
	pldd005b	0.050	0.010	1.000
	pldd007b	0.050	0.020	1.000
	rat1	0.120	0.110	0.230
	rlfddd	0.170	0.080	0.590
	rlfprim	0.780	0.220	0.920
	route	4.450	0.060	0.730

Table 4.18: Execution time for IPM, Primal and Dual (in seconds)

Category	**Problem**	**Primal**	**Dual**	**IPM**
	slptsk	0.480	0.300	0.620
	stat96v1	166.140	27.030	156.520
	sws	0.020	0.010	0.030
	ulevimin	19.860	6.930	7.000
	aa5	2.010	0.270	0.170
	aircraft	0.060	0.090	0.050
	co9	4.380	2.700	2.010
	dano3mip	4.930	6.020	2.000
	dbic1	15.550	18.490	8.360
	ex3sta1	0.920	7.020	0.610
	lp22	14.810	7.600	1.230
	mod2	30.260	10.780	2.140
	model10	3.350	3.230	1.170
	model4	1.040	1.220	1.010
	model5	1.200	1.170	1.150
	model6	1.090	1.280	1.000
	model7	1.310	1.450	1.060
	nemspmm1	1.250	1.440	1.000
	nemspmm2	1.620	1.370	1.170
	nemswrld	10.730	9.230	1.170
	nug08	1.000	1.290	0.380
	nug12	18.720	31.790	1.830
	nug15	220.460	484.200	7.320
	p05	0.160	0.090	0.080
	rat5	0.330	0.340	0.220
	rat7a	3.280	2.560	1.260
	rosen10	0.300	0.090	0.080
	rosen2	0.170	0.090	0.060
	seymourl	0.500	0.450	0.200
	south31	3.920	11.530	1.220
	stat96v4	117.720	42.350	1.950
	stat96v5	2.840	2.810	1.470
	t0331-4l	5.590	3.620	0.940
	world	41.680	12.890	1.900
	air02	0.010	0.030	0.980
	bas1lp	0.360	3.780	2.710
	cari	0.010	0.020	0.030
	complex	0.470	0.560	1.030
	crew1	0.030	0.060	0.050
	df2177	0.050	0.140	0.110
	dsbmip	0.010	0.030	1.010
	kl02	0.060	0.110	0.950
	large001	0.090	1.280	1.010
	lpl1	3.120	6.430	5.210
	lpl2	0.020	0.030	1.000
	lpl3	0.110	0.390	1.090
	model3	0.110	1.010	1.000
	model8	0.080	0.130	0.140
	nw14	0.270	0.280	0.870
	rlfdual	0.230	0.980	1.120
	us04	0.090	0.140	0.940

Table 4.18: Execution time for IPM, Primal and Dual (in seconds)

Category	**Problem**	**Primal**	**Dual**	**IPM**
Meszaros_Problematic	de080285	0.020	0.010	1.010
	gen	1.030	0.050	1.200
	gen1	1.030	0.050	1.060
	gen4	12.400	0.280	1.720
	gen2	10.550	3.370	1.280
	l30	60.720	4.460	2.540
Meszaros_Stochlp	fxm3_6	0.130	0.120	0.980
	fxm4_6	0.840	0.550	0.970
	pgp2	0.090	0.030	1.000
	scagr7-2b-64	0.050	0.030	1.000
	scagr7-2r-432	0.220	0.080	1.000
	scagr7-2r-864	0.940	0.380	1.000
	scfxm1-2b-16	0.060	0.030	0.050
	scfxm1-2b-64	0.900	0.660	1.010
	scfxm1-2r-128	1.250	0.640	1.010
	scfxm1-2r-32	0.090	0.080	1.000
	scfxm1-2r-64	0.330	0.270	1.000
	scrs8-2r-512	0.030	0.010	0.980
	scsd8-2b-64	0.300	0.060	0.220
	scsd8-2c-64	0.300	0.080	0.200
	scsd8-2r-108	0.060	0.010	0.670
	scsd8-2r-216	0.250	0.050	0.970
	scsd8-2r-432	0.870	0.110	0.950
	sctap1-2b-64	0.130	0.060	0.230
	sctap1-2r-108	0.030	0.020	0.090
	sctap1-2r-216	0.090	0.080	0.170
	sctap1-2r-480	0.500	0.280	0.950
	stormg2-125	2.540	2.120	2.610
	aircraft_stoch	0.060	0.090	0.030
	fxm2-16	0.080	0.140	0.060
	fxm3_16	2.840	2.460	1.250
	scfxm1-2r-256	2.820	1.940	1.060
	stormg2_1000	381.470	90.040	32.680
	cep1	0.010	0.020	0.030
	deter0	0.020	0.030	0.060
	deter1	0.060	0.140	0.980
	deter2	0.050	0.110	1.000
	deter3	0.080	0.140	0.980
	deter5	0.050	0.090	0.980
	deter6	0.030	0.060	1.000
	deter7	0.060	0.110	0.980
	deter8	0.030	0.060	1.000
	pltexpa3_16	0.060	0.280	1.110
	pltexpa4_6	0.060	0.410	1.000
	sc205-2r-800	0.030	0.050	0.060
	scagr7-2r-108	0.030	0.050	1.000
	scagr7-2r-216	0.050	0.130	1.000
	scagr7-2r-64	0.010	0.030	1.000
	scfxm1-2r-96	0.520	0.530	1.010
	stormg2-27	0.200	0.250	1.010
	stormg2-8	0.030	0.050	1.000

Table 4.18: Execution time for IPM, Primal and Dual (in seconds)

Category	**Problem**	**Primal**	**Dual**	**IPM**
Mittelmann	16_n14	261.860	22.370	43.030
	fome20	3.680	1.080	4.340
	fome21	10.140	3.790	6.750
	I_n13	2483.260	25.510	80.480
	lo10	239.460	3.090	418.930
	long15	191.400	6.270	2019.780
	neos	29.280	4.740	11.280
	neos3	56.040	2.340	20.610
	netlarge1	112.270	66.820	1043.320
	netlarge2	320.070	5.970	3410.430
	netlarge3	458.530	24.770	3870.600
	netlarge6	63.620	62.350	1343.710
	ns1688926	27.410	8.530	29.640
	pds-100	267.670	18.640	68.810
	pds-30	8.640	3.130	7.190
	pds-40	26.580	4.730	14.430
	pds-50	70.810	6.300	18.720
	pds-60	115.290	8.830	24.870
	pds-70	168.400	12.390	34.910
	pds-80	241.470	14.460	47.140
	pds-90	278.200	16.860	50.950
	square15	209.540	5.770	2219.410
	watson_1	23.880	3.260	5.150
	wide15	191.300	6.270	2017.720
	fome11	16.680	8.110	1.540
	fome12	44.550	21.120	3.540
	fome13	149.330	63.900	2.480
	L1_sixm250obs	2219.690	1597.370	128.530
	ns1687037	4541.880	5184.270	70.390
	rail2586	37.040	24.570	12.610
	rail4284	117.970	79.950	27.440
	watson_2	87.250	17.190	7.420
	cont1	112.150	216.170	182.830
	cont4	115.380	205.750	174.680
	neos1	1.400	2.890	4.760
	neos2	1.530	4.600	4.420
	rail507	0.390	0.450	0.940
	rail516	0.170	0.190	0.330
	rail582	0.340	0.380	0.940
	sgpf5y6	0.560	0.660	1.690

For this set of benchmark problems, the lower and upper values in the numbers of constraints, variables and nonzero elements of the constraint matrix and right-hand side vector, are given respectively in Table 4.19 below. Each minimum and maximum value may be related to different LPs, since these values are provided as reference and are not necessarily linked to the same LP.

Table 4.19: Lower and Upper values in examined LP characteristics

	Minimum	Maximum
Constraints	25	986, 069
Variables	882	15, 000, 000
Nonzero elements of constraint matrix	3, 108	30, 000, 000
Nonzero elements of right-hand side vector	0	512, 209
rankA	25	526, 121

The reason behind the selection of the specific dataset has been our interest to predict the performance of CPLEX's primal and dual simplex algorithms on well-known LPs, such as the ones described above. Including problems of different nature and structure (such as quadratic or mixed-integer problems) could increase the size of our dataset for training the examined ANNs, however, the diversity of the examined problems would not contribute to meaningful results regarding the problems' solution. In order to ensure that our dataset size is sufficient for training an artificial neural network, we took into consideration commonly applied rules-of-thumb, such as that the dataset size should be at least a factor of a) 50 to 1000 times the number of predicted classes [32], and b) 10 to 100 times the number of the examined features/characteristics ([58], [63], [92]).

4.3 Computing environment

For the implementation of the EPSA algorithm, we worked with MATLAB programming language, in MathWorks MATLAB environment (version R2014a) (Mathworks MATLAB [10]). MATLAB has been the most suitable option, due to its inherent capability for matrix and vector operations, regarding sparse matrices and large linear problems. Our code is designed to take advantage of Matlab's sparse matrix functions. The hardware and software characteristics of the specific computing environment in which our 6,780 LPs were generated and solved are fully described in the following table (Table 4.20) [70].

In order to create the predictive model for EPSA, we concluded that one of the most widely known statistical environments is the most appropriate, mainly because of the combination of analytical functions it supports; that is Minitab. Minitab is a general statistics software [11], originally developed in 1972, as a light version of OMNITAB, a statistical analysis program, in the National Institute of Standards and Technology (NIST) [12].

For the implementation of IPM, Primal and Dual algorithms, our study utilized the algorithms' versions supported by CPLEX 12.6.1 [9] to solve the respective benchmark problems, described in the previous section. CPLEX Optimizer includes several high-performance linear programming algorithms, supporting, among other methods, primal and dual variants of the simplex algorithm, as well as the interior point method. This solver offers efficient methods for model generation in order to overcome particularly complex optimization tasks, such as planning and scheduling. It is important to note that the algorithms were executed with the default options of CPLEX Optimizer, in order to minimize subjectivity in our observations, which may have resulted from different settings. This should be considered as an interesting area of research for the future, especially in combination with solver tuning techniques. Since the hardware and software characteristics are crucial parameters of a specific computing environment, which prove to have a significant impact on the performance of an algorithm [71], all experiments were conducted in the same environment, thus, this factor can be considered same for all problems, with no fluctuation from one problem to another. In case the computational experiments needed to be performed under different hardware conditions, a re-training process on the respective ANNs would be required.

Regarding the predictive models for IPM, Primal and Dual Simplex algorithms, these have been generated and examined with the use of the scikit-learn toolkit [84]. Scikit-learn toolkit integrates a plethora of widely use machine learning techniques, which can further be utilized for inferential statistical data analysis. In contrast to descriptive analysis, which focuses on the attributes of the sampled dataset only, inferential analysis is closer to the concept of a data population, which the observed data derives from. Scikit-learn supports numerous methods of supervised and unsupervised learning, model selection and evaluation and transformation of data. The characteristics of the computing environment are described in detail in the following table (Table 4.21).

The exact methodology and analysis steps that were followed in the above described environments is presented thoroughly in the next chapter, along with the generated models for the examined algorithms.

Table 4.20: Computing and model creation environment for EPSA

CPU	Intel® Xeon™, 3.00 GHz (2 processors)
RAM size	12,288 MB
L2 Cache size	2×1,024 KB
L1 Cache size	2×16 KB
Operating System	Microsoft Windows 7 Professional, SP1, 64-bit
MATLAB version	8.3.0.532 R2014a
Minitab version	16.1.1

Table 4.21: Computing and model creation environment for IPM, Primal and Dual Simplex

CPU	Intel Core i7, 3.40 GHz (8 cores)
Main memory	32 GB
Clock	3,700 MHz
L1 Code Cache size	32 KB \core
L1 Data Cache size	32 KB \core
L2 Cache size	256 KB \core
L3 Cache size	8 MB
Memory bandwidth	21 GB \s
Operating System	Microsoft Windows 7 Professional, SP1, 64-bit
IBM ILOG CPLEX	Optimization Studio V12.6.1
scikit-learn	0.23

Chapter 5

Predictive Models

This chapter includes the details of the analysis, conducted during our study. It provides the results regarding the generated predictive models for the computational behavior of the Exterior Point Simplex Algorithm, the Interior Point Method and the Primal and Dual Simplex algorithms. We are presenting the models in a sequential order, i.e. EPSA first, then IPM, and finally Primal and Dual Simplex. Each predictive model is followed by the respective statistics and validation findings, accompanied by comparative results among the tested models each time. It should be underlined that the generated models provided us with a great opportunity to understand the respective algorithms' performance and confidently support the models' predictive capabilities. Apart from the results we received for EPSA, the analysis performed among CPLEX's implementations for IPM, Primal and Dual Simplex algorithms was of vital significance. Our findings can have a major impact on the establishment of comparative thinking before selecting a particular algorithm for the solution of any LP problem. The main concern regarding CPLEX and any other mathematical solver has been that all examined algorithms may actually be able to solve the problems in question each time. However, so far, it had not been possible for any researcher that would wish to know upfront which algorithm is the most efficient one, to decide upon using a specific algorithm only. The researcher should select one algorithm and solve the given problem with no information on whether another algorithm would be much faster or consume different computing resources or be affected by different parameters. Researchers and software designers often perform extensive computational experiments to find the default values of parameters that will perform well on most instances of their data. Algorithm selection has been facilitated through algorithm tuning processes with several interesting studies conducted in this field ([19], [20], [47]). Meta-learning approaches have been utilized for tuning the performance of algorithms, mainly machine learning ones. The more similar those previous problems are, the better performance we can achieve [118]. Of course, there is no free lunch [120]. When a new problem comes in, leveraging prior experience may not be effective. Apart from tuning the performance of algorithms, meta-learning approaches have been also utilized for predicting their execution time [89, 25].

5.1 Exterior Point Simplex Algorithm (EPSA)

The first section of this chapter consists of a thorough explanation of the process we followed in order to create predictive models for the EPSA algorithm. We are presenting all details of our models, which are accompanied by the respective model validation part. In order to generate and examine our models, we have applied well established Regression Analysis techniques and metrics; here, we describe each regression model separately, through numerical and graphical representation of the results.

The most significant finding is the linear relation of the dependent with the independent variables in the model for the randomly generated sparse LP problems. While studying upon EPSA behavior, we focused more on the required number of iterations of the algorithm, rather than the CPU time. The problem with using regression analysis to study CPU time is that the memory hierarchy (caches, virtual memory system), creates a piece-wise cost function for time, with abrupt changes when the algorithm outgrows a level in the hierarchy. The location of the breaks, and the magnitude of the changes, are highly platform dependent. Therefore, time measurements on one platform would not be useful for predicting costs on another platform [28].

Regression models were generated and tested in Minitab, using the “Best subsets” model selection method. This is an automated process for identifying the best-fitting regression model for a given dataset, taking all available independent variables into account [54]. The main goal is to select a subset of independent variables that best satisfies specific statistical metrics, as these have already been presented in previous sections and will also be stated below. Since the fundamental aspects of Regression Analysis have already been described, we will now pinpoint the metrics used in order to evaluate the best regression model for this part of our study on EPSA. These metrics are listed below, followed by a brief explanation.

The metrics utilized for the purpose of our analysis are well-known and commonly used in the statistical research field for evaluation and model performance testing.

- R-squared (*R-Sq*): Coefficient of determination. Defines the degree of good fit of a statistical model on the examined data.
- Adjusted R-squared (*R-Sq(adj)*): Adjusted coefficient of determination. Useful for comparison of models with different number of predictors, due to the fact that it is adjusted according to the number of predictors in a model.
- Predicted R-squared (*R-Sq(pred)*): Indicates the predictive capability of a regression model for new observations. Useful for identifying if a model is not capable of providing valid predictions, even if it seems to fit to the original data.
- Standard error of regression (*S*): Measures the units of the response variable and represents the standard distance between data values and the estimated regression line.
- Standard error of the coefficient (*SE Coef*): Standard deviation of the estimate of a coefficient in a regression model. Measures the precision of the model in estimating the unknown value of the coefficient.

- Degrees of freedom (DF): Number of values in the final calculation of a statistic that are free to vary.
- Sum of Squares (SS): Indicates the deviation from the mean and is calculated as the sum of the squares of the differences from the mean.
- Mean Squares (MS): Calculated by dividing the respective sum of squares by the degrees of freedom. This metric is an estimate of the population variance.

As explained in Chapter 3, we pay attention to the F and the corresponding P values of each model. The F-test is the metric which determines if this relationship is statistically significant or not. Therefore, if the P value for the overall F-test is less than the applied significance level, then the specific regression model has statistically significant predictive capability [39]. The significance level in our study for EPSA has been set to 5%. Moreover, our models are accompanied by the respective probability plots (i.e. P-P plots), whose significance has been clarified earlier in this thesis. An extensive study of several possible regression models was conducted, so that we can reach a conclusion about the best suitable model for the sampled dataset. The best exponential and logarithmic models are included for comparative purposes.

5.1.1 Predictive model for randomly generated LP problems

The regression model for the randomly generated LP problems, solved by EPSA, reveals a linear positive relation between the number of iterations and the number of constraints (m) and decision variables (n). This indicates that a potential increase of the problem dimensions will result to an increase in the number of iterations as well.

The positive relation of the number of iterations with the problem sparsity reflects possible difficulties that the algorithm may face during the computation of specific mathematical equations in very sparse problems. Moreover, the condition of matrix A (*cond(A)*) is positively related to the number of iterations. This means that if the condition of our data (namely of matrix A) gets worse (i.e., the corresponding value of *cond(A)* increases), then the number of iterations will increase as well. The regression equation of the corresponding regression model is the following (Eq.5.1).

$$niter = -5,935.063 + 0.148m + 0.699n + 6,453.502sparsity + 1.150E - 006condA \tag{Eq.5.1}$$

A full description of the model's statistics is provided in the following table (Table 5.1). All independent variables that contribute to this predictive model are statistically significant with a P value lower than 0.05. The coefficient of determination (not only the *R-Sq*, but also the *R-Sq(adj)*), is approximately equal to 96% and the F-test is of significance $P < 0.001$, meaning that the model is suitable for the overall description and explanation of the variability of the whole dataset. Finally, the Durbin-Watson statistic [40] is approximately equal to 1.94, which is really close to 2 and indicates that there is no auto-correlation within the examined data.

Table 5.1: Statistical values of regression model

Predictor	Coef	SE Coef	T	P	
Constant	-5,935.063	56.482	-105.080	0.000	
m	0.148	0.003	48.887	0.000	
n	0.699	0.003	229.803	0.000	
sparsity	6,453.502	67.330	95.849	0.000	
cond(A)	1.150E-006	0.000	2.021	0.043	
R-Sq = 96.0%	R-Sq(adj) = 95.9%	S = 383.634			
R-Sq(pred) = 95.93%					
Analysis of Variance					
Source	DF	SS	MS	F	P
Regression	4	23,640,513,358.335	5,910,128,339.584	40,157.160	0.000
Residual Error	6,778	997,551,872.102	147,174.959		
Total	6,780	24,638,065,230.437			
Durbin-Watson	1.936				

The normal probability plot below (Fig. 5-1) provides a graphical representation of the standardized residuals, regarding the number of iterations. The X axis represents the Observed Cumulative Probability (Observed Cum Prob), which is based on the percentiles in the frequency distribution of the residuals. The Y axis, which represents the Expected Cumulative Probability (Expected Cum Prob), is based on the Standardized Residual (Z-score) and on the computation of the cumulative density from the normal distribution. If the residuals are normally distributed, then the values should fall exactly on the diagonal line. In our analysis, there is a deviation from the diagonal line, as shown in the (P-P) plot, which indicates a positive skewness of the distribution. This means that the right side tail of the curve, if this was depicted in a histogram, is longer than the left side tail and the mean is greater than the mode. Skewness is actually the asymmetry of a distribution and can be quantified to measure the extent to which this distribution is distorted and how much it differs from a normal distribution. This matter can be subject to further analysis in the future.

Moving forward, we are providing specific details of an exponential (Eq. 5.1) and a logarithmic (Eq. 5.2) model, which were found to be the best out of the rest models examined for the complete dataset (Tables 5.2 and 5.3).

As shown, the independent variables participating in the exponential model are only the number of decision variables (n) and the problem sparsity, while in the logarithmic model we see the number of constraints (m), the number of decision variables (n), the problem sparsity and the condition of matrix A (*cond(A)*). All independent variables in both predictive models are statistically significant with a P value lower than 0.05 and while the respective values of the coefficient of determination (*R-Sq*, *R-Sq(adj)* and *R-Sq(pred)*) are satisfying (i.e. approx. 94%

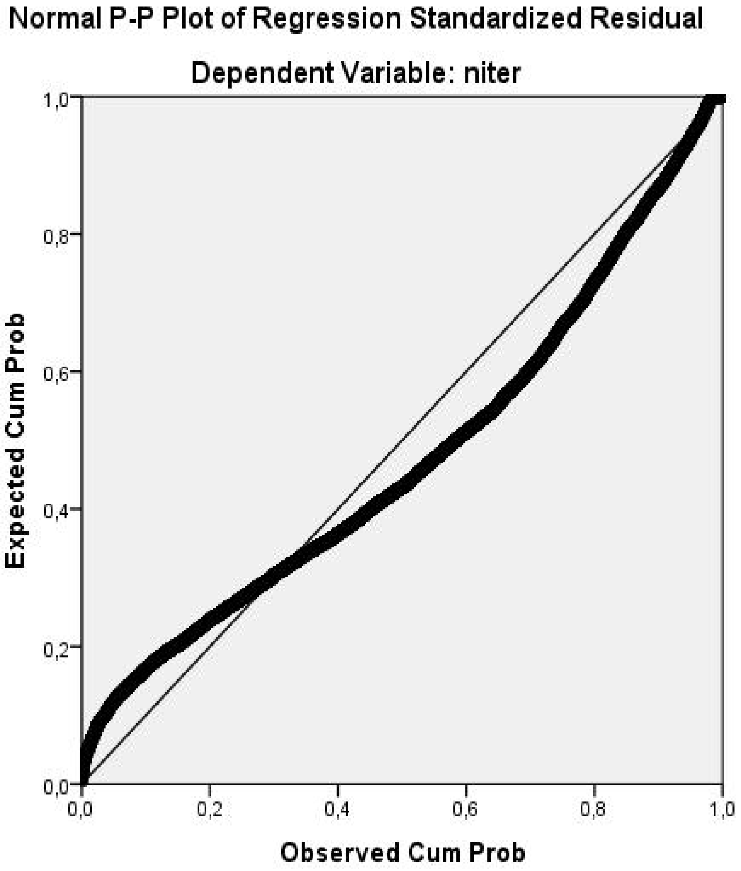

Figure 5-1: Normal Probability plot of iterations standardized residuals

and 88% for exponential and logarithmic model, respectively) they are still lower than the corresponding value of the linear model (i.e. approx. 96%). The F-test is of significance $P < 0.001$ in both models, meaning that they could be suitable the overall description of the dataset, however, still the standard error of regression (S) of the linear model is lower than the respective values of the exponential and logarithmic models. Finally, the Durbin-Watson statistic [40] is quite satisfying in both models (approx. 1.54 and 1.84 in the exponential and logarithmic models, respectively), although the corresponding value of the linear model remains closer to 2 (i.e. 1.94), indicating that in all cases, there is no auto-correlation in the examined dataset.

$$niter = -6,830 + 0.804n + 2,805exp(sparsity) \tag{5.1}$$

$$niter = -24,624+1,282log(m)+6,654log(n)+12,380log(sparsity)+91.4log(condA) \tag{5.2}$$

Table 5.2: Statistical values of exponential Regression model

Predictor	Coef	SE Coef	T	P	
Constant	-6,830.37	77.34	-88.32	0.000	
n	0.803768	0.002466	325.89	0.000	
exp(sparsity)	2,804.57	33.54	83.62	0.000	
R-Sq = 94.6%	R-Sq(adj) = 94.6%	S = 441.373			
R-Sq(pred) = 94.63%					
Analysis of Variance					
Source	DF	SS	MS	F	P
Regression	2	23,317,254,689	11,658,627,345	59846.20	0.000
Residual Error	6,780	1,320,810,541	194,810		
Total	6,782	24,638,065,230			
Durbin-Watson	1.542				

Table 5.3: Statistical values of logarithmic Regression model

Predictor	Coef	SE Coef	T	P	
Constant	-24,623.8	150.3	-163.81	0.000	
log(m)	1,281.69	55.03	23.29	0.000	
log(n)	6,653.55	55.14	120.66	0.000	
log(sparsity)	12,380.3	218.9	56.57	0.000	
log(cond(A))	91.402	5.216	17.52	0.000	
R-Sq = 88.4%	R-Sq(adj) = 88.4%	S = 648.856			
R-Sq(pred) = 88.40%					
Analysis of Variance					
Source	DF	SS	MS	F	P
Regression	4	21,784,430,694	5,446,107,674	12,935.69	0.000
Residual Error	6,778	2,853,634,536	421,014		
Total	6,782	24,638,065,230			
Durbin-Watson	1.839				

5.1.2 Validation of Predictive Model for randomly generated LP problems

Earlier in this section, the regression model which corresponds to the dataset of the randomly generated LPs was thoroughly presented. This section aims to provide a further validation of this model in order to ensure its accuracy and prediction ability. For this purpose, additional LPs were generated, following the same process that was applied during the creation of the initial dataset. In terms of statistical analysis, our initial LPs form the training dataset for our models, while the additional LPs, which are mentioned here, form the validation dataset, which confirms the regression model we created. The value ranges regarding the number of constraints (m), the number of variables (n), the problem density, as well as the type of constraints are the same to the ones applied in the initial dataset. During the creation of our validation dataset, we examined the same characteristics as the ones studied in the initial dataset. The additional LPs that were created during the validation step of our process are totally independent from the initial dataset.

The number of the additional LPs that were generated, covers approximately 10% of the total dataset (TDS). The number of LPs during validation is presented in the following table (Table 5.4).

Table 5.4: Validation Dataset

	Number of LPs	Percentage against TDS
Total	647	9.54

The validation process included the following steps:

1. Regression Model: The regression equation of the resulted model is given
2. Value Replacement: The independent variables (m, n, etc.) within the model are replaced by the corresponding observed values from the validation dataset
3. Calculation: The estimated value of number of iterations is calculated, based on our regression model
4. Comparison: The estimated value is compared to the observed one
5. Deviation computation: The deviation between the estimated and the observed value is calculated

For the purpose of this computation, we used the following ratio (Eq. 5.3):

$$DeviationRatio = (EstimatedValue - ObservedValue)/ObservedValue \qquad (5.3)$$

This process resulted in having a complete view of the efficiency of our models, both for representation and prediction of the number of iterations. Table 5.5 below, shows the maximum, minimum and average deviation from the observed values (in absolute numbers) (Eq. 5.3).

The regression model created for the number of iterations seems to be quite accurate and the actual average differences between the estimated and observed

Table 5.5: Deviation from Observed values in the resulted regression model

	Number of Iterations		
	max	min	average
Total	112.80%	0.02%	9.81%

values are remarkably small. Apart from the above mentioned analysis, though, we also examined how these instances are distributed and how they affect the average practical performance of our algorithm. EPSA performance seems to fall behind in extreme instances like the ones shown in the box plot on Fig. 5-3 below. This is also depicted on the respective histogram (Fig. 5-2), which indicates right skewed instances. This right or positive skewness is characterized by a "tail" of the instances to the right. This means, that there are many instances in the validation dataset, for which the number of iterations is relatively small, while in increasingly few instances the number of iterations increase significantly. The existence of a specific pattern of characteristics in these extreme instances, which may eventually affect the practical performance of EPSA, could be subject to further analysis.

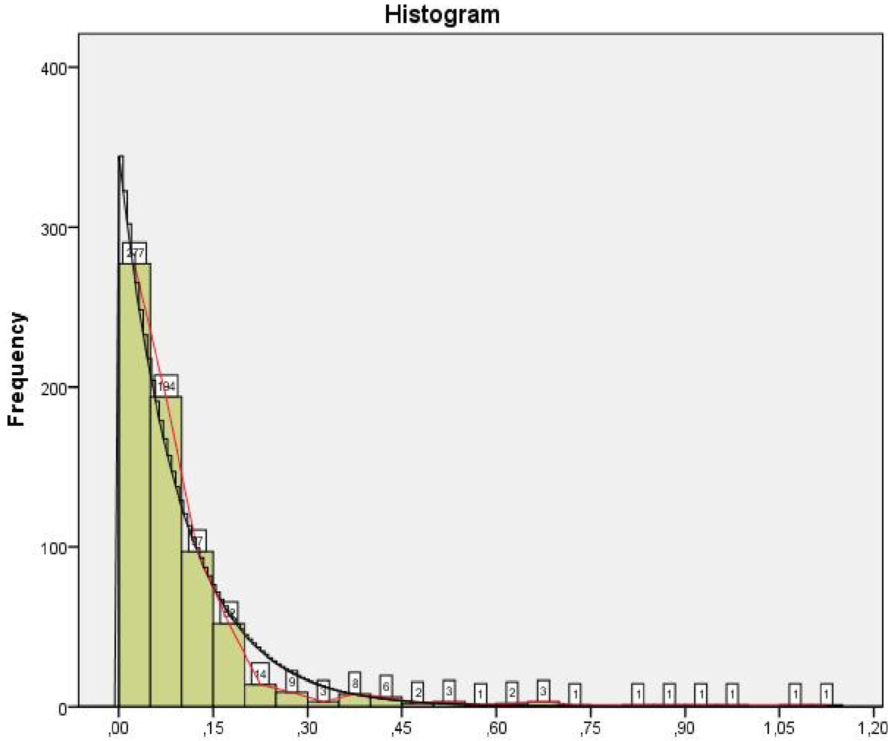

Figure 5-2: Deviation Histogram - Validation

5.1.3 Predictive model for benchmark LP problems

In the current section we are presenting our analysis of EPSA performance on benchmark dataset of 60 LP problems, as this has been described in Chapter 4.

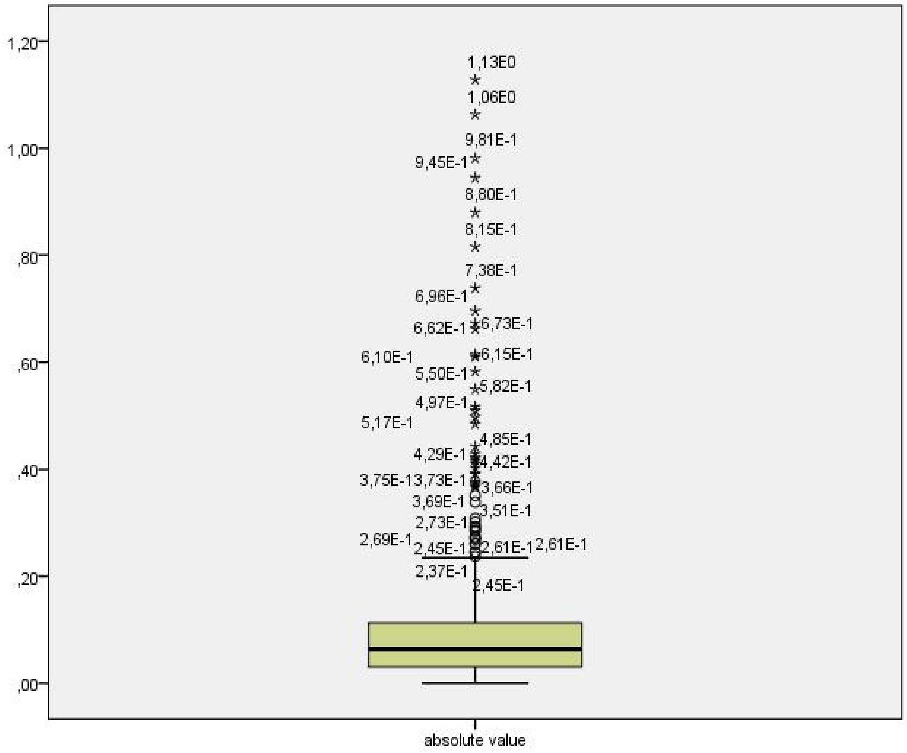

Figure 5-3: Outliers - Validation

Although the regression model included in this section is the best performing model out of the models tested, it actually indicates that the performance of our algorithm drops (comparing to the performance in the randomly generated sparse problems, described in previous sections). This can be explained by the nature of the benchmark problems of the dataset, which are degenerate; it was quite impressive that during the analysis seven LPs were automatically excluded from the analysis, since their condition number reached infinity. These problems could not, eventually, participate in the computation of the regression model. The corresponding details are shown below (Equation 5.4 and Table 5.6).

$$niter = 539 + 0.0296 sparsity \times nnz + 0.177 n \times sparsity \tag{5.4}$$

As shown, *R-Sq(adj)* reaches 60.3%, however *R-Sq(pred)* shows that the particular model is not suitable for prediction purposes. This is reasonable, if we take into consideration the structure and unique characteristics of benchmark problems, which are degenerate and ill-conditioned. As a result, it is difficult for EPSA to handle such problems and especially to have results that would form a satisfying regression model, which would then be used for predictive purposes, as well.

Table 5.6: Statistical values of Regression model - Benchmark

Predictor	Coef	SE Coef	T	P	
Constant	539.0	171.9	3.14	0.003	
sparsity × nnz	0.029562	0.005815	5.08	0.000	
n × sparsity	0.17708	0.08942	1.98	0.053	
R-Sq = 61.9%	R-Sq(adj) = 60.3%	S = 1064.60			
R-Sq(pred) = 23.71%					
Analysis of Variance					
Source	DF	SS	MS	F	P
Regression	2	91,940,699	45,970,349	40.56	0.000
Residual Error	50	56,669,020	1,133,380		
Total	52	148,609,719			
Durbin-Watson	1.892				

5.2 CPLEX - IPM, Primal and Dual Simplex Algorithms

As explained earlier in this study, LP problems can nowadays be solved by a plethora of algorithms which are supported in several mathematical solvers. Regardless of the number of available algorithms which can solve a specific LP problem though, it is often difficult to decide upon which algorithm would be the most appropriate and efficient to use, in terms of execution time for the solution of a problem each time. One of the most widely used mathematical solvers, IBM CPLEX Optimizer [9], includes several high-performance linear programming algorithms, supporting, among other methods, primal and dual variants of the simplex algorithm, as well as the interior point method.

In this section, we are analyzing the IPM, Primal and Dual CPLEX variants and elaborating thoroughly on the steps we followed in order to reach meaningful conclusions through the generated predictive models. For these algorithms, we are concentrating on the execution time, rather than the number of iterations for the solution of a LP problem (as examined for EPSA). Reason is the that execution time is of vital significance for all modern mathematical solvers, which have evolved to complex software systems consisting of various parameters that can be tuned. The configuration process of the solver's parameters is referred to as solver tuning [18]. Most software designers often perform extensive computational experiments to find the default values of parameters that will perform well on the majority of instances to be solved. Solver tuning has been applied successfully in various mathematical solvers ([16], [56], [55], [30], [68]). For CPLEX, we are looking into a different problem though, since we are interested in predicting the execution time of the solver for a specific problem instance. Knowing the execution time upfront, i.e. the time necessary to solve an instance, the mathematical solver may devise different options to solve this instance. Similar studies have been conducted in the past, such as [122], which aims to predict the solution time of Branch-and-Bound algorithms for mixed-integer programs (MIP) and proposes a double exponential smoothing technique, evaluating it with three MIP solvers. Such a performance modeling of software systems is mainly achieved through analytical modeling and machine learning [72, 109, 85, 14]. Analytical modeling exploits existing knowledge of the internal dynamics of the software system and can express the relationship of the inputs and outputs using a set of analytical equations. However, as stated already, software systems have become so complex over the years that applying analytical modeling techniques to predict the solvers' execution time does not yield good results anymore. Therefore, various researchers explore machine learning techniques to predict the execution time of software systems ([102], [113], [66], [101]). Based on our research, a performance modeling tool for any mathematical optimization solver has not been developed so far.

5.2.1 Predictive model for Interior Point Method

In regards to the analysis of the Interior Point Method, the neural network models we examined have been generated using the scikit-learn toolkit [84]. While testing several statistical environments, we found scikit-learn to be the most suitable for the purpose of our analysis, since it supports numerous methods of supervised and unsupervised learning, model selection and data evaluation and transformation. The algorithm used for the generation of our model is the Multi-layer Perceptron (MLP) and in particular, Multi-layer Perceptron Regressor (MLPRegressor). MLP is a supervised learning algorithm that can learn a function $f(x) : R^x \rightarrow R^o$ by training on a dataset, where x is the number of dimensions for input and o is the number of dimensions for output. Given a set of input features and an output target, MLP can learn a nonlinear function approximator for either classification (MLPClassifier) or regression (MLPRegressor). Although we experimented with other supervised learning methods available in scikit-learn toolkit, such as linear regression, lasso regression, ridge regression and decision trees, we obtained the best results using the MLP regression method.

To better understand its functionality, the parameters of MLPRegressor are described in detail below, including the number of hidden layers in ANNs, the activation function and ANN solver, the tolerance and alpha values, along with the scale of the examined data.

One of the most important parameters is the number of hidden layers that needs to be examined and defined, along with their size and the activation function which we have to choose. The activation function is responsible for converting the input signal of the last hidden layer to an output signal for the next layer. Commonly used activation functions are the hyperbolic tan function (*tanh*), the logistic sigmoid function (*logistic*) and the rectified linear unit function (*relu*). Numerous combinations of all supported activation functions with different number of hidden layers were tested for the purpose of this study.

It has also been mandatory to test the ANN solver of our model. Neural networks consist of a number of simple but highly interconnected nodes, the so-called “neurons”, which are organized in layers. Neural networks are extremely helpful in finding patterns that are too complex to be manually extracted and taught for any kind of machine. In an ANN, the input layer (which has one neuron for each element of the input data) communicates to one or more hidden layers that are present in the network. The hidden layers are actually the place where all information is processed, thus their name may not be so representative of their real significance; they are characterized as “hidden” only because they do not constitute the input or the output layer. The information is processed through weights and biases (commonly referred as W and b, respectively). In more detail, once the input is received, the neuron calculates a weighted sum (by adding also the bias) and according to the result and the preset activation function, it is activated or not. The neuron transfers this information to its connected neurons, ending up to the last hidden layer that is linked to the output layer, which has only one neuron (i.e. for the respective output). The ANN solver of the model is related to the weight optimization process that takes place while transmitting information through hidden layers. There are several solvers that can be used, such as *lbfgs*, an optimizer in the family of quasi-Newton methods and *sgd*, concerning stochastic gradient descent. For small datasets, *lbfgs* converges faster and performs better in general. Solver *lbfgs* uses a weighted linear summation to transform the input values of previous layers to output values for the next layer [84].

Another parameter is the tolerance value, which refers to the tolerance for the optimization. To explain, let us assume that upon a certain number of iterations, we fail to decrease the training loss or to increase the validation score by at least a value equal to tolerance; in this case, the convergence is considered to be reached and the training stops.

The alpha value refers to the L2 penalty parameter (Ridge regression; regularization technique used to address over-fitting and feature selection), while the maximum number of iterations indicates that the solver will iterate until convergence (determined by tolerance value) or this number of iterations.

Last but not least, one more aspect that needs to be taken into consideration is the scale of our data. The input parameters of a model may have different scales, which makes it difficult for the examined problems to be modeled. Scaling and normalizing the original data is a significant step we have taken while generating our models.

The greatest challenge during our analysis was related to the number of hidden layers, which had to be set while testing our models, along with the activation function we had to choose. It was noted that the more hidden layers we have in our models, the worse results we eventually get. Several activation functions (i.e. *tanh*, *logistic* and *relu*) were also tested while repeated and extensive testing was also performed on the number of hidden layers. The solver that was eventually selected for weight optimization is *lbfgs*. The ANN parameters selected in the best performing model are shown in Table 5.7.

The ratio between the training and test set was chosen to be 70 to 30. To evaluate the performance of our models, certain metrics were taken into consideration. The coefficient of determination (R^2, R-squared) provides an estimate of the strength of the relationship between a regression model and the dependent variable (output),

Table 5.7: Model parameters for the MLP method

Algorithm	MLPRegressor
Hidden layer sizes	20
Activation function	relu
Solver	lbfgs
Alpha value	$1e-5$
Maximum iterations	1000
Tolerance	0.0001

since it defines how well a statistical model fits the examined data (i.e. the bigger its value is, the better fitting the model has). The Root Mean Square Error (RMSE) is the standard deviation of the residuals (prediction errors) and measures their spread around the line of best fit. RMSE is considered as a measure of accuracy, used for comparison of different models generated for a particular dataset. This metric has non-negative values and a value of 0, although impossible to achieve in practice, would indicate a perfect fit to the data. In general, a lower RMSE value is always better than a higher one. It is important to note that this metric is not used between different datasets, as it is scale-dependent [57], thus comparisons of different types of data would not be valid since the measurement depends on the scale of the examined dataset numbers. We also include the mean absolute error that measures the average magnitude of the errors in a set of predictions, without considering their direction, and the median absolute error that is insensitive to outliers ([39], [67] and [90]). Table 5.8 presents the results for the training set, where the model achieved an RMSE value of 123.32 and an R^2 value of 0.78 and the test set, where the model achieved an RMSE value of 296.73 and an R^2 value of 0.72. Taking into account the features' variability of the 295 linear programming models and the metrics' values, it is shown that our model can explain the data reasonably well. As an example, an R^2 value of 1 would indicate a perfect fit of the data, so the current R^2 value proves goodness of fit of our model.

Table 5.8: MLPRegressor model for the IPM execution time

	Training set	Test set
Root Mean squared error	123.32	296.73
Absolute Mean error	54.31	97.54
Absolute Median error	7.12	9.49
R^2	0.78	0.72

A graphical representation of the comparison between the metrics measured for some of the models we tested follows in Figures 5-4 and 5-5 (different number of hidden layers and different activation functions, respectively). The term *units* as shown in Figure 5-4 below, refers to the actual layers of the model.

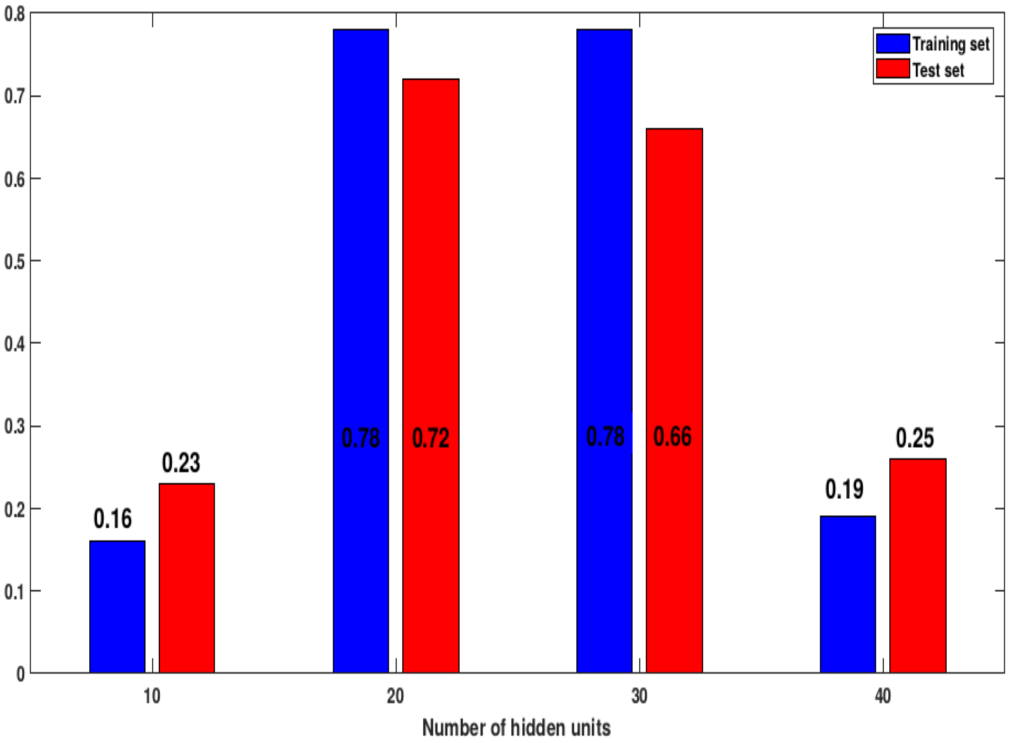

Figure 5-4: Regression model for interior point method - Tuning the number of neurons (1 hidden layer)

5.2.2 Predictive models for Primal and Dual Simplex Algorithm

For Primal and Dual Simplex algorithms, we extended the work performed for the Interior Point Method, by using the MLPRegressor algorithm for the generation of our predictive models. Our initial goal was to examine whether a regression model could also be built for these two algorithms, in the same way as for IPM. We also experimented with other supervised learning methods and more specifically, with classification methods available in scikit-learn, such as MLPClassifier and KNeighborsClassifier.

The findings of our computational study showed that, none of the formed regression models could either achieve goodness of fit for our data or stand as an accurate prediction model for the execution time of CPLEX's primal and dual simplex algorithms. In order to overcome the barrier introduced by this outcome, we attempted treating the problem as a classification problem; thus, instead of estimating the execution time, we attempted estimating the class under which the execution time will fall.

In the following paragraphs, we will briefly describe the concept of regression and classification techniques that were used and then, present the analysis we conducted to form regression models, using artificial neural networks, for the prediction of primal and dual simplex algorithms' execution time. A thorough description of the

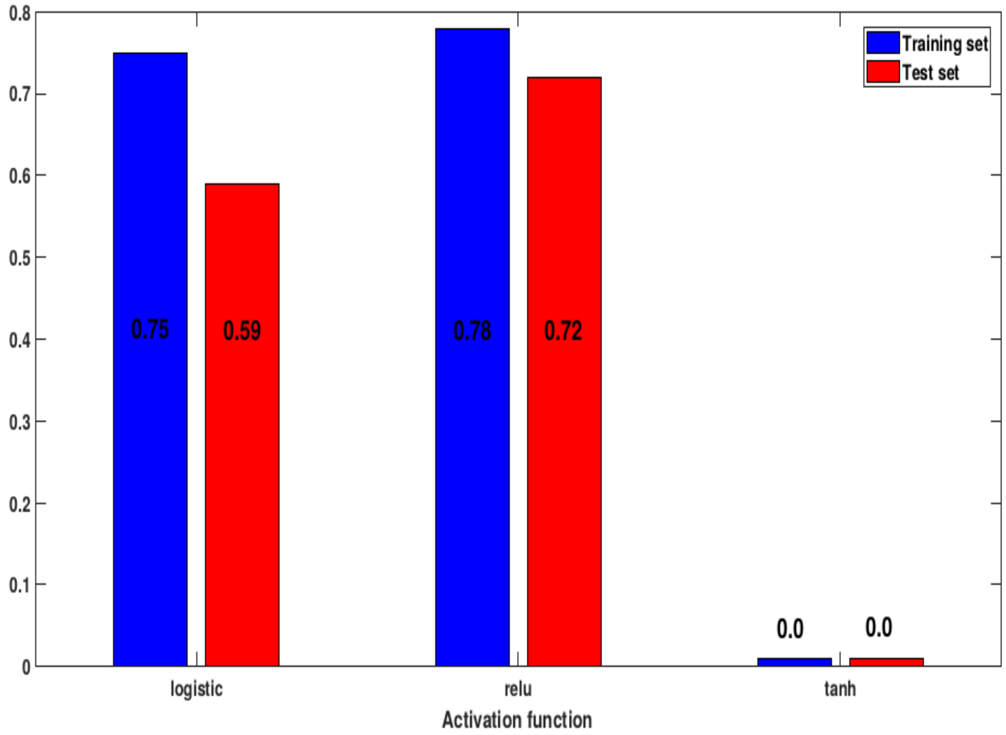

Figure 5-5: Regression model for interior point method - Tuning the activation function

models generated using classification techniques, follows next, along with the respective results and a graphical representation of the comparative analysis among the generated models in order to select the most appropriate one for each algorithm.

As explained in previous sections, regression and classification belong to the broader family of supervised machine learning techniques, utilizing the concept of using known datasets (i.e. training datasets) to make predictions about new incoming data. Considering that an input variable x and an output variable y are available, a supervised learning algorithm aims to "teach" a mapping function (that is $y = f(x)$) from the input variable x to the output variable y. This way, whenever there is a new input data x, the respective output variable y will be predicted, with the help of regression or classification predictive models. Although these techniques share the same objective, regression and classification have a main difference, which is that the output variable for classification is categorical (or discrete), while in regression it is numerical (or continuous). Classification predicts a discrete class label, while regression a continuous quantity. There are some algorithms, though, which can be used both for classification and regression, with only slight modifications, such as Artificial Neural Networks (ANNs) and decision trees. Classification predictive models can be evaluated using the accuracy value, whereas regression predictive models are evaluated through other metrics, such as the respective coefficient of determination and the root mean squared error (i.e. quantities that cannot be measured for classification predictions).

Evaluating the metrics of each model for the primal and dual simplex algorithm separately, our models were formed, testing and tuning the parameters shown in Table 5.9.

Table 5.9: MLP model parameters used for primal and dual simplex algorithms

Algorithm	Primal, Dual
Hidden layers	[1-3]
Hidden layer sizes	[10-100 neurons/layer]
Activation function	relu, tanh, logistic
Solver	lbfgs, sgd
Alpha value	$1e-5$
Maximum iterations	$1,000$
Tolerance	0.0001

To split our dataset into training and test set, a ratio of 75 to 25 was selected. The training and test sets were formed through cross validation, as supported for MLPRegressor by the scikit-learn library [84]. Similarly to IPM, the formulated models were evaluated upon standard metrics, such as the coefficient of determination and root mean square error. As explained, the coefficient of determination (R-squared, R^2) estimates the strength of the relationship between a regression model and the dependent variable (output) and defines how well a statistical model fits the examined data (i.e., the bigger R^2 value is, the better is the fitting of the examined model). Root Mean Square Error (RMSE) is the standard deviation of the residuals (prediction errors), which measures their spread around the line of best fit. RMSE is considered to be a measure of accuracy and it is used for comparison of different models generated for a particular dataset. This metric has non-negative values and a value of 0 would indicate a perfect fit to the data, however, this is quite impossible to achieve in practice. In general, a lower RMSE value is always better than a higher one. It is important to note that this metric should not be used between different datasets, as it is scale-dependent [57]. Comparisons of different types of data would not be valid since the measure depends on the scale of the examined dataset numbers. Apart from these metrics, we also measured the mean absolute error that measures the average magnitude of the errors in a set of predictions, without considering their direction, and the median absolute error that is insensitive to outliers [39, 67, 90].

Table 5.10 presents the results of our neural networks, both for primal and dual simplex algorithms on the examined dataset.

For the primal simplex algorithm, the model that showed the best performance, compared to the rest models that were formed and tested, achieved an RMSE value of 342.08 and an R^2 value of 0.79, while for the test set the model achieved an RMSE value of 1302.25 and an R^2 value of 0.21. The model was set to work with 1 hidden layer of size equal to 30 neurons, logistic activation function, and the *lbfgs* solver.

As for the dual simplex algorithm, the results the best fitting model achieved, were an RMSE value of 345.28 and an R^2 value of 0.66 in the training set, while in the test set, these values reached 1260.39 and 0.05, respectively. In this case, there

Table 5.10: MLPRegressor model for the execution time of the primal and dual simplex algorithms

	Primal		Dual	
	Training set	Test set	Training set	Test set
Root Mean squared error	342.08	1302.25	345.28	1260.39
Absolute Mean error	9.60	25.07	11.35	25.16
Absolute Median error	3.26	14.75	3.93	16.05
R^2	0.79	0.21	0.66	0.05

was 1 hidden layer with 20 neurons, while the activation function and solver were the logistic and *lbfgs*, similarly to the model formulated for the primal simplex algorithm.

Taking into account the variability in the features of the 295 LP problems of our dataset and the metrics' values, our models proved to perform lower than our initial expectations. It became clear that they cannot explain the data well, showing a significant discrepancy between the metrics' values of the training and the test set. An R^2 value of 1 would indicate a perfect fit of the data, so the current R^2 values of the training sets for both algorithms prove a certain level of goodness of fit of our models, which, however, cannot be verified or validated further. This is confirmed by the test sets, where the R^2 values drop significantly, while the corresponding RMSE values increase tremendously, comparing to the ones of the training set.

These values reveal models that cannot be utilized for prediction of the execution time needed for the solution of LP problems by the primal and dual simplex algorithms. It is quite interesting to show that these results were considered the "best" after extensive and thorough testing, with different numbers of hidden layers and neurons per layer ($1-3$ and $10-100$, respectively), different activation functions (*relu*, *tanh* and *logistic*), and solvers (*lbfgs* and *sgd*). A graphical representation of the results we received with only some of the different models formed with MLPRegressor follows further below (Figures 5-6 and 5-7 for primal simplex and Figures 5-8 and 5-9 for dual simplex algorithms, respectively).

The models we tested showed worse performance, while some of them were even characterized by negative values of R^2 for the test set, which could not be interpreted to lead in meaningful and useful results. More specifically, since R^2 compares the fit of the chosen model with that of a horizontal straight line (the null hypothesis), if the model fits worse than a horizontal line, then R^2 is negative, meaning that the chosen model does not follow the trend of the data, so would not be useful for prediction purposes. Moreover, models formed with *sgd* solver showed non-goodness-of-fit (R^2 values were below 0.06 and 0.20 for the primal and dual simplex algorithms, respectively), thus they are not included in the graphical representation.

The example of Figure 5-6 presents the R^2 value for several numbers of neurons in models for the primal simplex algorithm, using 1 hidden layer, the logistic activation function, and the *lbfgs* solver. The R^2 value is given both for the training and test sets. In Figure 5-7, the R^2 value of several models is given, using the *relu*, *tanh*, and *logistic* activation functions and having 30 neurons in 1 hidden layer with *lbfgs* as solver.

Figure 5-8 presents the R^2 value for several number of neurons in models for the dual simplex algorithm, using 1 hidden layer, the logistic activation function and the *lbfgs* solver. In Figure 5-9, the R^2 value of several models is given, using the *relu*, *tanh*, and *logistic* activation functions and having 20 neurons in 1 hidden layer with *lbfgs* as solver. The R^2 value is given both for the training and test sets below, for all presented samples of our comparative analysis.

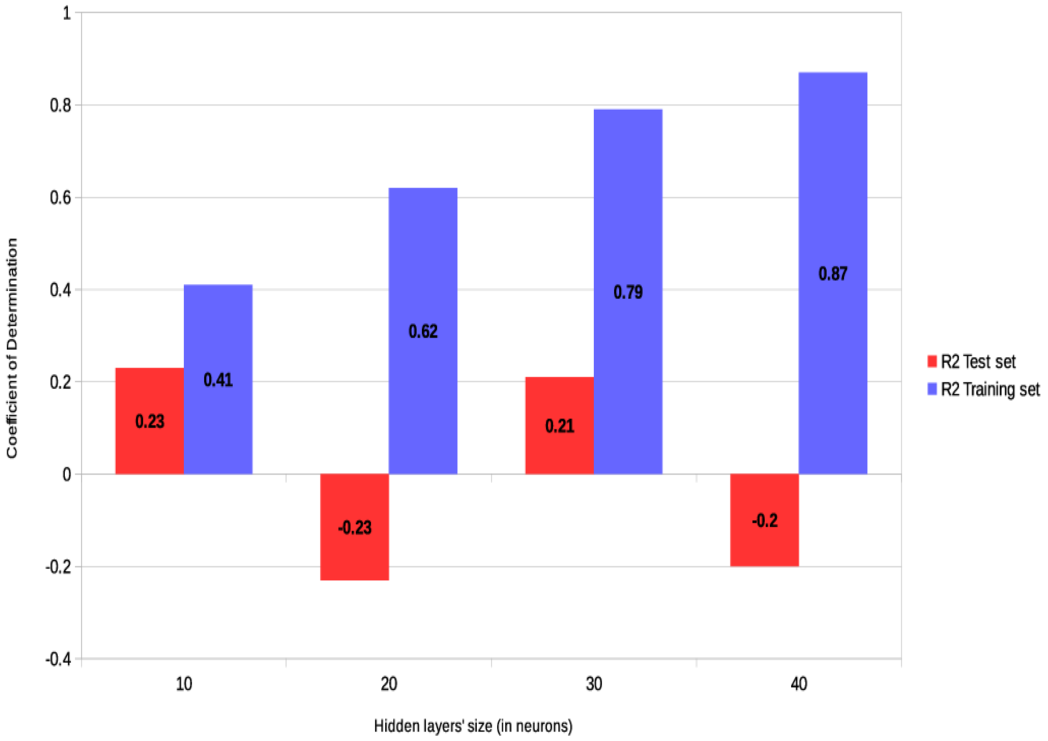

Figure 5-6: Regression model for primal method - Tuning the number of neurons (1 hidden layer)

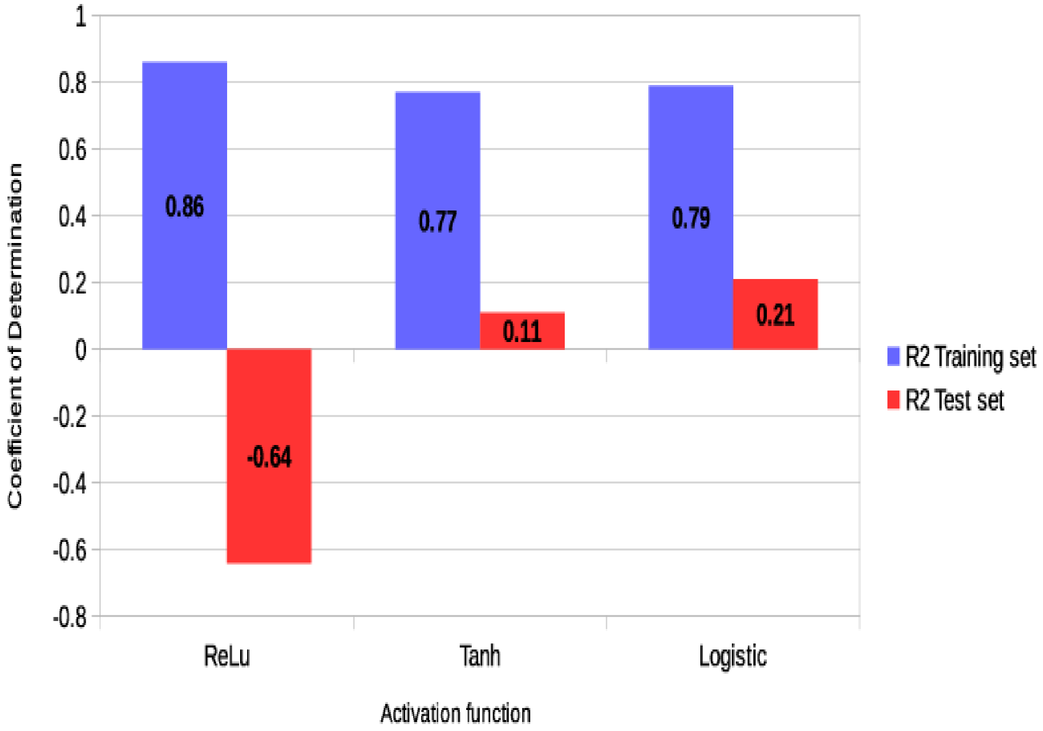

Figure 5-7: Regression model for primal method - Tuning the activation function

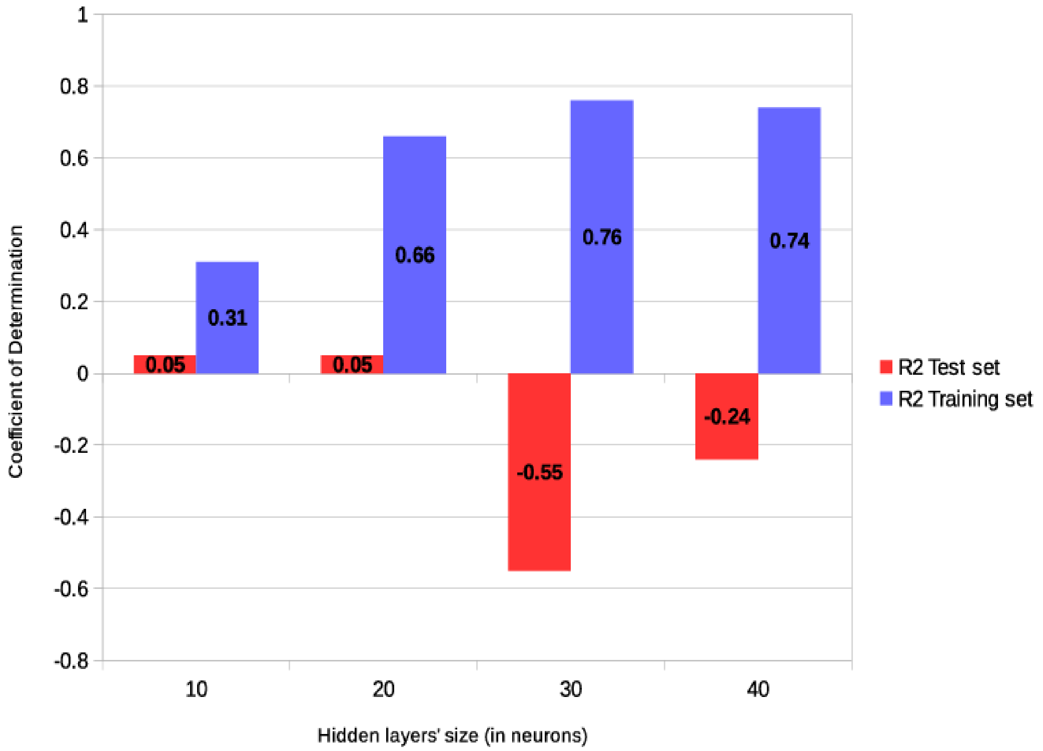

Figure 5-8: Regression model for dual method - Tuning the number of neurons (1 hidden layer)

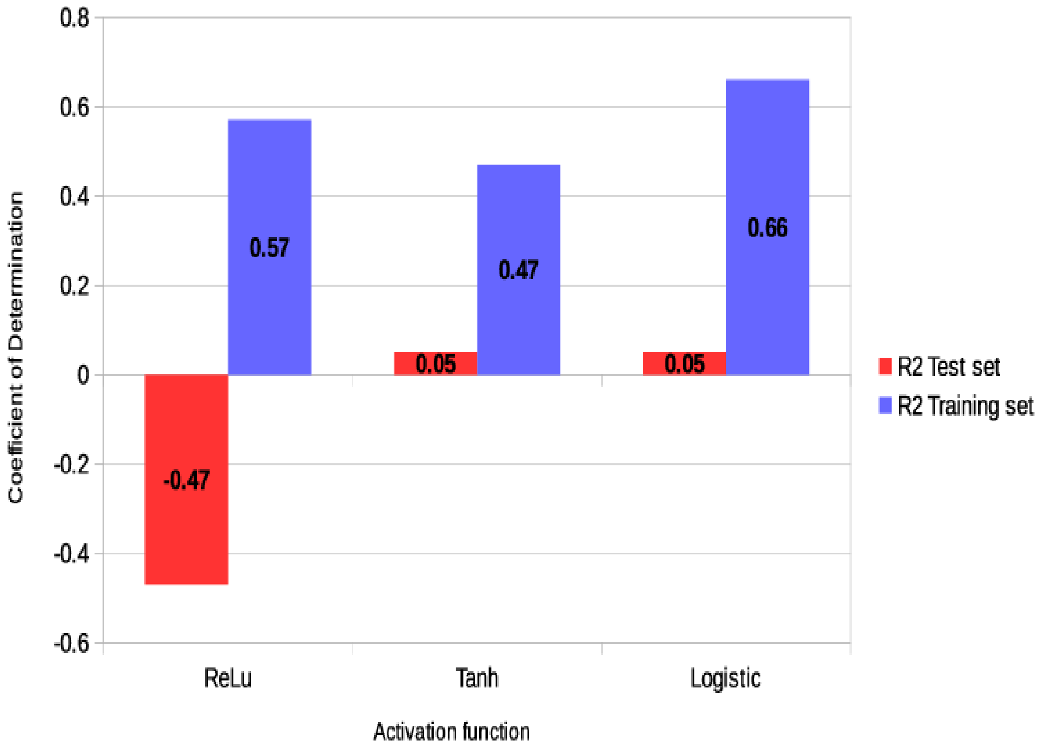

Figure 5-9: Regression model for dual method - Tuning the activation function

Elaborating further on the concept of regression, we extended our analysis to more regression algorithms, such as Decision Tree, ElasticNet, Lasso, Random Forest, Ridge, Support Vector and Linear Regressor. Although scikit's GridSearch function was utilized to identify the best regression model generated from each algorithm, the models that were eventually formulated, could not be used for prediction purposes. A remarkable exception was reported for Random Forest ANN model, which may result in better values of the evaluation metrics than MLPRegressor, but shows the same significant discrepancies between the training and test set. More specifically, the reported R^2 values of the training set for primal and dual simplex algorithms, show goodness of fit for the relevant Random Forest ANN models, in combination with the values of the rest metrics, as well. However, this fact cannot be validated through the test set, since the R^2 values decrease significantly, along with the rest error metrics' values which increase to a high extend, comparing to the ones of the training set. The values of the metrics, which were used to evaluate the respective ANN regression models for primal and dual simplex algorithms are included in Tables 5.11 and 5.12, separated between training and test set.

Table 5.11: Other Regression models for the execution time of the primal simplex algorithm

	Training set				Test set			
	RMSE	MAE	MedAE	R^2	RMSE	MAE	MedAE	R^2
Decision Tree	1441.9	31.77	26.02	0.01	2098.2	37.06	28.00	-0.03
ElasticNet	1483.2	32.11	24.28	0.07	1717.2	35.04	27.27	-0.01
Lasso	1470.6	32.24	35.39	0.05	1797.1	35.17	24.92	-0.002
Linear	1669.9	34.95	27.82	0.05	1177.7	29.90	24.39	-0.03
Random Forest	209.4	10.28	7.57	0.88	954.20	22.52	17.53	0.15
Ridge	1491.6	32.10	24.50	0.04	1809.10	36.41	28.00	-0.01
Support Vector	1663.0	27.55	17.89	-0.08	2138.28	31.92	18.46	-0.14

Table 5.12: Other Regression models for the execution time of the dual simplex algorithm

	Training set				Test set			
	RMSE	MAE	MedAE	R^2	RMSE	MAE	MedAE	R^2
Decision Tree	1125.2	28.02	23.90	0.02	998.6	26.98	25.90	-0.08
ElasticNet	1014.7	26.16	20.90	0.03	1206.8	27.90	20.76	0.04
Lasso	940.2	24.01	18.77	0.09	1327.0	28.93	20.92	-0.08
Linear	1040.6	25.87	17.59	0.10	1006.7	25.99	22.91	-0.10
Random Forest	148.4	8.68	6.77	0.87	618.9	17.29	13.12	0.34
Ridge	1033.3	26.21	19.78	0.11	997.9	24.84	18.60	-0.10
Support Vector	1299.2	21.26	8.06	-0.17	1356.96	24.15	12.20	-0.30

This outcome turned our focus more on classification techniques. Now, instead of trying to predict an exact value, such as the execution time of an algorithm, we are concentrating on predicting the class under which the value of the execution time may fall. As shown in the following sections, classification techniques work more efficiently with our dataset. We concluded that keeping the models that showed the best performance as prediction models for the execution time of the primal and dual simplex algorithms, would be neither a useful nor a valid choice.

Therefore, we tested two of the most commonly used classification algorithms supported by scikit-learn toolkit, such as Multi-layer Perceptron Classifier (MLPClassifier) and KNeighborsClassifier. Unlike other classification algorithms, such as Naive Bayes Classifier, MLPClassifier performs the task of classification, based on an underlying neural network. The process of classification using ANNs may seem theoretically complex and difficult to implement and interpret while it surely requires extensive testing to tune (offering a plethora of tuning options to prevent over- or under-fitting). However, this still cannot change the fact that it proves to be a powerful tool towards dealing with complex relations and functions that connect the examined input and output variables, while it is effective for high-dimensional problems. The parameters described earlier for MLPRegressor are also present in the use of MLPClassifier. Therefore, we proceeded with exhaustive testing of several models using different numbers of hidden layers and numbers of neurons, different activation functions, and solvers. The exact ranges of values are presented in Table 5.9. To measure the validity and accuracy of the models, which were generated by MLPClassifier and the rest classifiers, we analyzed the respective confusion matrix and the accuracy value of each model, along with the classification report that is created upon testing the model. In statistical classification and machine learning, a confusion matrix supports the visualization of a supervised learning algorithm's performance, by showing the instances in a predicted class in each row of the matrix and the instances in an actual class in each column of the matrix (or vice versa). The confusion matrix can show how many instances were mis-classified for each class. Accuracy is another significant metric for evaluating classification models, which, in general, can be considered as the amount of predictions that the examined model identified correctly. More specifically, accuracy could be defined through the following equation:

$$Accuracy = (Total\ number\ of\ correct\ predictions\ /\ Total\ number\ of\ predictions) \tag{5.5}$$

We could explain the concept of accuracy in classification problems with the help of some simple but really useful terms such as True and False Positives and True and False Negatives. A True Positive (TP) is an instance that exists in an actual class of our dataset and has also been correctly predicted by our examined model. A True Negative (TN) is an instance that does not exist in the actual class of our dataset and here again, it is also correctly predicted by our examined model. A False Positive (FP) is an instance that does not exist in the class, but our model has predicted it incorrectly, while a False Negative (FN) refers to an instance that exists in a class of the examined dataset, however, it is incorrectly predicted (i.e that it does not exist). As a fraction, which includes TP, TN, FP and FN amounts, accuracy could be expressed as follows:

$$\mathit{Accuracy} = (TP + TN) \;/\; (TP + TN + FP + FN) \tag{5.6}$$

However, it would not be safe to consider that the confusion matrix and accuracy value can stand alone as proof of the validity and good performance of the examined models, thus we proceeded with further analysis of the generated classification reports. A classification report includes the precision and recall values and the F_1 score and support scores of a classification model. Comparing to a plain accuracy value, we could say that the classification report offers a deeper understanding of the classifier's behavior and can also help select the most effective model for the examined dataset (for instance, the model with the "strongest" values of classification metrics). Before presenting the results of our models in this section, we are including a brief description of the metrics we used to compare our results. The precision value is representative of the classifier's ability to avoid marking a negative instance as positive. For each class of a given dataset, the precision is defined as the ratio of TPs to the sum of TPs and FPs, as shown below:

$$\mathit{Precision} = TP \;/\; (TP + FP) \tag{5.7}$$

Moving on, the recall value depicts the classifier's ability to find all positive instances. For each class of the examined dataset, recall is calculated by the ratio of TPs to the sum of TPs and FNs, as shown below:

$$\mathit{Recall} = TP \;/\; (TP + FN) \tag{5.8}$$

Furthermore, F_1 score is the harmonic mean of precision and recall, with its best value reaching 1 (i.e., perfect precision and recall) and its worst at 0. Although F_1 cannot be used alone to describe the accuracy of a classification model, it can certainly be a useful tool while comparing several models. Last but not least, the support value stands for the number of actual instances of each class in the examined dataset. This value provides us with a clear picture of the "balance" in our dataset, meaning how balanced the separation of the instances among the classes of our dataset is. Unbalanced training data may result in weaknesses in the reported scores of the classifier, which would result to the need for stratified sampling or even, rebalancing [42].

Proceeding with the classification models for the primal and dual Simplex algorithms, we are including detailed examples of our comparative analysis to select the most efficient models for the examined dataset. The selected model for the primal simplex algorithm uses *tanh* activation function, *lbfgs* solver, and 2 hidden layers of 100 neurons each. Similarly, the selected model for the dual simplex algorithm uses *lbfgs* solver and 2 hidden layers of 100 neurons each, with the only exception being the activation function, which is now *relu* instead of *tanh*. The execution time of primal and dual simplex algorithms has been separated in 4 classes, which are defined as shown in Table 5.13.

Table 5.13: Classes of the primal and dual simplex algorithms execution time (in seconds)

Class	Primal	Dual
0	$0 < \text{time} < 0.1$	$0 < \text{time} < 0.1$
1	$0.1 \leq \text{time} < 0.5$	$0.1 \leq \text{time} < 1$
2	$0.5 \leq \text{time} < 4$	$1 \leq \text{time} < 10$
3	$4 \leq \text{time}$	$10 \leq \text{time}$

Class 0 represents LPs that are easy to solve, with the time needed for their solution being less than 0.1 seconds, for both algorithms. Class 1 consists of LPs that are relatively easy to solve, with the execution time falling in a range of $0.1 - 0.5$ and $0.1 - 1$ seconds, for primal and dual simplex algorithms, respectively. In the same concept, Class 2 stands for the LPs that seem to require more time to solve (i.e., execution time for primal and dual simplex reported in ranges of $0.5 - 4$ and $1 - 10$ seconds, respectively). Finally, Class 3 consists of LPs that can be considered rather difficult and time-consuming, with the relevant execution time exceeding 4 and 10 seconds for primal and dual simplex algorithms, respectively. The classes were formulated after extensive sampling of the given dataset. We also experimented with different number of classes. The generated classification model for the execution time of the primal simplex algorithm reaches an accuracy value of 0.83, while the generated classification model for the execution time of the dual simplex algorithm has an accuracy value of 0.84.

The respective confusion matrices are available in Tables 5.14 and 5.15, while the classification reports in Table 5.16. It is shown that the model for the primal simplex algorithm mis-classifies only 2 instances in Class 0, 3 instances in Class 1 and Class 2, while 7 instances are mis-classified in Class 3. The model for the dual simplex algorithm classifies all 33 instances correctly in Class 0, mis-classifies 4 instances in Class 1 and 6 instances in Class 2, while Class 3 turns out to be the most challenging one with 4 instances mis-classified out of a total of 11 instances. The precision, recall, and F_1 scores for each examined class are quite satisfying, with the average scores confirming the accuracy of the generated models.

These results were extracted after extensive testing with several combinations of activation functions, solvers, number of hidden layers and neurons, and different classification algorithms (e.g., KNeighborsClassifier). The following figures include a graphical representation of examples from various tests which were performed and used for comparative analysis, before we reach the final models of this study. As shown, although we have an accuracy score of 0.88 in Figure 5-13, the respective model is not selected eventually. The reason for this is that the rest of its characteristics (precision, recall, F_1 score) indicate a unsuitable model for our dataset, i.e. precision and F_1 value are ill-defined and thus, they are set to 0.0 in labels with no predicted samples.

Table 5.14: Confusion matrix for the primal simplex algorithm execution time

		Actual Class			
		0	1	2	3
Predicted Class	0	22	2	0	0
	1	2	21	0	1
	2	0	0	20	3
	3	0	0	7	11

Table 5.15: Confusion matrix for the dual simplex algorithm execution time

		Actual Class			
		0	1	2	3
Predicted Class	0	33	0	0	0
	1	4	20	0	0
	2	0	3	15	3
	3	0	2	2	7

Table 5.16: Classification reports for the primal and dual simplex algorithms execution time

Class (Primal)	Precision	Recall	F_1	Support
0	0.92	0.92	0.92	24
1	0.91	0.88	0.89	24
2	0.74	0.87	0.80	23
3	0.73	0.61	0.67	18
avg/total	0.83	0.83	0.83	89

Class (Dual)	Precision	Recall	F_1	Support
0	0.89	1.00	0.94	33
1	0.80	0.83	0.82	24
2	0.88	0.71	0.79	21
3	0.70	0.64	0.67	11
avg/total	0.84	0.84	0.84	89

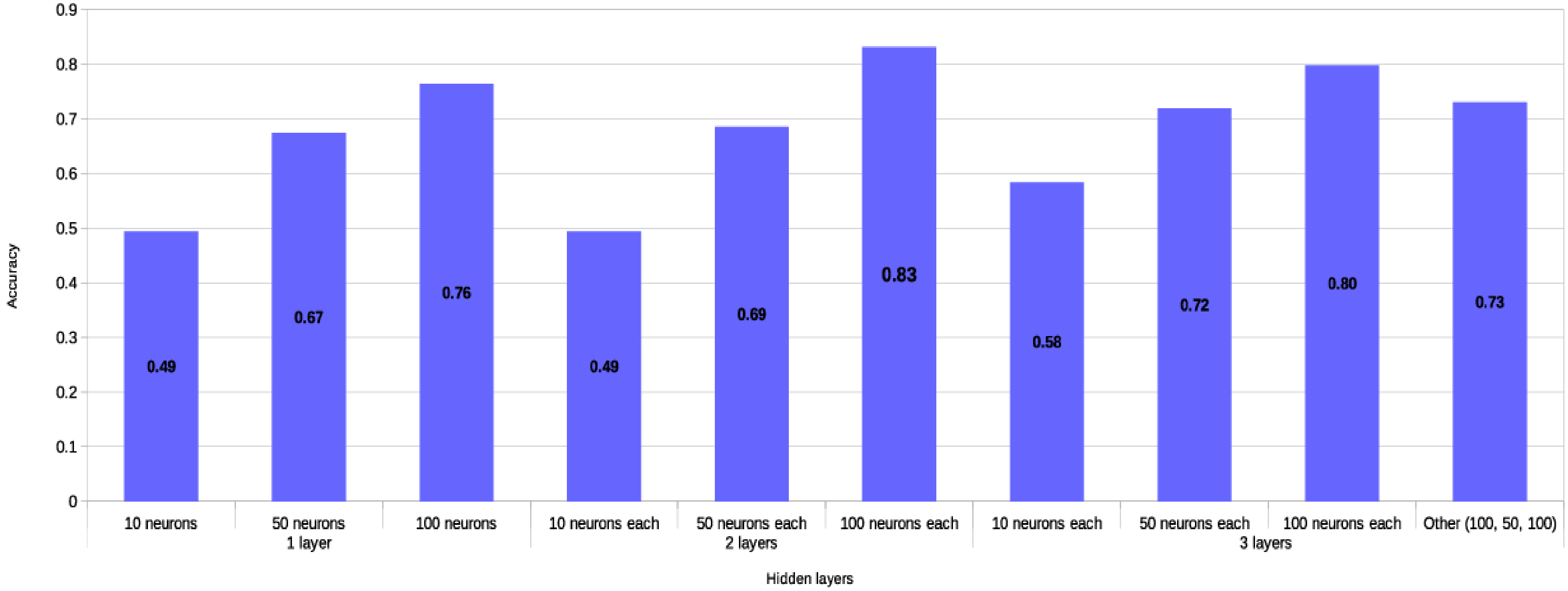

Figure 5-10: Classification model for primal method - Tuning the number of hidden layers and neurons (*tanh* activation function, *lbfgs* solver)

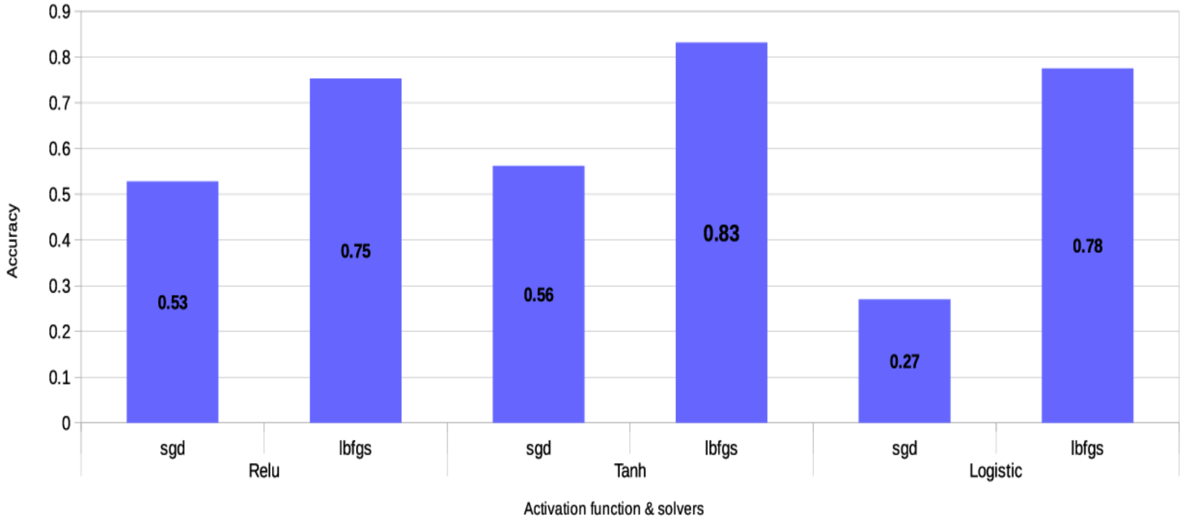

Figure 5-11: Classification model for primal method - Tuning the activation function and solver (2 hidden layers, 100 neurons each)

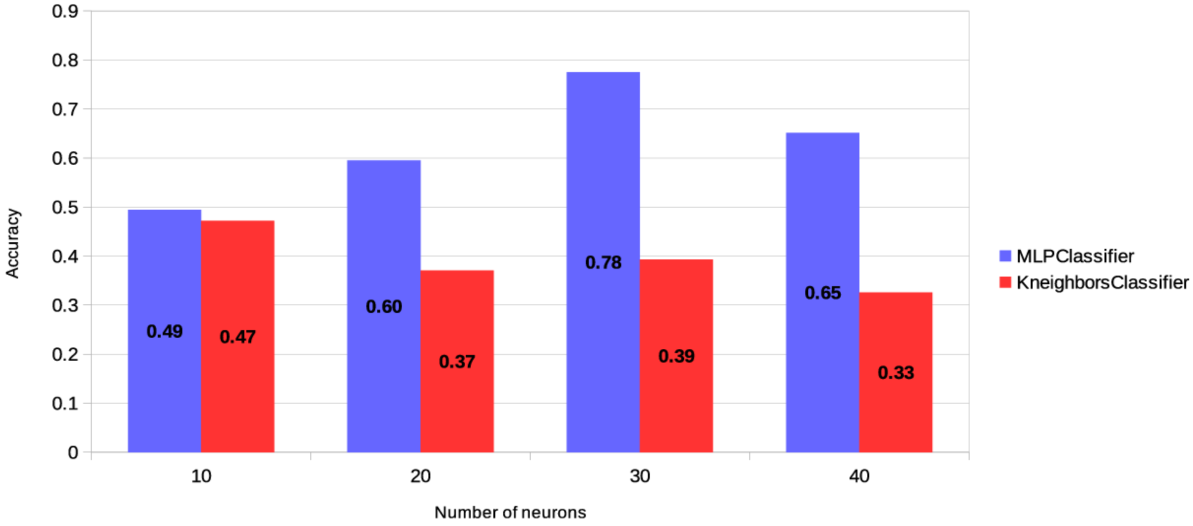

Figure 5-12: Classification model for primal method - Testing different classification algorithms (1 hidden layer, *tanh* activation function, *lbfgs* solver)

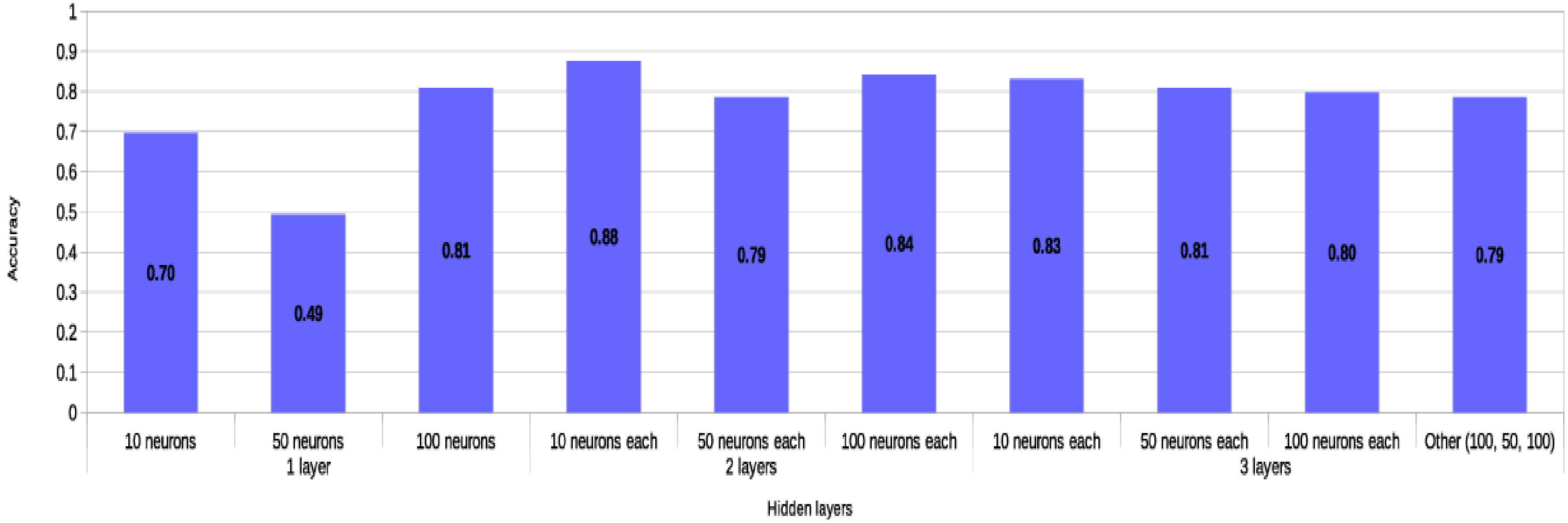

Figure 5-13: Classification model for dual method - Tuning the number of hidden layers and neurons (*relu* activation function, *lbfgs* solver)

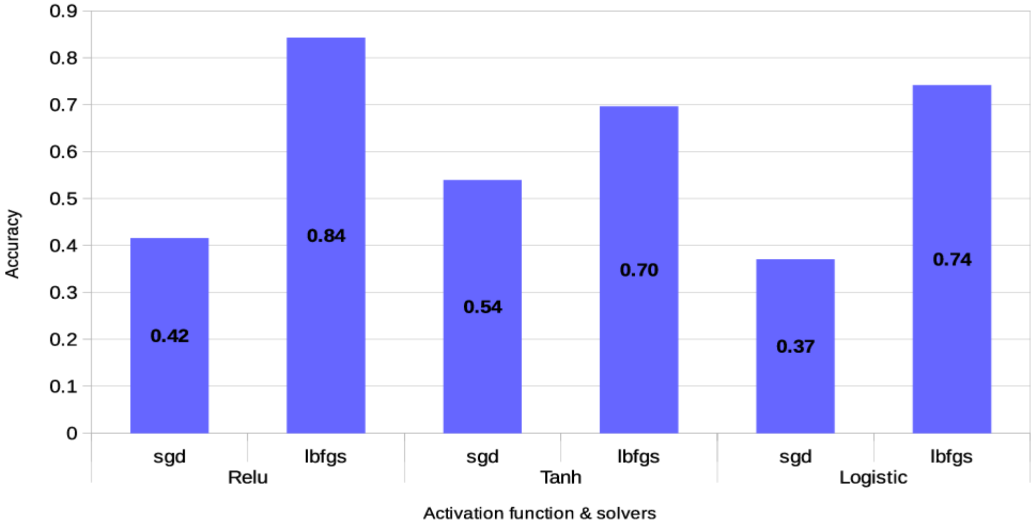

Figure 5-14: Classification model for dual method - Tuning the activation function and solver (2 hidden layers, 100 neurons each)

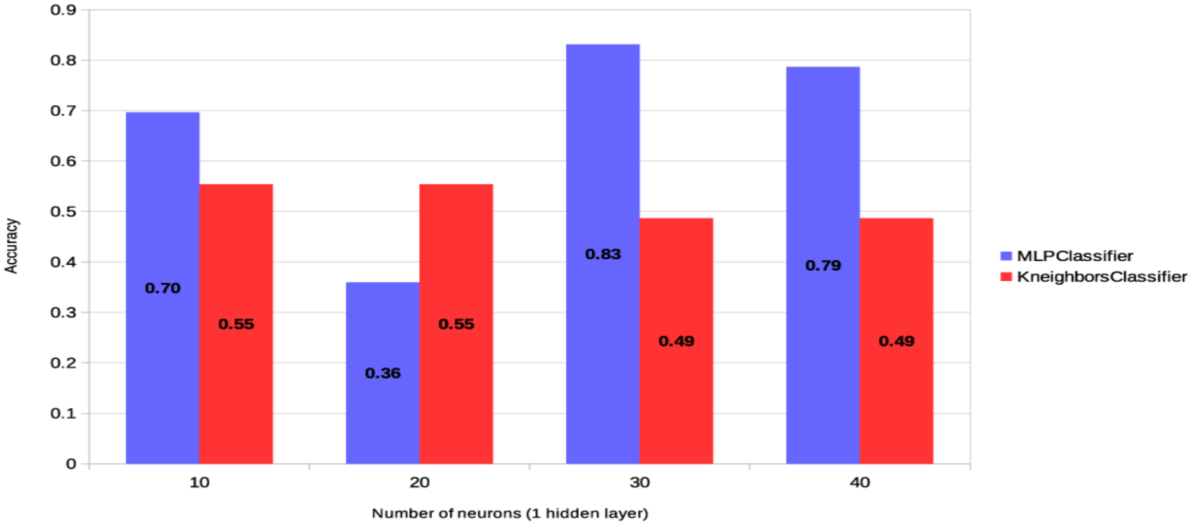

Figure 5-15: Classification model for dual method - Testing different classification algorithms

Chapter 6

Conclusions

In the final chapter of this thesis, we are summarizing our findings and providing specific suggestions for future work and additional research. As discussed, our main area of interest has been the computational complexity of four of the most commonly used Linear Programming Optimization algorithms; the Exterior Point Simplex algorithm (EPSA), the Interior Point Method and the Primal and Dual Simplex algorithms. For the analysis of the last three algorithms, we utilized the implementations supported by IBM CPLEX [9]. Our purpose has been the creation of representative statistical models, capable of predicting the computational performance of the examined algorithms for the solution of Linear Programming (LP) problems.

The predictive model for the computational performance of **Exterior Point Simplex algorithm** (EPSA) was created through extensive regression analysis and was distinguished in two specific models, based on the examined datasets; a) the randomly generated LP problems and b) the specific benchmark problems from *netlib*, *kennington*, *Mészáros* and *Mittelman* LP problem libraries. It is of major significance that the regression model for the randomly generated LP problems, proves the linear relation between the number of iterations that EPSA needs to perform for the solution of a given LP problem with specific LP problem attributes; namely the number of constraints (m) and variables (n), the problem sparsity and the condition of matrix A (*cond(A)*). Another important conclusion is that, for the examined benchmark problems, a respective predictive model, being statistically significant and representative of the examined dataset, was not possible to be created.

In our analysis for EPSA, the generated Normal Probability Plot of iterations standardized residuals, revealed a slight deviation from the diagonal line. If the residuals had been distributed normally, the values would fall exactly on the diagonal line. This deviation indicated a positive skewness of the distribution. Positive skewness was confirmed by our validation process and the respective histogram chart, where the right side tail of the curve was longer than the left side tail and the mean was greater than the mode. Skewness could be considered as the asymmetry of a distribution and can be quantified to measure the extent to which this distribution is distorted and how much it differs from a normal distribution.

The fact that there are many instances in the validation dataset, for which the number of iterations is relatively small, while in increasingly few instances the number of iterations increase significantly, reveals the existence of a specific pattern of characteristics in these extreme instances, which may eventually affect the practical performance of EPSA. This finding could be subject to further analysis in the future.

Apart from the above, though, that there are many aspects, which can be improved and offer a great opportunity for future study. Applying the above methodology to non-feasible algorithms as well, could provide some useful results about their practical performance in real-world applications. Furthermore, it was noticed that our algorithm tends to have a slightly worse than its usual good performance, when it comes to extreme conditions, as already explained. Therefore, it remains to be examined how can we improve this aspect of EPSA, in order to take advantage of its capabilities and achieve the best possible outcome for any LP problem given.

In the second part of our study, we focused on the problem of algorithm selection, as this concerns the majority of researchers using modern linear programming solvers. Most of these solvers support a heuristic procedure to select the most appropriate algorithm, based on the characteristic of the input linear programming problem. Through our analysis, we experimented with the use of a neural network for predicting the execution time of CPLEX's **Interior Point Method** (IPM). Our dataset consisted of a large pool of benchmark problems from the *netlib*, *kennington*, *Mészáros* and *Mittelman* LP problem libraries. The generated results showed that our model achieves an R^2 value of 78% for the training set and 72% for the test set. Taking into account the variability in the features of the examined benchmark LP problems and the metrics used for the comparison of the generated models, the current model proved to have a good fit on the data and thus, can be used for further prediction of the algorithm's efficiency.

Encouraged by the results for IPM, we extended our work by examining the use of neural networks for predicting the execution time of CPLEX's **Primal and Dual Simplex algorithms**. As explained, the results we received from the regression process were not satisfying enough to support a prediction model for the execution time of each algorithm. Thus, we further experimented with a classification approach, which led to meaningful results about the generated models. Through classification, we managed to form a predictive model about the class under which a specific problem can be classified. This piece of information alone provided us with sufficient insight about the time we will need for the solution of the problem, selecting one of the two examined Simplex methods. The accuracy of the our model for Primal Simplex reached a value of 0.83, while for Dual Simplex, 0.84. Moreover, acting as the main driving tool for our analysis, AI algorithms have been utilized to tackle one of the major questions in research community, when it comes to algorithm selection for the solution of linear programming problems, using one of the most widely used mathematical solvers. Thus, we believe that this study makes a significant contribution, outlining the importance and necessity of AI algorithms in solver tuning field.

In future work, it would be crucial to continue building prediction models for more LP Optimization algorithms, supported by widely used mathematical solvers, using supervised learning methods. The models could be formed after a extensive study of several model parameters (i.e. number of hidden layers, number of neurons, activation functions, scaling and normalization techniques), similarly to the process followed for our analysis. Even if this effort seems big, building accurate models for the prediction of the execution of LP Optimization algorithms will enable any linear programming solver to select the most efficient algorithm for a given LP problem. This step will open new ways for remarkable time and cost savings in solving linear programming problems.

Bibliography

[1] Netlib Repository. Netlib LP problems. http://www.netlib.org/lp/data. Online; Last accessed.

[2] Netlib Repository. Kennington LP problems. http://www.netlib.org/lp/data/kennington. Online; Last accessed.

[3] Netlib Repository. infeasible section of Netlib LP problems. http://www.netlib.org/lp/infeas. Online; Last accessed.

[4] Mészáros C. Linear Programming Test Problems. *Miscellaneous* section. $http : //www.sztaki.hu/*meszaros/public_{f}tp/lptestset/misc$. Online; Last accessed on 05 Oct 2016.

[5] Mészáros C. Linear Programming Test Problems. *Problematic* section. $http : //www.sztaki.hu/*meszaros/public_{f}tp/lptestset/problematic$. Online; Last accessed on 05 Oct 2016.

[6] Mészáros C. Linear Programming Test Problems. *Stochlp* section. $http : //www.sztaki.hu/*meszaros/public_{f}tp/lptestset/stochlp$. Online; Last accessed on 05 Oct 2016.

[7] Mittelman H. Mittelman LP problems. $http : //plato.asu.edu/ftp/lptestset$. Online; Last accessed on 05 Oct 2016.

[8] Netlib Benchmark LP problems. http://www.netlib.org/benchmark. Online; Last accessed on 05 Oct 2016.

[9] IBM ILOG CPLEX: Cplex 12.6.0 user manual. http://www-01.ibm.com/support/knowledgecenter/SSSA5P _12.6.1/ilog.odms.studio.help/Optimization_Studio/topics/COS_home.html?lang=en (2017). Online.

[10] Mathworks MATLAB. http://www.mathworks.com/products/matlab. Online; Last accessed on 05 Oct 2016.

[11] Minitab. http://www.minitab.com/en-us/academic. Online. Last accessed on 05 Oct 2016.

[12] NIST, National Institute of Standards and Technology. http://www.nist.gov. Online. Last accessed on 5 Oct 2016.

[13] C. Aggarwal. *Neural Networks and Deep Learning. A Textbook.* Springer International Publishing, 2018.

[14] M. Amaris, R.Y. de Camargo, M. Dyab, A. Goldman and D. Trystram. A comparison of GPU execution time prediction using machine learning and analytical modeling. In Proceedings of the 2016 IEEE 15th International Symposium on Network Computing and Applications (NCA), Cambridge, MA, USA, 31 October–2 November, 2016; pp. 326–333.

[15] R. Amarasingham, P. Patel, K. Toto, L. Nelson, T. Swanson, B. Moore, B. Xie, S. Zhang, K. Alvarez, Y. Ma, M. Drazner, U. Kollipara, and E. Halm. Allocating scarce resources in real-time to reduce heart failure readmissions: A prospective, controlled study. *BMJ Quality and Safety*, 22(12):998–1005, 2013.

[16] C. Audet and D. Orban. Finding optimal algorithmic parameters using derivative-free optimization. *SIAM Journal on Optimization*, 17(3):642–664, 2006.

[17] L. Barrault, O. Bojar, M. Costa-jussà, C. Federmann, M. Fishel, Y. Graham, B. Haddow, M. Huck, P. Koehn, S. Malmasi, and C. Monz. Findings of the 2019 conference on machine translation (wmt19). In *Proceedings of the Fourth Conference on Machine Translation (Volume 2: Shared Task Papers, Day 1). Florence, Italy: Association for Computational Linguistics*, pages 1–61, 2019.

[18] M. Barry, H. Abgottspon, and R. Schumann. Solver tuning and model configuration. In *Joint German/Austrian Conference on Artificial Intelligence (Künstliche Intelligenz)*, pages 141–154, Cham, 2018. Springer.

[19] M. Baz and B. Hunsaker. Automated tuning of optimization software parameters. technical report. Technical report, Department of Industrial Engineering, University of Pittsburgh, Pittsburgh, PA, 2007.

[20] M. Baz, B. Hunsaker, and O. Prokopyev. How much do we "pay" for using default parameters? *Comput. Optim. Appl.*, 48:91–108, 2011.

[21] E. Berenguer and L. Smith. The expected number of extreme points of a random linear program. *Mathematical Programming*, 35:129–134, 1986.

[22] K. Borgwardt. The average number of pivot steps required by the simplex-method is polynomial. *Mathematical Methods of Operations Research*, 26(1):157–177, 1982.

[23] K. Borgwardt. Some distribution-independent results about the asymptotic order of the average number of pivot steps of the simplex method. *Mathematics of Operations Research*, 7(3):441–462, 1982.

[24] L. Bottaci, P. Drew, J. Hartley, M. Hadfield, R. Farouk, P. Lee, I. Macintyre, G. Duthie, and J. Monson. Artificial neural networks applied to outcome prediction for colorectal cancer patients in separate institutions. *The Lancet*, 350(9076):469–472, 1997.

[25] P. Brazdil, C.G. Carrier, C. Soares, and R. Vilalta. *Metalearning: Applications to Data Mining*; Springer-Verlag: Berlin Heidelberg, 2009.

[26] O. Bretscher. *Linear Algebra With Applications (3rd ed.)*. Upper Saddle River, NJ: Prentice Hall, 1995.

[27] R. Bronson and G. Naadimuthu. *Schaum's Outline of Operations Research 2nd Edition*. Schaum's Outlines. McGraw-Hill Education; 2 edition, second edition, 22 July 1997. Greek edition by Pr. N. Samaras.

[28] C. McGeoch C, P. Sanders, R. Fleischer, PR. Cohen PR, and D. Precup. *Using finite experiments to study asymptotic performance*, volume 2547 of *Fleischer R., Moret B., Schmidt E.M. (eds) Experimental Algorithmics. Lecture Notes in Computer Science*, pages 93–126. Springer, Berlin, Heidelberg, 2002.

[29] J. Castle, J. Doornik, and D. Hendry. Evaluating automatic model selection. *Journal of Time Series Econometrics*, 3(1):1–33, 2011.

[30] W. Chen, Z. Shao, K. Wang, X. Chen, and L. Biegler. Random sampling-based automatic parameter tuning for nonlinear programming solvers. *Industrial & Engineering Chemistry Research*, 50(7):3907–3918, 2011.

[31] A-L. Cholesky. Note Sur Une Méthode de Résolution des équations Normales Provenant de L'Application de la MéThode des Moindres Carrés a un Système D'équations Linéaires en Nombre Inférieur a Celui des Inconnues. Application de la Méthode a la Résolution d'un Système Defini d'équations linéaires. *Géodésique 2*, pp. 67–77, 1924.

[32] D. C. Cireşan, U. Meier, and J. Schmidhuber. Transfer learning for latin and chinese characters with deep neural networks. In *The 2012 International Joint Conference on Neural Networks (IJCNN)*, 2012.

[33] D. Cook. Detection of influential observations in linear regression. *Technometrics. American Statistical Association*, 19(1):15–18, 1977.

[34] D. Cook. Influential observations in linear regression. *Journal of the American Statistical Association. American Statistical Association*, 74(365):169–174, 1979.

[35] G. Dantzig. Linear programming and extensions. 1963.

[36] G. B. Dantzig. Programming in a linear structure. technical report. Technical report, Comptroller, US Air Force, Washington, DC., 1948.

[37] G. B. Dantzig. Programming of interdependent activities: Ii, mathematical model. pages 200–211, 1949.

[38] D. den Hertog. *Interior Point Approach to Linear, Quadratic and Convex Programming: Algorithms and Complexity*, volume 277 of *Mathematics and Its Applications*. Springer Netherlands, 1994.

[39] N. Draper and H. Smith. *Applied regression analysis, 3rd edn.* Wiley Series in Probability and Statistics. Wiley, New York, 1998.

[40] J. Durbin and G. Watson. Testing for serial correlation in least squares regression. *I. Biometrika*, 37(3,4):409–428, 1950.

[41] J. Erickson. Lecture Notes on Linear Programming (https://courses.engr.illinois.edu/cs498dl1/sp2015/notes/27-simplex.pdf) - Online; Last accessed.

[42] T. Fawcett. An introduction to roc analysis. *Pattern Recognition Letters*, 27(8):861–874, 2006.

[43] S. Finlay. *Predictive Analytics, Data Mining and Big Data. Myths, Misconceptions and Methods.* Business in the Digital Economy. Palgrave Macmillan UK, 2014.

[44] R. A. Fisher. On the interpretation of χ^2 from contingency tables and the calculation of p. *Journal of the Royal Statistical Society*, 85(1):87–94, 1922.

[45] R. A. Fisher. On the mathematical foundations of theoretical statistics. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, 222:309–368, 1922.

[46] J-B. J. Fourier. Histoire de l'academie royale des sciences de l'institut de france. *Analyse des travaux de I'Acadamie Royale des Sciences pendant I'annee 1824 Partie mathematique*, 7:xlvii–lv, 1827.

[47] A. Franzin, L. Pérez Cáceres, and T. Stützle. Effect of transformations of numerical parameters in automatic algorithm configuration. *Optim Lett*, 12:1741–1753, 2018.

[48] J. Gondzio. Interior point methods 25 years later. *European Journal of Operational Research*, 218(3):587–601, 2012.

[49] J. Gondzio and A. Grothey. Direct solution of linear systems of size 10^9 arising in optimization with interior point methods. In N. Meyer R. Wyrzykowski, J. Dongarra and J. Wasniewski, editors, *Parallel Processing and Applied Mathematics*, number 3911 in Lecture Notes in Computer Science, pages 513–525, Berlin, 2006. Springer-Verlag.

[50] J. A. J. Hall. Towards a practical parallelisation of the simplex method. *Computational Management Science*, 7:139–170, 2010.

[51] J. A. J. Hall and K. I. M. McKinnon. Hyper-sparsity in the revised simplex method and how to exploit it. *Computational Optimization and Applications*, 32:259–283, 2005.

[52] S. Haykin. *Neural Networks and Learning Machines.* Pearson; 3 edition, 2011.

[53] J. K. Ho and R. P. Sundarraj. A timing model for the revised simplex method. *Operations Research Letters*, 13:67–73, 1993.

[54] D. Hosmer, B. Jovanovic, and S. Lemeshow. Best subsets logistic regression. *Biometrics*, 45(4):1265–1270, 1989.

[55] F. Hutter, H. Hoos, and K. Leyton-Brown. Automated configuration of mixed integer programming solvers. In *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*, pages 186–202, Cham, 2010. Springer.

[56] F. Hutter, H. Hoos, K. Leyton-Brown, and T. Stützle. Paramils: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36:267–306, 2009.

[57] R. Hyndman and A. Koehler. Another look at measures of forecast accuracy. *International Journal of Forecasting*, 22(4):679–688, 2006.

[58] A. K. Jain and B. Chandrasekaran. Dimensionality and sample size considerations in pattern recognition practice. *Handb. Stat*, 39(2):835–855, 1982.

[59] B. Jain and B. Nag. Performance evaluation of neural network decision models. *Journal of Management Information Systems*, 14(2):201–216, 1997.

[60] L. Kaelbling, M. Littman, and A. Moore. Reinforcement learning. a survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[61] L. V. Kantorovich. Mathematical methods of organising and planning production. 1939. Translated in Management Science, vol.6, no.4, 366-422, 1960.

[62] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–395, 1984.

[63] T. Kavzoglu and P. M. Mather. The use of backpropagating artificial neural networks in land cover classification. *Int. J. Rem. Sens.*, 24(23):4907–4983, 2003.

[64] L. G. Khachiyan. A polynomial algorithm in linear programming. *Doklady Akademii Nauk SSSR*, 244:1093–1096, 1979. Translated in Soviet Mathematics Doklady 20, 191-194, 1979.

[65] V. Klee and G. J. Minty. *How good is the simplex algorithm.* Academic Press, New York and London, 1972.

[66] S. Krishnaswamy, S. Loke, and A. Zaslavsky. Estimating computation times of data-intensive applications. *IEEE Distributed Systems Online*, 5(4), 2004.

[67] M. Kutner, J. Neter, C. Nachtsheim, and W. Wasserman. *Applied linear statistical models, 5th edn.* McGraw-HillfIrwin series Operations and decision sciences. McGraw-Hill, New York, 2004.

[68] J. Liu, N. Ploskas, and N. Sahinidis. Tuning baron using derivative-free optimization algorithms. *Journal of Global Optimization*, 74(4):611–637, 2019.

[69] A. Makam, O. Nguyen, B. Moore, Y. Ma, and R. Amarasingham. Identifying patients with diabetes and the earliest date of diagnosis in real time: an electronic health record case-finding algorithm. *BMC Medical Informatics and Decision Making*, 13(81), 2013.

[70] I. Maros and M. Khaliq. Advances in design and implementation of optimization software. technical report. Technical report, Imperial College, London, 1999.

[71] I. Maros and M. H. Khaliq. Advances in design and implementation of optimization software. *European Journal of Operational Research*, 140(2):322–337, 1999.

[72] A. Matsunaga and J.A. Fortes. On the use of machine learning to predict the time and resources consumed by applications. In Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, Melbourne, VIC, Australia, 17–20 May, 2010; pp. 495–504.

[73] W. McCulloch and W. Pitts. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5(4):115–133, 1943.

[74] N. Megiddo. *On the complexity of linear programming*, pages 225–268. T. Bewley (Ed.), Advances in Economic Theory: Fifth World Congress (Econometric Society Monographs). Cambridge: Cambridge University Press, 1987.

[75] S. Mehrotra. On the implementation of a primal-dual interior point method. *SIAM Journal on Optimization*, 2(4):575–601, 1992.

[76] F. Ordó nez and R. Freund. Computational experience and the explanatory value of condition measures for linear optimization. *SIAM Journal on Optimization*, 14(2):307–333, 2003.

[77] K. Paparrizos. An infeasible (exterior point) simplex algorithm for assignment problems. *Mathematical Programming*, 51(1):45–54, 1991.

[78] K. Paparrizos. An exterior point simplex algorithm for (general) linear programming problems. *Annals of Operations Research*, 46-47(2):497–508, 1993.

[79] K. Paparrizos, N. Samaras, and G. Stephanides G. A method for generating random optimal linear problems and a comparative computational study. In *Proceedings of the 13th national conference of the Hellenic operational research society, Piraeus (in Greek)*, pages 785–794, 2000.

[80] K. Paparrizos, N. Samaras, and G. Stephanides. An efficient simplex type algorithm for sparse and dense linear programs. *European Journal of Operational Research*, 148(2):323–334, 2003.

[81] K. Paparrizos, N. Samaras, and C. Triantafyllidis. A computational study of exterior point simplex algorithm variations. In *Proceedings of 20th Hellenic Operational Research Society*, 2008.

[82] K. Paparrizos, N. Samaras, and K. Tsiplidis. Some results on the finiteness of an exterior point simplex algorithm. In *Proceedings of the 3rd Balkan Conference on Operations Research*, 1995.

[83] K. Pearson. Notes on regression and inheritance in the case of two parents. In *Proceedings of the Royal Society of London*, number 58, pages 240–242, 1895.

[84] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[85] I. Pietri, G. Juve, E. Deelman, and R. Sakellariou. A performance model to estimate execution time of scientific workflows on the cloud. In Proceedings of the 2014 9th Workshop on Workflows in Support of Large-Scale Science, New Orleans, LA, USA, 16–21 November, 2014; pp. 11–19.

[86] N. Ploskas and N. Samaras. Pivoting rules for the revised simplex algorithm. *Yugoslav Journal of Operations Research*, 24(3):321–332, 2014.

[87] N. Ploskas and N. Samaras. *Linear Programming Using MATLAB®*. Springer Optimization and Its Applications. Springer International Publishing, 2017.

[88] F. Potra and S. J. Wright. Interior-point methods. *Journal of Computational and Applied Mathematics*, 124(1-2):281–302, 2000.

[89] R. Priya, B.F. de Souza, A.L. Rossi, and A.C. de Carvalho. Predicting execution time of machine learning tasks using metalearning. In Proceedings of the 2011 World Congress on Information and Communication Technologies, Mumbai, India, 11–14 December, 2011; pp. 1193–1198.

[90] C. Rao. *Linear statistical inference and its applications, 2nd edn.* Wiley Series in Probability and Statistics. Wiley, New York, 1973.

[91] C. Rao. and J. Rawlings Linear Programming and Model Predictive Control. *Journal of Process Control*, 10:283–289, 2000.

[92] S. J. Raudys and A. K. Jain. Small sample size effects in statistical pattern recognition: recommendations for practitioners. *IEEE Trans. Pattern Anal. Mach. Intell.*, 13(3):252–264, 1991.

[93] D. Rumelhart and J. McClelland. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations.* MIT Press, 1987.

[94] D. Rumelhart and J. McClelland. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Psychological and Biological Models.* MIT Press, 1987.

[95] N. Samaras. Lecture Notes on Linear and Network Programming (http://opencourses.uom.gr/courses/efarmosmenhs-plhroforikhs/576-grammikos-diktyakos-programmatismos/enothtes) - Online; Last accessed.

[96] N. Samaras and A. Sifaleras. A comparative computational study of exterior point algorithms for the assignment problem. In *Proceedings of 19th national conference of Hellenic Operational Research Society (HELORS)*, 2007.

[97] N. Samaras, A. Sifaleras, and C. Triantafyllidis. A primal–dual exterior point algorithm for linear programming problems. *Yugoslav Journal of Operations Research*, 19(1):123–132, 2009.

[98] J. Schmidhuber. Learning complex, extended sequences using the principle of history compression. *Neural Computation*, 4:234–242, 1992.

[99] J. Schmidhuber. Deep learning. *Scholarpedia*, 10(11):85–117, 2015.

[100] I. N. Silva, D. H. Spatti, R. A. Flauzino, L. H. Bartocci Liboni, and S. F. dos Reis Alves. *Artificial Neural Networks. A Practical Course.* Springer International Publishing, 2017.

[101] W. Smith. Prediction services for distributed computing. In *IEEE International Parallel and Distributed Processing Symposium*, pages 1–10, 2007.

[102] W. Smith, I. Foster, and V. Taylor. Predicting application run times with historical information. *Journal of Parallel and Distributed Computing*, 64(9):1007–1016, 2004.

[103] D. Spielman and SH. D. Teng. Smoothed analysis of termination of linear programming algorithms. *Math. Program., Ser. B 97*, 375–404, 2003.

[104] D. Spielman and SH. D. Teng. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *Journal of the ACM*, 51(3):385–463, 2004.

[105] D. Spielman and SH. D. Teng. Smoothed analysis: an attempt to explain the behavior of algorithms in practice. *Communications of the ACM*, 52(10):76–84, 2009.

[106] S. M. Stigler. Gauss and the invention of least squares. *Ann. Stat.*, 9(3):465–474, 1981.

[107] S. M. Stigler. Francis galton's account of the invention of correlation. *Statistical Science*, 4(2):73–79, 1989.

[108] Student. The probable error of a mean. *Biometrika*, 6(1):1–25, 1908.

[109] J. Sun, G. Sun, S. Zhan, J. Zhang, and Y. Chen. Automated performance modeling of HPC applications using machine learning. *IEEE Trans. Comput.* **2020**, 69:749–763.

[110] T. Terlaky and S. Zhang. Pivot rules for linear programming: A survey on recent theoretical developments. *Annals of Operations Research*, 46(1):203–233, 1993.

[111] M. E. Thomadakis. Implementation and evaluation of primal and dual simplex methods with different pivot-selection techniques in the lpbench environment, a research report. Technical report, Texas A & M University, 1994.

[112] C. Triantafyllidis and N. Samaras. Three nearly scaling invariant versions of an exterior point algorithm for linear programming. *Optimization: A Journal of Mathematical Programming and Operations Research*, 64(10):2136–2181, 2015.

[113] D. Tsafrir, Y. Etsion, and D. Feitelson. Backfilling using system-generated predictions rather than user runtime estimates. *IEEE Transactions on Parallel and Distributed Systems*, 18(6):789–803, 2007.

[114] R. J. Vanderbei. *Linear Programming: Foundations and Extensions, Fourth Edition.* International Series in Operations Research and Management Science. Springer US, fourth edition, 2014.

[115] I. Vlahavas, P . Kefalas, N . Bassiliades, F . Kokkoras, and . Sakellariou. *Artificial Intelligence - 3rd edition.* University of Macedonia Press, 2011.

[116] J. von Neumann. Zur theorie der gesellschaftsspiele. *Math. Ann.*, 100:295–320, 1928.

[117] J. von Neumann and O. Morgenstern. *The Theory of Games and Economic Behavior.* Princeton University Press, 1944.

[118] J. Vanschoren. Meta-learning: A survey. *arXiv* **2018**, arXiv:1810.03548.

[119] P. J. Werbos. Applications of advances in nonlinear sensitivity analysis. In *Drenick R.F., Kozin F. (eds) System Modeling and Optimization. Lecture Notes in Control and Information Sciences, vol 38. Springer, Berlin, Heidelberg*, 1982.

[120] D.H. Wolpert and W.G. Macready. *No Free Lunch Theorems for Search.* Technical Report; Technical Report SFI-TR-95-02-010; Santa Fe Institute:1399 Hyde Park Road, Santa Fe, NM, 87501, 1995.

[121] S. J. Wright. *Primal-Dual Interior-Point Methods.* Society for Industrial and Applied Mathematics, 3600 University City Science Center Philadelphia, PA, United States, 1997.

[122] O. Özaltın, B. Hunsaker, and A. Schaefer. Predicting the solution time of branch-and-bound algorithms for mixed-integer programs. *INFORMS Journal on Computing*, 23(3):392–403, 2011.